



## **BTS 5110 Group Project StarAIS**

### **GROUP 12**

**(A0155573Y) Goh Dai Yong, Adison**

**(A0280519B) Rizon Agustan Sinaga**

**(A0280536A) Nur Iryani Binti Halip**

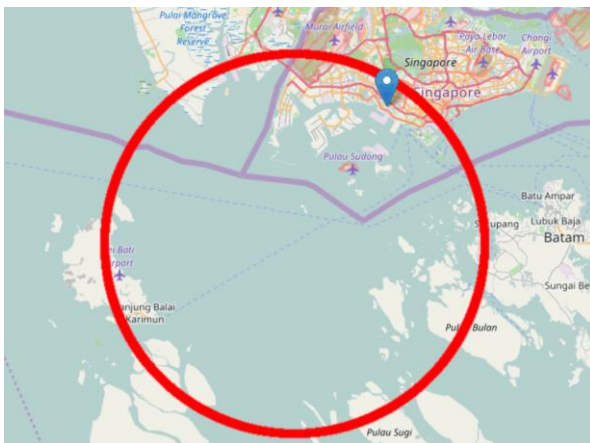
**(A0280538Y) Gary Wong Yue Whay**

## **BACKGROUND**

From 2002, the International Maritime Organisation (IMO) requested that vessels over 300GT and all passenger vessels install the Automatic Identification System (AIS). AIS information, which includes dynamic ship information (e.g., position, speed over ground, course over ground, rate of turn) and static information (e.g., Maritime Mobile Service Identity (MMSI) code, vessel name, ship type), is broadcast through very high-frequency signals. The AIS has greatly helped to promote ship navigation safety.

For this project, we have used historical AIS information received from an antenna position at School of Computing, National University of Singapore. The red circle in Figure 1 roughly delimits the covered zone which is primarily the western part of the Singapore Strait. We approximated the coordinates bound by the covered zone to be Latitude between 0.83 – 1.39 and Longitude between 103.38 – 103.94 (rounded to 2 decimal place).

*Figure 1. Covered zone of the AIS information*



## **OBJECTIVE**

The primary aim of this project is to design and populate a database warehouse for AIS messages and build an interface to query the data warehouse. The AIS database warehouse serves as a structured data environment that is optimised for aggregation (“roll-up”) and de-aggregation (“drill down”) of information along any specific dimension (i.e., “slicing and dicing”). The end goal is to enable business users to easily analyse the AIS data and reap maximum value from the large volumes of AIS data.

## **REPORT STRUCTURE**

We structured our report according to Kimball’s 4 Steps Dimensional Design Process:

1. We will first identify the business process and outline the business objectives of the AIS database warehouse.
2. Next, we will explain the choice of AIS scope and steps taken to stage and clean the raw AIS data – this will establish the “grain” of the business process (as Step 2 of Kimball’s 4 Steps Dimensional Design Process).
3. In step 3 of Dimensional Design Process, we will explain the dimension tables and population process.
4. As the last step of Dimensional Design Process, we outline the facts and measures in our relational fact table.
5. We will then present our Star Schema and an illustration of the multi-dimensional data cube of our schema.
6. Lastly, we will propose 4 business questions with 9 sub-queries on the database warehouse to address the business objectives and demonstrate the use of an interface to access and query the database warehouse.

## **1. BUSINESS OBJECTIVE OF AIS DATABASE WAREHOUSE**

Our client (Maritime Port Authority of Singapore) would like to leverage the AIS database warehouse to understand the marine traffic in Singapore waters (Straits of Singapore, Straits of Malacca, South China Sea). Since larger vessels account for most of the marine traffic in Singapore waters, they would like to focus on larger vessels (i.e., Passenger Ships, Tankers and Cargo). Their top 4 business questions (which we will cover in “6. SQL database queries & user interface” section) are:

- What is the impact of COVID-19 on maritime traffic in Singapore waters?
- What is the maritime traffic density in Singapore waters and ports for safety enhancement?
- How “Green” are the vessels in Singapore waters (as part of SG Green 2030 Maritime target status check)?
- How to track the movement of a particular vessel in Singapore waters?

## **2. AIS DATA SCOPE & DATA STAGING / CLEANING**

### **a. AIS DATA SCOPE**

The downloaded AIS messages included 4 months of data – Dec 19, Jan 20, Dec 20, Jan 21 consisting of a total of 23.77 million messages. This choice is to cover the following time periods:

- Pre-CV19 (Dec 19 & Jan 20) vs CV-19 (Dec 20 & Jan 21) – as this is a focus for our client.
- Holiday (Dec) vs Post-holiday (Jan) trend

Within these messages, there are:

- 21.08 million (or 88.68%) of message types 1, 2 and 3 and the remaining 2.69 million (or 11.32%) were message types 5, 18 and 19.
  - Message types 1,2 and 3 provided dynamic data on vessel positions from Class A type vessels.
  - Message type 5 contained static data on vessel information from Class A type vessels.
  - Message types 18 and 19 contained dynamic data on vessel positions from Class B type vessels.
- We will focus on message types 1, 2 and 3 for vessel positions and leverage on message type 5 for basic static vessel information. Additionally, we will focus on ship types 60 – 69 (Passenger ship), 70 – 79 (Cargo) and 80 – 89 (Tanker) as aligned with our client’s focus on larger vessels in our data staging.

### **b. DATA STAGING AND CLEANING STEPS**

1. The individual JSON data files for each month were first unzipped and imported into Python.
2. AIS messages of types 1, 2, 3 and 5 were filtered, as per our scope and separated into two tables (one for vessel positions with type 1, 2, 3 messages, and another for static information with only type 5 messages).
3. The following observations in Table 1 were made during the initial exploration of the data frame which prompted our data cleaning steps performed in python:

Table 1. Table of observations from types 1, 2, 3 and 5 messages with corresponding data cleaning steps performed

Category	Observation	Data Cleaning Steps
MMSI	Some MMSI are not 9 digits (standard)	Remove ships with blank or incomplete MMSI
Speed	Null values	Replace with 102.3 (to indicate not available) as suggested by US Coastal Guard Navigation Centre
Course	Null values	Replace with 360.0 (to indicate not available) as suggested by US Coastal Guard Navigation Centre
Heading	Null values	Replace with 511 (to indicate not available) as suggested by US Coastal Guard Navigation Centre
Longitude & Latitude coordinates	Some reported coordinates are erroneously transmitted, for example: – outside covered zone (e.g., Lon of -175) – coordinates on land instead of sea (this was done iteratively after the ‘Spatial’ dimension table connecting GPS coordinates with “Land” or “Sea” reference was set up)	Remove messages with erroneous coordinates.
Ship names / Call sign	Special characters were observed	Remove these special characters to prevent errors downstream (eg. when querying)
IMO	Some IMOs had >7 digits or are empty	Remove ships with incorrect or null IMO values
Ship type	Has values outside the range of 20 to 99	Change the values to 0 (representing ‘Others’)

4. AIS messages of types 1, 2 and 3 are then further filtered to include only Passenger, Cargo and Tanker. This step was done in two phases:
  - i. Because ship type information is found in message type 5, we first filtered out distinct MMSI for ship types 60 – 69 (Passenger ship), 70 – 79 (Cargo) and 80 – 89 (Tanker) in message type 5.
  - ii. These distinct MMSI are then used to filter messages of types 1, 2 and 3 for our vessels of interest.
5. Rows with missing values are dropped. Data in the time stamp column was converted to the “timestamp” data type. Finally, we added a unique running record id for each row in the vessel positions table.

After data staging and cleaning, we have a total of 6.63 million messages of types 1, 2 and 3 and 6,791 vessels (based on unique IMO, MMSI number and ship names from type 5 messages).

Refer to Appendix A for the Python codes used for data staging and cleaning.

### **3. DIMENSION TABLES**

According to Kimball, “dimension tables are integral companions to a fact table and contain the textual descriptors of the business. Dimension table attributes play a vital role in the database warehouse as the entry points into the fact table. Robust dimension attributes deliver robust analytic slicing and dicing capabilities.”

Taking this into consideration, we created the following dimension tables. Detailed classification or categorisation of the dimension tables and web-scraping or API call codes are explained in Appendix B.

Vessel Dimension: This dimension contains detailed information of each vessel derived from the static data of AIS message type 5 and complemented by other vessel information obtained through API calls to the Datalastic Vessel

Ownership API. Each ship is assigned a unique Ship ID key. The attributes include IMO No., MMSI, Ship name, Callsign, Country Flag, Ship Type, Ship Category and sub-category, Ship Type Description, Ship dimensions (to bow, to stern, to port, to starboard, vessel length, vessel beam and deadweight tonnage), Ship owner information (Beneficial owner name and country, operator name and country, technical manager name and country, commercial manager name and country), Class 1 code and Built Year.

Speed (SOG) Dimension: This is to provide speed classification (e.g., Slow, Medium, High, Very High)

Course (COG) Dimension: This is to classify the vessel course into 8 directions (e.g., North, North-East, East, etc).

Spatial Dimension: The covered zone (Lat: 0.83-1.39, Long: 103.38–103.94) is transformed into corresponding geographic cells of 0.01° x 0.01°. This is a *finer* resolution than the minimum requirement of 1 nautical mile which is equivalent to 0.0167°. Each cell is assigned a unique Geo\_Cell\_ID key and categorised into ‘Land’, ‘Sea’ or ‘Port’, with another attribute to define the name of the Singapore ports (Tuas, Jurong Island, Pasir Panjang, Tanjong Pagar, Brani, Bukom, Marina Cruise Centre). The detailed location of each cell is derived from web-scraping through API calls to OpenCage Geocoding API.

Status Dimension: Descriptions for each navigational status as outlined in the Spire Maritime Documentation.

Date Dimension: Since our date scope covers 3 years (2019 – 2021), we included 3 years of dates with detailed attributes referencing Kimball’s recommendations. The holiday indicator is based on Singapore’s public holiday as published in the Ministry of Manpower website. Additionally, we included a ‘COVID-19 indicator’ defined as “Pre CV-19” before 11 Mar 20 (when WHO declares COVID-19 as pandemic) and as “CV-19” on and after 11 Mar 20 since our client is interested to understand trends before and during COVID -19.

Time Dimension: This dimension covers 24 hours in a day with the following attributes: Hour, AM/PM indicator, Day Part (Early Morning, Morning, Afternoon, Evening, Night, Late Night).

#### **4. RELATIONAL FACT TABLES**

According to Kimball “Fact tables are the foundation of the data warehouse. They contain the fundamental measurements of the enterprise and are the ultimate target of most data warehouse queries. The aim of the fact table is to be the repository of the numeric facts that are observed during the measurement event. The first and most important design step is declaring the fact table grain.”

In this project, we built three fact tables: (1) Transaction fact table; and (2) Measured Speed fact table and (3) Measured Ship Count fact table, where (2) and (3) are Periodic Snapshot fact tables.

##### **a. TRANSACTION FACT TABLE**

The basis or grain of this fact table is the AIS message types 1, 2 and 3 for 4 months (Dec 19, Jan 20, Dec 20 & Jan 21) and passenger ship, cargo & tanker ship types which resulted in a total of approximately 6.6 million messages.

We also included other attributes in the fact table such as the primary key for the fact table (Rec\_id) and foreign key references to the dimension tables (eg. shipid, Geo\_Cell\_ID). Table 2 shows the list of attributes in the transaction fact table.

Table 2. Transaction Fact Table

Attribute	In AIS msg	Description
Rec_id	Added	This is a surrogate key created to identify each message in the fact table
Type	In AIS msg	This refers to the message type: 1/2/3
MMSI	In AIS msg	This is 9-digit vessel unique number (Maritime Mobile Service Identities)
Status	In AIS msg	This refers to the navigational status
Turn	In AIS msg	This refers to the rate of turn of the vessel
Speed	In AIS msg	This refers to the speed over ground ranging from 0.0 to 102.2
Accuracy	In AIS msg	This refers to the position accuracy and takes values 0 or 1
Lon	In AIS msg	This refers to the longitude coordinates position of the vessel
Lat	In AIS msg	This refers to the latitude coordinates position of the vessel
Course	In AIS msg	This refers to the course over ground
Heading	In AIS msg	This refers to the true heading of the vessel
Timestamp	In AIS msg	This records the receive time of the AIS record
Date	Added	This is to split the received timestamp into “date” component for reference to the date dimension table
Time	Added	This is to split the received timestamp into “time” component for reference to the time dimension table
shipid	Added	For reference to vessel dimension table. This is a surrogate key created in the vessel dimension table to identify each vessel because the MMSI may not be unique
Geo_Cell_ID	Added	For reference to spatial dimension table. This is a surrogate key created in the spatial dimension table to identify location of the ship in a 0.01° x 0.01° grid in the covered zone.

## b. MEASURED SPEED FACT TABLE

According to Kimball, “a row in a periodic snapshot fact table captures some sort of periodic data for instance, a daily snapshot of financial metrics. In other words, the ‘grain’ or ‘level of resolution’ is the period, not the individual transaction. If no transactions occur during a certain period, a new row must be inserted into the periodic snapshot table, even if every fact that is saved is a null.”

Since *speed* in the transaction table is not additive, we created a measured speed fact table (as our periodic snapshot fact table) to calculate key descriptive statistics (average, standard deviation, minimum, maximum) of speed for each day in each location (sea or identified port e.g., Port Tuas, Port Pasir Panjang) by ship category. Table 3 outlines the attributes in the measured speed fact table. Refer to Appendix C Figure 1 for the SQL query used to create the measured speed fact table.

Table 3. Measured Speed Fact Table

Attribute	Description
Date	This is the ‘grain’ because we are interested in the speed statistics on each day
Geo_SubCategory	This is to understand the speed statistics at each location (sea or port)
ShipCategory	This is to aggregate the measured speed at the ship category level (Tanker, Cargo, Passenger)
Average Speed	These are the calculated descriptive statistics of speed.
SD Speed	
Min & Max Speed	

### c. MEASURED SHIP COUNT FACT TABLE

In view of the large volumes of messages in the fact table, we created a separate measured ship count fact table to streamline query of ship count by day, time, geographical location and ship category. Table 3 outlines the attributes in the measured ship count fact table. Refer to Appendix C Figure 2 for the SQL query used to create the measured ship count fact table.

Table 4. Measured Ship Count Fact Table

Attribute	Description
Date	Together these two form the 'grain' because we are interested in the count statistics on each date by day part (e.g., 10 Dec 19 Early Morning, Morning, etc)
Day Part	
Geo_SubCategory	This is to understand the speed statistics on each location (sea or port)
ShipCategory	This is to aggregate the measured speed at the ship category level (Tanker, Cargo, Passenger)
Unique Ship Count	This is the calculated statistics

## 5. MULTI-DIMENSIONAL DATA WAREHOUSE MODEL - STAR SCHEMA

According to Kimball, "these dimensionally modelled tables are referred to as star schemas. If the presentation area is based on multidimensional database or online analytic processing (OLAP) technology, then the data is stored in cubes."

Putting the relational fact table and dimension tables together, Figure 3 (next page) shows the AIS database warehouse star schema model. As the database warehouse is set up in a star schema design, it can be viewed in the form of multi-dimensional data cube which facilitates the ease of querying and analysing the large volumes of AIS data. For example, Figure 2 shows how a business user can query the data warehouse by "slicing and dicing" and "drilling down" (say count of distinct ships) to a particular period (pre Covid-19), of a particular ship category (Tanker) at a particular location (Port Tuas).

Figure 2. Multi-Dimensional Data Cube Example

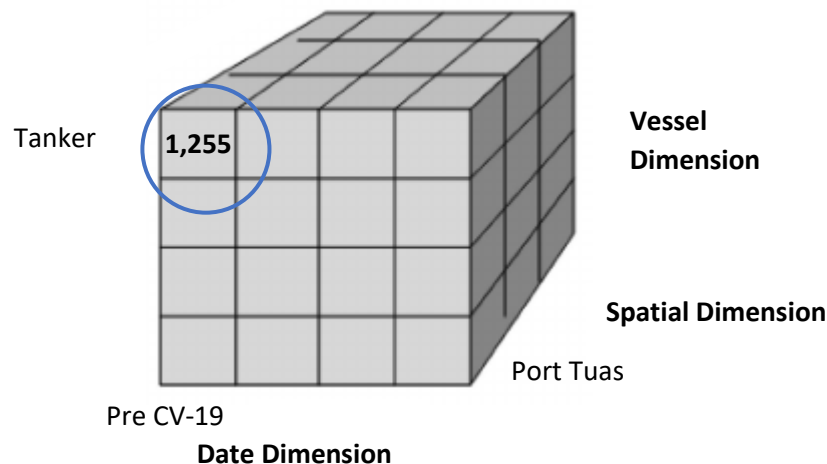
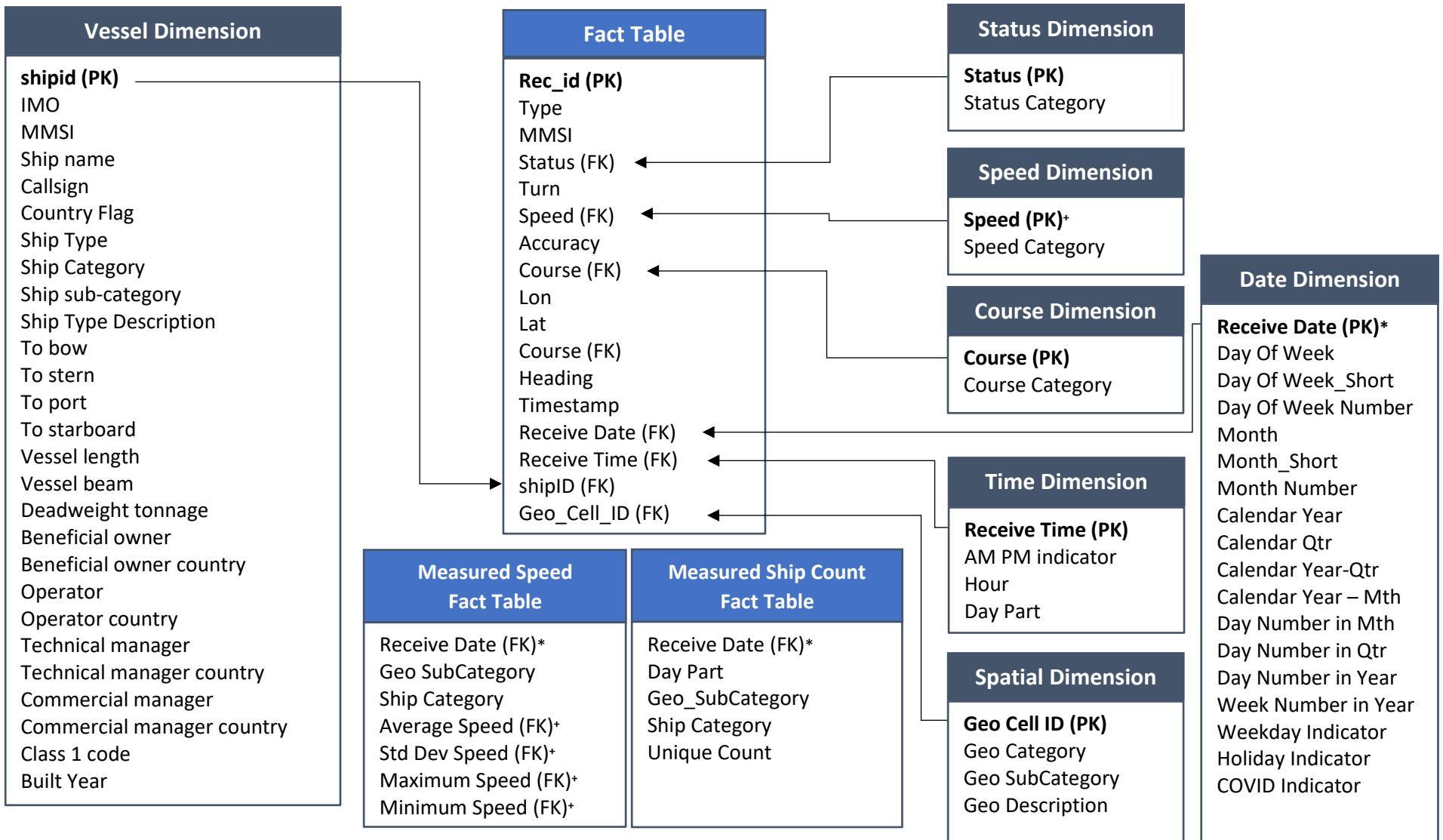




Figure 3. AIS Database Warehouse Star Schema



\* Denotes PK-FK link between the date dimension & measured fact tables

+ Denotes PK-FK link between the speed dimension & measured fact table



## **6. SQL DATABASE QUERIES & USER INTERFACE**

Nine SQL queries were set up to address the four business questions discussed in section 1. Detailed SQL query codes and sample query results are outlined in Appendix D. In addition, two dynamic user interfaces were created: (1) a data visualisation interface using Power BI to visualise key trends (Appendix E) and (2) a web-based application using Flask to generate tabular reports (Appendix F).

### **BUSINESS QUESTION 1: WHAT IS THE IMPACT OF COVID-19 ON MARITIME TRAFFIC IN SINGAPORE WATERS?**

Query 1.1: We analysed the total unique ships in Singapore waters in Dec 19 & Jan 20 (as pre COVID-19 baseline) and Dec 20 & Jan 21 (as COVID-19 period for comparison). From this query, we found that there is a slight drop of 6% in total unique ships in Singapore waters COVID-19 vs Pre COVID-19. (See Appendix D Figure 1)

Query 1.2: Next, we drilled down into ship count by location (Singapore waters vs in Port). From this query, we observed that comparing pre-COVID-19 to during COVID-19, there was larger drop in number of ships stopping in our ports (12%) vs ships in the sea (6%). (See Appendix D Figure 2)

Query 1.3: Next, we drilled down into ship count by port location (Singapore waters vs in Port). From this query, we observed that port Tanjong Pagar is the most impacted with 34% reduction from 224 vessels pre-COVID-19 to 147 during COVID-19. (See Appendix D Figure 3)

Query 1.4: We further drilled down into the different ports and ship categories to understand what ship category is contributing to this reduction in each port. The reduction at Port Tanjong Pagar was driven by Tanker ship category. (See Appendix D Figure 4)

Figure 1 of Appendix E is the Power BI interface to address this business question with the flexibility for users to select the Ship Category and Ship Country.

### **BUSINESS QUESTION 2: WHAT IS THE TRAFFIC DENSITY IN SINGAPORE WATERS AND PORT WITHIN A MONTH OR DAY?**

Before identifying safety enhancement measures, our client needs to understand the maritime traffic density:

Query 2.1: We analysed the number of total unique ships based on ship category and their speed measure (e.g., average speed, standard deviation speed) present in Singapore waters for a particular month (e.g., Dec 19) daily. From this query (using Dec 19 as an example), we found that the busiest period in Dec 19 for Cargo ships were before Christmas, and the average speed during busy periods is also the highest (refer to Appendix D Figure 5). Figure 2 of Appendix E shows the Power BI interface which allows the user to make different month selection and ship category for analysis.

Query 2.2: We analysed the number of unique ships and by ship category in each Singapore port during each part of day (e.g., morning, late morning etc) for a particular month (e.g., Dec 19). From this query, we observed that the busiest parts of the day are 'Afternoon' and 'Late Night', and this is consistent across all ports and ship categories. (See Appendix D Figure 6). Figure 3 of Appendix E is the Power BI interface which allows the user to make different month, port and ship category selection for "drill down".

Appendix F illustrates the use of a web-based Flask application to generate tabular reports based on a user's input. The queries used to generate these reports are based on variations of queries 2.1 (2 variations) and 2.2 (4 variations). The Flask application has two key features. The first feature allows users to find the number of ships

in Singapore waters & their speed statistics based on the selected year, month, and day. The second feature allows users to find the number of ships docked in Singapore ports by time of the day, users may search based on individual or all ports, as well as by day, month, and year.

### **BUSINESS QUESTION 3: HOW “GREEN” ARE THE VESSELS IN SINGAPORE WATERS?**

Taking into consideration Singapore Green Plan 2030 sustainable maritime target, our client would like to understand the current “green” status of vessels in Singapore waters. The cut-off for vessels which may not be able to retrofit to meet the “green” criteria is 15 years of age or older.

Query 3.1: We analysed the number of ships in Singapore waters which are 15 years or older (i.e., Built Year less than 2005 since our data scope is year 2020). From this query, we found that there were 1,480 ships which are 15 years or older – this is 22% of total ships passing Singapore waters in the selected period. (See Appendix D Figure 7)

Query 3.2: We then drilled down to analyse the ship category and ship country for these older ships. From this query, we found slightly more (56%) are Cargo ships and they are mainly from Singapore. (See Appendix D Figure 7)

Figure 4 of Appendix E shows the power BI interface which gives the user the flexibility to select (1) the ‘Built Year’ range in case our client would like to change the criteria for older ships; (2) Date range to narrow our analysis to a particular period of interest.

### **BUSINESS QUESTION 4: HOW TO TRACK THE MOVEMENT OF A PARTICULAR VESSEL IN SINGAPORE WATERS?**

Our client would like to monitor a particular vessel (via IMO number) and the route it takes in Singapore waters.

Query 4.1: We analysed the GPS coordinates for a particular vessel in a particular month (e.g., Dec’19). Additionally, we also extracted the vessel speed at each position. (See Appendix D Figure 8)

Figure 5 of Appendix E shows the movement of a particular vessel (identified by IMO 9006239) through Singapore waters in the month of Dec 19.

In Power BI interface, we further expanded the scope to give users the flexibility to filter for a date range, ship category, course categories and speed categories. For example, Figure 6 of Appendix E shows the number of Cargo vessels on 1 Dec 19 by different speed categories. Expectedly, the ships travelling through Singapore will have medium to high speed (i.e., 8 – 20 knots).

## **7. CONCLUSION**

AIS data provides robust and real time information about ship position data, speed, heading, MMSI, IMO number, ship type. The value of the large volumes of AIS data can be harnessed via building a database warehouse according to the Star Schema together with analytics dashboard tool (like Power BI and Flask) as demonstrated by the result of this project.

Following Kimball’s 4 Steps of Dimensional Design Process, we first understood the key business questions; then identified our scope or ‘grain’ for extracting, transforming, and loading the raw AIS data PostgreSQL. We then designed the dimensions tables and populated them with as much information from web-scraping, especially for

the vessel dimension and spatial dimension, resulting in 7 dimension tables, which included 6791 vessels, 3 years of date, 24 hours of time, more than 3000 geographical cells, speed, course, navigational status classification. Next, we defined the relational fact tables which included one transaction fact table of AIS message types 1, 2 and 3 (total 6.6 million messages) and two measured fact tables (speed and ship count). Finally, we built a Power BI dashboard which covered the 8 SQL queries to address the key business questions and included an interactive map dashboard for vessel traffic monitoring.

Further works to consider: (1) Explore PostgreSQL extension for Geographic Information System (GIS) to better map the geographical space; (2) Include longer periods of time for analysis (e.g., all months of 2019 to 2021) albeit we included 4 months of data as proof of concept; and (3) Include weather conditions as additional attribute in the “Date” dimension table to allow users to analyse by weather conditions.

## **REFERENCES**

*Class A AIS Position Report (Messages 1, 2, and 3) | Navigation Center.* (n.d.). <https://www.navcen.uscg.gov/ais-class-a-reports>

Datalastic Ship Data API & Vessel Database. (2023, October 6). *Vessels Ownership API - Datalastic*. Datalastic. <https://datalastic.com/vessels-ownership-api/>

Kimball, R., & Ross, M. (1996). *The Data Warehouse Toolkit: the complete guide to dimensional modeling*. <http://ci.nii.ac.jp/ncid/BA75756421>

Nofandi, F., Widyaningsih, U., Rakhman, R. A., Mirianto, A., Zuhri, Z., & Harini, N. V. (2022). Case Study of Ship Traffic Crowds in The Malacca Strait-Singapore by Using Vessel Traffic System. *IOP Conference Series, 1081(1)*, 012009. <https://doi.org/10.1088/1755-1315/1081/1/012009>

*OpenCage Geocoding API Documentation.* (n.d.). <https://opencagedata.com/api>

*Ministry of Manpower Public holidays.* (n.d.). <https://www.mom.gov.sg/employment-practices/public-holidays>

*Our targets.* (n.d.). <https://www.greenplan.gov.sg/targets/>

*Review of Maritime Transport 2023.* (2023, September 27). UNCTAD. <https://unctad.org/publication/review-maritime-transport-2023>

*Oliveira, S. (2022, September 22). AIS Fundamentals.* Spire Maritime Documentation. <https://documentation.spire.com/ais-fundamentals/how-to-interpret-navigational-status-codes/>

## APPENDIX A: PYTHON CODES FOR DATA STAGING AND CLEANING

### 1. Loading Json Data to Python data frame

```
# Load Json data to dataframe,
# to read from the file at a time to manage memory use

# Define the path to each JSON file
file_paths = [
    'ais-processed-log-2019-12.json', # Dec19 4.7 million rows
    'ais-processed-data_location_202001_filtered.json', #prefiltered Jan20 3.61 million rows
    'ais-processed-log-2020-12.json', # Dec20 3.4 million rows
    'ais-processed-log-2021-01.json', # Jan21 2.6 million rows
]

# Initialize an empty list to store data
data_list = []
error_count = 0

# Chunk size for processing data
chunk_size = 1000000

# Iterate through each file and load the data
for file_path in file_paths:
    with open(file_path, 'r') as file:
        try:
            data = json.load(file)
            if isinstance(data, list):
                data_list.extend(data)
            elif isinstance(data, dict):
                data_list.append(data)
        except json.JSONDecodeError as e:
            error_count += 1

# Convert the List of dictionaries to a DataFrame
df = pd.DataFrame(data_list)

# Print error count and the number of rows for each file
print(f"File: {file_path}, Cumulative Error Count: {error_count}, Cumulative Rows: {df.shape[0]}")

#print('error:', error_count)
#print('df.shape', df.shape)

File: ais-processed-log-2019-12.json, Cumulative Error Count: 0, Cumulative Rows: 4700810
File: ais-processed-data_location_202001_filtered.json, Cumulative Error Count: 0, Cumulative Rows: 8314545
File: ais-processed-log-2020-12.json, Cumulative Error Count: 0, Cumulative Rows: 11762425
File: ais-processed-log-2021-01.json, Cumulative Error Count: 0, Cumulative Rows: 14380370
```

---

### 2. Cleaning The Data

#### 2.1. Filter only message type of 1, 2, 3 since we predetermine it as our scope

```
# Create dataframe df with only type 1, 2, 3 inside
df_123 = df[df['type'].isin([1,2,3])]

# Comparison before and after (for checking)
b = df.shape
a = df_123.shape
diff_rows = a[0] - b[0]
diff_cols = a[1] - b[1]
print('Data type 1,2, and 3')
print('before: ', b)
print('after : ', a)
print('row differences: ', diff_rows)
print('columns differences: ', diff_cols)
print()

Data type 1,2, and 3
before: (14380370, 25)
after : (13110203, 25)
row differences: -1270167
columns differences: 0
```

---

## 2.2. Remove all columns which have all zero values

```
# Drop columns with all null values
df_123 = df_123.drop(df_123.columns[df_123.isnull().all()], axis=1)
```

## 2.3. Replace missing values in rows with default-not-available-numbers per maritime industry standards

```
# Drop rows with Null values in given columns
```

```
# Drop rows with missing values
df_123['heading'].fillna(511, inplace=True)
df_123['speed'].fillna(102.3, inplace=True)
df_123['course'].fillna(360.0, inplace=True)
a = df_123.shape
```

```
# Comparison before and after (For checking)
diff_rows = a[0] - b[0]
diff_cols = a[1] - b[1]
print('df_123')
print('before: ', b)
print('after: ', a)
print('row differences: ', diff_rows)
print('columns differences: ', diff_cols)
```

```
df_123
before: (14380370, 25)
after : (12275963, 11)
row differences: -2104407
columns differences: -14
```

```
df_123.describe().round(0)
```

	type	mmsi	speed	accuracy	lon	lat	course	heading	status	turn
count	12275963.0	12275963.0	12275963.0	12275963.0	12275963.0	12275963.0	12275963.0	12275963.0	12275963.0	12275963.0
mean	1.0	523517727.0	3.0	0.0	104.0	1.0	187.0	250.0	2.0	-33.0
std	1.0	95754239.0	6.0	0.0	5.0	1.0	106.0	175.0	4.0	68.0
min	1.0	0.0	0.0	0.0	-180.0	-90.0	0.0	0.0	0.0	-128.0
25%	1.0	538003681.0	0.0	0.0	104.0	1.0	92.0	88.0	0.0	-128.0
50%	1.0	563067200.0	0.0	0.0	104.0	1.0	199.0	237.0	0.0	0.0
75%	1.0	565410000.0	6.0	1.0	104.0	1.0	283.0	357.0	3.0	0.0
max	3.0	999486810.0	102.0	1.0	181.0	91.0	360.0	511.0	15.0	127.0

## 2.4. Remove abnormal GPS coordinate and add location's information based on 0.01 nautical grid

```
# Drop rows when Lon and Lat outside the given range
df_123 = df_123[(df_123['lon'].between(103.38, 103.94)) & (df_123['lat'].between(0.83, 1.39))]

# Create the grid
lon_grid = [i * 0.01 + 103.38 for i in range(int((103.94 - 103.38) / 0.01))]
lat_grid = [i * 0.01 + 0.83 for i in range(int((1.39 - 0.83) / 0.01))]

# Create a function for the filter
def encoding_filter(lon, lat):
    if (103.38 <= lon < 103.94) and (0.83 <= lat < 1.39):
        lon_index = int((lon - 103.38) / 0.01)
        lat_index = int((lat - 0.83) / 0.01)
        return lat_index * len(lon_grid) + lon_index + 1
    else:
        return None

# Apply the filter to df_123
df_123['Geo_Cell_ID'] = df_123.apply(lambda row: encoding_filter(row['lon'], row['lat']), axis=1)

# Print the example
print(df_123[['lat', 'lon', 'Geo_Cell_ID']])
```

```
lat lon Geo_Cell_ID
1 1.29001 103.76168 2615.0
4 1.13256 103.75723 1718.0
5 1.20683 103.63240 2098.0
7 1.29129 103.73411 2612.0
8 1.29499 103.76037 2615.0
... ..
14380362 1.31055 103.72688 2723.0
14380363 1.22833 103.66174 2213.0
14380364 1.22832 103.75379 2222.0
14380365 1.29355 103.64766 2603.0
14380366 1.30580 103.75349 2670.0
```

```
[12238506 rows x 3 columns]
```

## 2.5. Include ShipID in the data frame

```
# Load vessel dimension in to photon that contains ship ID number
df_imo = pd.read_excel('unique_mmsi_imo_values_v2.xlsx')
df_imo
```

```
1]:
```

	shipId	imo	mmsi	vessel_name	callsign
0	1	9726671	105792957	MAERSK STADELHORN	9V5223
1	2	9235268	205408000	SAEUROPE	ONCP
2	3	9230050	205421000	EXCALIBUR	ONCE
3	4	9444649	205553000	EXEMPLAR	ONFZ
4	5	9416733	205559000	FRATERNITY	ONGB
...	...	...	...	...	...
6786	6787	9453872	710029870	MACHADO DE ASSIS	PPBC
6787	6788	9453884	710032130	MILTON SANTOS	PPBL
6788	6789	9547673	710033180	CASTRO ALVES	PU2179
6789	6790	9547685	710033240	CARLOS DRUMMOND DE ANDRADE	PU3965
6790	6791	9453810	710239000	ZUMBI DOS PALMARES	PPNM

6791 rows × 5 columns

```
# Adding ship ID by joining main data frame df and vessel information data frame
df_123 = df_123.merge(df_imo[['mmsi', 'shipId']], on='mmsi', how='inner') #include only ships that have shipId

# Comparison before and after (For checking)
a = df_123.shape
diff_rows = a[0] - b[0]
diff_cols = a[1] - b[1]
print('df_123')
print('before: ', b)
print('after : ', a)
print('row differences: ', diff_rows)
print('columns differences: ', diff_cols)
```

```
df_123
before: (14380370, 25)
after : (6398806, 15)
row differences: -7981564
columns differences: -10
```

```
# Check the number of unique value of vessel's mmsi
unique_mmsi = df_123['shipId'].unique()
unique_mmsi.shape
```

```
]: (6777,)
```



## 2.6. Convert column 'time' to date format and break it to time and date

```
# Convert 'time' column to datetime format
df_123['time'] = pd.to_datetime(df_123['time'])

# Create column 'time_' and 'date_' column make a specific datetime format
df_123['time_'] = df_123['time'].dt.strftime('%H:%M')
df_123['date_'] = df_123['time'].dt.strftime('%y/%m/%d')

df_123[['time_', 'date_']]
```

	time	time_	date_
1	2019-12-01 00:00:07+00:00	00:00	19/12/01
4	2019-12-01 00:00:30+00:00	00:00	19/12/01
5	2019-12-01 00:00:30+00:00	00:00	19/12/01
7	2019-12-01 00:00:53+00:00	00:00	19/12/01
8	2019-12-01 00:00:53+00:00	00:00	19/12/01
...	...	...	...
14380362	2021-01-31 23:59:58+00:00	23:59	21/01/31
14380363	2021-01-31 23:59:58+00:00	23:59	21/01/31
14380364	2021-01-31 23:59:58+00:00	23:59	21/01/31
14380365	2021-01-31 23:59:58+00:00	23:59	21/01/31
14380366	2021-01-31 23:59:58+00:00	23:59	21/01/31

12275963 rows × 3 columns

## 27. Create Records Id

```
#Add record_id as a primary key
df_123['rec_id'] = range(1, len(df_123) + 1)
df_123['rec_id']
```

```
4]: 0      1
     1      2
     2      3
     3      4
     4      5
     ...
6398801  6398802
6398802  6398803
6398803  6398804
6398804  6398805
6398805  6398806
Name: rec_id, Length: 6398806, dtype: int64
```

## 28. Rearrange the sequence of columns

```
#Rearrange the columns
df_123_new = df_123_new.reindex(columns=['rec_id', 'type', 'mmsi', 'status', 'turn', 'speed', 'accuracy', 'lon', 'lat', 'course', 'heading', 'time', 'new_date', 'time_', 'shipId', 'Geo_Cell_ID'])

#Final Description of final cleaned data frame
print('Final statistics Descriptive of the cleaned data (type 1,2,3)\n')
df_123_new.describe().T
```

Final statistics Descriptive of the cleaned data (type 1,2,3)

	count	mean	std	min	25%	50%	75%	max
type	6631022.0	1.383035e+00	7.861344e-01	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	3.000000e+00
mmsi	6631022.0	5.096755e+08	1.063713e+08	2.054080e+08	4.775049e+08	5.630751e+08	5.656030e+08	7.102390e+08
speed	6631022.0	3.738638e+00	4.962701e+00	0.000000e+00	0.000000e+00	3.000000e-01	7.300000e+00	1.023000e+02
accuracy	6631022.0	4.676071e-01	4.989496e-01	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00
lon	6631022.0	1.037119e+02	7.885579e-02	1.033800e+02	1.036830e+02	1.037187e+02	1.037644e+02	1.039400e+02
lat	6631022.0	1.223228e+00	5.087235e-02	8.301400e-01	1.192140e+00	1.225300e+00	1.261110e+00	1.387010e+00
course	6631022.0	1.903175e+02	1.083338e+02	0.000000e+00	9.130000e+01	2.008000e+02	2.852750e+02	3.600000e+02
heading	6631022.0	1.991079e+02	1.475523e+02	0.000000e+00	7.400000e+01	1.410000e+02	2.990000e+02	5.110000e+02
status	6631022.0	1.012448e+00	2.371854e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.500000e+01
turn	6631022.0	-1.343748e+01	5.152305e+01	-1.280000e+02	0.000000e+00	0.000000e+00	0.000000e+00	1.270000e+02
shipId	6631022.0	4.323718e+03	1.663314e+03	2.000000e+00	3.414000e+03	5.041000e+03	5.473000e+03	6.791000e+03
Geo_Cell_ID	6631021.0	2.207960e+03	2.869202e+02	1.500000e+01	2.046000e+03	2.218000e+03	2.430000e+03	3.117000e+03

```
#Number of unique values of each column in the dataframe
df_123_new.apply(lambda x: x.nunique())
```

```
type          3
mmsi          6777
speed         735
accuracy       2
lon           55526
lat           32418
course        3601
heading       361
time         1830388
status        16
turn          227
shipId        6777
time_         1440
date_         124
Geo_Cell_ID   1608
dtype: int64
```

## 29. Export Cleaned data frame to csv

```
#Export to csv

#df_edit.to_json('output.json', orient='records', lines=True)
df_123_new.to_csv('df_123_new8.csv', index=False)
```

## **APPENDIX B: DIMENSION TABLE DESCRIPTION & WEB SCRAPPING CODES**

### Appendix B1: Vessel Dimension Attributes Description & Source:

• IMO	Ship identification	AIS message type 5
• MMSI	Ship identification	AIS message type 5
• Ship name	Ship identification	AIS message type 5
• Callsign	Ship identification	AIS message type 5
• Country Flag	Ship Flag	Datalastic API call
• Ship Type	2 digit to define ship type e.g. 60 – 69 for passenger	AIS message type 5
• Ship Type Description	Description of the 2 digit ship type	Datalastic API call
• Ship Category	Tanker, Cargo, Passenger	Datalastic API call
• Ship Subcategory	More details on the ship category e.g. Liquid Gas Carrier (LNG)	Datalastic API call
• Ship Dimensions (to bow, to stern, to port, to starboard, length, beam, deadweight tonnage)	Numeric measurement of ship	AIS message type 5 & Datalastic API call
• Ship owner information (Beneficial owner name and country, operator name and country, technical manager name and country, commercial manager name and country)	Additional ownership information	Datalastic API call
• Class 1 code	Code for ships engaged on long international voyages.	Datalastic API call
• Built Year	Year when vessel is built	Datalastic API call

## Appendix B2: Vessel Dimension Attributes - Datalastic Vessels Ownership API Call Codes:

```
import requests
import json
import pandas as pd

# Read the CSV file containing unique imo values into a DataFrame
data = pd.read_csv("imo_values.csv")
df_imo = data['imo']

# Create an empty DataFrame with columns based on the values in ship_cols, which will be used to store data from the API call
ship_cols = ['id', 'imo', 'vessel_name', 'beneficial_owner', 'beneficial_owner_country',
             'operator', 'operator_country', 'flag_name', 'vessel_type_code', 'built_year',
             'buyer', 'dwt_design', 'class1_code', 'technical_manager', 'technical_manager_country',
             'commercial_manager', 'commercial_manager_country', 'modified_at']
df_shipdata = pd.DataFrame(columns=ship_cols)

# Iterate through each IMO value, make an API call & store the output as a row in df_shipdata :
for imo in df_imo:
    params = {
        'api-key': 'fe65ed94-b963-4335-9ec1-0a186213b2d9',
        'imo': imo
    }

    # make API call, result returned as a python dict
    method = 'vessel'
    api_base = 'https://api.datalastic.com/api/maritime_reports/ownership?'
    api_result = requests.get(api_base+method, params)
    api_response = api_result.json()

    # Extract the list of dictionaries under the 'data' key
    data_list = api_response['data']

    # Create a new DataFrame from the data_list
    new_data = pd.DataFrame(data_list)

    # Append the new data to the existing df_shipdata DataFrame
    df_shipdata = pd.concat([df_shipdata, new_data], ignore_index=True)

# Export the output from the API call into a CSV file
df_shipdata.to_csv('datalastic_ship_data.csv', index=False)
```

## Appendix B3: Speed On Ground (SOG) Dimension Classification:

- 0 knots: Stationary
- >0 to <=8 knots: Slow
- >8 to <=13 knots: Medium
- >13 to <=20 knots: High
- > 20 knots: Very High
- 102.3 knots: Not Available

## Appendix B4: Course On Ground (COG) Dimension Classification:

- N (337.5° – 22.5°)
- NE (22.5° – 67.5°)
- E (67.5° – 112.5°)
- SE (112.5° – 157.5°)
- S (157.5° – 202.5°)
- SW (202.5° – 247.5°)
- W (247.5° – 292.5°)
- NW (292.5° - 337.5°)

## Appendix B5: Spatial Dimension Geographic Cell Detailed Location Web Scrapping Codes:

```
import requests
import pandas as pd

# Replace 'YOUR_API_KEY' with your actual OpenCage Data API key
api_key = '939af1c742df4ad9bd4d0cc366a39a3b'

def get_location_names(start_lat, start_lon, end_lat, end_lon, step):
    base_url = 'https://api.opencagedata.com/geocode/v1/json'
    location_data = []

    current_lat = start_lat
    current_lon = start_lon

    while current_lat <= end_lat:
        while current_lon <= end_lon:
            params = {
                'q': f'{current_lat},{current_lon}',
                'key': api_key,
            }

            response = requests.get(base_url, params=params)

            if response.status_code == 200:
                data = response.json()
                if 'results' in data and len(data['results']) > 0:
                    location = data['results'][0]['formatted']
                    location_data.append((current_lat, current_lon, location))
                else:
                    location_data.append((current_lat, current_lon, "Location not found"))
            else:
                location_data.append((current_lat, current_lon, "Error fetching data from the API"))

            current_lon += step

        current_lat += step
        current_lon = start_lon

    return location_data

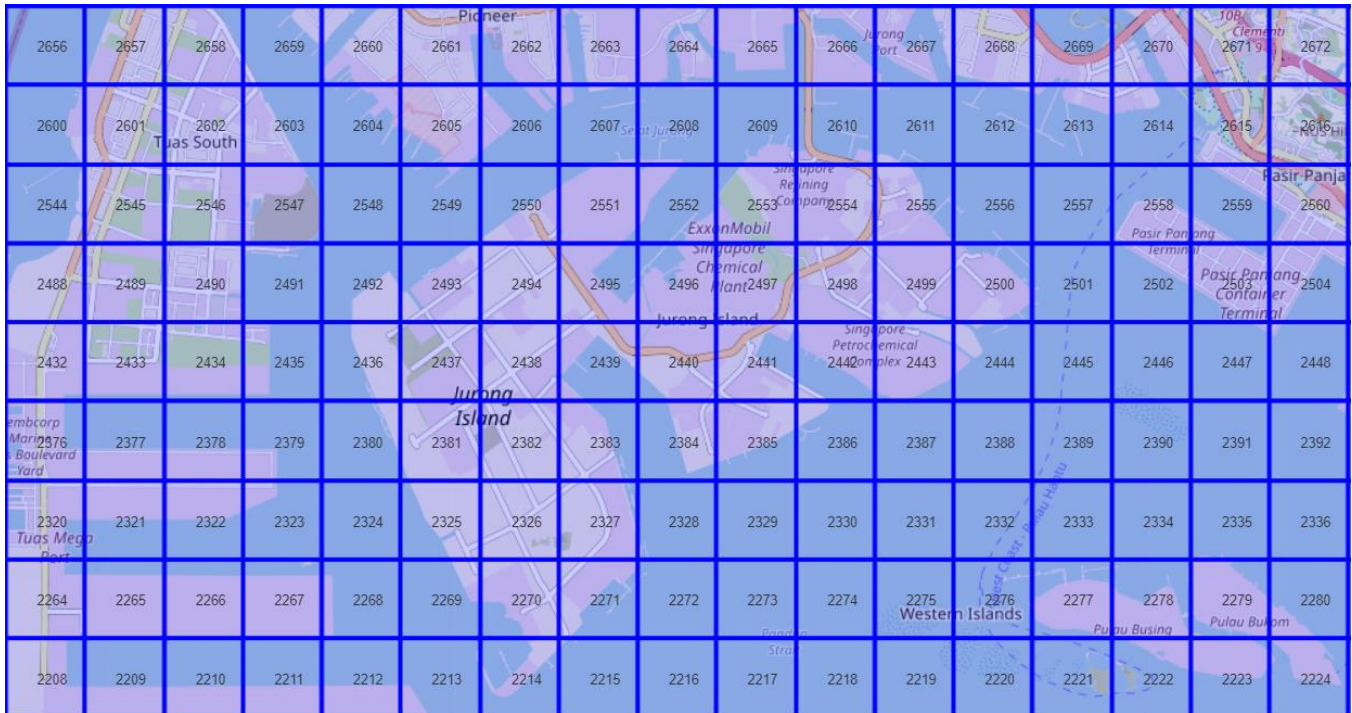
# Example usage with a range of coordinates and a step of 0.01 degrees:
start_latitude = 1.115
start_longitude = 103.385
end_latitude = 1.395
end_longitude = 103.945
step = 0.01

location_data = get_location_names(start_latitude, start_longitude, end_latitude, end_longitude, step)

# Create a DataFrame
df = pd.DataFrame(location_data, columns=['Latitude', 'Longitude', 'Location Name'])

# Save the DataFrame to a CSV file
df.to_csv('location_data1.csv', index=False)
```

## Appendix B6: Spatial Dimension Illustration of Geographic Cells and Allocation of Geo\_Cell\_ID:










## APPENDIX C: SQL QUERY CODES TO CREATE MEASURED FACT TABLES

Figure 1. SQL Query for the Creation of the Measured Speed Fact Table

Query: Query History

```
1 SELECT table1.date_, table1.geo_subcategory, table1.shipcategory, table2.avgspeed, table2.sdspeed, table2.minspeed, table2.maxspeed
2 FROM
3     (SELECT *
4      FROM
5          (SELECT DISTINCT f.date_ FROM facts f) AS ff,
6          (SELECT DISTINCT s.geo_subcategory FROM spatial s WHERE s.geo_subcategory <> 'Land') AS ss,
7          (SELECT DISTINCT v.shipcategory FROM vessels v) AS vv
8      ) AS table1
9 LEFT OUTER JOIN
10     (SELECT f.date_, s.geo_subcategory, v.shipcategory, avg(f.speed) AS avgspeed, stddev(f.speed) AS sdspeed, min(f.speed) AS minspeed, max(f.speed) AS maxspeed
11      FROM facts f, spatial s, vessels v
12      WHERE f.shipid = v.shipid
13            AND f.geo_cell_id = s.geo_cell_id
14            AND f.speed <> 102.3 -- exclude speeds that are unknown (coded as 102.3)
15      GROUP BY f.date_, s.geo_subcategory, v.shipcategory) AS table2
16 ON table1.date_ = table2.date_
17 AND table1.geo_subcategory = table2.geo_subcategory
18 AND table1.shipcategory = table2.shipcategory;
```

Data Output Messages Notifications



	date_ character varying (10)	geo_subcategory character varying	shipcategory character varying	avgspeed double precision	sdspeed double precision	minspeed double precision	maxspeed double precision
1	1/1/2020	Port Bukom	CARGO	6.894444444444445	2.9881082392833767	1.6	12.1
2	1/1/2020	Port Bukom	TANKER	4.465217391304348	3.7707218649819554	0	13.1
3	1/1/2020	Port Bukom	PASSENGER	[null]	[null]	[null]	[null]
4	1/1/2020	Port Tanjong Pagar	CARGO	3.05	2.845464226917405	0	5.9
5	1/1/2020	Port Tanjong Pagar	TANKER	[null]	[null]	[null]	[null]
6	1/1/2020	Port Tanjong Pagar	PASSENGER	[null]	[null]	[null]	[null]

Figure 2. SQL Query for the Creation of the Measured Ship Count Fact Table

Query
Query History

```

1 SELECT ddg.date_, ddg.day_part_segment, ddg.geo_subcategory, ddg.shipcategory, COALESCE(t1.noOfShips, 0) AS shipCount
2 FROM
3     (SELECT *
4      FROM
5          (SELECT DISTINCT ff1.date_ FROM facts ff1) AS udate,
6          (SELECT DISTINCT tt1.day_part_segment FROM time_ tt1) AS uday,
7          (SELECT DISTINCT sp1.geo_subcategory FROM spatial sp1 WHERE sp1.geo_subcategory <> 'Land') AS ugeo,
8          (SELECT DISTINCT v1.shipcategory FROM vessels v1) AS vcat
9      ) AS ddg
10 LEFT OUTER JOIN
11     (SELECT dd.date, tt.day_part_segment, sp.geo_subcategory, v.shipcategory, COUNT(DISTINCT f.shipid) AS noOfShips
12      FROM facts f, vessels v, spatial sp, date_ dd, time_ tt
13      WHERE f.shipid = v.shipid
14            AND f.geo_cell_id = sp.geo_cell_id
15            AND f.date_ = dd.date
16            AND f.time_ = tt.time
17      GROUP BY dd.date, tt.day_part_segment, sp.geo_subcategory, v.shipcategory
18      ORDER BY dd.date ASC, sp.geo_subcategory ASC, tt.day_part_segment ASC
19     ) AS t1
20 ON ddg.date_ = t1.date
21    AND ddg.day_part_segment = t1.day_part_segment
22    AND ddg.geo_subcategory = t1.geo_subcategory
23    AND ddg.shipcategory = t1.shipcategory
24 ORDER BY ddg.date_ ASC, ddg.day_part_segment ASC, ddg.geo_subcategory ASC, ddg.shipcategory ASC;
25

```

Data Output
Messages
Notifications

	date_ character varying (10)	day_part_segment character varying	geo_subcategory character varying	shipcategory character varying	shipcount bigint
1	1/1/2020	Afternoon	Port Brani	CARGO	4
2	1/1/2020	Afternoon	Port Brani	PASSENGER	0
3	1/1/2020	Afternoon	Port Brani	TANKER	1
4	1/1/2020	Afternoon	Port Bukom	CARGO	2
5	1/1/2020	Afternoon	Port Bukom	PASSENGER	0
6	1/1/2020	Afternoon	Port Bukom	TANKER	12



## APPENDIX D: SQL QUERY CODES AND SAMPLE QUERY RESULTS

Figure 1 Query 1.1

Query	Query History
1	SELECT dd.covid_indicator, COUNT(DISTINCT v.shipid)
2	FROM facts f, vessels v, date_ dd
3	WHERE f.date_ = dd.date
4	AND f.shipid = v.shipid
5	GROUP BY dd.covid_indicator

Data Output	Messages	Notifications
<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>		
	covid_indicator character varying	count bigint
1	CV19	4784
2	Pre-CV19	5082

Figure 2 Query 1.2

Query

Query History

```
1 SELECT dd.covid_indicator, sp.geo_category, COUNT(DISTINCT f.shipid)
2 FROM facts f, date_dd, spatial sp
3 WHERE f.date_ = dd.date
4 AND f.geo_cell_id = sp.geo_cell_id
5 GROUP BY dd.covid_indicator, sp.geo_category
6 ORDER BY sp.geo_category ASC, dd.covid_indicator DESC;
```

Data Output

Messages

Notifications

	covid_indicator character varying	geo_category character varying	count bigint
1	Pre-CV19	Port	2576
2	CV19	Port	2266
3	Pre-CV19	Sea	5071
4	CV19	Sea	4777

Figure 3 Query 1.3

Query

Query History

1

SELECT dd.covid\_indicator, sp.geo\_subcategory, COUNT(DISTINCT v.shipid)

2

FROM facts f, vessels v, spatial sp, date\_ dd

3

WHERE f.date\_ = dd.date

4

AND f.shipid = v.shipid

5

AND f.geo\_cell\_id = sp.geo\_cell\_id

6

AND sp.geo\_category = 'Port'

7

GROUP BY dd.covid\_indicator, sp.geo\_subcategory

8

ORDER BY sp.geo\_subcategory ASC, dd.covid\_indicator DESC;

Data Output

Messages

Notifications

covid\_indicator

character varying

geo\_subcategory

character varying

count

bigint

1

Pre-CV19

Port Brani

238

2

CV19

Port Brani

188

3

Pre-CV19

Port Bukom

954

4

CV19

Port Bukom

794

5

Pre-CV19

Port Jurong Island

1831

6

CV19

Port Jurong Island

1623

7

Pre-CV19

Port Marina Bay Cruise Centre

16

8

CV19

Port Marina Bay Cruise Centre

2

9

Pre-CV19

Port Pasir Panjang

1549

10

CV19

Port Pasir Panjang

1377

11

Pre-CV19

Port Tanjong Pagar

224

Total rows: 14 of 14

Query complete 00:00:02.085

Figure 4 Query 1.4

Query

Query History

```
1 SELECT cvgeocat.covid_indicator, cvgeocat.geo_subcategory, cvgeocat.shipcategory,
2 COALESCE(counts.shipcount, 0) AS shipcount
3 FROM
4 (SELECT *
5 FROM
6 (SELECT DISTINCT dd1.covid_indicator FROM date_dd1) AS cv,
7 (SELECT DISTINCT sp1.geo_subcategory FROM spatial sp1 WHERE sp1.geo_category = 'Port')
8 AS sp1,
9 (SELECT DISTINCT v1.shipcategory FROM vessels v1) AS vc)
10 AS cvgeocat
11 LEFT OUTER JOIN
12 (SELECT dd.covid_indicator, sp.geo_subcategory, v.shipcategory, COUNT(DISTINCT v.shipid)
13 AS shipcount
14 FROM facts f, vessels v, spatial sp, date_dd
15 WHERE f.date_ = dd.date
16 AND f.shipid = v.shipid
17 AND f.geo_cell_id = sp.geo_cell_id
18 AND sp.geo_category = 'Port'
19 GROUP BY dd.covid_indicator, sp.geo_subcategory, v.shipcategory
20 ORDER BY sp.geo_subcategory ASC, dd.covid_indicator DESC, v.shipcategory ASC) AS counts
21 ON cvgeocat.covid_indicator = counts.covid_indicator
22 AND cvgeocat.geo_subcategory = counts.geo_subcategory
23 AND cvgeocat.shipcategory = counts.shipcategory;
```

Data Output

Messages

Notifications

	covid_indicator character varying	geo_subcategory character varying	shipcategory character varying	shipcount bigint
1	CV19	Port Brani	CARGO	157
2	Pre-CV19	Port Brani	CARGO	184
3	CV19	Port Brani	TANKER	31
4	Pre-CV19	Port Brani	TANKER	54
5	CV19	Port Brani	PASSENGER	0
6	Pre-CV19	Port Brani	PASSENGER	0
7	CV19	Port Bukom	CARGO	319
8	Pre-CV19	Port Bukom	CARGO	421
9	CV19	Port Bukom	TANKER	475
10	Pre-CV19	Port Bukom	TANKER	531
11	CV19	Port Bukom	PASSENGER	0

Total rows: 42 of 42

Query complete 00:00:01.934

Ln 15,

Figure 5 Query 2.1

Query

Query History

```
1 -- Find the number of unique ships (ie. traffic) across singapore waters on a daily basis
2 -- for Dec 2019, and their average speed, etc.
3 SELECT d.date, d.weekday_indicator, d.holiday_indicator, v.shipcategory, COUNT(DISTINCT f.shipid),
4 mf.avgspeed, mf.stdspeed, mf.minspeed, mf.maxspeed
5 FROM facts f, date_d, vessels v, measuredspeedfacts mf
6 WHERE f.date_ = d.date
7       AND d.calendar_year = 2019
8       AND d.month_number = 12
9       AND f.shipid = v.shipid
10      AND mf.date = d.date
11      AND mf.shipcategory = v.shipcategory
12      AND mf.geo_subcategory = 'Sea'
13      AND f.date_ = mf.date
14 GROUP BY d.day_number_in_year, d.date, d.weekday_indicator, d.holiday_indicator,
15 v.shipcategory, mf.avgspeed, mf.stdspeed, mf.minspeed, mf.maxspeed
16 ORDER BY d.day_number_in_year;
```

Data Output

Messages

Notifications

	date character varying (10)	weekday_indicator character varying	holiday_indicator character varying	shipcategory character varying	count bigint	avgspeed double precision	stdspeed double precision
1	1/12/2019	Weekend	NonHoliday	CARGO	129	7.714967811	5.57282785
2	1/12/2019	Weekend	NonHoliday	PASSENGER	1	0	
3	1/12/2019	Weekend	NonHoliday	TANKER	302	3.006264407	4.28715622
4	2/12/2019	Weekday	NonHoliday	CARGO	138	6.761970614	5.74632172

Total rows: 77 of 77

Query complete 00:00:02.393

Ln 5, Col 48

Figure 6 Query 2.2

Query

Query History

```

1  -- In the month of Dec 2019, at each port, based on each type of ship,
2  -- find the busiest period of the day:
3  SELECT mscf.geo_subcategory, mscf.shipcategory, mscf.day_part_segment, SUM(mscf.shipcount)
4  FROM measuredshipcountfacts mscf, date_ d
5  WHERE mscf.geo_subcategory LIKE 'Port %'
6         AND mscf.date = d.date
7         AND d.month_number = 12
8         AND d.calendar_year = 2019
9  GROUP BY mscf.day_part_segment, mscf.geo_subcategory, mscf.shipcategory
10 ORDER BY mscf.geo_subcategory ASC, mscf.shipcategory ASC, mscf.day_part_segment ASC;

```

Data Output

Messages

Notifications

+

📄

📄

🗑️

📄

📄

📄

	geo_subcategory character varying	shipcategory character varying	day_part_segment character varying	sum bigint
1	Port Brani	CARGO	Afternoon	61
2	Port Brani	CARGO	Early_Morning	30
3	Port Brani	CARGO	Evening	37
4	Port Brani	CARGO	Late_Night	45
5	Port Brani	CARGO	Morning	38
6	Port Brani	CARGO	Night	37
7	Port Brani	PASSENGER	Afternoon	0
8	Port Brani	PASSENGER	Early_Morning	0
9	Port Brani	PASSENGER	Evening	0

Total rows: 126 of 126

Query complete 00:00:00.067

Ln 2

Figure 7 Query 3.1 & 3.2

Query

Query History

```
1  WITH ShipCount AS (
2      SELECT v.flag_name,
3             COUNT(DISTINCT f.shipid) AS ship_count
4      FROM facts f, vessels v
5      WHERE v.shipid = f.shipid AND v.built_year <= 2005
6      GROUP BY v.flag_name
7      ORDER BY ship_count DESC
8  ),
9  TotalCount AS (
10     SELECT COUNT(DISTINCT f.shipid) AS total_ships
11     FROM facts f, vessels v
12     WHERE v.shipid = f.shipid AND v.built_year <= 2005
13 )
14
15 SELECT sc.flag_name,
16        sc.ship_count,
17        tc.total_ships,
18        (sc.ship_count * 100.0 / tc.total_ships) AS percentage
19 FROM ShipCount sc
20 CROSS JOIN TotalCount tc limit 10;
```

Data Output

Messages

Notifications

	flag_name character varying	ship_count bigint	total_ships bigint	percentage numeric
1	Singapore	381	1480	25.7432432432432432
2	Malta	236	1480	15.9459459459459459
3	Marshall Islands	186	1480	12.5675675675675676
4	Cyprus	107	1480	7.2297297297297297
5	Denmark	101	1480	6.8243243243243243
6	Greece	85	1480	5.7432432432432432
7	Liberia	48	1480	3.2432432432432432
8	Isle of Man	46	1480	3.1081081081081081
9	Panama	41	1480	2.7702702702702703
10	France	24	1480	1.6216216216216216
Total rows: 10 of 10    Query complete 00:00:03.002				

Figure 8 Query 4.1

Query

Query History

1

SELECT v.imo, f.time, f.speed, f.lon, f.lat

2

FROM facts f, vessels v, date\_ d

3

WHERE f.shipid = v.shipid

4

AND f.date\_ = d.date

5

AND v.imo = 9006239

6

AND d.calendar\_year = 2019 AND d.month\_number =12;

Data Output

Messages

Notifications

	imo bigint	time timestamp with time zone	speed double precision	lon double precision	lat double precision
1	9006239	2019-12-12 06:38:27+08	6.6	103.85207	1.19757
2	9006239	2019-12-12 06:55:27+08	4.9	103.82627	1.18788
3	9006239	2019-12-12 06:59:27+08	10.3	103.82011	1.18548
4	9006239	2019-12-12 07:08:49+08	5.1	103.80342	1.17946
5	9006239	2019-12-12 07:10:26+08	4.9	103.80073	1.17863
6	9006239	2019-12-12 07:19:20+08	3.9	103.79051	1.17626
7	9006239	2019-12-12 07:29:26+08	7.6	103.78059	1.17337
8	9006239	2019-12-12 07:30:37+08	10.8	103.7778	1.17195
9	9006239	2019-12-12 07:36:06+08	12.9	103.76119	1.16211
10	9006239	2019-12-12 07:38:27+08	13.1	103.75399	1.15762
11	9006239	2019-12-12 07:40:26+08	13.3	103.74793	1.15356
12	9006239	2019-12-12 07:42:26+08	12.5	103.7412	1.1531
13	9006239	2019-12-12 07:44:00+08	11	103.73505	1.15206
Total rows: 33 of 33					

Query complete 00:00:01.186

## APPENDIX E: POWER BI INTERFACE

Figure 1 Power BI interface for Business Question 1

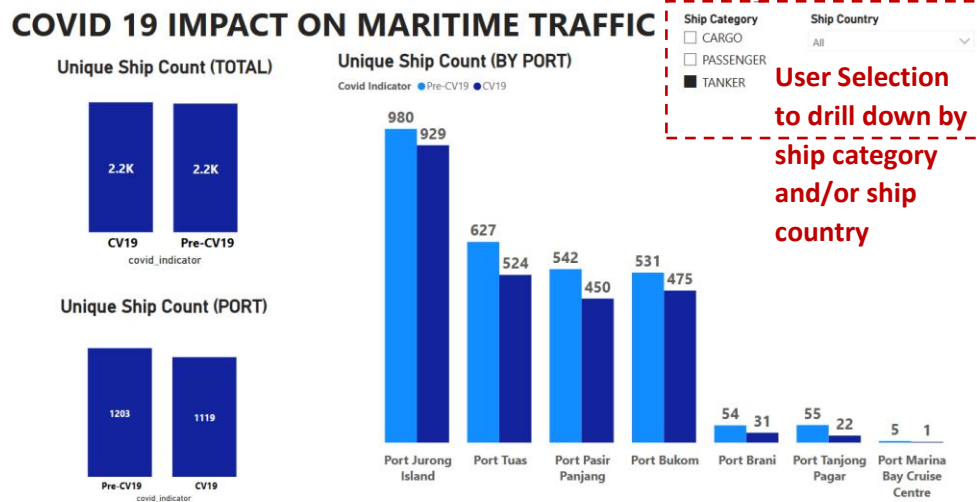


Figure 2 Power BI interface for Query 2.1

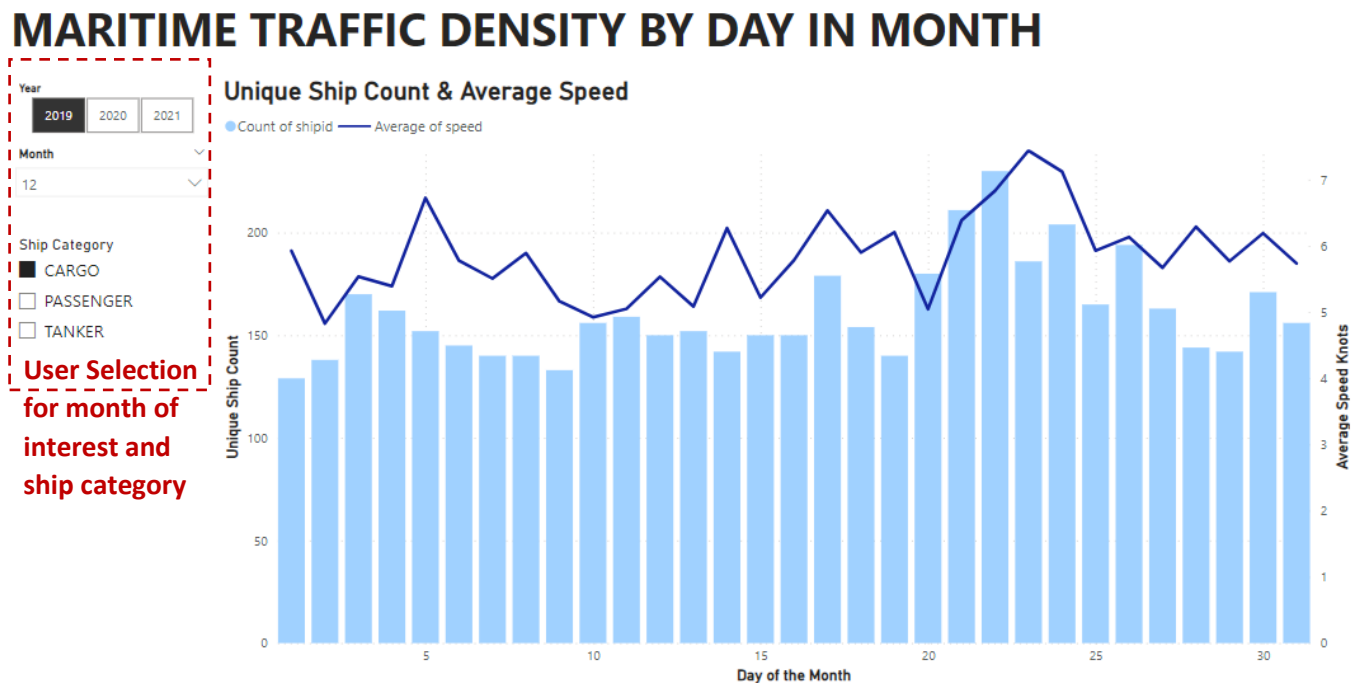


Figure 3 Power BI interface for Query 2.2

## VESSEL TRAFFIC AT PORT BY DAY PART

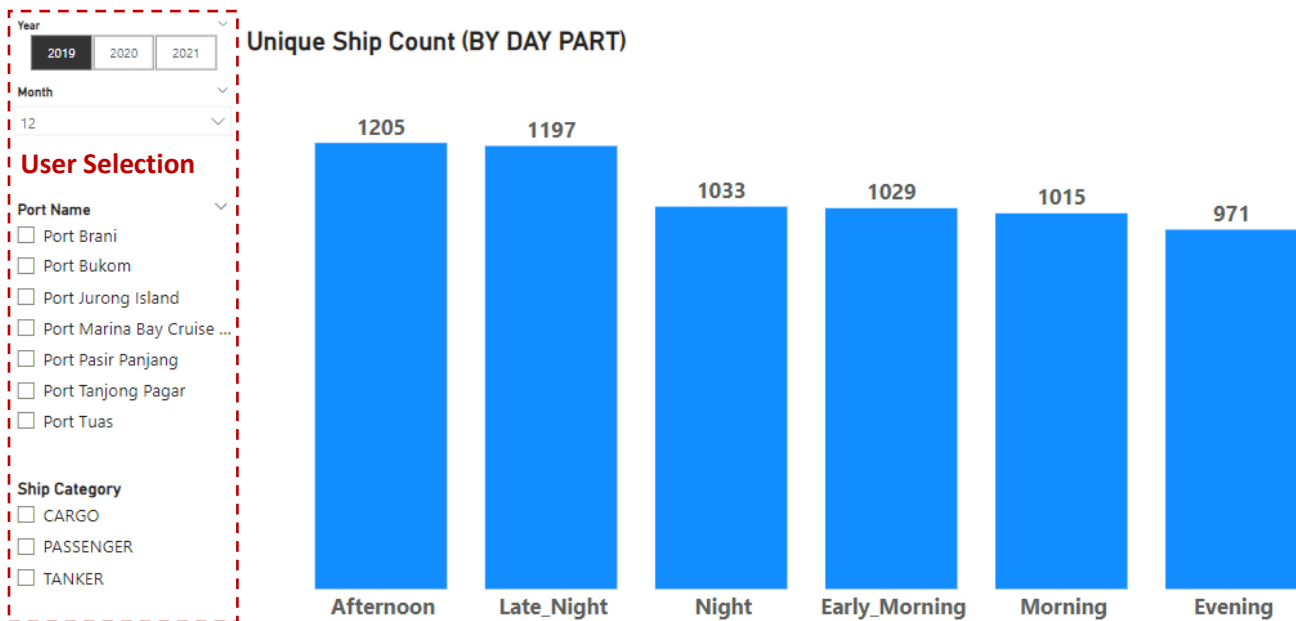


Figure 4 Power BI interface for Business Question 3

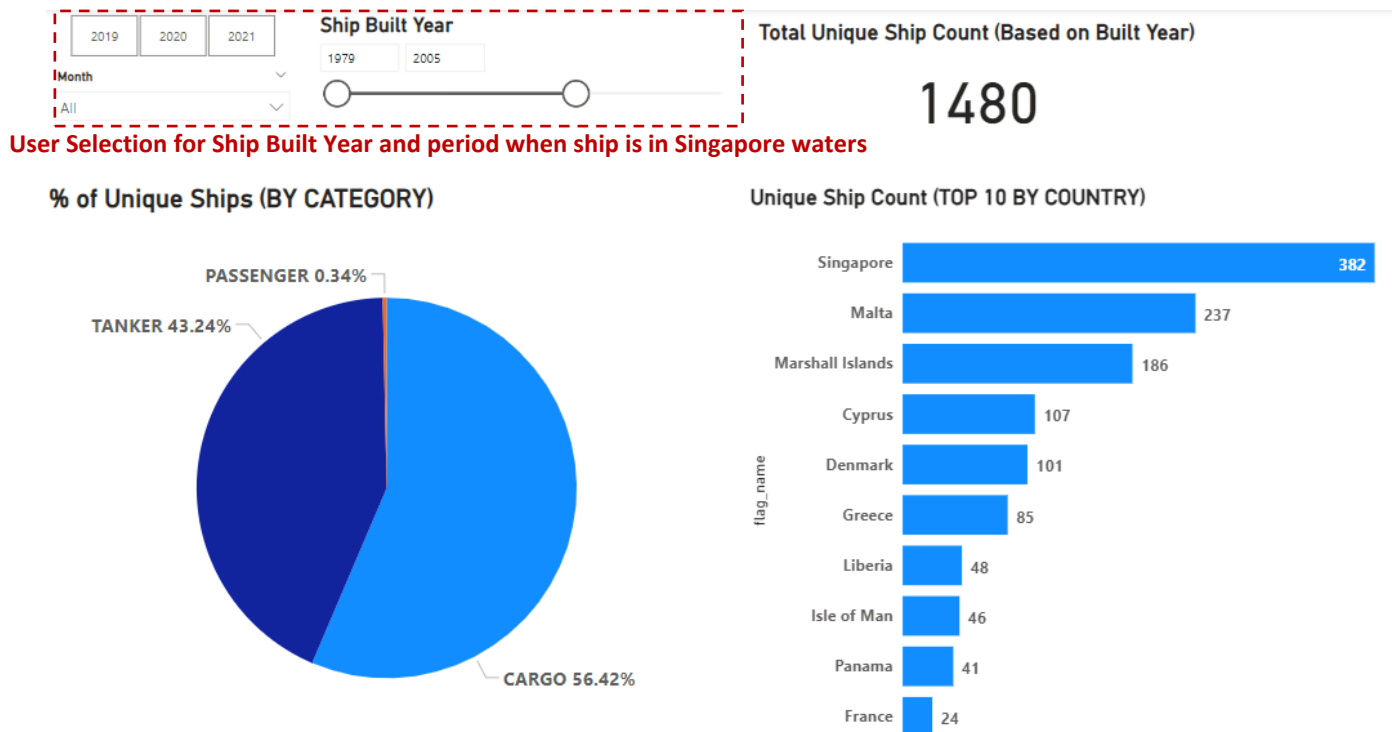


Figure 5 Power BI interface for monitoring a particular vessel.

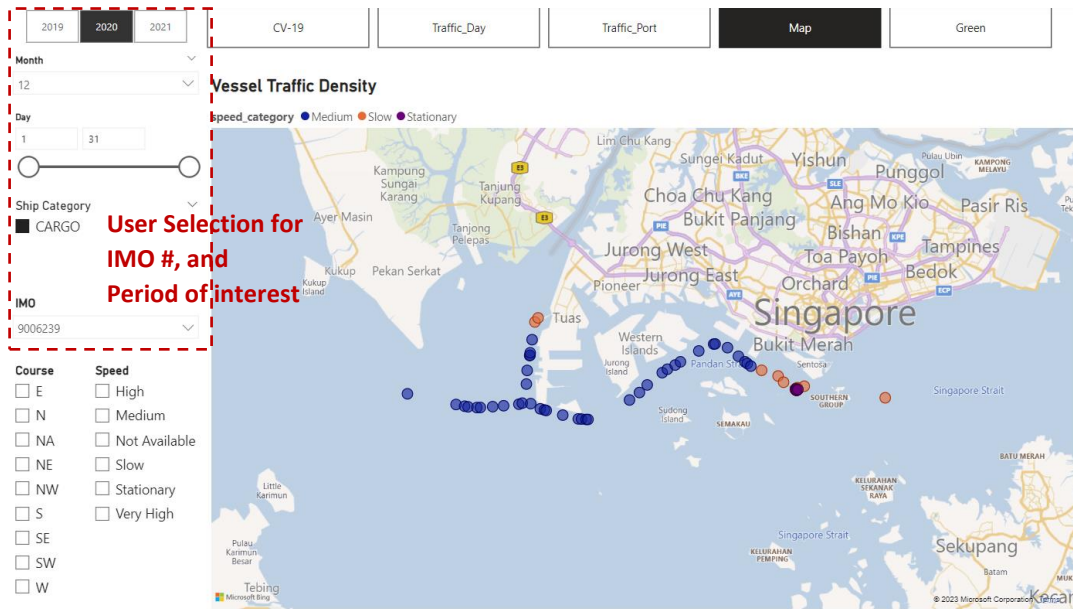
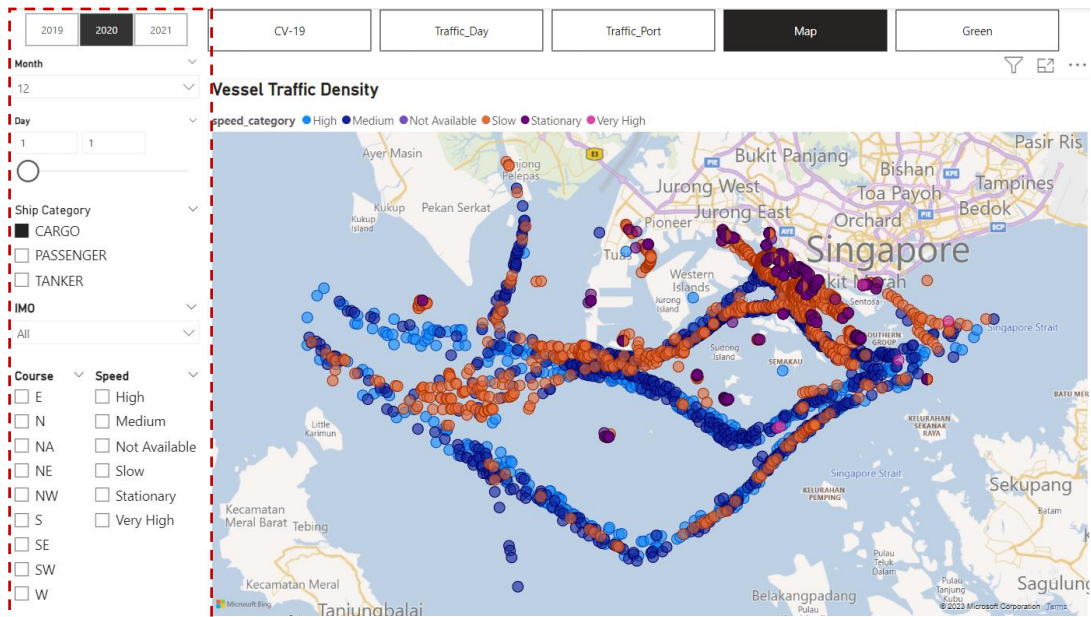


Figure 6 Power BI interface for vessel traffic monitoring

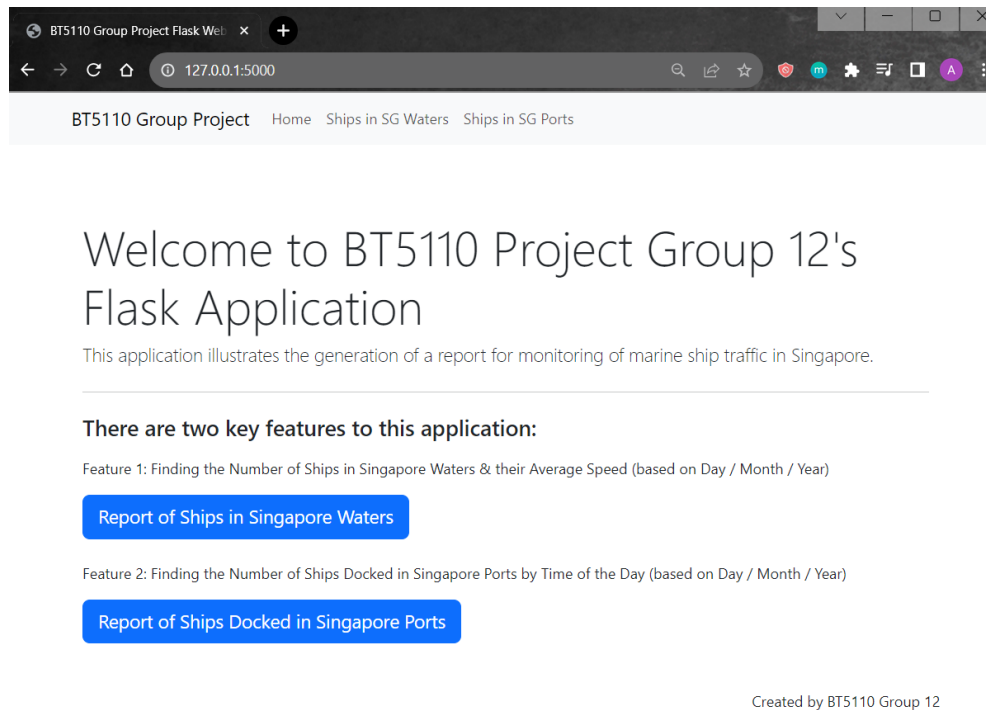


**User Selection for period of interest, ship category, speed category and course heading category**

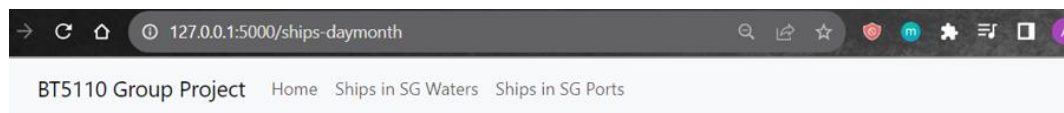


## APPENDIX F: FLASK APPLICATION

Step 1: Accessing the front page (index.html) of the Flask web application via a local server. The application has two key features (covered in steps 2 & 3).



Step 2: Access feature 1, which allows users to find the number of ships in Singapore waters & their speed statistics based on the selected year, month and day. Users can choose to run a report by either (i) month + year or (ii) day + month + year. In this example, the user searched by "Day + Month + Year", for 31 December 2019.



## Flask Application: Generation of Report from StarAIS database

Report on the Number of Ships in Singapore Waters & their Average Speed (based on Day / Month / Year)

Enter Query Parameters (By Day Month)

Search by:

Calendar Year:

Month Number:

Day Number:

Created by BT5110 Group 12

Step 2.1: Upon clicking ‘Search’, a report based on the user’s search parameters will be generated.

127.0.0.1:5000/ships-daymonth-results

BT5110 Group Project
Home
Ships in SG Waters
Ships in SG Ports

## Flask Application: Generation of Report from StarAIS database

### Report on the Number of Ships in Singapore Waters & their Average Speed (based on Day / Month / Year)

Display Query Results:

Date	Weekday Indicator	Holiday Indicator	Ship Category	Unique Ship Count	Average Speed	SD Speed	Min Speed	Max Speed
31/12/2019	Weekday	NonHoliday	CARGO	156	7.468491699	5.447429759	0.0	66.5
31/12/2019	Weekday	NonHoliday	PASSENGER	1	19.36	0.892306296	17.0	20.2
31/12/2019	Weekday	NonHoliday	TANKER	334	3.81261586	4.717611464	0.0	79.9

Created by BT5110 Group 12

Step 2.2: Different SQL queries are executed depending on whether the user searches for “Month + Year” or “Day + Month + Year”.

```

if search_type == "month_year":
    query = """
    SELECT d.date, d.weekday_indicator, d.holiday_indicator, v.shipcategory, COUNT(DISTINCT f.shipid), mf.avgspeed, mf.stdspeed, mf.minspeed, mf.maxspeed
    FROM facts f, date_d, vessels v, measuredfacts mf
    WHERE f.date_ = d.date
        AND d.calendar_year = %s
        AND d.month_number = %s
        AND f.shipid = v.shipid
        AND mf.date = d.date
        AND mf.shipcategory = v.shipcategory
        AND mf.geo_subcategory = 'Sea'
        AND f.date_ = mf.date
    GROUP BY d.day_number_in_year, d.date, d.weekday_indicator, d.holiday_indicator, v.shipcategory, mf.avgspeed, mf.stdspeed, mf.minspeed, mf.maxspeed
    ORDER BY d.day_number_in_year ASC;
    """
    cur.execute(query, (calendar_year, month_number))

if search_type == "day_month_year":
    query = """
    SELECT d.date, d.weekday_indicator, d.holiday_indicator, v.shipcategory, COUNT(DISTINCT f.shipid), mf.avgspeed, mf.stdspeed, mf.minspeed, mf.maxspeed
    FROM facts f, date_d, vessels v, measuredfacts mf
    WHERE f.date_ = d.date
        AND d.calendar_year = %s
        AND d.month_number = %s
        AND d.day_number_in_month = %s
        AND f.shipid = v.shipid
        AND mf.date = d.date
        AND mf.shipcategory = v.shipcategory
        AND mf.geo_subcategory = 'Sea'
        AND f.date_ = mf.date
    GROUP BY d.day_number_in_year, d.date, d.weekday_indicator, d.holiday_indicator, v.shipcategory, mf.avgspeed, mf.stdspeed, mf.minspeed, mf.maxspeed
    ORDER BY d.day_number_in_year ASC;
    """
    cur.execute(query, (calendar_year, month_number, day_number_in_month))
  
```

Step 3: Access feature 2, which allows users to find the number of ships docked in Singapore ports by time of the day, users may search based on individual or all ports, as well as by day, month and year.

Step 3.1: In this example, a search is performed by individual port (Port Tuas), for the month of January 2021.

BT5110 Group Project Home Ships in SG Waters Ships in SG Ports

## Flask Application: Generation of Report from StarAIS database

### Report on the Number of Ships Docked in Singapore Ports by Time of the Day (based on Day / Month / Year)

Enter Query Parameters (By Time Period)

Search by: Individual Port; Month + Year

Calendar Year: 2021

Month Number: 1

Day Number:

Select Port: Port Tuas

Search

Individual Port; Month + Year  
Individual Port; Day + Month + Year  
All Ports; Month + Year  
All Ports; Day + Month + Year

Port Brani  
Port Bukom  
Port Jurong Island  
Port Marina Bay Cruise Centre  
Port Pasir Panjang  
Port Tanjong Pagar  
Port Tuas

Created by BT5110 Group 12

Step 3.1.1: Report based on search parameters from step 3.1.

BT5110 Group Project Home Ships in SG Waters Ships in SG Ports

## Flask Application: Generation of Report from StarAIS database

### Report on the Number of Ships Docked in Singapore Ports by Time of the Day (based on Day / Month / Year)

Display Query Results:

Month	Year	Port	Ship Type	Day Part	Count
Jan	2021	Port Tuas	CARGO	Afternoon	74
Jan	2021	Port Tuas	CARGO	Early_Morning	52
Jan	2021	Port Tuas	CARGO	Evening	48
Jan	2021	Port Tuas	CARGO	Late_Night	86
Jan	2021	Port Tuas	CARGO	Morning	42
Jan	2021	Port Tuas	CARGO	Night	54
Jan	2021	Port Tuas	PASSENGER	Afternoon	15
Jan	2021	Port Tuas	PASSENGER	Early_Morning	11
Jan	2021	Port Tuas	PASSENGER	Evening	10
Jan	2021	Port Tuas	PASSENGER	Late_Night	13
Jan	2021	Port Tuas	PASSENGER	Morning	8
Jan	2021	Port Tuas	PASSENGER	Night	10
Jan	2021	Port Tuas	TANKER	Afternoon	221

Step 3.2: In another example, a search is performed with all ports, for 9 December 2020.

→ ↺ ⌂ ⓘ 127.0.0.1:5000/ships-timeperiod 🔍 ↗ ☆ 🛡️ m ⚙️ ☰ 🗖️ A

BT5110 Group Project   Home   Ships in SG Waters   Ships in SG Ports

## Flask Application: Generation of Report from StarAIS database

---

### Report on the Number of Ships Docked in Singapore Ports by Time of the Day (based on Day / Month / Year)

Enter Query Parameters (By Time Period)

Search by: All Ports; Day + Month + Year ▾

Calendar Year: 2020 ▾

Month Number: 12 ▾

Day Number: 9

Select Port: ▾

Search

Step 3.2.1: Report based on search parameters from step 3.2.

→ ↺ ⌂ ⓘ 127.0.0.1:5000/ships-timeperiod-results 🔍 ↗ ☆ 🛡️ m ⚙️ ☰ 🗖️ A

BT5110 Group Project   Home   Ships in SG Waters   Ships in SG Ports

### Report on the Number of Ships Docked in Singapore Ports by Time of the Day (based on Day / Month / Year)

Display Query Results:

Date	Week	Weekday?	Holiday?	Port	Ship Type	Day Part	Count
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	CARGO	Afternoon	1
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	CARGO	Early_Morning	1
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	CARGO	Evening	0
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	CARGO	Late_Night	2
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	CARGO	Morning	4
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	CARGO	Night	2
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	PASSENGER	Afternoon	0
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	PASSENGER	Early_Morning	0
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	PASSENGER	Evening	0
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	PASSENGER	Late_Night	0
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	PASSENGER	Morning	0
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	PASSENGER	Night	0
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	TANKER	Afternoon	0
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	TANKER	Early_Morning	0
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	TANKER	Evening	1
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	TANKER	Late_Night	0
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	TANKER	Morning	0
9/12/2020	Wed	Weekday	NonHoliday	Port Brani	TANKER	Night	1
9/12/2020	Wed	Weekday	NonHoliday	Port Bukom	CARGO	Afternoon	3

Step 3.3: Different SQL queries are executed depending on whether the user searches for “Individual Port: Month + Year” or “Individual Port: Day + Month + Year” or “All Ports: Month + Year” or “All Ports: Day + Month + Year”.

```

if search_type == "month_year_port":
    query = """
    SELECT d.month_abb, d.calendar_year, mscf.geo_subcategory, mscf.shipcategory, mscf.day_part_segment, SUM(mscf.shipcount)
    FROM measuredshipcountfacts mscf, date_d
    WHERE mscf.geo_subcategory = %s
        AND mscf.date = d.date
        AND d.month_number = %s
        AND d.calendar_year = %s
    GROUP BY d.month_abb, d.calendar_year, mscf.day_part_segment, mscf.geo_subcategory, mscf.shipcategory
    ORDER BY mscf.geo_subcategory ASC, mscf.shipcategory ASC, mscf.day_part_segment ASC;
    """
    cur.execute(query, (port_name, month_number, calendar_year))
    column_names = ["Month", "Year", "Port", "Ship Type", "Day Part", "Count"]

elif search_type == "day_month_year_port":
    query = """
    SELECT d.date, d.dayofweek_abb, d.weekday_indicator, d.holiday_indicator, mscf.geo_subcategory, mscf.shipcategory, mscf.day_part_segment, SUM(mscf.shipcount)
    FROM measuredshipcountfacts mscf, date_d
    WHERE mscf.geo_subcategory = %s
        AND mscf.date = d.date
        AND d.month_number = %s
        AND d.calendar_year = %s
        AND d.day_number_in_month = %s
    GROUP BY d.date, d.dayofweek_abb, d.weekday_indicator, d.holiday_indicator, mscf.day_part_segment, mscf.geo_subcategory, mscf.shipcategory
    ORDER BY mscf.geo_subcategory ASC, mscf.shipcategory ASC, mscf.day_part_segment ASC;
    """
    cur.execute(query, (port_name, month_number, calendar_year, day_number_in_month))
    column_names = ["Date", "Week", "Weekday?", "Holiday?", "Port", "Ship Type", "Day Part", "Count"]

elif search_type == "month_year_all":
    query = """
    SELECT d.month_abb, d.calendar_year, mscf.geo_subcategory, mscf.shipcategory, mscf.day_part_segment, SUM(mscf.shipcount)
    FROM measuredshipcountfacts mscf, date_d
    WHERE (mscf.geo_subcategory = 'Port Brani'
        OR mscf.geo_subcategory = 'Port Bukom'
        OR mscf.geo_subcategory = 'Port Jurong Island'
        OR mscf.geo_subcategory = 'Port Marina Bay Cruise Centre'
        OR mscf.geo_subcategory = 'Port Pasir Panjang'
        OR mscf.geo_subcategory = 'Port Tanjong Pagar'
        OR mscf.geo_subcategory = 'Port Tuas')
        AND mscf.date = d.date
        AND d.month_number = %s
        AND d.calendar_year = %s
    GROUP BY d.month_abb, d.calendar_year, mscf.day_part_segment, mscf.geo_subcategory, mscf.shipcategory
    ORDER BY mscf.geo_subcategory ASC, mscf.shipcategory ASC, mscf.day_part_segment ASC;
    """
    cur.execute(query, (month_number, calendar_year))
    column_names = ["Month", "Year", "Port", "Ship Type", "Day Part", "Count"]

else:
    query = """
    SELECT d.date, d.dayofweek_abb, d.weekday_indicator, d.holiday_indicator, mscf.geo_subcategory, mscf.shipcategory, mscf.day_part_segment, SUM(mscf.shipcount)
    FROM measuredshipcountfacts mscf, date_d
    WHERE (mscf.geo_subcategory = 'Port Brani'
        OR mscf.geo_subcategory = 'Port Bukom'
        OR mscf.geo_subcategory = 'Port Jurong Island'
        OR mscf.geo_subcategory = 'Port Marina Bay Cruise Centre'
        OR mscf.geo_subcategory = 'Port Pasir Panjang'
        OR mscf.geo_subcategory = 'Port Tanjong Pagar'
        OR mscf.geo_subcategory = 'Port Tuas')
        AND mscf.date = d.date
        AND d.month_number = %s
        AND d.calendar_year = %s
        AND d.day_number_in_month = %s
    GROUP BY d.date, d.dayofweek_abb, d.weekday_indicator, d.holiday_indicator, mscf.day_part_segment, mscf.geo_subcategory, mscf.shipcategory
    ORDER BY mscf.geo_subcategory ASC, mscf.shipcategory ASC, mscf.day_part_segment ASC;
    """
    cur.execute(query, (month_number, calendar_year, day_number_in_month))
    column_names = ["Date", "Week", "Weekday?", "Holiday?", "Port", "Ship Type", "Day Part", "Count"]

```