Documentation for SQL query to track and analyze ship voyage events:

Introduction

This SQL script is designed to analyze voyage events for a specific vessel. It involves creating a table to store voyage data, inserting sample data, and processing this data to calculate distances, sailing times, and port stay durations. The script uses temporary tables to handle intermediate calculations and produce the final results.

Creating the Voyages Table

The Voyages table is created using CREATE TABLE to hold data on different events that take place during the ship voyages. The table structure includes fields for event type, timestamp, location, and vessel identifiers. Unlike the sample dataset, it is advised to import data from CSV files straight into the table for larger datasets. Here a sample data is inserted into the voyages table using the INSERT INTO statement.

Dropping Temporary Tables

The script eliminates old temporary tables (if any exists) before generating new ones to prevent conflicts.

Creating Temporary Tables

Temporary tables can be indexed, which can significantly improve performance for complex queries involving large datasets and multiple operations. Temporary tables remain accessible for the duration of the session or until explicitly dropped, allowing multiple queries or sessions to utilize them without recalculating, thus making it more preferable to use here than the CTE.

First Temporary Table: tmp_voyage_events

This temporary table stores voyage events with additional calculated fields, it uses window functions to obtain previous event data for each row.

Conversion of event timestamp into UTC:

It is derived by combining ‘datestamp’ and ‘timestamp’ from the Voyages table and converting it into a UTC timestamp.

Ex: 43831, 0.708333

Here,

43831 represents the serialized number date from the year 1900-01-01 and 0.708333 represents the timestamp in decimal number, where integer part represents hours and fractional part represent minutes and seconds.

43831 days after 1900-01-01 is 2020-01-03 and 0,708333*24 is 17:00:00 PM, i.e. 2020-01-03 17:00:00 PM is the UTC timestamp for the example.

The window function (Lag) enables seamless retrieval of data from preceding rows based on timestamp ordering (**event_utc**).

Second Temporary Table: tmp_voyage_segments

This Identifies and segment different voyage stages based on a series of 'SOSP' and 'EOSP' events. A segment is identified from the start of a SOSP to the end of next EOSP. A segment id is attached to each segment.

This also calculates the,

- **Distance between two geographical coordinates**: the distance is calculated at EOSP, distance travelled by the vessel in the voyage from one port to other.

  Haversine Distance Formula:
  2 * 6371 * ASIN (SQRT (POWER (SIN (RADIANS (lat - prev_lat) / 2), 2) + COS (RADIANS (prev_lat)) * COS (RADIANS (lat)) * POWER (SIN (RADIANS (lon - prev_lon) / 2), 2)))

Ex: Distance between A (44.8038121, -40.0564167) and B (13.4864496,37.0820249)

h = Sin^2((latB-latA)/2) + cos(latA)*cos(latB) * sin^2((lonB-lonA)/2)

= 0.0728493 + 0.7095238*0.9724251*0.3887019 = 0.3410376

theta = 2*sin^(-1)(sqrt (0.3410376)) = 1.24726 Rad = 71.46 deg

Distance d = theta*6371 = 7946.27 Km.

This determines the great circle distance between two points on a sphere given their longitudes and latitudes. We consider the radius of earth (6371 Km) as the radius of the circle and find the distance between two geographical coordinates.

- **Sailing time hours:** the sailing time hours is calculated at the EOSP, time taken for the vessel to travel from one port to other.
  It is calculated using TIMESTAMPDIFF () function between current event_utc and prev_event-utc.

- **Port stay duration:** the port stay duration is calculated at SOSP, amount of time stayed at a port
  It is calculated using TIMESTAMPDIFF () function between current event_utc and prev_event-utc.

Final Query

The final query selects the basic fields in addition with distance travelled, sailing time, port stay duration.

Distance travelled is converted into nautical miles from kilometers by multiplying the value with 0.535597. 1 Km = 0.535597 nautical miles

Documentation for the Python Script to track and analyze ship voyage events:

Introduction

With the help of this Python script, voyage data for ships can be processed and visualised, providing a thorough picture of their movements and operations. In order to do this, the script generates sample journey data, computes timings and distances, processes the data to separate voyages, and then displays the voyage timeline.

Data Generation

Unlike the sample data, data import of a CSV file should be done when dealing with larger set of datasets. For practical purposes, data should be read from a CSV file using pandas; nonetheless, this example contains a method called **generate_sample_data()** to construct a tiny sample dataset.

Distance Calculation Function

Similar to the SQL query, the **haversine_distance(lat1, lon1, lat2, lon2)** function calculates the distance between two geographical points on the Earth's surface based on the hypothetical latitudes and longitudes using the haversine formula. This is particularly useful for determining the distance travelled by a ship between two ports.

The haversine formula is given by:

$a=\sin^2(\Delta\phi/2) +\cos(\phi1) \cdot\cos(\phi2) \cdot\sin^2(\Delta\lambda/2)$

$c=2\cdot\text{atan2}(\sqrt{a}, \sqrt{1-a}) = 2*\text{asin}(\sqrt{a})$

$d=R\cdot c$

where:

- $\phi 1$ and $\phi 2$ are the latitudes of the two points (in radians).
- $\lambda 1$ and $\lambda 2$ are the longitudes of the two points (in radians).
- $\Delta\phi = \phi 2 - \phi 1$ is the difference in latitudes.
- $\Delta\lambda = \lambda 2 - \lambda 1$ is the difference in longitudes.
- R is the Earth's radius (mean radius = 6,371 km, but for nautical miles, we use 3,440.065 nautical miles).
- d is the distance between the two points

Here first we convert the latitudes and longitudes from degrees to radians, then we calculate the difference in latitudes and longitudes from one point to other. After this apply the haversine formula and compute the central angle c. Multiplying c by the radius of earth will give us the geographical distance between two points.

Data Processing

In order to prepare and transform the raw voyage data into a structured format appropriate for analysis and visualisation, the process_voyages_data(df) function is essential. A thorough description of each stage of the data processing is provided below:

- **Conversion of event timestamp into UTC:**
  the raw data contains serialized datestamp and timestamp columns, which need to be converted into a standard datetime format.

  ```
  df['event_utc'] = pd.to_datetime('1900-01-01') + pd.to_timedelta(df['dateStamp'], unit='D') + pd.to_timedelta(df['timeStamp'] * 24, unit='H')
  ```

pd.to_datetime('1900-01-01'): Creates a base datetime object from which we will calculate the actual event times.
pd.to_timedelta(df['dateStamp'], unit='D'): Converts the dateStamp (days since 1900-01-01) to a timedelta object.
pd.to_timedelta(df['timeStamp'] * 24, unit='H'): Converts the timeStamp (fraction of the day) to hours and then to a timedelta object.
Combining these components adds the base date, the days, and the hours to get the full datetime in UTC.

- **Shifting columns for previous function:**
  To calculate distances and time differences between consecutive events, we need information about the previous event. This is achieved by using the shift(1) function. Shift (1) function moves the column down by one row, so each row contain the value of the previous event's column.

- **Calculating Distance in Nautical Miles:**
  using the haversine formula, we calculate the distance between two geographical coordinates, but only at the EOSP events because, we need the distance travelled by the vessel from one port to other which should be calculated at the EOSP. Apply the haversine_distance function row wise, where the event is EOSP to calculated the distance travelled between each port

- **Calculating Sailing time in hours**
  The sailing time between ports is calculated by taking the difference between the current and previous event's UTC timestamps, but only for 'EOSP' events. It is the time taken for the vessel to reach the end of the sea passage
  `(df['event_utc'] - df['prev_event_utc']).dt.total_seconds() / 3600`:
  Calculates the time difference in seconds and converts it to hours.

- **Calculate the Port stay duration in days:**
  The port stay duration is calculated for 'SOSP' (Start of Sea Passage) events. For simplicity, if the ship starts at 'Port A', the stay duration is set to 0.0 days. It is the amount of time a ship stayed at a port before the start of another sea passage. It is the difference between current SOSP event and previous EOSP event
  (df['event_utc'] - df['prev_event_utc']).dt.total_seconds() / (60 * 60 * 24):
  Calculates the time difference in seconds and converts it to days

- **Segmenting the Voyages:**
  Each voyage segment is identified by a series of 'SOSP' (start) and 'EOSP' (end) events. A new segment starts at each 'SOSP' event. Loop through each row, checks if the event is SOSP and increment the segment id. Add this segment id to the list which is then assigned to a new column

Finally, after processing the data, the processed DataFrame is returned after removing rows with missing values in the event column and resetting the index

Data Visualization

This code visualizes the voyages showing the events, duration of port stay days, the distance travelled, and segments of the voyage. This plot is a timeline that helps in understanding the sequence and duration of events in visual format.

The function plot_voyage_timeline(df) takes the processed DataFrame as input and generates a timeline plot of the voyages.

- Create a figure and a set of subplots with specified size. Fig is the figure object and ax is subplot.
- Generate a colormap to ensure each segment of the voyage is plotted with unique color.

- Plotting the events, iterate through each row of the processed DataFrame and plot the necessary symbols with appropriate annotations wherever required to visualize the data efficiently.
- For SOSP events, plot a green upward triangle indicating the start of the sea passage, annotate the port stay duration in days, port name and segment ID here.
- For the EOSP events, plot a red downward triangle indicating the end of the sea passage, annotate the distance travelled in nautical miles, sailing time in hours, port name and segment id here.
- Connect the ports with lines showing the movement of the vessel, if two consecutive ports are in same segment connect them with a dashed line (---) indicating the movement of the ship from one port to other and if they are in different segments connect them with a straight line (-) indicates the ship stayed at the port for the entire duration.
- Label X-axis to Date, set the title of the plot to 'Voyage Timeline
- Add a customized legend clearly explaining all the symbols and markers we took in the plot.
- Display the plot with **plot.show()**