

MET CS 689 Designing and Implementing a Data Warehouse

Assignment 3B: ETL - SQL



April 7, 2023

Vincent Nguyen

nvui@bu.edu

# Overview of the Assignment:

There is more than one way to complete ETL. This assignment will go through the same steps as Assignment 3A, except we will now perform all steps using SQL so that you will get a feel for how to perform the same process within two different languages. Your learning outcome will provide you tools to complete ETL via Python and or SQL

# Part 1 –Extract, Staging

We will start by working with the original source files Ships.csv and CLIWOC15.csv.

The first step is “staging” the data within our SQL database itself by creating the staging tables directly within the relational database itself. The idea of staging tables is to keep the data as close to the source format as possible.

1. *Load the* ***Ships.csv file*** *into the staging table Ships\_Staging on your DBMS* 
   * **SQL Server data load tutorial:** <https://learn.microsoft.com/en-us/sql/integration-services/import-export-data/start-the-sql-server-import-and-export-wizard?view=sql-server-ver15#sql-server-management-studio-ssmsb>.
     1. Hint for SQL Server, on the Modify Columns screen, make sure to select allow NULLs for all three columns.
   * **PostgreSQL data load tutorial:** <https://www.postgresqltutorial.com/postgresql-tutorial/import-csv-file-into-posgresql-table/>
     1. use the Import CSV file into a table using pgAdmin
     2. You will need to create the table first before using the import wizard

Screenshots of the data load of Ships\_Staging:

* **For SQL Server:** provide the Summary and Results screenshots of the Import Flat File Wizard. You do not need to create the table first as the import wizard will do it for you.
* For PostgreSQL: provide your DDL SQL Code to create Ships\_Staging table, as well as a screenshot of the successfully completed confirmation screen similar to the one at the end of the tutorial above.

1. *Verify that there is data Ships\_Staging table by selecting data from it.*

**SQL command:**

*# Create Table*

**CREATE TABLE Ships\_Staging (**

**ShipName Varchar (500),**

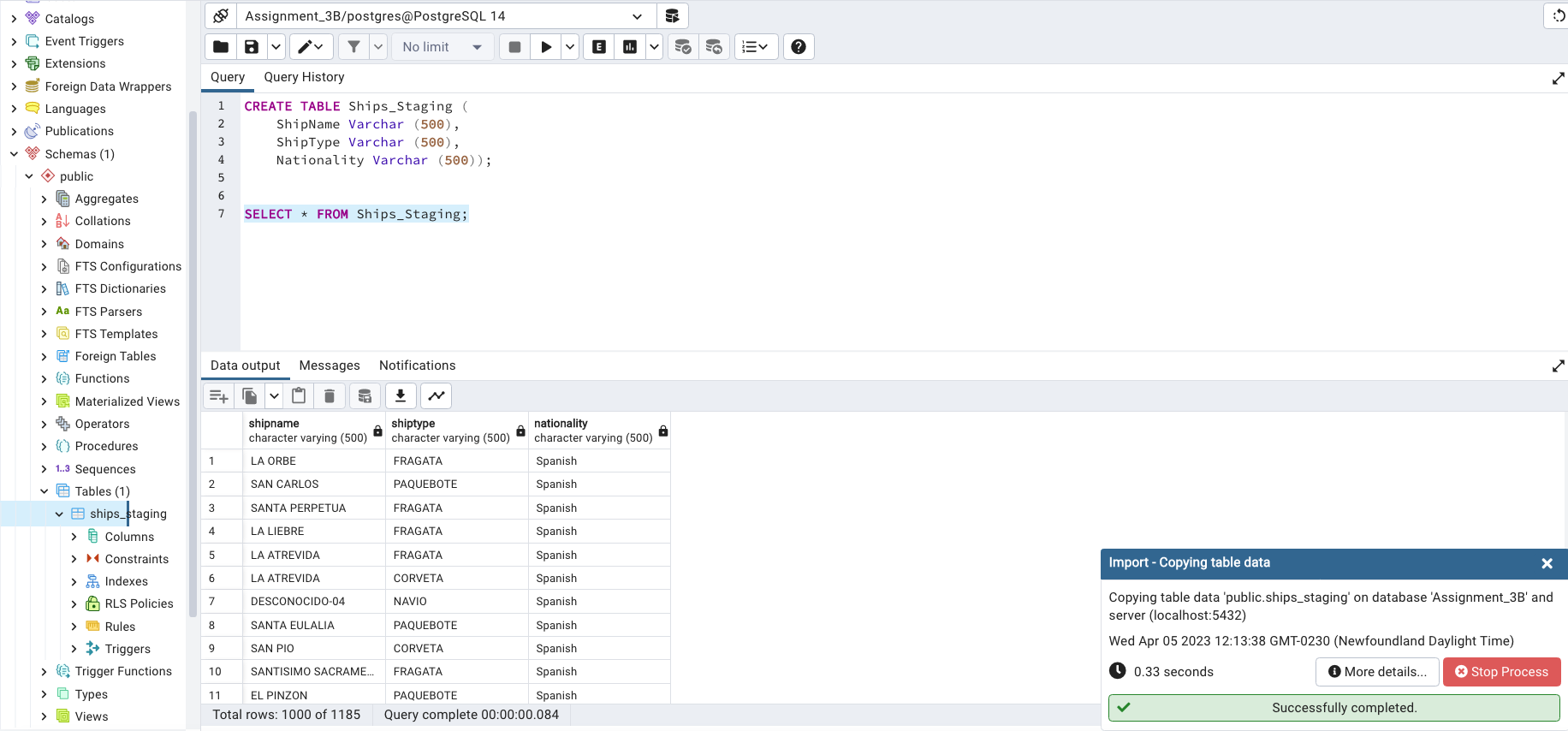
**ShipType Varchar (500),**

**Nationality Varchar (500));**

*# Select table to view results.*

**SELECT \* FROM Ships\_Staging;**

**Screenshots of the executed command:**



1. *Load the Trips csv data from the CLIWOC15.csv file into Trips\_Staging table. You are free to pick the method of loading this staging table including Python, SQL and or a combination of both.*
   1. Hints: Both SQL Server and PostgreSQL will have trouble loading the full file using the method in the previous question where you loaded the Ships\_Staging table. This is mostly due to the variation of the data in some of the columns, we would need to know the appropriate lengths.
   2. Suggested approach: Use any method you like, however what might be the easiest is to load the Trips data into a data frame as python is more forgiving, filter out the columns that you need (recall you will need ShipName, ShipType, and Nationality, RecID, Year, Month, Day, and the three measures you selected in the previous assignment. Create the Trips\_Staging and load using the same method that you loaded the TripFact table at the end of the last assignment, however here you are loading the original data from the CLIWOC15.csv file, beyond the selection of the columns and the load there are no other steps on the Python side.

**SQL/Python commands:**

**import pandas as pd**

*# Read the csv file*

**Trip\_Staging\_read\_df = pd.read\_csv('CLIWOC15.csv')**

*# Filter needed columns for Trips\_Staging*

**TripStaging\_df = Trip\_Staging\_read\_df.filter(['ShipName','ShipType','Nationality', 'RecID', 'Year',**

**'Month', 'Day', 'Distance', 'ShipSpeed', 'drLatDeg'])**

**TripStaging\_df**

*# Connect to DBMS*

**import psycopg2 as pg**

**conn = pg.connect(user = "postgres",**

**password = "BU669",**

**host = "127.0.0.1",**

**port = "5432",**

**database = "Assignment\_3B")**

**cur = conn.cursor()**

*# Create Trips\_Staging in database*

**cur.execute("""CREATE TABLE IF NOT EXISTS Trips\_Staging(**

**ShipName varchar ,ShipType varchar, Nationality varchar,**

**RecID int, Year int, Month int, Day int,**

**Distance Float, ShipSpeed Float, drLatDeg Int);""")**

**conn.commit()**

*#Load data into DBMS*

**for index, row in TripStaging\_df.iterrows():**

**ShipName = row.ShipName if not pd.isna(row.ShipName) else None**

**ShipType = row.ShipType if not pd.isna(row.ShipType) else None**

**Nationality = row.Nationality if not pd.isna(row.Nationality) else None**

**Distance = row.Distance if not pd.isna(row.Distance) else None**

**ShipSpeed = row.ShipSpeed if not pd.isna(row.ShipSpeed) else None**

**drLatdeg = row.drLatDeg if not pd.isna(row.drLatDeg) else None**

**cur.execute("""INSERT INTO Trips\_Staging (ShipName, ShipType, Nationality,**

**RecID, Year, Month, Day,**

**Distance, ShipSpeed, drLatDeg)**

**values (%s,%s,%s,%s,%s,%s,%s,%s,%s, %s)""",**

**(ShipName, ShipType, Nationality, row.RecID, row.Year,**

**row.Month, row.Day, Distance, ShipSpeed, drLatdeg))**

**conn.commit()**

**Screenshots showing the data load:**

Graphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated

Text

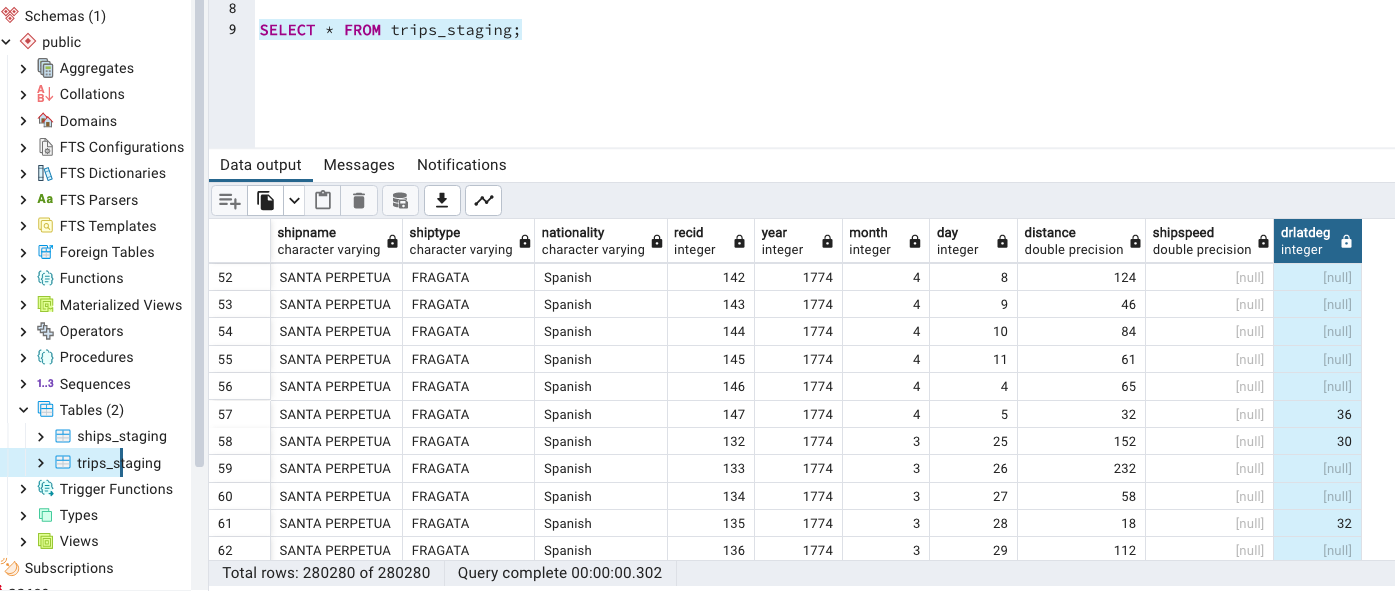
Description automatically generated

1. *Verify that there is data in the Trips\_staging table by selecting data from it.*

**SQL command:**

**SELECT \* FROM trips\_staging;**

**Screenshots of the executed command:**



1. *Once you load the two tables: How many rows are in each staging table? Did your row count match to the loaded data frames in the previous assignment?* Yes, my row count match to the loaded data frames in the previous assignment.

|  |  |
| --- | --- |
|  | Rows |
| Ships\_staging | 1185 |
| Trips\_staging | 280280 |

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

# Part 2 –Creating SCD1 Ship Dimension, Loading and Key maintenance.

1. *Our end goal for this section of the assignment is to create a Ship dimension table DimShipSQL which will be SCD type1, we will need to determine unique records from the Ships\_Staging table first.*

*Select unique records Ships\_Staging using SQL*

**SQL command:**

**SELECT ShipName, ShipType, nationality**

**From ships\_staging**

**Group by ShipName, ShipType, Nationality;**

**Screenshots of the executed command:**

Table

Description automatically generated

*How many unique rows are there?* **1174**

1. *Recall that Trips\_Staging has a few additional dimensional records. Write a query which gets additional unique ships (ShipName, ShipType, Nationality) which are not in Ships\_Staging.* 
   1. Hint, this can be done via a sub-query – this is probably preferred, or using a left join filtering out the nulls.

**SQL command:**

**SELECT ShipName, ShipType, Nationality**

**FROM Trips\_staging**

**EXCEPT**

**SELECT ShipName, ShipType, Nationality**

**FROM ships\_staging**

**Group by ShipName, ShipType, Nationality**

**Screenshots of the executed command:**

Table

Description automatically generated

*How many additional unique ship rows are there in the Trips\_Staging?* **11**

1. *Combine the two queries from question 5 and 6 using the UNION command. You know have your dimensional records.*

**SQL command:**

**(SELECT ShipName, ShipType, nationality**

**From Ships\_staging**

**Group by ShipName, ShipType, Nationality)**

**UNION**

**(SELECT ShipName, ShipType, Nationality**

**FROM Trips\_staging**

**EXCEPT**

**SELECT ShipName, ShipType, Nationality**

**FROM ships\_staging**

**Group by ShipName, ShipType, Nationality);**

**Screenshots of the executed command:**

Table

Description automatically generated

*How many unique rows are there?* **1185**

1. *Create the DimShipSQL table – it will be the same structure as ShipDim from Assignment 3A – however look to implement key management for DimShipID surrogate key using Identity (SQL Server) or serial (PostgreSQL)*

**SQL command:**

**CREATE TABLE DimShipSQL (**

**DimShipID Serial PRIMARY KEY,**

**ShipName Varchar (500),**

**ShipType Varchar (500),**

**Nationality Varchar (500)**

**);**

**Screenshots of the executed command:**

Text

Description automatically generated

1. *Load the data from the results of the previous question #8 into the DimShipSQL*
   1. Hint, use the insert into command.

**SQL command:**

*---- Create a Ship\_Distinct View*

**CREATE VIEW Ship\_Distinct AS**

**(SELECT ShipName, ShipType, nationality**

**From Ships\_staging**

**Group by ShipName, ShipType, Nationality)**

**UNION**

**(SELECT ShipName, ShipType, Nationality**

**FROM Trips\_staging**

**EXCEPT**

**SELECT ShipName, ShipType, Nationality**

**FROM ships\_staging**

**Group by ShipName, ShipType, Nationality)**

*# Insert into DimShipSQL from the Ship\_Distinct view*

**INSERT INTO DimShipSQL(ShipName, shiptype, nationality)**

**SELECT Ship\_Distinct.ShipName, Ship\_Distinct.Shiptype, Ship\_Distinct.Nationality**

**From Ship\_Distinct;**

**Screenshots of the executed command:**

Text

Description automatically generated

*Graphical user interface, text, application, email

Description automatically generated*

*Display the loaded data from the DimShipSQL table and check the record count to make sure it matches*

**SQL command:**

*# Display the loaded data*

**SELECT \* FROM DimShipSQL;**

*# Count number of rows*

**SELECT Count(\*) As Number\_rows**

**From DimShipSQL;**

**Screenshots of the executed command:**

Table

Description automatically generated

*How many rows are there in the resulting DimShipSQL?* **1185**

Graphical user interface, text, application

Description automatically generated

Congratulations, you have now created a clean SCD1 table called DimShipSQL which includes distinct record combinations as well as a primary key.

# Part 3 –Creating Fact table by joining and transforming.

Now that we have a DimShipSQL dimension, lets again focus on creating a FactTripSQL table. Note the three or four numeric columns from the original Trip\_Staging table that you are using.

1. *List the three measures columns you will use:*

* **Measure 1: Distance**
* **Measure 2: ShipSpeed**
* **Measure 3: drLatDeg**

1. *Create the TripFactSQL table – it will be the same structure as TripDim from Assignment 3A – however look to implement key management using Identity (SQL Server) or serial (PostgreSQL) for the primary key. Note that in some fact table designs, there is no surrogate PK used, we will use one here.*

*Your TripFactSQL should include the following attributes*

* 1. *TripFactSQL PK using identity/serial for key management*
  2. *DimShipID FK which should not be NULL*
  3. *TripRecID (the source column for this will be RecID)*
  4. *TripDate (date column)*
  5. *The three/four measures you have selected.*

**SQL command:**

**CREATE TABLE TripFactSQL (**

**TripFactSQL Serial PRIMARY KEY,**

**DimShipID Int NOT NULL,**

**TripRecID Int,**

**TripDate Date,**

**Distance Int,**

**ShipSpeed Int,**

**drLatDeg Int,**

**Constraint fk\_DimShip**

**Foreign Key (DimShipID)**

**References DimShipSQL(DimShipID));**

**Screenshots of the executed command:**

Graphical user interface, text, application

Description automatically generated

1. *We will begin to shape the data which will be inserted into the TripFactSQL table by joining Trip\_Staging to the DimShipSQL – recall using the three columns that define the ship dimension as the join condition. Your resulting query will include the following fields from the Trip\_Staging table: TripRecID, RecID, Year, Month, Day, the three/four measures of your choice, and DimShipID from the DimShip table.*
   1. Hint: it might be helpful at first to see the Ship data fields from both tables first to make sure the join is working, however you won’t need them for the final result.

**SQL command:**

**Select Trip.RecID as TripRecID, Trip.RecID, Ship.DimShipID, Trip.Year,**

**Trip.Month, Trip.Day, Trip.Distance, Trip.Shipspeed, Trip.drlatdeg**

**From Trips\_Staging Trip**

**JOIN DimShipSQL Ship ON coalesce(Ship.nationality,'') = coalesce(Trip.nationality,'')**

**AND COALESCE(Ship.shipname,'') = COALESCE(Trip.shipname,'')**

**AND COALESCE(Ship.shiptype,'') = COALESCE(Trip.shiptype,'')**

*# Create TripShips\_view*

**CREATE VIEW TripShips\_View AS**

**Select Trip.RecID as TripRecID, Trip.RecID, Ship.DimShipID, Trip.Year,**

**Trip.Month, Trip.Day, Trip.Distance, Trip.Shipspeed, Trip.drlatdeg**

**From Trips\_Staging Trip**

**JOIN DimShipSQL Ship ON coalesce(Ship.nationality,'') = coalesce(Trip.nationality,'')**

**AND COALESCE(Ship.shipname,'') = COALESCE(Trip.shipname,'')**

**AND COALESCE(Ship.shiptype,'') = COALESCE(Trip.shiptype,'')**

**Screenshots of the executed command:**

Table

Description automatically generated

*Table

Description automatically generated with medium confidence*

*How many rows are there?* **280280**

*Compare this number of rows to how many rows there are in the Trip\_Staging table:*

I have same number of rows with Trip\_Staggin table which is 280280.

*Graphical user interface, text, application

Description automatically generated*

Graphical user interface, text, application

Description automatically generated

1. *We are almost ready to load this data. Let’s finish the prep work by transforming the month day year fields into a date. Select the RecID, Year, Month, Day as well as your calculated transformed date field called TripDate. Your screenshot can include the first 15-20 rows.*
   1. Hints: It might be helpful to limit your results as you are building your formula
   2. Look at SQL String to date functions, you might need to concatenate the data first, there are a few approaches here.
   3. If there are any null values or errors, you can ignore these. For example, you can use a case statement to check each date part for nulls, and only put the date together if all three parts are not null.

**SQL command:**

*-- Create Function to handle invalid date like 28 Feb, 1850-11-31*

**Create or REPLACE function date\_transform (str text, format text)**

**returns date as**

**$$**

**begin**

**return to\_date(str,format);**

**exception**

**When OTHERS then return null;**

**end;**

**$$ language plpgsql;**

*---- Create view TripShip\_date\_transformed*

**CREATE VIEW TripShip\_date\_transformed As**

**SELECT DimShipId,TripRecID,RecID,Year,Month,Day,Distance,Shipspeed,drlatdeg,**

**CASE WHEN Year IS NOT NULL AND Month IS NOT NULL AND Day IS NOT NULL THEN**

**date\_transform(CONCAT(Year, '-', Month, '-', Day), 'YYYY-MM-DD')**

**ELSE NULL**

**END AS TripDate**

**FROM Tripships\_view**

*----- View the results limit 20 rows*

**SELECT Recid, YEAR, MONTH, DAY, TripDate**

**FROM TripShip\_date\_transformed**

*----- View the results which show 31/11/1850 is handled.*

**SELECT Recid, YEAR, MONTH, DAY, TripDate**

**FROM TripShip\_date\_transformed**

**where year = 1850 and day = 31;**

**Screenshots of the executed command:**

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

Graphical user interface, application, table

Description automatically generated

Table

Description automatically generated

Transforming date is a single example of transformation, usually this is one of the more complicated steps. In your project, you will want to focus transforming strings and aggregating data to create measures as part of transformation instead of just extracting measures from the source file.

1. *Now we are ready to load our TripFactSQL table. Take your query from question 13, and instead of the separate date fields, use the calculated/transformed Trip date, and insert the results of this query into the TripFactSQL table.*

**SQL command:**

*--- insert data into TripFactSQL*

**INSERT INTO TripFactSQL(dimshipid,triprecid,tripdate,distance,shipspeed,drlatdeg)**

**SELECT TripShip\_date\_transformed.dimshipid, TripShip\_date\_transformed.triprecid,**

**TripShip\_date\_transformed.tripdate, TripShip\_date\_transformed.distance,**

**TripShip\_date\_transformed.shipspeed, TripShip\_date\_transformed.drlatdeg**

**From TripShip\_date\_transformed;**

**Screenshots of the executed command:**

**Graphical user interface, text, application, email

Description automatically generated**

1. *Display the loaded data from the TripFactSQL table and check the record.*

**SQL command:**

*----- Display load results*

**Select \* from TripFactSQL;**

*---- Check count*

**Select Count(\*) as number\_rows from TripFactSQL;**

*--- TripFactSQL has 280280 records which is same table when joining Trip\_Staging to the DimShipSQL (280280 records)*

**Screenshots of the executed command:**

Table

Description automatically generated

Table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Congratulations, you have now performed ETL in both SQL and Python

1. *Within no more than two-three sentences compare and contrast the process between SQL and Python, what was easier in one process then the other? There is no right answer here, we just want to see you reflect on the process!*

From my point of view, Processing through Python is easier than SQL. Firstly, it’s easier to load and process data into data frame and then load it to DBMS, we can handle the data types easier in python. Secondly, it causes a lot of errors if we load data directly into DBMS because data types. Thirdly, it is easier to transform date in python compared to SQL.

# Extra Credit - Extending the dimension

(Up-to 5 extra credit points)

Similar to the extra credit in 3A, perform the same steps utilizing SQL as much as possible. Some of the fields in the CLIWOC.csv could be a new dimension, or part of the DimShipSQL table. Outline the new dimension you want to create, extract, transform and load both the dimension and the fact appropriately. A suggestion is to focus on some complexity within transformation. This will give you practice and prepare you for the term project.

Show the commends and appropriate screenshots demonstrating your work and that the data has been loaded into the database.

**SQL/Python commands:**

**Select \* from Trips\_staging\_weather**

*# Create Dim\_Weather*

**Create table Dim\_Weather (**

**weather\_id Serial Primary Key,**

**weather varchar(500),**

**windforce varchar (500),**

**winddirection varchar (500),**

**precipitationdescription varchar (1000)**

**)**

*# Create view Weather\_distinct*

**Create VIEW Weather\_distinct AS**

**Select weather, windforce, winddirection,precipitationdescriptor**

**From Trips\_Staging\_weather**

**Group By weather, windforce, winddirection,precipitationdescriptor;**

*# View results*

**Select \* from weather\_distinct;**

*--- Load Data into dim\_weather*

**INSERT INTO dim\_weather (weather, windforce, winddirection,precipitationdescription)**

**Select weather\_distinct.weather, weather\_distinct.windforce, weather\_distinct.winddirection, weather\_distinct.precipitationdescriptor**

**From weather\_distinct;**

*#View Dim\_weather data*

**Select \* from Dim\_Weather;**

*# Create TripShip\_Weather view*

**CREATE View TripShip\_Weather\_load AS**

**Select TW.RecID As TripRecId, Ship.DimShipId, TW.distance, TW.shipspeed, TW.drlatdeg, DW.weather\_id, TW.weather,TW.winddirection, TW.windforce, TW.precipitationdescriptor, TW.Year, TW.Month,TW.Day,**

**CASE WHEN Year IS NOT NULL AND Month IS NOT NULL AND Day IS NOT NULL THEN**

**date\_transform(CONCAT(Year, '-', Month, '-', Day), 'YYYY-MM-DD')**

**ELSE NULL**

**END AS TripDate**

**From trips\_staging\_weather TW**

**Join DimShipSQL Ship ON coalesce(Ship.nationality,'') = coalesce(TW.nationality,'')**

**AND COALESCE(Ship.shipname,'') = COALESCE(TW.shipname,'')**

**AND COALESCE(Ship.shiptype,'') = COALESCE(TW.shiptype,'')**

**Join dim\_weather DW on coalesce(DW.weather,'') = COALESCE(TW.weather,'')**

**And coalesce(DW.winddirection,'') = coalesce(TW.winddirection,'')**

**AND coalesce(DW.windforce,'') = coalesce(TW.windforce,'')**

**AND COALESCE(DW.precipitationdescription,'') = COALESCE(TW.precipitationdescriptor,'')**

*# View Tripship\_Weather\_load*

**select \* from tripship\_weather\_load;**

*# Create TripFact\_WeatherSQL*

**CREATE TABLE TripFact\_WeatherSQL (**

**TripFactSQL Serial PRIMARY KEY,**

**DimShipID Int NOT NULL,**

**Weather\_id Int Not Null,**

**TripRecID Int,**

**TripDate Date,**

**Distance Int,**

**ShipSpeed Int,**

**drLatDeg Int,**

**Constraint fk\_DimShip**

**Foreign Key (DimShipID)**

**References DimShipSQL(DimShipID),**

**Constraint fk\_dim\_weather**

**Foreign Key (weather\_id)**

**References dim\_weather(weather\_id)**

**);**

*# Load data into Tripfact\_weatethersql*

**INSERT INTO TripFact\_WeatherSQL(dimshipid,weather\_id,triprecid,tripdate,distance,shipspeed,drlatdeg)**

**SELECT TL.dimshipid, TL.weather\_id,TL.triprecid,**

**TL.tripdate, TL.distance, TL.shipspeed, TL.drlatdeg**

**From tripship\_weather\_load TL;**

*# View data*

**Select \* from tripfact\_weathersql**

**# Count rows**

**Select count(\*) from tripfact\_weathersql** *# Has 280280 rows which same as trips\_staging\_weather*

**Screenshots of the executed commands:**

Table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

Table

Description automatically generated with low confidence

Table

Description automatically generated

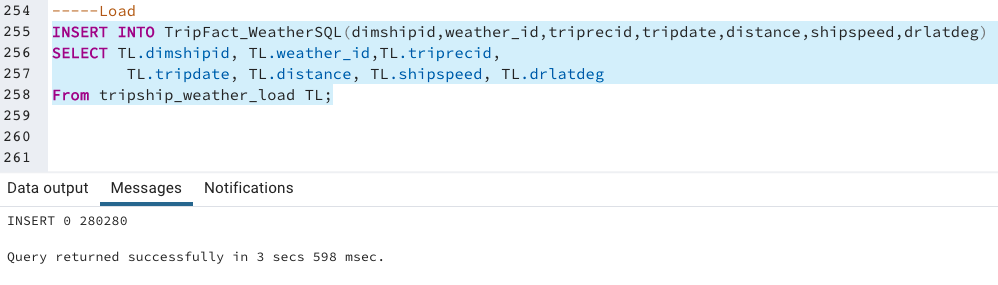
Graphical user interface, text, application

Description automatically generatedGraphical user interface, table

Description automatically generated

Graphical user interface, text

Description automatically generated



Table

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Use the **Ask the Teaching Team Discussion Forum** if you have any questions regarding the how to approach this assignment.

Save your assignment as ***lastnameFirstname\_assign3\_B.docx*** and submit it in the *Assignments* section of the course.

For help uploading files please refer to the *Technical Support* page in the syllabus.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Criterion | A | B | C | D | F | Letter Grade |
| Correctness and Completeness of Results (70%) | All steps' results are entirely complete and correct. | About ¾ of the steps' results are correct and complete | About half of the steps' results are correct and complete | About ¼ of the steps' results are correct and complete | Virtually none of the step's results are correct and complete |  |
| Constitution of SQL/Python and Explanations (30%) | Excellent use and integration of appropriate SQL/Python constructs and supporting explanations | Good use and integration of appropriate SQL/Python constructs and supporting explanations | Mediocre use and integration of appropriate SQL/Python constructs and supporting explanations | Substandard use and integration of appropriate SQL/Python constructs and supporting explanations | Virtually all SQL/Python constructs and supporting explanations are unsuitable or improperly integrated |  |
|  |  |  |  |  | Assignment Grade: |  |