# 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](https://learn.microsoft.com/en-us/sql/integration-services/import-export-data/start-the-sql-server-import-and-export-wizard?view=sql-server-ver15" \l "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:

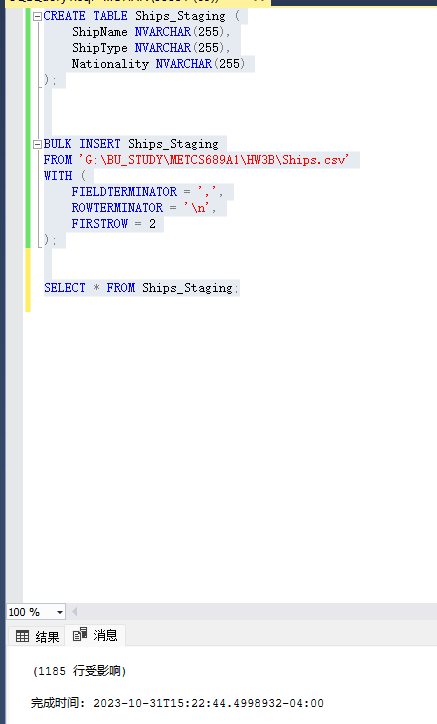
* **For SQL Server:** provide the Summary and Results screenshots of the Import Flat File Wizard. You do not need to create the table first as the import wizard will do it for you.
* For PostgreSQL: provide your DDL SQL Code to create Ships\_Staging table, as well as a screenshot of the successfully 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 Ships\_Staging (  ShipName NVARCHAR(255),  ShipType NVARCHAR(255),  Nationality NVARCHAR(255)  );  BULK INSERT Ships\_Staging  FROM 'G:\BU\_STUDY\METCS689A1\HW3B\Ships.csv'  WITH (  FIELDTERMINATOR = ',',  ROWTERMINATOR = '\n',  FIRSTROW = 2  );  SELECT \* FROM Ships\_Staging; |

Screenshots of the executed command:



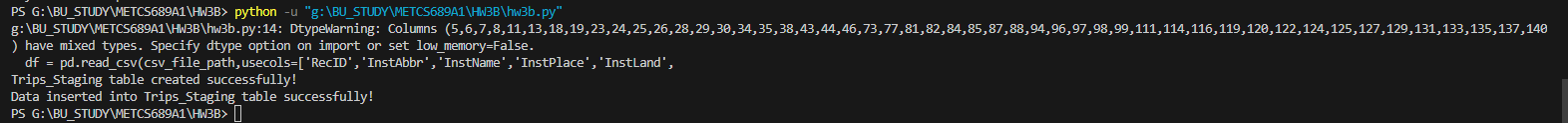


1. Load the Trips csv data from the CLIWOC15.csv file into Trips\_Staging table. You are free to pick the method of loading this staging table including Python, SQL and or a combination of both.
   1. Hints: Both SQL Server and PostgreSQL will have trouble loading the full file using the method in the 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  import pyodbc as db  conn = db.connect('Driver={SQL Server};'                  'Server=XUYUHAN;'                 'Server=SQLServer-PC;'                 'Database=hw3b;'                  'Trusted\_Connection=yes;')  cursor = conn.cursor()  csv\_file\_path = 'G:/BU\_STUDY/METCS689A1/HW3B/CLIWOC15.csv'  df = pd.read\_csv(csv\_file\_path,usecols=['RecID','InstAbbr','InstName','InstPlace','InstLand',  'NumberEntry','NameArchiveSet','ArchivePart','Specification','LogbookIdent',  'LogbookLanguage','EnteredBy','DASnumber','ImageNumber','VoyageFrom',  'VoyageTo','ShipName','ShipType','Company','OtherShipInformation',  'Nationality','Name1','Rank1','Name2','Rank2',  'Name3','Rank3','ZeroMeridian','StartDay','TimeGen',  'ObsGen','ReferenceCourse','ReferenceWindDirection','DistUnits','DistToLandmarkUnits',  'DistTravelledUnits','LongitudeUnits','VoyageIni','UnitsOfMeasurement','Calendar',  'Year','Month','Day','DayOfTheWeek','PartDay',  'TimeOB','Watch','Glasses','UTC','CMG',  'ShipSpeed','Distance','drLatDeg','drLatMin','drLatSec',  'drLatHem','drLongDeg','drLongMin','drLongSec','drLongHem',  'LatDeg','LatMin','LatSec','LatHem','LongDeg','LongMin','LongSec','LongHem','Lat3','Lon3',  'LatInd','LonInd','PosCoastal','EncName','EncNat','EncRem','Anchored','AnchorPlace','LMname1','LMdirection1',  'LMdistance1','LMname2','LMdirection2','LMdistance2','LMname3','LMdirection3','LMdistance3','EstError','ApplError','WindDirection',  'AllWindDirections','WindForce','WindForceScale','AllWindForces','WindScale','Weather','ShapeClouds','DirClouds','Clearness','PrecipitationDescriptor',  'CloudFrac','Gusts','Rain','Fog','Snow','Thunder','Hail','SeaIce','Duplicate','Release',  'SSTReading','SSTReadingUnits','StateSea','CurrentDir','CurrentSpeed','TairReading','AirThermReadingUnits','ProbTair','BaroReading','AirPressureReadingUnits',  'BarometerType','BarTempReading','BarTempReadingUnits','HumReading','HumidityUnits','HumidityMethod','PumpWater','WaterAtThePumpUnits','LifeOnBoard','LifeOnBoardMemo',  'Cargo','CargoMemo','ShipAndRig','ShipAndRigMemo','Biology','BiologyMemo','WarsAndFights','WarsAndFightsMemo','Illustrations','TrivialCorrection','OtherRem'])  df.fillna('unknown', inplace=True)  create\_table\_statement = '''  IF OBJECT\_ID('Trips\_Staging', 'U') IS NOT NULL      DROP TABLE Trips\_Staging      CREATE TABLE Trips\_Staging (      RecID NVARCHAR(4000),      InstAbbr NVARCHAR(4000),      InstName NVARCHAR(4000),      InstPlace NVARCHAR(4000),      InstLand NVARCHAR(4000),      NumberEntry NVARCHAR(4000),      NameArchiveSet NVARCHAR(4000),      ArchivePart NVARCHAR(4000),      Specification NVARCHAR(4000),      LogbookIdent NVARCHAR(4000),      LogbookLanguage NVARCHAR(4000),      EnteredBy NVARCHAR(4000),      DASnumber NVARCHAR(4000),      ImageNumber NVARCHAR(4000),      VoyageFrom NVARCHAR(4000),      VoyageTo NVARCHAR(4000),      ShipName NVARCHAR(4000),      ShipType NVARCHAR(4000),      Company NVARCHAR(4000),      OtherShipInformation NVARCHAR(4000),      Nationality NVARCHAR(4000),      Name1 NVARCHAR(4000),      Rank1 NVARCHAR(4000),      Name2 NVARCHAR(4000),      Rank2 NVARCHAR(4000),      Name3 NVARCHAR(4000),      Rank3 NVARCHAR(4000),      ZeroMeridian NVARCHAR(4000),      StartDay NVARCHAR(4000),      TimeGen NVARCHAR(4000),      ObsGen NVARCHAR(4000),      ReferenceCourse NVARCHAR(4000),      ReferenceWindDirection NVARCHAR(4000),      DistUnits NVARCHAR(4000),      DistToLandmarkUnits NVARCHAR(4000),      DistTravelledUnits NVARCHAR(4000),      LongitudeUnits NVARCHAR(4000),      VoyageIni NVARCHAR(4000),      UnitsOfMeasurement NVARCHAR(4000),      Calendar NVARCHAR(4000),      Year NVARCHAR(4000),      Month NVARCHAR(4000),      Day NVARCHAR(4000),      DayOfTheWeek NVARCHAR(4000),      PartDay NVARCHAR(4000),      TimeOB NVARCHAR(4000),      Watch NVARCHAR(4000),      Glasses NVARCHAR(4000),      UTC NVARCHAR(4000),      CMG NVARCHAR(4000),      ShipSpeed NVARCHAR(4000),      Distance NVARCHAR(4000),      drLatDeg NVARCHAR(4000),      drLatMin NVARCHAR(4000),      drLatSec NVARCHAR(4000),      drLatHem NVARCHAR(4000),      drLongDeg NVARCHAR(4000),      drLongMin NVARCHAR(4000),      drLongSec NVARCHAR(4000),      drLongHem NVARCHAR(4000),      LatDeg NVARCHAR(4000),      LatMin NVARCHAR(4000),      LatSec NVARCHAR(4000),      LatHem NVARCHAR(4000),      LongDeg NVARCHAR(4000),      LongMin NVARCHAR(4000),      LongSec NVARCHAR(4000),      LongHem NVARCHAR(4000),      Lat3 NVARCHAR(4000),      Lon3 NVARCHAR(4000),      LatInd NVARCHAR(4000),      LonInd NVARCHAR(4000),      PosCoastal NVARCHAR(4000),      EncName NVARCHAR(4000),      EncNat NVARCHAR(4000),      EncRem NVARCHAR(4000),      Anchored NVARCHAR(4000),      AnchorPlace NVARCHAR(4000),      LMname1 NVARCHAR(4000),      LMdirection1 NVARCHAR(4000),      LMdistance1 NVARCHAR(4000),      LMname2 NVARCHAR(4000),      LMdirection2 NVARCHAR(4000),      LMdistance2 NVARCHAR(4000),      LMname3 NVARCHAR(4000),      LMdirection3 NVARCHAR(4000),      LMdistance3 NVARCHAR(4000),      EstError NVARCHAR(4000),      ApplError NVARCHAR(4000),      WindDirection NVARCHAR(4000),      AllWindDirections NVARCHAR(4000),      WindForce NVARCHAR(4000),      WindForceScale NVARCHAR(4000),      AllWindForces NVARCHAR(4000),      WindScale NVARCHAR(4000),      Weather NVARCHAR(4000),      ShapeClouds NVARCHAR(4000),      DirClouds NVARCHAR(4000),      Clearness NVARCHAR(4000),      PrecipitationDescriptor NVARCHAR(4000),      CloudFrac NVARCHAR(4000),      Gusts NVARCHAR(4000),      Rain NVARCHAR(4000),      Fog NVARCHAR(4000),      Snow NVARCHAR(4000),      Thunder NVARCHAR(4000),      Hail NVARCHAR(4000),      SeaIce NVARCHAR(4000),      Duplicate NVARCHAR(4000),      Release NVARCHAR(4000),      SSTReading NVARCHAR(4000),      SSTReadingUnits NVARCHAR(4000),      StateSea NVARCHAR(4000),      CurrentDir NVARCHAR(4000),      CurrentSpeed NVARCHAR(4000),      TairReading NVARCHAR(4000),      AirThermReadingUnits NVARCHAR(4000),      ProbTair NVARCHAR(4000),      BaroReading NVARCHAR(4000),      AirPressureReadingUnits NVARCHAR(4000),      BarometerType NVARCHAR(4000),      BarTempReading NVARCHAR(4000),      BarTempReadingUnits NVARCHAR(4000),      HumReading NVARCHAR(4000),      HumidityUnits NVARCHAR(4000),      HumidityMethod NVARCHAR(4000),      PumpWater NVARCHAR(4000),      WaterAtThePumpUnits NVARCHAR(4000),      LifeOnBoard NVARCHAR(4000),      LifeOnBoardMemo NVARCHAR(4000),      Cargo NVARCHAR(4000),      CargoMemo NVARCHAR(4000),      ShipAndRig NVARCHAR(4000),      ShipAndRigMemo NVARCHAR(4000),      Biology NVARCHAR(4000),      BiologyMemo NVARCHAR(4000),      WarsAndFights NVARCHAR(4000),      WarsAndFightsMemo NVARCHAR(4000),      Illustrations NVARCHAR(4000),      TrivialCorrection NVARCHAR(4000),      OtherRem NVARCHAR(4000)  )  '''  cursor.execute(create\_table\_statement)  conn.commit()  cursor.close()  conn.close()  print("Trips\_Staging table created successfully!")  conn1 = db.connect('Driver={SQL Server};'                  'Server=XUYUHAN;'                 'Server=SQLServer-PC;'                 'Database=hw3b;'                  'Trusted\_Connection=yes;')  cursor1 = conn1.cursor()  insert\_query = """  INSERT INTO Trips\_Staging (RecID,InstAbbr,InstName,InstPlace,InstLand,NumberEntry,NameArchiveSet,ArchivePart,Specification,LogbookIdent,LogbookLanguage,EnteredBy,DASnumber,ImageNumber,VoyageFrom,VoyageTo,ShipName,ShipType,Company,OtherShipInformation,Nationality,Name1,Rank1,Name2,Rank2,Name3,Rank3,ZeroMeridian,StartDay,TimeGen,ObsGen,ReferenceCourse,ReferenceWindDirection,DistUnits,DistToLandmarkUnits,DistTravelledUnits,LongitudeUnits,VoyageIni,UnitsOfMeasurement,Calendar,Year,Month,Day,DayOfTheWeek,PartDay,TimeOB,Watch,Glasses,UTC,CMG,ShipSpeed,Distance,drLatDeg,drLatMin,drLatSec,drLatHem,drLongDeg,drLongMin,drLongSec,drLongHem,LatDeg,LatMin,LatSec,LatHem,LongDeg,LongMin,LongSec,LongHem,Lat3,Lon3,LatInd,LonInd,PosCoastal,EncName,EncNat,EncRem,Anchored,AnchorPlace,LMname1,LMdirection1,LMdistance1,LMname2,LMdirection2,LMdistance2,LMname3,LMdirection3,LMdistance3,EstError,ApplError,WindDirection,AllWindDirections,WindForce,WindForceScale,AllWindForces,WindScale,Weather,ShapeClouds,DirClouds,Clearness,PrecipitationDescriptor,CloudFrac,Gusts,Rain,Fog,Snow,Thunder,Hail,SeaIce,Duplicate,Release,SSTReading,SSTReadingUnits,StateSea,CurrentDir,CurrentSpeed,TairReading,AirThermReadingUnits,ProbTair,BaroReading,AirPressureReadingUnits,BarometerType,BarTempReading,BarTempReadingUnits,HumReading,HumidityUnits,HumidityMethod,PumpWater,WaterAtThePumpUnits,LifeOnBoard,LifeOnBoardMemo,Cargo,CargoMemo,ShipAndRig,ShipAndRigMemo,Biology,BiologyMemo,WarsAndFights,WarsAndFightsMemo,Illustrations,TrivialCorrection,OtherRem)  VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)  """  for index, row in df.iterrows():      Key = index + 1        cursor1.execute(insert\_query, row['RecID'],row['InstAbbr'],row['InstName'],row['InstPlace'],row['InstLand'],row['NumberEntry'],row['NameArchiveSet'],row['ArchivePart'],row['Specification'],row['LogbookIdent'],row['LogbookLanguage'],row['EnteredBy'],row['DASnumber'],row['ImageNumber'],row['VoyageFrom'],row['VoyageTo'],row['ShipName'],row['ShipType'],row['Company'],row['OtherShipInformation'],row['Nationality'],row['Name1'],row['Rank1'],row['Name2'],row['Rank2'],row['Name3'],row['Rank3'],row['ZeroMeridian'],row['StartDay'],row['TimeGen'],row['ObsGen'],row['ReferenceCourse'],row['ReferenceWindDirection'],row['DistUnits'],row['DistToLandmarkUnits'],row['DistTravelledUnits'],row['LongitudeUnits'],row['VoyageIni'],row['UnitsOfMeasurement'],row['Calendar'],row['Year'],row['Month'],row['Day'],row['DayOfTheWeek'],row['PartDay'],row['TimeOB'],row['Watch'],row['Glasses'],row['UTC'],row['CMG'],row['ShipSpeed'],row['Distance'],row['drLatDeg'],row['drLatMin'],row['drLatSec'],row['drLatHem'],row['drLongDeg'],row['drLongMin'],row['drLongSec'],row['drLongHem'],row['LatDeg'],row['LatMin'],row['LatSec'],row['LatHem'],row['LongDeg'],row['LongMin'],row['LongSec'],row['LongHem'],row['Lat3'],row['Lon3'],row['LatInd'],row['LonInd'],row['PosCoastal'],row['EncName'],row['EncNat'],row['EncRem'],row['Anchored'],row['AnchorPlace'],row['LMname1'],row['LMdirection1'],row['LMdistance1'],row['LMname2'],row['LMdirection2'],row['LMdistance2'],row['LMname3'],row['LMdirection3'],row['LMdistance3'],row['EstError'],row['ApplError'],row['WindDirection'],row['AllWindDirections'],row['WindForce'],row['WindForceScale'],row['AllWindForces'],row['WindScale'],row['Weather'],row['ShapeClouds'],row['DirClouds'],row['Clearness'],row['PrecipitationDescriptor'],row['CloudFrac'],row['Gusts'],row['Rain'],row['Fog'],row['Snow'],row['Thunder'],row['Hail'],row['SeaIce'],row['Duplicate'],row['Release'],row['SSTReading'],row['SSTReadingUnits'],row['StateSea'],row['CurrentDir'],row['CurrentSpeed'],row['TairReading'],row['AirThermReadingUnits'],row['ProbTair'],row['BaroReading'],row['AirPressureReadingUnits'],row['BarometerType'],row['BarTempReading'],row['BarTempReadingUnits'],row['HumReading'],row['HumidityUnits'],row['HumidityMethod'],row['PumpWater'],row['WaterAtThePumpUnits'],row['LifeOnBoard'],row['LifeOnBoardMemo'],row['Cargo'],row['CargoMemo'],row['ShipAndRig'],row['ShipAndRigMemo'],row['Biology'],row['BiologyMemo'],row['WarsAndFights'],row['WarsAndFightsMemo'],row['Illustrations'],row['TrivialCorrection'],row['OtherRem'])  conn1.commit()  cursor1.close()  conn1.close()  print("Data inserted into Trips\_Staging table successfully!") |

Screenshots showing the data load:

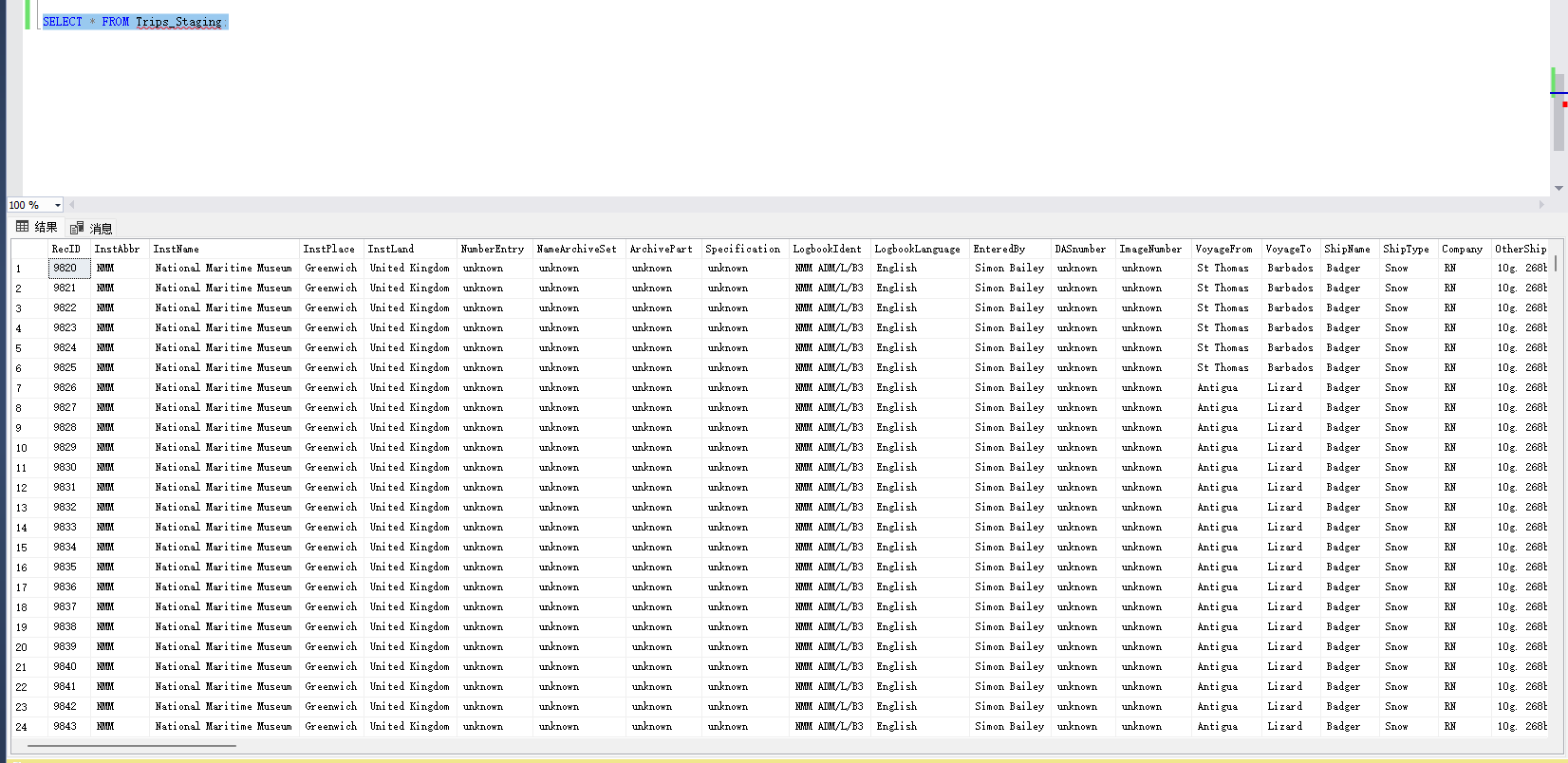


1. Verify that there is data in the Trips\_staging table by selecting data from it.

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?

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

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

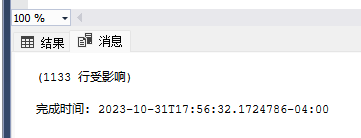
1. Our end goal for this section of the assignment is to create a Ship dimension table DimShipSQL which will be SCD type1, we will need to determine unique records from the Ships\_Staging table first.

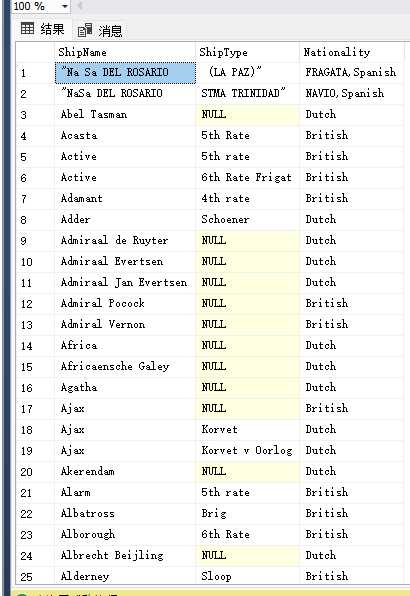
Select unique records Ships\_Staging using SQL

SQL command:

|  |
| --- |
| SELECT DISTINCT ShipName, ShipType, Nationality  FROM Ships\_staging; |

Screenshots of the executed command:





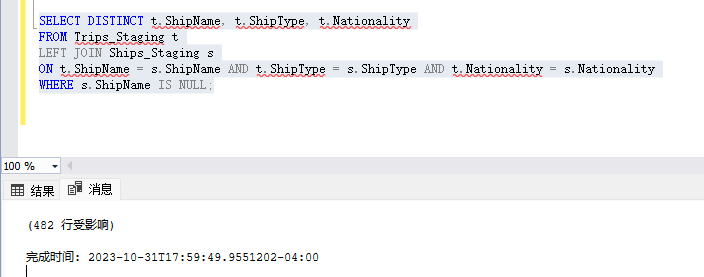
How many unique rows are there? \_1133\_\_

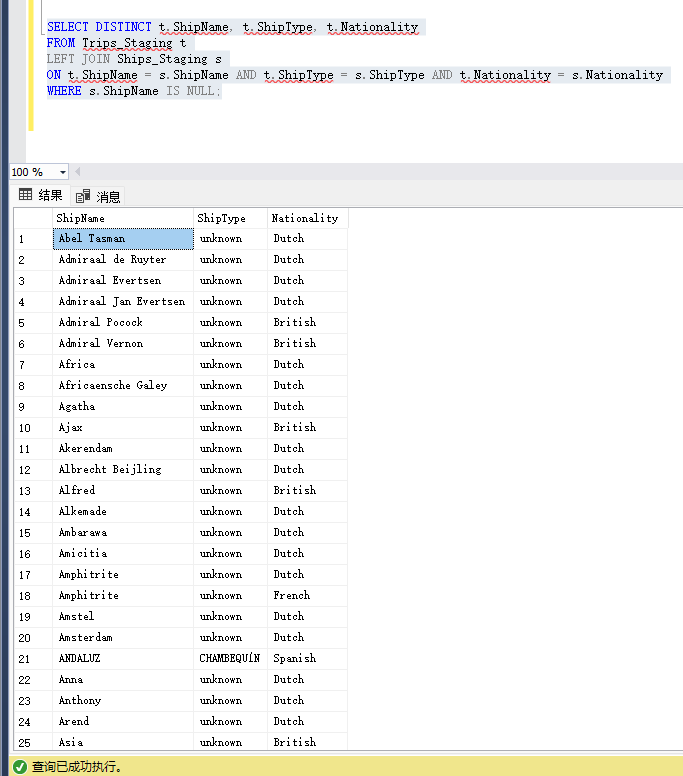
1. Recall that Trips\_Staging has a few additional dimensional records. Write a query which gets additional unique ships (ShipName, ShipType, Nationality) which are not in Ships\_Staging.
   1. Hint, this can be done via a sub-query – this is probably preferred, or using a left join filtering out the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

SQL command:

|  |
| --- |
| SELECT DISTINCT t.ShipName, t.ShipType, t.Nationality  FROM Trips\_Staging t  LEFT JOIN Ships\_Staging s  ON t.ShipName = s.ShipName AND t.ShipType = s.ShipType AND t.Nationality = s.Nationality  WHERE s.ShipName IS NULL; |

Screenshots of the executed command:





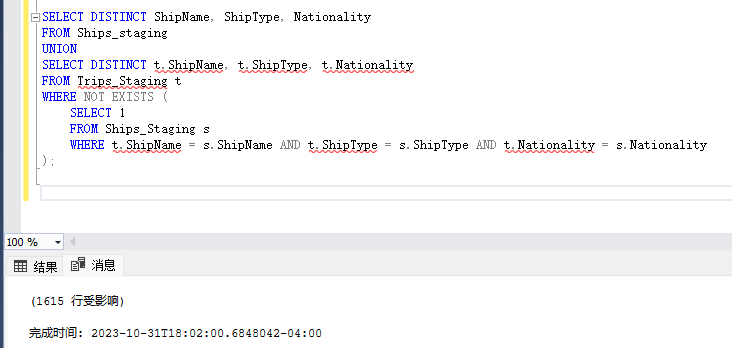
How many additional unique ship rows are there in the Trips\_Staging? \_482\_\_

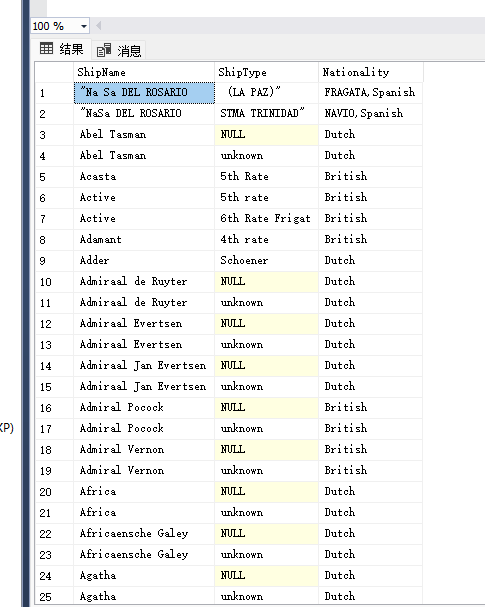
1. Combine the two queries from question 5 and 6 using the UNION command. You know have your dimensional records.

SQL command:

|  |
| --- |
| -- Selecting unique ships from Ships\_Staging  SELECT DISTINCT ShipName, ShipType, Nationality  FROM Ships\_staging  UNION  -- Selecting unique ships from Trips\_Staging that are not in Ships\_Staging  SELECT DISTINCT t.ShipName, t.ShipType, t.Nationality  FROM Trips\_Staging t  WHERE NOT EXISTS (  SELECT 1  FROM Ships\_Staging s  WHERE t.ShipName = s.ShipName AND t.ShipType = s.ShipType AND t.Nationality = s.Nationality  ); |

Screenshots of the executed command:





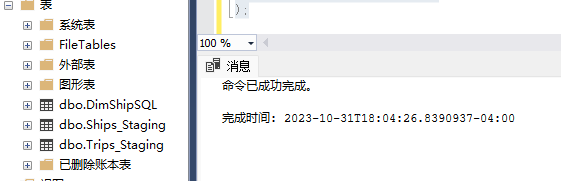
How many unique rows are there? \_1615\_\_

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 INT PRIMARY KEY IDENTITY(1,1),  ShipName VARCHAR(255),  ShipType VARCHAR(255),  Nationality VARCHAR(255)  ); |

Screenshots of the executed command:

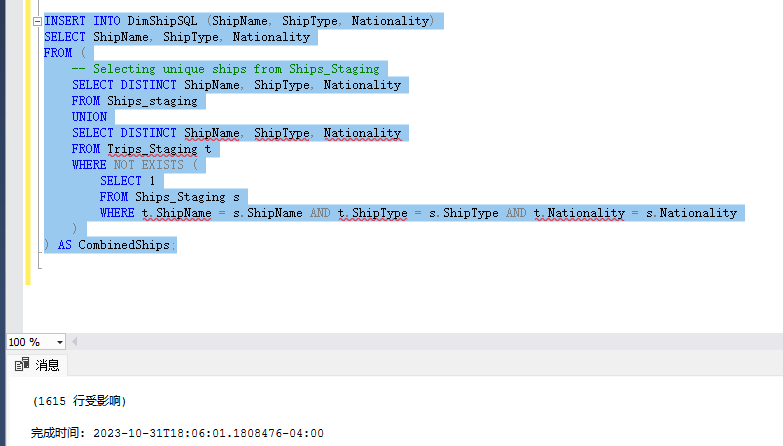


1. Load the data from the results of the previous question #8 into the DimShipSQL
   1. Hint, use the insert into command.

SQL command:

|  |
| --- |
| INSERT INTO DimShipSQL (ShipName, ShipType, Nationality)  SELECT ShipName, ShipType, Nationality  FROM (  -- Selecting unique ships from Ships\_Staging  SELECT DISTINCT ShipName, ShipType, Nationality  FROM Ships\_staging  UNION  SELECT DISTINCT ShipName, ShipType, Nationality  FROM Trips\_Staging t  WHERE NOT EXISTS (  SELECT 1  FROM Ships\_Staging s  WHERE t.ShipName = s.ShipName AND t.ShipType = s.ShipType AND t.Nationality = s.Nationality  )  ) AS CombinedShips; |

Screenshots of the executed command:

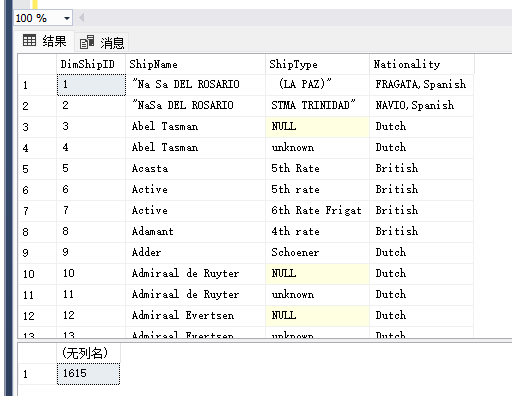


Display the loaded data from the DimShipSQL table and check the record count to make sure it matches

SQL command:

|  |
| --- |
| SELECT \* FROM DimShipSQL;  SELECT COUNT(\*) FROM DimShipSQL; |

Screenshots of the executed command:



How many rows are there in the resulting DimShipSQL? Your rows count should match the count in question 7 \_1615\_

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：Description: The Distance column quantifies the total navigational miles covered by the ship during a specific trip segment.
* Measure 2: ShipSpeed：Description: The ShipSpeed column measures the vessel's velocity, providing insights into its performance and efficiency over the journey.
* Measure 3:WindForce：Description: The WindForce column assesses the intensity of wind experienced during the trip, crucial for understanding the weather conditions and their impact on the voyage.

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.

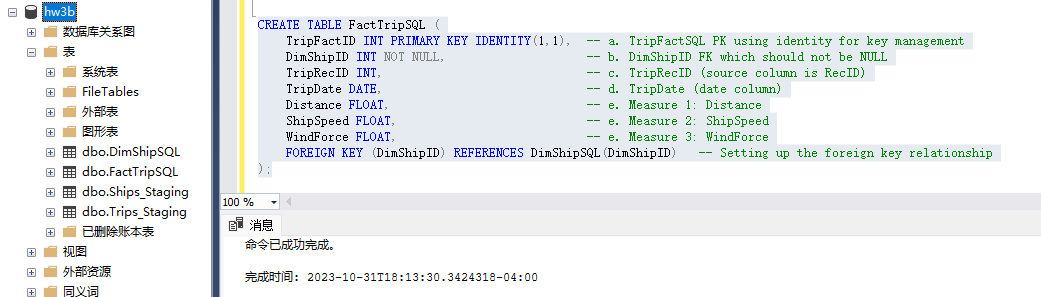
Your TripFactSQL should include the following attributes

* 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 FactTripSQL (  TripFactID INT PRIMARY KEY IDENTITY(1,1), -- a. TripFactSQL PK using identity for key management  DimShipID INT NOT NULL, -- b. DimShipID FK which should not be NULL  TripRecID INT, -- c. TripRecID (source column is RecID)  TripDate DATE, -- d. TripDate (date column)  Distance FLOAT, -- e. Measure 1: Distance  ShipSpeed FLOAT, -- e. Measure 2: ShipSpeed  WindForce FLOAT, -- e. Measure 3: WindForce  FOREIGN KEY (DimShipID) REFERENCES DimShipSQL(DimShipID) -- Setting up the foreign key relationship  ); |

Screenshots of the executed command:

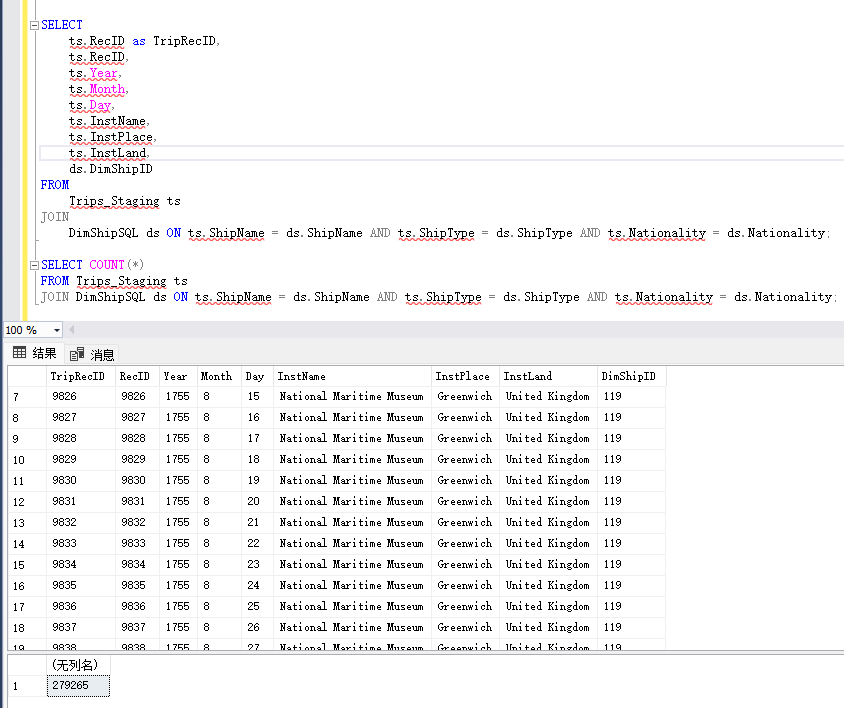


1. We will begin to shape the data which will be inserted into the TripFactSQL table by joining Trip\_Staging to the DimShipSQL – recall using the three columns that define the ship dimension as the join condition. Your resulting query will include the following fields from the Trip\_Staging table: TripRecID, 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  ts.RecID as TripRecID,  ts.RecID,  ts.Year,  ts.Month,  ts.Day,  ts.InstName,  ts.InstPlace,  ts.InstLand,  ds.DimShipID  FROM  Trips\_Staging ts  JOIN  DimShipSQL ds ON ts.ShipName = ds.ShipName AND ts.ShipType = ds.ShipType AND ts.Nationality = ds.Nationality;  SELECT COUNT(\*)  FROM Trips\_Staging ts  JOIN DimShipSQL ds ON ts.ShipName = ds.ShipName AND ts.ShipType = ds.ShipType AND ts.Nationality = ds.Nationality; |

Screenshots of the executed command:



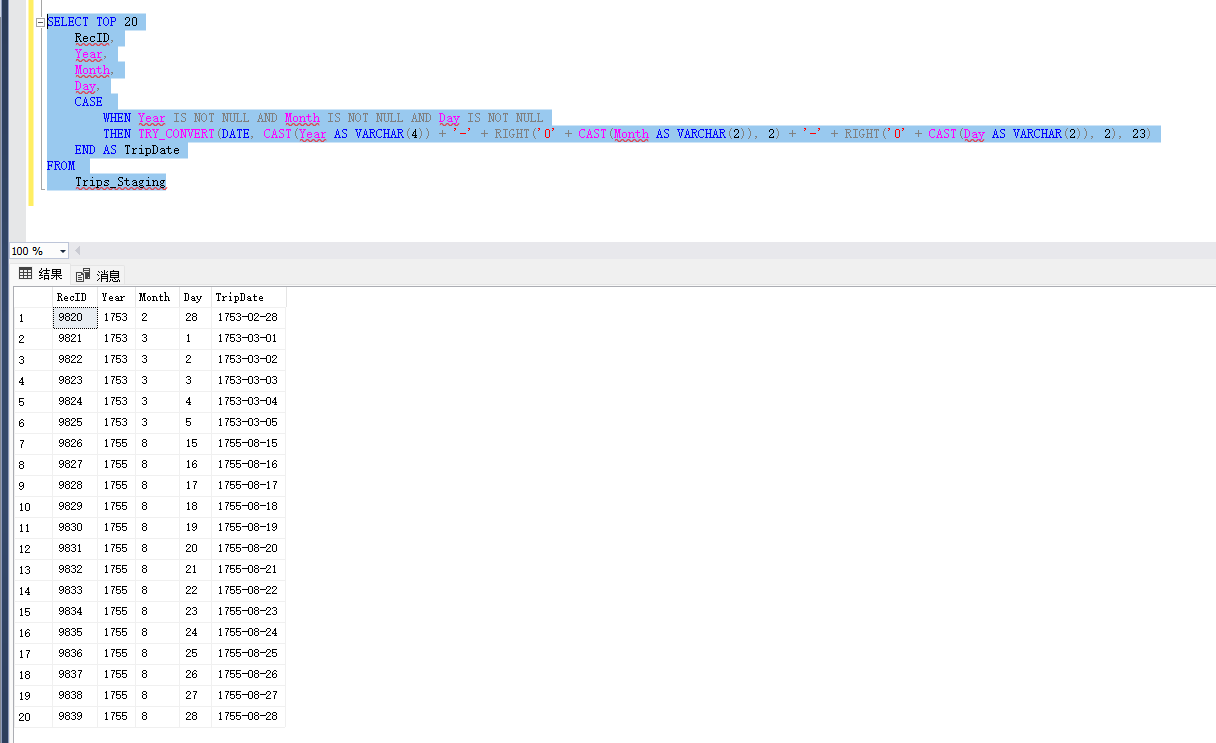
How many rows are there? \_279265\_ Compare this number of rows to how many rows there are in the Trip\_Staging table, you should have the same amount of rows.

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:

|  |
| --- |
| SELECT TOP 20  RecID,  Year,  Month,  Day,  CASE  WHEN Year IS NOT NULL AND Month IS NOT NULL AND Day IS NOT NULL  THEN TRY\_CONVERT(DATE, CAST(Year AS VARCHAR(4)) + '-' + RIGHT('0' + CAST(Month AS VARCHAR(2)), 2) + '-' + RIGHT('0' + CAST(Day AS VARCHAR(2)), 2), 23)  END AS TripDate  FROM  Trips\_Staging |

Screenshots of the executed command:



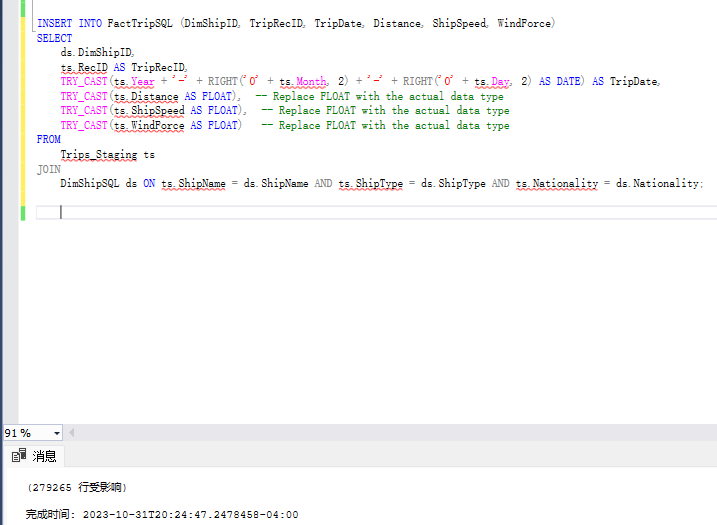
Transforming date is a single example of transformation, usually this is one of the more complicated steps. In your project, you will want to focus transforming strings and aggregating data to create measures as part of transformation instead of just extracting measures from the source file.

1. Now we are ready to load our FactTripSQL 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 FactTripSQL table.

SQL command:

|  |
| --- |
| INSERT INTO FactTripSQL (DimShipID, TripRecID, TripDate, Distance, ShipSpeed, WindForce)  SELECT  ds.DimShipID,  ts.RecID AS TripRecID,  TRY\_CAST(ts.Year + '-' + RIGHT('0' + ts.Month, 2) + '-' + RIGHT('0' + ts.Day, 2) AS DATE) AS TripDate,  TRY\_CAST(ts.Distance AS FLOAT), -- Replace FLOAT with the actual data type  TRY\_CAST(ts.ShipSpeed AS FLOAT), -- Replace FLOAT with the actual data type  TRY\_CAST(ts.WindForce AS FLOAT) -- Replace FLOAT with the actual data type  FROM  Trips\_Staging ts  JOIN  DimShipSQL ds ON ts.ShipName = ds.ShipName AND ts.ShipType = ds.ShipType AND ts.Nationality = ds.Nationality; |

Screenshots of the executed command:

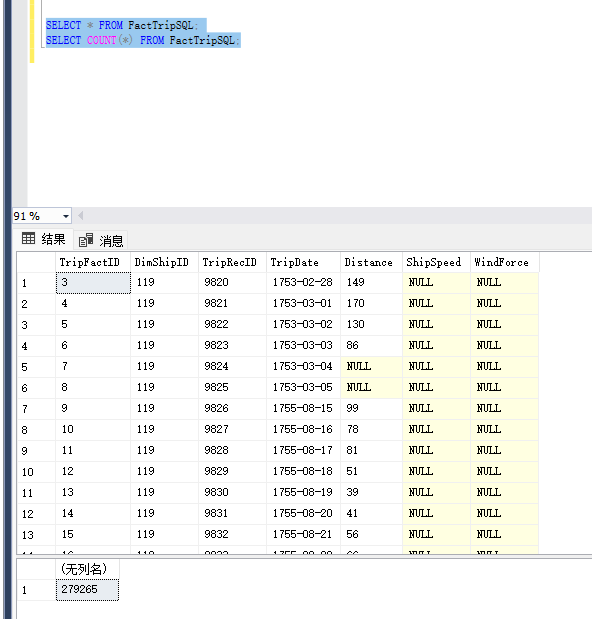


1. Display the loaded data from the FactTripSQL table and check the record count to make sure it matches to results of question 13.

SQL command:

|  |
| --- |
| SELECT \* FROM FactTripSQL;  SELECT COUNT(\*) FROM FactTripSQL; |

Screenshots of the executed command:



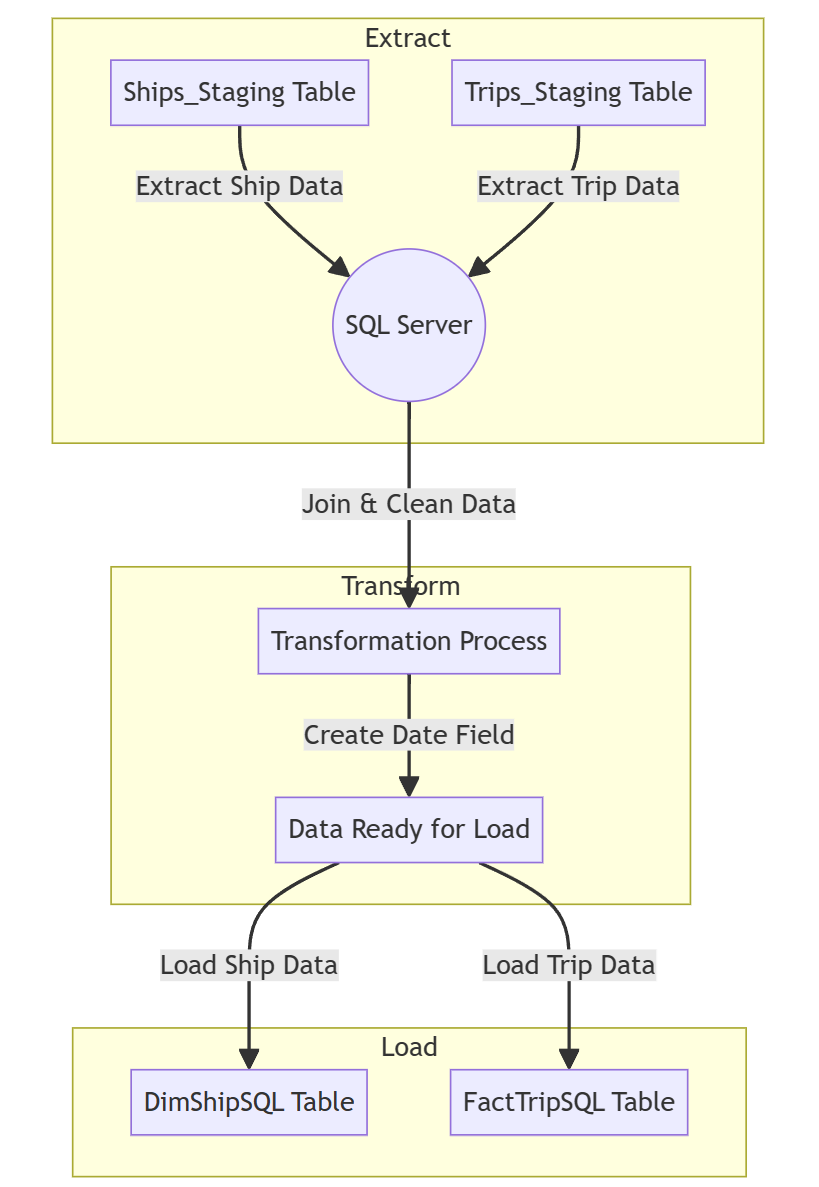
Congratulations, you have now performed ETL in both SQL and Python

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!

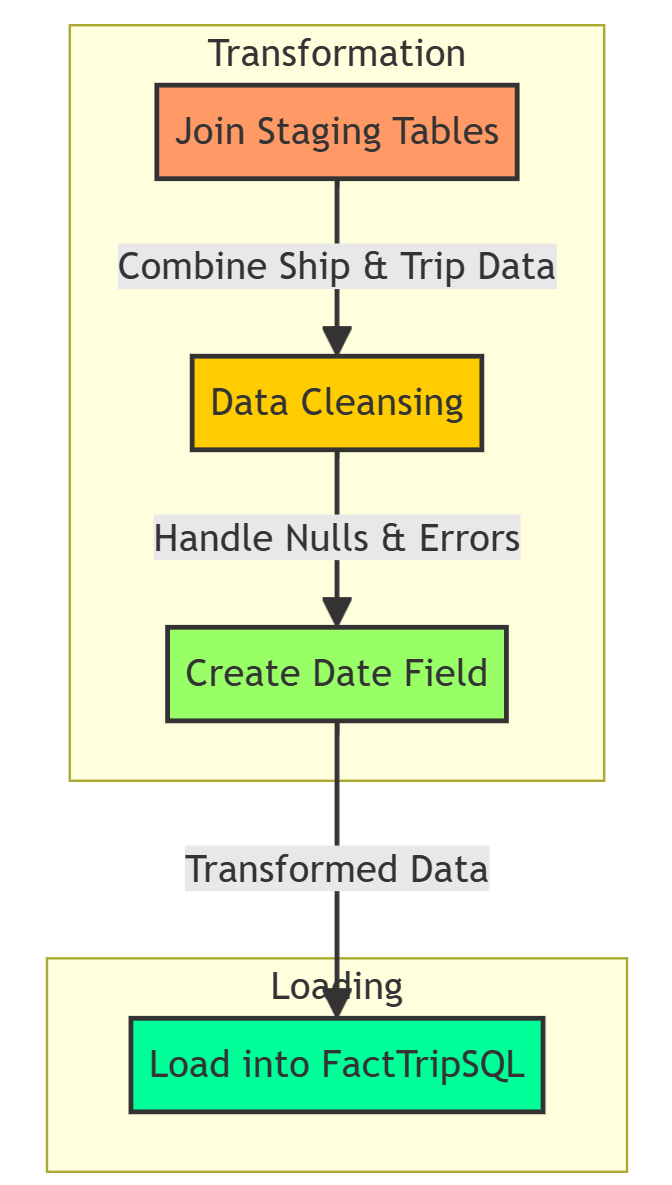
SQL excels in handling and querying relational database data directly, providing robust and efficient ways to filter, aggregate, and join tables, especially for large datasets, but can become complex for intricate data transformations. In contrast, Python, with its extensive libraries and user-friendly syntax, offers more flexibility and ease for complex data manipulation, analysis, and visualization, although it might not handle large datasets as efficiently as SQL without loading the data into memory. The ease of use in Python for complex manipulations and the efficiency of SQL for database operations make them complementary tools in data processing workflows.

# Part 4 Data flow

1. Create a physical data flow diagram depicting the ETL process in this assignment.



1. Expand on one **physical data flow** – the one where your DBMS was involved and create a **logical data flow** depicting the process. Focus and limit this to 3 logical steps within your design (not the entire transform and load process)



# 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.

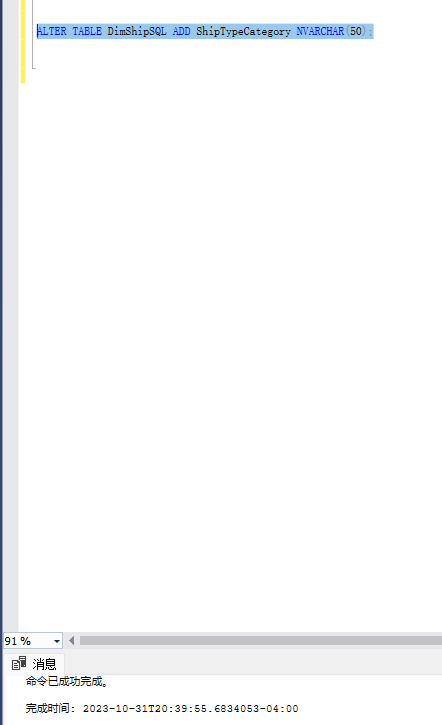
Show the commends and appropriate screenshots demonstrating your work and that the data has been loaded into the database.

SQL/Python commands:

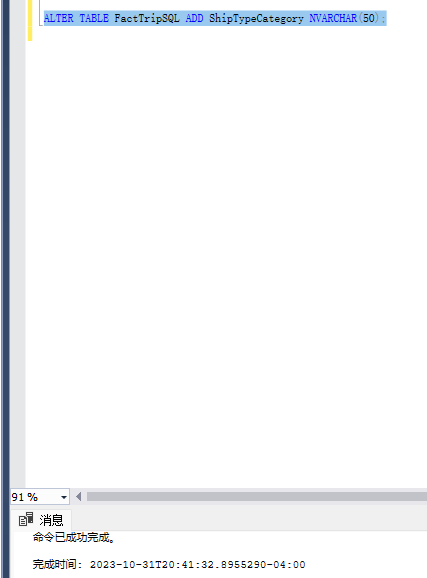
|  |
| --- |
| Add a New Column for Ship Type Category in DimShipSQL:  ALTER TABLE DimShipSQL ADD ShipTypeCategory NVARCHAR(50); |
| Update the DimShipSQL Table to Categorize Ship TypeUPDATE FactTripSQL  UPDATE DimShipSQL  SET ShipTypeCategory =  CASE  WHEN ShipType = 'Frigate' THEN 'Warship'  WHEN ShipType = 'Brig' OR ShipType = 'Schooner' THEN 'Merchant Ship'  ELSE 'Other'  END; |
| Add New Column for Ship Type Category in FactTripSQL  ALTER TABLE FactTripSQL ADD ShipTypeCategory NVARCHAR(50); |
| Update the FactTripSQL Table  UPDATE FactTripSQL  SET ShipTypeCategory = ds.ShipTypeCategory  FROM FactTripSQL ft  JOIN DimShipSQL ds ON ft.DimShipID = ds.DimShipID; |
| Validate  SELECT \* FROM DimShipSQL;  SELECT \* FROM FactTripSQL; |

Screenshots of the executed commands:

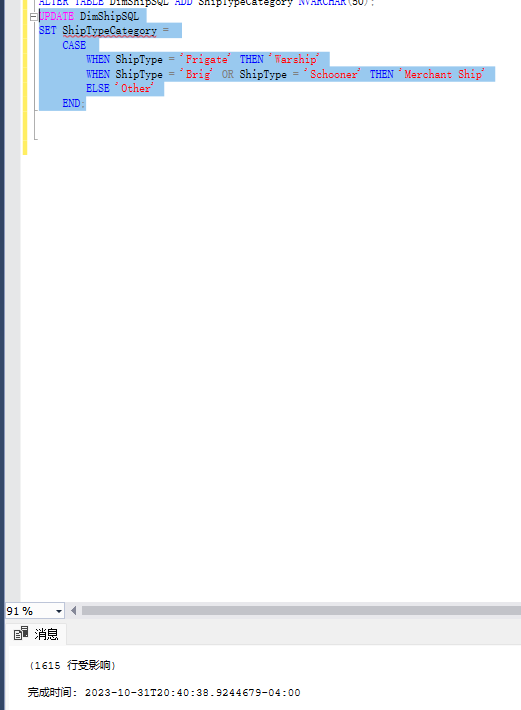
Add a New Column for Ship Type Category in DimShipSQL:



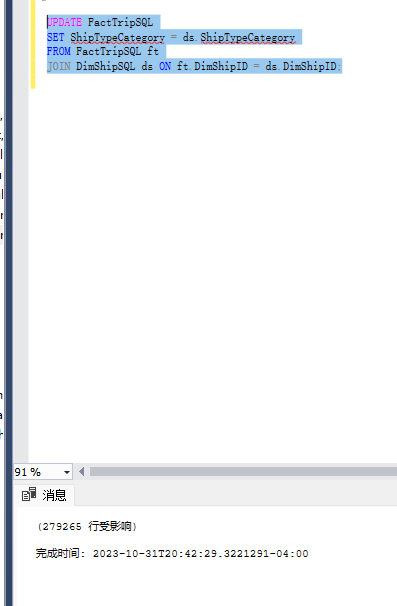
Update the DimShipSQL Table to Categorize Ship TypeUPDATE FactTripSQL



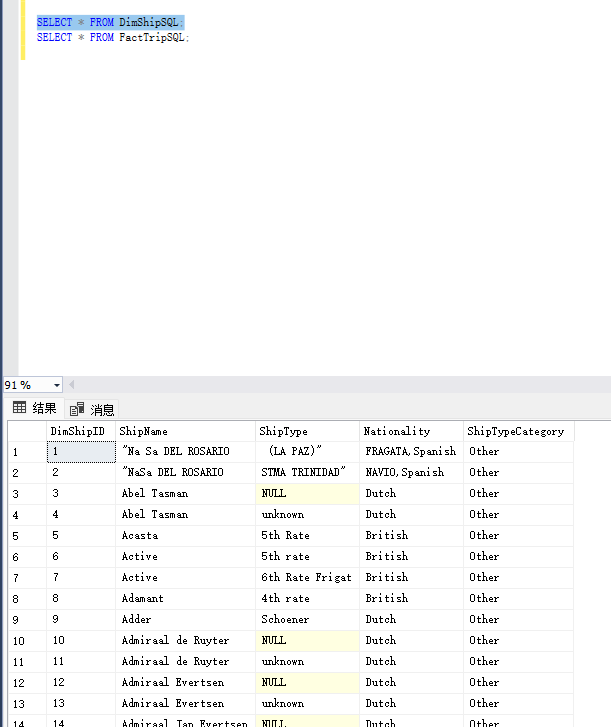
Add New Column for Ship Type Category in FactTripSQL

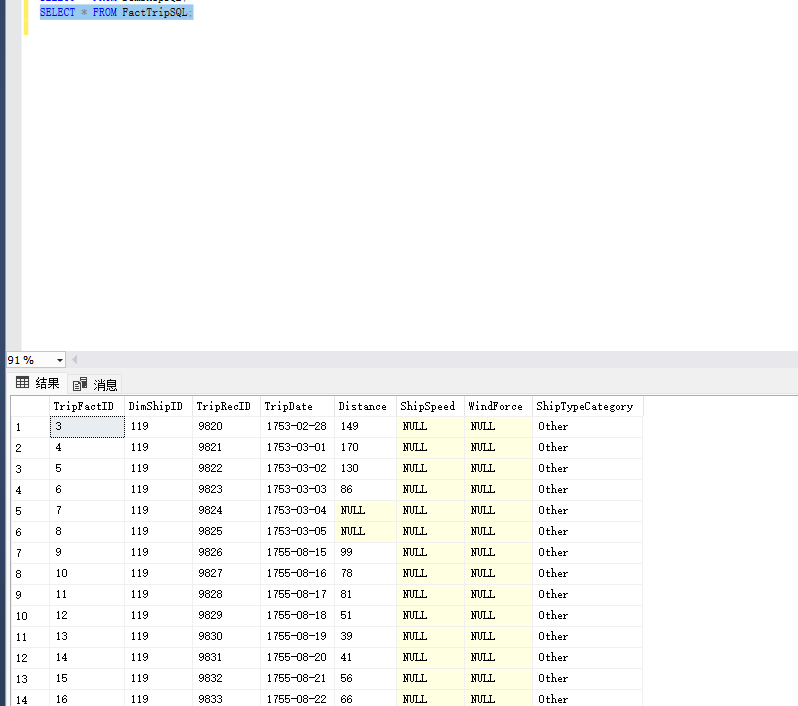


Update the FactTripSQL Table



Validate





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: |  |