# Overview of the Assignment:

ETL takes most of the data warehouse developers’ and administrators’ time. This assignment will go through some of the more common ETL processes using Python. We will explore ETL via SQL in Assignment 3B.

**Notice: All the code in this file are continued, for example, the code in question 3 is continued with the code in Question 1 and 2.**

# Part 1 –Extract, Staging

Download the two csv files Ships and CLIWOC15.csv. You just completed the “Extract” phase, it does get more complicated, but we keep it simple here.

We will begin by “staging” the data. The first step before “Transformation” is to get the data prepared or what’s referred to as “staged” before we can load it, and that initial step is to load the data into “staging tables” in this case these are going to be staging data frames. The idea of staging tables is to keep the data as close to the source format as possible. We will begin by using Python, to load the two files into two data frames called Ship\_df and Trip\_df, respectively. Review the data and note that the two files share three columns: ShipName, ShipType, and Nationality. These will be used as “natural composite key” columns to join the two data frames (tables).

Pandas Reference/Hints/Notes

* [Pandas - DataFrame Reference (w3schools.com)](https://www.w3schools.com/python/pandas/pandas_ref_dataframe.asp) is a great reference to Pandas methods
* Review run LoadTitanic.py from assignment 1 on how to create a data frame from a CSV file. Note that you don’t have to have a separate Load.py, you can paste the load command directly in the notebook, however having separate load.py source files allow you to build modularity.
* display(df) method shows all the data in the data frame – this is very helpful in understanding the data
* df.head()method allows you to inspect the header and the first 10 rows – this is very helpful in understanding the data with performance in mind.
* df.count() method will show count of records, it’s good to know what was loaded or not.
* List(df.columns.values) method shows the attributes
* Loading Trips\_df might give you an error, understand why you are getting the error, and research how to solve it (you can use the “lazy option”, and not convert the data types- especially for the initial staging table)

1. Once you load the two data frames: How many rows and columns are in each data frame?

|  |  |  |
| --- | --- | --- |
|  | Rows | Columns |
| Ship\_df | 1185 | 3 |
| Trip\_df | 280280 | 141 |

Show the record count of both Ship\_df and Trip\_df data frames as well as the count.

**Python command:**

|  |
| --- |
| import pandas as pd  # Load the CSV files into dataframes  Ship\_df = pd.read\_csv('G:\BU\_STUDY\METCS689A1\HW3A\Ships.csv')  Trip\_df = pd.read\_csv('G:\BU\_STUDY\METCS689A1\HW3A\CLIWOC15.csv')  # Show the first few rows of the dataframes to review the data  print("Ship\_df:\n",Ship\_df)  print()  print("Trip\_df:\n",Trip\_df) |

**Screenshots of the executed command:**

|  |
| --- |
|  |

# Part 2 –Creating SCD1 Dimension and Key maintenance

1. Our end goal for this section of the assignment is to create a Ship dimension table ShipDim. In this case we are going to keep it simple and use SCD type1, meaning overwrite if there are any changes, or add if it’s a new record. Recall that SCD type 1 needs to have a unique instance of each record, so let’s check if there are any duplicates in the Ship\_df

Hints:

* Review the value\_counts() function – this might only help in seeing the count for a single attribute.
* Review the groupby() function – make sure to include all three attributes as we want to find a distinct combination of all three.
* You can apply multiple functions together on a data frame. For example, you can apply both groupby first and value\_counts after to get a count of distinct values.

Show if there are any duplicate combination of all three attributes in the Ship\_df, this can just show the counts of combinations.

**Python command:**

|  |
| --- |
| The code here is continued with the code in Task 1    duplicate\_counts = Ship\_df.groupby(['ShipName', 'ShipType', 'Nationality']).size()  # Filter the groups where the count is more than 1, indicating duplicates  duplicates = duplicate\_counts[duplicate\_counts > 1]  # Show the duplicates  print("Duplicate Combinations:")  print(duplicates) |

**Screenshots of the executed command:**

|  |
| --- |
|  |

1. You will notice that Ship\_df has some duplicates that need to be removed before that data can be used to populate a ShipDim dimension table for it to be in SCD1 format. Use pandas to drop the duplicates and store the result into another data frame labelled ShipDistinct\_df.

* Hint: Review the drop\_duplicates() function.

Show the above operation to create a new ShipDistinct\_df data frame by removing the duplicate records.

**Python command:**

|  |
| --- |
| The code here is continued with the code in Task 1,2    # Remove duplicate rows based on 'ShipName', 'ShipType', and 'Nationality'  ShipDistinct\_df = Ship\_df.drop\_duplicates(subset=['ShipName', 'ShipType', 'Nationality'])  # Show the first few rows of the new dataframe to review the data  print("ShipDistinct\_df:")  print(ShipDistinct\_df.head()) |

**Screenshots of the executed command:**

|  |
| --- |
|  |

1. Show new ShipDistinct\_df data frame record count which has no duplicates, use the commands you used in question 1 and 3 on this new data frame.

**Python command:**

|  |
| --- |
| The code here is continued with the code in Task 1,2,3    # Show the number of rows in ShipDistinct\_df  num\_rows = ShipDistinct\_df.shape[0]  print("Number of rows in ShipDistinct\_df:", num\_rows)  # Check for duplicates based on 'ShipName', 'ShipType', and 'Nationality'  duplicates\_check = ShipDistinct\_df.duplicated(subset=['ShipName', 'ShipType', 'Nationality'])  any\_duplicates = duplicates\_check.any()  # Show if there are any duplicates  print("Are there any duplicates in ShipDistinct\_df based on 'ShipName', 'ShipType', and 'Nationality'? :", any\_duplicates) |

**Screenshots of the executed command:**

|  |
| --- |
|  |

How many rows are in ShipDistinct\_df after duplicates have been dropped? \_1174\_

Note: the new count should make sense in reviewing groupby count results between the two data frames.

1. Now let’s focus on the Trip\_df. Trip has some additional ships (ShipName, ShipType, Nationality) that do not currently appear in ShipDistinct\_df. These new dimension rows need to be pulled and added to the dimension table. Inspect the column names of the Trips\_df and provide screenshot of the columns.

Hint: Look into columns.tolist() function, if not all columns display, modify settings to display all columns by using pd.set\_option('display.max\_columns', None)

**Python command:**

|  |
| --- |
| The code here is continued with the code in Task 1,2,3,4    # Set pandas option to display all columns  pd.set\_option('display.max\_columns', None)  # Get and display all column names from Trip\_df  column\_names = Trip\_df.columns.tolist()  print("Column names in Trip\_df:")  print(column\_names) |

**Screenshots of the executed command:**

|  |
| --- |
|  |

1. Our goal is to identify any Ships in Trips\_df that is not in ShipDistinct. We can use a LEFT JOIN on these dataframes to determine which values of (ShipName, ShipType, Nationality) are in Trip\_df but not in ShipDistinct\_df.

First create a joined data frame called ShipTrips\_df by using the merge() function on the Trips\_df and the ShipDistinct\_df; use a left join on (ShipName, ShipType, Nationality) and set the indicator to True.

Show the merge command to create the ShipTrips\_df data frame.

**Python command:**

|  |
| --- |
| The code here is continued with the code in Task 1,2,3,4,5    # Perform a left join on (ShipName, ShipType, Nationality)  ShipTrips\_df = pd.merge(Trip\_df, ShipDistinct\_df, on=['ShipName', 'ShipType', 'Nationality'], how='left', indicator=True)  # Show the first few rows of the resulting dataframe to verify the join  print("ShipTrips\_df:")  print(ShipTrips\_df.head()) |

**Screenshots of the executed command:**

|  |
| --- |
| 屏幕截图 2023-10-22 000614  屏幕截图 2023-10-22 000627  屏幕截图 2023-10-22 000634 |

1. Inspect the resulting ShipsTrips\_df data frame (use the head() function), specifically scroll all the way to the right and note the \_merge column that has been added. Let’s determine the unique combinations of \_merge column by using the value\_counts() function.

Show the value\_counts() of the \_merge column.

**Python command:**

|  |
| --- |
| # Show the first few rows of ShipTrips\_df with all columns, focusing on the \_merge column  print("ShipTrips\_df:")  print(ShipTrips\_df.head())  # Determine the unique combinations in the \_merge column  merge\_counts = ShipTrips\_df['\_merge'].value\_counts()  print("\nCounts of unique combinations in the \_merge column:")  print(merge\_counts)  print() |

**Screenshots of the executed command:**

|  |
| --- |
|  |

For each of the resulting values (you should see three) of the results above**, very briefly explain what it means – short, bulleted list.**

* **Your explanation goes here….**

|  |
| --- |
| The \_merge column in the resulting data frame from a merge operation in pandas indicates the source of each row. Here are the meanings of the unique values found in the \_merge column:  both:  Rows with this value indicate that the row’s key is present in both the left and right dataframes.  In this specific case, there are 277,150 such rows, meaning a vast majority of rows in the ShipsTrips data frame had matching keys in the other dataframe involved in the merge.  left\_only:  Rows with this value indicate that the row’s key is only present in the left dataframe.  There are 3,130 rows like this in the ShipsTrips data frame, suggesting that there were log entries that did not have corresponding information in the other dataframe.  right\_only:  Rows with this value indicate that the row’s key is only present in the right dataframe.  There are 0 rows like this in the ShipsTrips data frame, meaning there was no information in the right dataframe that didn’t have a corresponding log entry in the ShipsTrips dataframe.  In summary, the \_merge column provides a straightforward way to understand the results of a merge operation, showing clearly which rows come from which dataframes or both. In this case, the vast majority of the data was matched (both), but there were some entries unique to the ShipsTrips data frame (left\_only), and none unique to the right dataframe (right\_only). |

1. Now we can filter out the new records we will need to bring into our Ships from the joined data frame, decide the filter condition based on the results from question 7.

Hint: Look into query and filter functions to use on the ShipsTrips\_df. The filter function will show the attributes that we want to look at, while the query function will help us filter the results

Provide the function call for the ShipsTrips\_df showing only the columns that we need (ShipName, ShipType, Nationality), as well as the \_merge column. The function call should filter (query) \_merge column as outlined in the directions above.

**Python command:**

|  |
| --- |
| new\_records\_df = ShipTrips\_df.query('\_merge == "left\_only"')[['ShipName', 'ShipType', 'Nationality', '\_merge']]  # Show the filtered DataFrame  print("New records to bring into Ships:")  print(new\_records\_df.head())  print() |

**Screenshots of the executed command:**

|  |
| --- |
|  |

1. Your rows count should match the count in question 7. How many new records were found? \_3130\_

|  |
| --- |
| num\_new\_records = len(new\_records\_df)  print("Number of new records found:", num\_new\_records)  print() |

1. Now let’s find the distinct instances of the ShipName, ShipType, and Nationality. Perform the same operations as outlined in steps 2 through 4 to create a ShipsTrips\_Distinct\_df (this new data frame should not have any duplicates)

**Python command:**

|  |
| --- |
| # Group by 'ShipName', 'ShipType', and 'Nationality', then count the size of each group  ships\_trips\_duplicate\_counts = new\_records\_df.groupby(['ShipName', 'ShipType', 'Nationality']).size()  # Filter the groups where the count is more than 1, indicating duplicates  ships\_trips\_duplicates = ships\_trips\_duplicate\_counts[ships\_trips\_duplicate\_counts > 1]  # Show the duplicates  print("Duplicate Combinations in ShipsTrips:")  print(ships\_trips\_duplicates)  print()  # 10.2 Remove Duplicate Records  # Remove duplicate rows based on 'ShipName', 'ShipType', and 'Nationality'  ShipsTrips\_Distinct\_df = new\_records\_df.drop\_duplicates(subset=['ShipName', 'ShipType', 'Nationality'])  # Show the first few rows of the new dataframe to review the data  print("ShipsTrips\_Distinct\_df:")  print(ShipsTrips\_Distinct\_df.head())  print()  # 10.3 Verify Removal of Duplicates  # Show the number of rows in ShipsTrips\_Distinct\_df  num\_rows\_distinct = ShipsTrips\_Distinct\_df.shape[0]  print("Number of rows in ShipsTrips\_Distinct\_df:", num\_rows\_distinct)  # Check for duplicates based on 'ShipName', 'ShipType', and 'Nationality'  duplicates\_check\_distinct = ShipsTrips\_Distinct\_df.duplicated(subset=['ShipName', 'ShipType', 'Nationality'])  any\_duplicates\_distinct = duplicates\_check\_distinct.any()  # Show if there are any duplicates  print("Are there any duplicates in ShipsTrips\_Distinct\_df based on 'ShipName', 'ShipType', and 'Nationality'? :", any\_duplicates\_distinct)  print() |

**Screenshots of the executed command:**

|  |
| --- |
|  |

1. Show the resulting data frame – list all the data.

**Python command:**

|  |
| --- |
| # To show the entire DataFrame  print(ShipsTrips\_Distinct\_df)  # To get the number of rows (records)  num\_records = ShipsTrips\_Distinct\_df.shape[0]  print("Number of new records in ShipsTrips\_Distinct\_df:", num\_records)  print() |

**Screenshots of the executed command:**

|  |
| --- |
|  |

How many new records are in the ShipsTrips\_Distinct\_df? \_\_11\_

1. Combine the two distinct data frames (ShipsDistinct\_df and ShipsTrips\_Distinct\_df ) into DimShip data frame. Hint: Use the pandas append() or the pd.concat() function, look to ignore the existing index as we will create a new surrogate primary key in the next step.

Show the command combining the data frames into a single DimShip.

**Python command:**

|  |
| --- |
| # Combining the data frames into a single DimShip data frame  DimShip = pd.concat([ShipDistinct\_df, ShipsTrips\_Distinct\_df], ignore\_index=True)  # Showing the first few rows of the DimShip data frame  print(DimShip.head())  print()  num\_records = len(DimShip)  print("Number of records in DimShip:", num\_records)  print() |

**Screenshots of the executed command:**

|  |
| --- |
|  |

How many records are in DimShip now? \_1185\_\_

1. Now we need to create a surrogate primary key for the DimShip data frame. Use the reset\_index() function to add a column to the DimShip data frame and call the column “Id”, start the index at 1. Hint: investigate how to add a new column to the existing data frame.

Show the command creating the surrogate key for DimShip and a separate command showing the new index column. Use display() method.

**Python command:**

|  |
| --- |
| # Adding a new column 'Id' that serves as a surrogate primary key, starting the index at 1  DimShip = DimShip.reset\_index(drop=True)  DimShip['Id'] = DimShip.index + 1  # Displaying the DimShip data frame to show the new 'Id' column  print(DimShip)  print() |

**Screenshots of the executed command:**

|  |
| --- |
| 屏幕截图 2023-10-22 002645 |

Congratulations, you have now created a clean data frame called DimShip which includes distinct record combinations as well as a primary key.

# Part 3 –Creating Fact data frame

Now that we have a DimShip dimension, lets focus on creating a FactTrip dat aframe. There are many columns in the Trip\_df – if you recall fact tables contain measures. Some of the columns are good candidates for additional dimensions, we are going to keep it simple for now and focus on creating a Fact table. Select three or four numeric columns from the original Trip\_df dataframe which you will use.

1. List the three measures columns you will use:

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

1. Create a new dataframe called FactTrip that includes the following attributes:
   1. ShipName, ShipType, Nationality – these are our natural key to connect the fact table to the ShipDim
   2. RecID, Year, Month, Day (we will work with dates in a later question)
   3. The Id from the DimShip dataframe – you will need to join to the DimShip dataFrame to get this attribute. Make sure to call this attribute DimShipId
   4. Three numeric values which you selected in question 15.

Provide the command(s) creating the FactTrip data frame and the results of displaying some sample data from the FactTrip. Provide a third screenshot verifying counts between the original Trip\_df and FactTrip.

**Python commands:**

|  |
| --- |
| # Selecting necessary attributes from Trip\_df  selected\_columns = Trip\_df[['ShipName', 'ShipType', 'Nationality', 'RecID', 'Year', 'Month', 'Day', 'ShipSpeed', 'WindForce', 'BaroReading']]  # Merging with DimShip to get DimShipId  FactTrip = pd.merge(selected\_columns, DimShip[['ShipName', 'ShipType', 'Nationality', 'Id']],                      on=['ShipName', 'ShipType', 'Nationality'], how='left')  # Renaming the Id column to DimShipId  FactTrip = FactTrip.rename(columns={'Id': 'DimShipId'})  # Displaying sample data from FactTrip  print("FactTrip DataFrame:")  print(FactTrip.head())  # Verify counts between the original Trip\_df and FactTrip  print("\nCount of rows in Trip\_df:", len(Trip\_df))  print("Count of rows in FactTrip:", len(FactTrip)) |

**Screenshots of the executed command:**

|  |
| --- |
|  |

Which counts specifically from the two data frames gives you confidence in the FactTrip data frame? (Short answer- single sentence)

**Your answer goes here:**The matching row counts between the Trip\_df DataFrame and the FactTrip DataFrame give confidence in the FactTrip data frame, ensuring that all trips from the original dataset are represented.

1. Add a surrogate key to the FactTrip data frame like we did in step 13 for the DimShip data frame.
2. Provide the command and results of creating the surrogate key for FactTrip and a second command and results showing the new index column including several rows of the FactTrip data frame. Use display() method.

**Python commands:**

|  |
| --- |
| # Resetting the index of the FactTrip DataFrame  FactTrip = FactTrip.reset\_index(drop=True)  # Adding a new column 'Id' as the surrogate key  FactTrip['Id'] = FactTrip.index + 1  # Displaying the FactTrip DataFrame to show the new 'Id' column  display(FactTrip.head()) |

**Screenshots of the executed commands:**

|  |
| --- |
|  |

# Part 4 – Transformation: Dates

There are a few columns in the FactTrip data frame that, together, indicate a specific date: Year, Month, Day. We will use the next step to transform those columns into a date column.

1. Add two new columns to the data frame and populate one with a string for the date and the second with the date calculated from the string. Hint, look into date formatting and string conversion as well as the to\_datetime() function and pay attention to how to handle errors (errors='coerce') and format.

**Python command:**

|  |
| --- |
| FactTrip['DateString'] = pd.to\_datetime(FactTrip[['Year', 'Month', 'Day']].astype(str).agg('-'.join, axis=1), errors='coerce', format='%Y-%m-%d')  FactTrip['Date'] = pd.to\_datetime(FactTrip['DateString'], errors='coerce')  print(FactTrip) |

**Screenshots of the executed command:**

|  |
| --- |
|  |

1. Take a screen shot showing the first 10-20 rows of the updated FactTrip data frame.

**Python command:**

|  |
| --- |
| print(FactTrip.head(20).to\_string()) |

**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.

# Part 5 – Load the data frames into the database tables

Create a new database in your system and create a table called Dim\_Ship and load it with the DimShip data frame. The Dim\_Ship table should have all the attributes from the DimShip including the primary key. You can choose to use SQL or Python to create the Dim\_Ship table. Using Python, populate the new Dim\_Ship table.

1. Take a screen shot of your command to create the Dim\_Ship table in your database

**Python/or SQL command:**

|  |
| --- |
| import pyodbc as db  # SQL Server  conn = db.connect('Driver={SQL Server};'                  'Server=XUYUHAN;'                 'Server=SQLServer-PC;'                 'Database=hw3;'                  'Trusted\_Connection=yes;')  cursor = conn.cursor()  stateQuery = "IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Dim\_Ship') DROP TABLE Dim\_Ship"  cursor.execute(stateQuery)  conn.commit()  create\_table\_query = """  CREATE TABLE Dim\_Ship (      ShipKey INT PRIMARY KEY,      ShipName VARCHAR(255),      ShipType VARCHAR(255),      Nationality VARCHAR(255)  );"""  cursor.execute(create\_table\_query)  conn.commit()  print("Dim\_Ship table created successfully!")  cursor.close()  conn.close()  print() |

**Screenshots of the executed command:**

|  |
| --- |
|  |

1. Show your command of loading the DimShip into the database using Python.

**Python command:**

|  |
| --- |
| DimShip1 = DimShip.drop(['\_merge'], axis = 1)  print(DimShip1)  import pyodbc as db  # SQL Server  conn1 = db.connect('Driver={SQL Server};'                  'Server=XUYUHAN;'                 'Server=SQLServer-PC;'                 'Database=hw3;'                  'Trusted\_Connection=yes;')  cursor1 = conn1.cursor()  insert\_query = """  INSERT INTO Dim\_Ship (ShipKey ,ShipName, ShipType, Nationality)  VALUES (?, ?, ?, ?)  """  # Iterate over the rows of the DataFrame and insert each row into the database  for index, row in DimShip1.iterrows():      Ship\_Key = index + 1        cursor1.execute(insert\_query, row['Id'], row['ShipName'], row['ShipType'], row['Nationality'])  # Commit the transaction  conn1.commit()  # Close the cursor and connection  cursor1.close()  conn1.close()  print("Data inserted into Dim\_Ship table successfully!") |

**Screenshots of the executed command:**

|  |
| --- |
| 屏幕截图 2023-10-22 020000屏幕截图 2023-10-22 005932 |

1. Display the loaded data from the Dim\_Ship table in your Database using SQL

**SQL command:**

|  |
| --- |
| SELECT \* FROM Dim\_Ship |

**Screenshots of the executed command:**

|  |
| --- |
| 屏幕截图 2023-10-22 020123 |

Next, we will create a table called Fact\_Trip and load it with the FactTrip data frame data. Fact\_Trip table will contain the following attributes from the FactTrip data frame The primary key, for foreign key to the Dim\_Ship, the Date you converted, and the three measures. You will want to skip loading ShipName, ShipType and Nationality fields as you now have a foreign key to it and these are no longer needed. You can choose to use SQL or Python to create the table. Use Python to populate the FactTrip table.

1. Take a screen shot of your command to create the Fact\_Trip table in your database

**Python/SQL command:**

|  |
| --- |
| print(FactTrip)  FactTrip1 = FactTrip.drop(['ShipSpeed'], axis = 1)  FactTrip1 = FactTrip1.drop(['BaroReading'], axis = 1)  conn2 = db.connect('Driver={SQL Server};'                  'Server=XUYUHAN;'                 'Server=SQLServer-PC;'                 'Database=hw3;'                  'Trusted\_Connection=yes;')  cursor2 = conn2.cursor()  stateQuery = "IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Fact\_Trip') DROP TABLE Fact\_Trip"  cursor2.execute(stateQuery)  conn2.commit()  create\_table\_query = """  CREATE TABLE Fact\_Trip (      ShipKey INT PRIMARY KEY,      ShipName VARCHAR(255),      ShipType VARCHAR(255),      Nationality VARCHAR(255),      RecID INT,      Year INT,      Month INT,      Day INT,      WindForce VARCHAR(255),  DimShipId INT  );"""  cursor2.execute(create\_table\_query)  conn2.commit()  print("Dim\_Ship table created successfully!")  cursor2.close()  conn2.close()  print() |

**Screenshots of the executed command:**

|  |
| --- |
|  |

1. Take a screen shot of your command loading the FactTrip into the database using Python

**Python command:**

|  |
| --- |
| conn3 = db.connect('Driver={SQL Server};'                  'Server=XUYUHAN;'                 'Server=SQLServer-PC;'                 'Database=hw3;'                  'Trusted\_Connection=yes;')  cursor3 = conn3.cursor()  insert\_query = """  INSERT INTO Fact\_Trip (ShipKey, ShipName, ShipType, Nationality, RecID, Year, Month, Day, WindForce, DimShipId)  VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)  """  # Iterate over the rows of the DataFrame and insert each row into the database  for index, row in FactTrip1.iterrows():      cursor3.execute(insert\_query, row['Id'], row['ShipName'], row['ShipType'], row['Nationality'], row['RecID'], row['Year'], row['Month'], row['Day'], row['WindForce'], row['DimShipId'])  # Commit the transaction  conn3.commit()  # Close the cursor and connection  cursor3.close()  conn3.close()  print("Data inserted into Fact\_Trip table successfully!") |

**Screenshots of the executed command:**

|  |
| --- |
|  |

1. Take a screen shot selecting the loaded data from the Fact\_Trip table in your Database using SQL

**SQL command:**

|  |
| --- |
| SELECT \* from Fact\_Trip |

**Screenshots of the executed command:**

|  |
| --- |
|  |

# Extra Credit - Extending the dimension

(Up-to 5 extra credit points)

Some of the fields in the CLIWOC.csv could be a new dimension, or part of the Dim\_Ship 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.

**Python commands:**

|  |
| --- |
| conn4 = db.connect('Driver={SQL Server};'                  'Server=XUYUHAN;'                 'Server=SQLServer-PC;'                 'Database=hw3;'                  'Trusted\_Connection=yes;')  cursor4 = conn4.cursor()  # 1. Check the first few rows in Dim\_Ship  print("Dim\_Ship Table:")  cursor4.execute("SELECT \* FROM Dim\_Ship;")  for row in cursor4.fetchmany(5):      print(row)  print("\nFact\_Trip Table:")  # 2. Check the first few rows in Fact\_Trip  cursor4.execute("SELECT \* FROM Fact\_Trip;")  for row in cursor4.fetchmany(5):      print(row)  # 3. Find the top 5 most common ship types  print("\nTop 5 Most Common Ship Types:")  query = """  SELECT ShipType, COUNT(\*) as ShipCount  FROM Dim\_Ship  GROUP BY ShipType  ORDER BY ShipCount DESC  """  cursor4.execute(query)  for row in cursor4.fetchmany(5):      print(row)  # 4. Find the average number of trips per ship  print("\nAverage Number of Trips per Ship:")  query = """  SELECT AVG(TripCount) as AvgTripsPerShip  FROM (SELECT ShipKey, COUNT(\*) as TripCount        FROM Fact\_Trip        GROUP BY ShipKey) as SubQuery  """  cursor4.execute(query)  for row in cursor4.fetchall():      print(row)  cursor4.close()  conn4.close() |

**Screenshots of the executed commands:**

|  |
| --- |
|  |

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\_A.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: |  |