Extracting Facts and Dimensions using SQL

Processes to be performed

1. **Examine the data file and identify the entities present. You may do this by examining the text file or by importing the data into SQL Server and examining it there.**
2. **Using either top-down design or bottom-up normalisation, identify the entities present and their relationships.**
3. **Design tables to hold the data.**
4. **Populate the tables.**

Instructions

1. Create an empty database in SQL Server and name it SalesDW. We will import the data file into this database.
2. Import the ProductSales.csv file into the SalesDW database. Pay attention to column data types. Ensure you set the following columns accordingly:
    - RowID – Two byte unsigned unteger (DT_UI2)
    - CustID – Two byte unsigned unteger (DT_UI2)
    - UnitsSold – Two byte unsigned integer (DT_UI2)
    - DateSold – Database Date (DT_DBDATE)
    - StdCost - Numeric (with Precision 8 and Scale 2)
    - StdPrice – Numeric (with Precision 8 and Scale 2)
3. List the data elements (entities) present. For each entity, state whether it has a suitable primary key. You should have a list like shown in Figure 1:

Entity	Primary Key?
Customer	**Yes**
Product	**Yes**
Country	**No**
Region	**No**
Sales Channel	**No**
Sale data	**No**

Figure 1

4. The next stage is to create tables to hold the data. In some cases, primary keys will need to be added. You need to create and populate an entity that's lacking a primary key.
5. Create a SalesChannel table using SQL. The SQL in Figure 2 will create a primary key as the data does not have this already, and setting the column as an identity column will arrange values automatically as you add rows. Add two timestamp columns for CreateTimestamp and Updatetimestamp. These are used in data warehouses to keep track of when records were loaded and updated in the data warehouses (as opposed to when they were created or updated in the operational source systems).

```
-- Create SalesChannel table
CREATE TABLE SalesChannel
(
	ChannelID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
	ChannelName VARCHAR(10),
	CreateTimestamp DATETIME,
	UpdateTimestamp DATETIME
)
```

Figure 2

6. Populate the table using SQL. The SQL in Figure 3 uses the DISTINCT keyword to ensure only one of each value is inserted. The identity column is populated automatically and so should not be included in the INSERT statement. The CURRENT_TIMESTAMP is a SQL Server function we can use to generate the timestamp for the Create and UpdateTimestamp columns. They return the current timestamp from SQL Server at the point they are called as part of the INSERT statement.

```
-- Insert SalesChannel Data
INSERT INTO SalesChannel
SELECT DISTINCT [SalesChannel],
CURRENT_TIMESTAMP AS CreateTimestamp,
CURRENT_TIMESTAMP AS UpdateTimestamp
FROM SalesDW.[dbo].[ProductSales]
```

Figure 3

7. Specify a CREATE TABLE statement for the Region table. You will need to specify the columns yourself. Make sure you include the two timestamps for Create and Update Timestamp. Every table should have these two timestamp columns as part of data housekeeping.

8. Before you load the Region data, take a moment to review the data quality; you will notice there is an issue with the Region Name. Can you identify what the root cause of this issue is? As you don't have control over the source of this data, you'll have to perform a fix in your staging table. A good method for this it to create a new column to hold the clean version of the region table. This allows you to always have a copy of the original Region column should we need to make further amendments.

    - Create a new column in your Staging Table for RegionClean (Figure 4).

```
ALTER TABLE ProductSales
ADD  RegionClean varchar(50)
```

Figure 4

    - Use a SQL Update statement to populate the column using the amendment shown in Figure 5. The other Region values can stay as they are.
    Hint: Use a CASE statement in the UPDATE statement

Original Value	Amended Value
Central America and the C	Central America and the Caribbean
Middle East and North Afr	Middle East and North Africa

Figure 5

    - Review your new column to ensure the update has worked successfully.

9. Some tables have suitable primary keys already. Figure 6 shows an example of how to create and populate the Customer table.

```
-- Create Customer Table
CREATE TABLE Customer
(
    CustID INT NOT NULL PRIMARY KEY,
    CustName VARCHAR(50),
    CreateTimestamp DATETIME,
    UpdateTimestamp DATETIME
)


--Insert Data
INSERT INTO Customer
SELECT DISTINCT [CustID],
[CustName],
CURRENT_TIMESTAMP AS CreateTimestamp,
CURRENT_TIMESTAMP AS UpdateTimestamp
FROM SalesDW.[dbo].[ProductSales]
```

Figure 6

8. Now do the same for the Product table and insert the data. You need to specify the columns yourself.

9. You could create the Country table in the same way as the Region table but, as a country will always be part of a region, it makes sense to use this relationship. This is a little trickier as you have to look up the ID of the region that the country is in when you populate the table (so you must have populated the region table FIRST). Create the Country table with an extra column for the foreign key (Figure 7).

```
-- Create Country Table
CREATE TABLE Country
(
	CountryID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
	CountryName VARCHAR(50),
	RegionID INT FOREIGN KEY REFERENCES Region(RegionID),
	CreateTimestamp DATETIME,
	UpdateTimestamp DATETIME
)
```

Figure 7

There are a couple of ways that you can make sure that the relationship between the country and Region tables is implemented correctly.

Method 1: Using a sub-query (Figure 8)

A normal insert except a sub-query looks up the ID of the matching region name in the Region table.

```
-- Insert Country Data
INSERT INTO Country
SELECT DISTINCT [Country],
 (
	SELECT RegionID
	FROM Region
	WHERE RegionName = S.RegionClean
) ,
CURRENT_TIMESTAMP AS CreateTimestamp,
CURRENT_TIMESTAMP AS UpdateTimestamp
FROM [SalesDW].[dbo].[ProductSales] AS S
```

Note: You're using RegionClean in the WHERE

Figure 8

Method 2: Using a Join (Figure 9)

This method would be more efficient for large amounts of data as it does not have a lookup query for each inserted row.

```
-- Insert Country Data
```

```
INSERT INTO Country
SELECT DISTINCT S.Country,
R.RegionID,
CURRENT_TIMESTAMP AS CreateTimestamp,
CURRENT_TIMESTAMP AS UpdateTimestamp
FROM Region AS R
INNER JOIN SalesDW.dbo.ProductSales AS S ON R.RegionName = S.RegionClean
```

Figure 9

The final data population is of the Sale table which links the other tables together. In Figure 10, just as with the Figure 9 example, you join the original raw data table to the new dimension tables. This enables you to get the ID values for the foreign key columns. The screenshot from the Query Designer (Figure 12) shows the joins visually.

```
 -- Create Sale Table
CREATE TABLE Sale
(
    SaleID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    DateSold Date NOT NULL,
    ProductID VARCHAR(8) NOT NULL FOREIGN KEY REFERENCES Product(ProductID),
    CustID INT NOT NULL FOREIGN KEY REFERENCES Customer(CustID),
    CountryID INT NOT NULL FOREIGN KEY REFERENCES Country(CountryID),
    ChannelID INT NOT NULL FOREIGN KEY REFERENCES SalesChannel(ChannelID),
    UnitsSold INT NOT NULL,
    CreateTimestamp DATETIME,
    UpdateTimestamp DATETIME
)
```

Figure 10

With the Sale table created, you now need to insert the data using the ProductSales table as your source joined with the dimension tables to get the primary key values (Figure 11).

```
-- Insert Sale Data
INSERT INTO Sale
SELECT
       S.dateSold,
       S.productID,
       S.custID,
       C.CountryID,
       SC.ChannelID,
       S.unitsSold,
       CURRENT_TIMESTAMP AS CreateTimestamp,
       CURRENT_TIMESTAMP AS UpdateTimestamp
FROM
       ProductSales AS S
       INNER JOIN Country AS C ON S.Country = C.CountryName
       INNER JOIN SalesChannel AS SC ON S.SalesChannel = SC.ChannelName
```

Figure 11

You should have the following counts for your tables. Run a series of counts yourself to check you have the same.

Table	Count
Customer	**744**
Product	**9**
Country	**150**
Region	**8**
SalesChannel	**3**
Sale	**1303**

Figure 12

You can also view the final schema by going to Database Diagrams (Figure 13) underneath your SalesDW database node in SQL Server Management Studio.

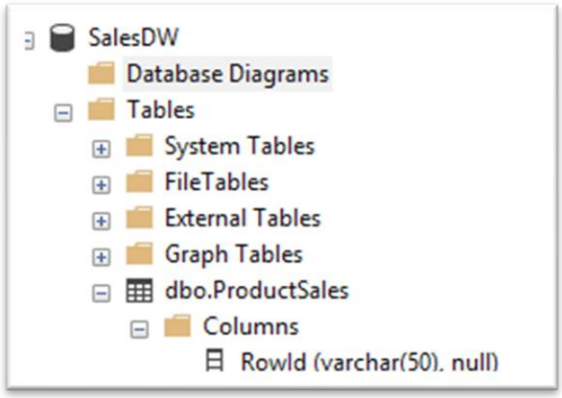

Figure 13

Right click on Database Diagrams > New Database Diagram.

You may get an error at this point stating that you do not have sufficient privileges. If so, enter the following SQL into a new query window:

```
alter authorization on database::salesdw to sa
```

With the privileges now set, try again.

- Right click on Database Diagrams > New Database Diagram

- Highlight the tables to view (do not include the ProductSales table) and click Add. You should see something similar to Figure 14:

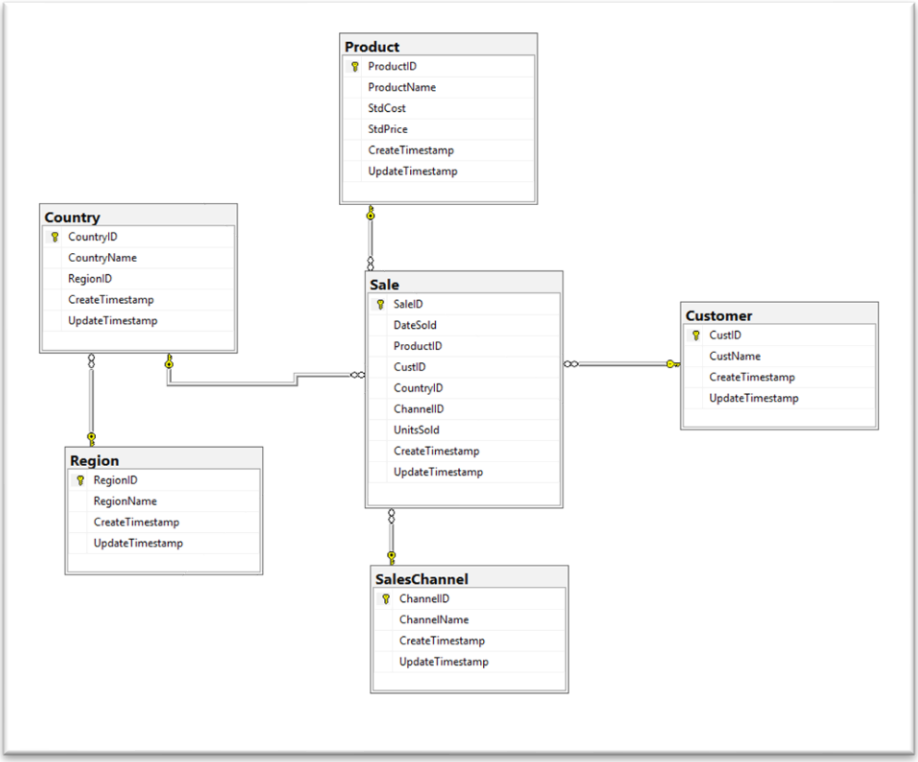

Figure 14

The region table linked to the country table rather than the Sale table means that this is a snowflake schema rather than a star schema.

Extension

1. Could you have used the RowId as a primary key?
2. Rename the tables to follow the naming convention for a data warehouse – don't forget all your SQL.

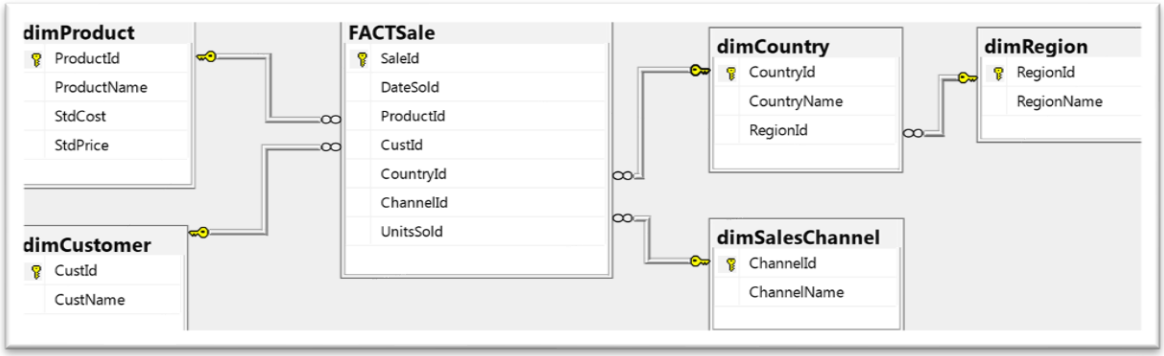

Figure 15

Useful SQL

You can use the Figure 16 SQL to obtain the maximum length of each data column from the imported data. Just bear in mind that you will need to do a sense check on the numbers that are returned from these queries. These queries will return the length of the dataset you have imported. Think about what the lengths need to be to cover all possible scenarios you might get in the future.

```
SELECT MAX(LEN(Country))
FROM  [SalesDW].[dbo].[ProductSales]

SELECT MAX(LEN(ProductName))
FROM  [SalesDW].[dbo].[ProductSales]

SELECT MAX(LEN(CustName))
FROM  [SalesDW].[dbo].[ProductSales]
```

Figure 16

Rename a table (Figure 17):

```
EXEC sp_rename 'Old table name', 'New table name'
```

Figure 17