# 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.

This exercise will involve repetition of some processes. In some cases, your tutor will show you how to do the first one and you will then apply this to others.

## Instructions

1. Create an empty database in SQL Server and name it SalesDW. We will import the data file ProductSales.csv into this database.

USE master

GO

DROP DATABASE IF EXISTS SalesDW

GO

CREATE DATABASE SalesDW

GO

USE SalesDW

GO

1. Import the ProductSales.csv file into the SalesDW database. Pay attention to column data types. Ensure you set the following columns accordingly:

*RowId – int (Primary key)*

*Country – varchar(50)*

*Region – varchar(50)*

*CustID – int*

*CustName – varchar(50)*

*ProductSold – varchar(50)*

*SalesChannel – varchar(50)*

*UnitsSold – int*

*DateSold – date*

*ProductID – varchar(10)*

*ProductName – varchar(50)*

*StdCost – decimal(8,2)*

*StdPrice – decimal(8,2)*

Once the operation is complete, if you don’t see the new table, select **SalesDW** and hit the refresh button.

1. 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

1. 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.
2. Create a SalesChannel table using SQL. The SQL code below 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) PRIMARY KEY,

ChannelName VARCHAR(10) NOT NULL,

CreateTimestamp DATETIME,

UpdateTimestamp DATETIME

)

1. Populate the table using SQL. The SQL code below 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 used 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]

1. Specify a CREATE TABLE statement for the Region table. 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.

-- create Region table

CREATE TABLE Region(

RegionId INT IDENTITY(1,1) PRIMARY KEY,

RegionName VARCHAR(50) NOT NULL,

CreateTimestamp DATETIME,

UpdateTimestamp DATETIME

)

1. Before you insert the Region data, 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 ProductSales, the staging table. A good method for this it to create a new column to hold the clean version of the Region column. This allows you to always have a copy of the original Region column should you need to make further amendments.
   * Create a new column RegionClean

ALTER TABLE ProductSales

ADD RegionClean varchar(50)

* + Use a SQL Update statement to populate the column using the amendment shown in the table below. The other Region values can stay as they are

|  |  |
| --- | --- |
| 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 2

UPDATE ProductSales

SET RegionClean = ( CASE

WHEN Region = 'Central America and the C'

THEN 'Central America and the Caribbean'

WHEN Region = 'Middle East and North Afr'

THEN 'Middle East and North Africa'

ELSE Region

END )

* + Review the new column to ensure the update has worked successfully
  + Populate the Region table

-- Insert Region Data

INSERT INTO Region

SELECT DISTINCT [RegionClean],

CURRENT\_TIMESTAMP AS CreateTimestamp,

CURRENT\_TIMESTAMP AS UpdateTimestamp

FROM SalesDW.[dbo].[ProductSales]

1. Some tables (Customer, Product) have suitable primary keys already. This is how to create and populate the Customer table.

-- Create Customer Table

CREATE TABLE Customer(

CustID INT PRIMARY KEY,

CustName VARCHAR(50) NOT NULL,

CreateTimestamp DATETIME,

UpdateTimestamp DATETIME

)

--Insert Customer Data

INSERT INTO Customer

SELECT DISTINCT [CustID], [CustName],

CURRENT\_TIMESTAMP AS CreateTimestamp,

CURRENT\_TIMESTAMP AS UpdateTimestamp

FROM SalesDW.[dbo].[ProductSales]

1. And now create and populate the Product table.

-- Create Product Table

CREATE TABLE Product(

ProductId VARCHAR(10) PRIMARY KEY,

ProductName VARCHAR(50) NOT NULL,

stdCost DECIMAL(5,2) NOT NULL,

stdPrice DECIMAL(5,2) NOT NULL,

CreateTimestamp DATETIME,

UpdateTimestamp DATETIME

)

--Insert Product Data

INSERT INTO Product

SELECT DISTINCT [productID], [productName], [stdCost], [stdPrice],

CURRENT\_TIMESTAMP AS CreateTimestamp,

CURRENT\_TIMESTAMP AS UpdateTimestamp

FROM SalesDW.[dbo].[ProductSales]

1. 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. It is necessary to look up the ID of the region that the country is in when populating the table (so the region table must be populated FIRST). Create the Country table with an extra column for the foreign key.

-- Create Country Table

CREATE TABLE Country

(

CountryID INT IDENTITY(1,1) PRIMARY KEY,

CountryName VARCHAR(50) NOT NULL,

RegionID INT FOREIGN KEY REFERENCES Region(RegionID),

CreateTimestamp DATETIME,

UpdateTimestamp DATETIME

)

1. We will use a join to correctly populate the table.

-- 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

1. The final data population is of the Sale table – the fact table which links the other tables together. We will join the original raw data table to the new dimension tables in order to get the ID values for the foreign key columns.

-- Create Sale Table

CREATE TABLE Sale

(

SaleID INT IDENTITY(1,1) PRIMARY KEY,

DateSold Date NOT NULL,

ProductID VARCHAR(10) 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

)

1. With the Sale table created, it is now necessary to insert the data using the ProductSales table as a source, joined with the dimension tables to get the primary key values.

-- 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

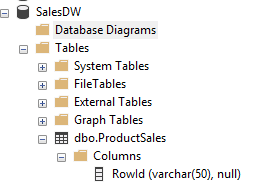
INNER JOIN SalesChannel AS SC ON S.SalesChannel = SC.ChannelName

* You should have the following counts for your tables.

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

Figure 3

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



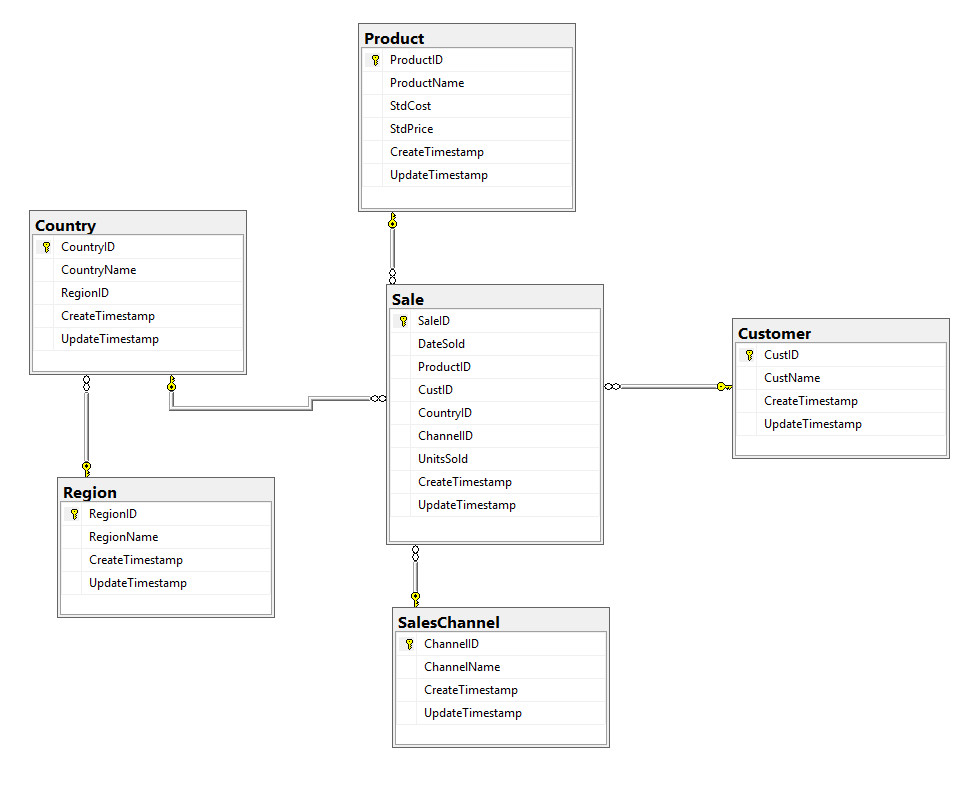
*Figure 4*

* 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 5:



*Figure 5*

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.

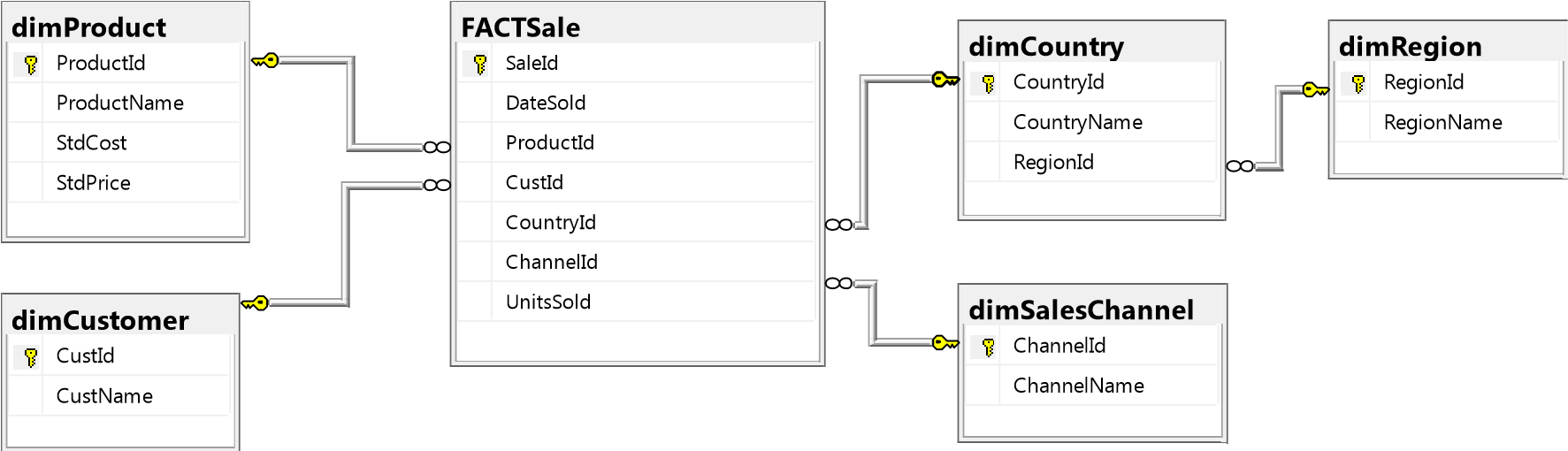


Figure 6

## Useful SQL

You can use the following SQL code 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]

Rename a table:

EXEC sp\_rename 'Old table name', ‘New table name'