INFO 5707

Final Project

**Full Custom Database**

**Alex Benitez**

# Image result for antique car logo"

# 

# 

# 

**Table of Contents:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Topic** | **Page No** |
| 1. | Database overview including objectives and scope | 3 |
| 2. | User Requirements | 4 |
| 3. | Business Rules | 4 |
| 4. | Entity-Relationship Diagram (ERD) | 5 |
| 5. | Data Dictionary | 6 |
| 6. | Database creation and Insertion using MYSQL statements with screenshots | 7-14 |
| 7. | MYSQL query statements with screenshots | 15-17 |

# 

# 

# Database Overview:

The database’s main objective is to help organize the data that is being generated from the sales of the company called “Full Custom”. This company sells not just antique and custom cars but also motorcycles, planes, ships, trains, trucks, and buses. This variety of vehicles are shipped globally and requires that the customers see the status of their shipment, information such as shipping location, product code, order date, quantity ordered. We can organize the data generated by the sales, shipping, and handling departments to properly display this information to the customer and keep the data for business analysis when needed.

The data consist of 2824 instances or transactions with over 28 different attributes that help keep track of customer's information and the orders they place. The 28 attributes are as follows:

* “Transaction\_ID” - used to keep track of the individual transactions that are generated by the customer when ordering. These are individually generated for each item that is created or data point.
* “OrderNumber” - used to keep track of the entire order placed by the customer these can contain multiple transactions and order lines.
* “QuantityOrdered” – used to keep track of the entire shipment that is placed when compiling the entire order. This can also be seen by the customer when checking on the status of the shipment.
* “PriceEach” – represents the price at which each item was purchased and is included in the information of each transaction so the customer can see the price paid.
* “OrderLineNumber” – used to keep track of how many items were in each order.
* “Sales” – this is the total sale of the “OrderLineNumber” that sums the “PriceEach” column for each order.
* “OrderDate” – used to keep track of when the order was received.
* “StatusCode” – used to keep track of the status of the order for customer and sales department use.
* “Qtr\_ID” – used to keep track of the quarter of which the order was made.
* “Month\_ID” – used to keep track of the month that the order was placed in.
* “Year\_ID” – used to keep track of the year the order was placed in.
* “ProductLine” - used to describe the item purchased by specifying what type of vehicle was purchased. This may be one of the following motorcycles, classic cars, vintage cars, planes, ships, trains, trucks, and buses.
* “MSRP” – used as a suggested sale price from the manufacturer also called the sticker price.
* “ProduceCode” – a specific code that is given to a product
* “CustomerName” – is the customer's business name affiliated with the order.
* “Phone” – phone number of the customers' contact
* “AddressLine1” – is the first line of the customer's address
* “AddressLine2” – is the second line of the customer’s address.
* “City” – is the customer’s shipping city.
* “StateCode” - is the customer’s state represented by the abbreviation.
* “PostalCode” – is the customer’s zip code used for shipping.
* “Country” – The name of the country the customers shipping address is located.
* “Territory” – The territory name of the customer’s residence.
* “Contact\_FName” – is the contact’s first name for the customer.
* “Contact\_LName” – is the contact’s last name for the customer.
* “Deals” – a category specified by the user to group order lines into small, medium or large deal sizes.
* “Customer\_ID” – a unique id that identifies each customer.
* “Shipping\_ID” – a unique id that identifies the order lines, shipping group.

URL: <https://www.kaggle.com/kyanyoga/sample-sales-data>

# User Requirements:

Correctly splitting up the data into the appropriate tables will allow for easier navigation of the data. Customers will be able to see their order details appropriately and see the status of their shipment. Shipping can see addresses and contact information accordingly to fill orders.

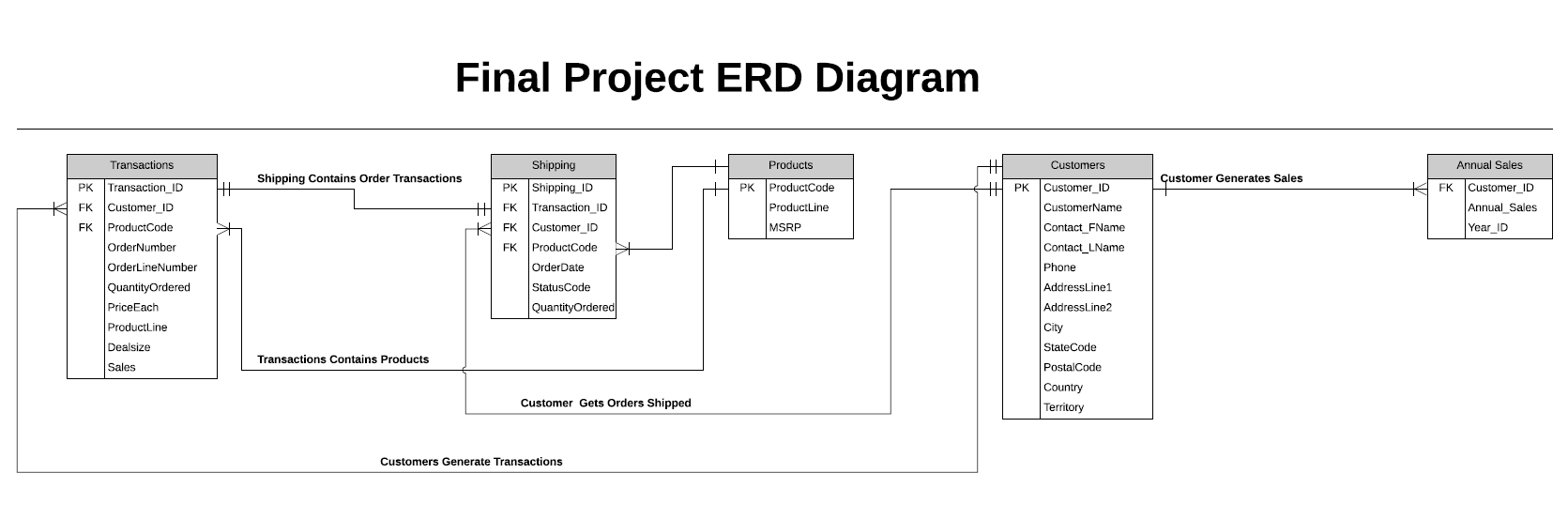
Sales can keep track of units sold, price, customer preference.

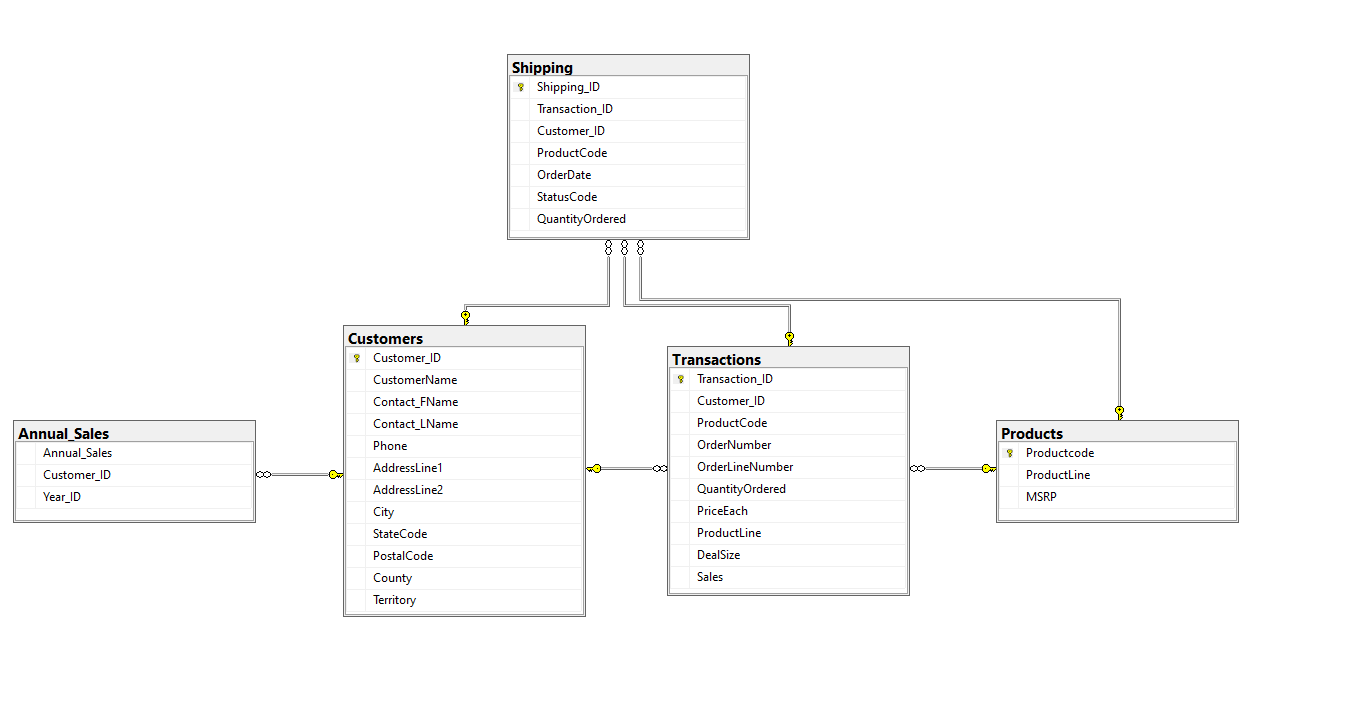
# Business Rules:

1. Each transaction will have one and only one shipping ID
2. Each shipping will have one and only one transaction ID
3. Each product will have multiple shipping ID’s
4. Each shipping will have one product
5. Each transaction will have one product
6. Each product will have one or many transactions
7. Each shipping will have one and only one customer
8. Each customer will have one or many shipping
9. Each customer will have one or many sales
10. Each sale will have one and only customer

# 

# Entity-Relationship Diagram:





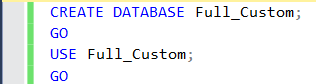
# Data Dictionary:

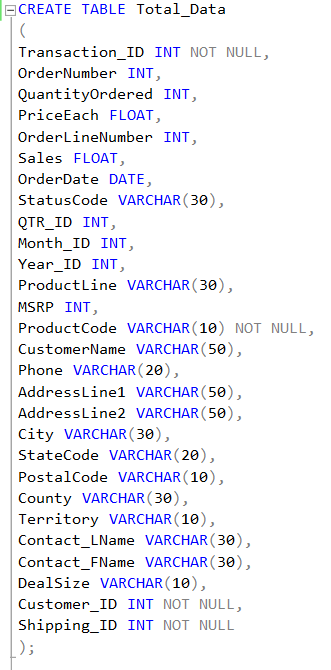
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Table | Attributes | Datatype | Format  (Eg) | Range | Constraint | Reference table |
| Transactions | Transaction\_ID  Customer\_ID  ProductCode  OrderNumber  OrderLineNumber  QuantityOrdered  price each  ProductLine  Dealsize  Sales | INT(50)  VARCHAR(20)  VARCHAR(20)  INT(50)  INT(50)  INT(50)  INT(50)  VARCHAR(20)  INT(50)  VARCHAR(20) | 99999  Abc123  Abc123  99999  99999  99999  99999  Abc123  99999  Abc123 | 00000-99999  00000-99999  00000-99999  00000-99999  00000-99999  00000-99999 | PK  FK  FK | Shipping  Product |
| Shipping | Shipping\_ID  Transaction\_ID  Customer\_ID  ProductCode  OrderDate  StatusCode  QuantityOrdered | VARCHAR(20) VARCHAR(20) VARCHAR(20) VARCHAR(20)  DATETIME  VARCHAR(50)  INT(50) | Abc123  Abc123  Abc123  Abc123  YYYY-MM-DD hh:mm: ss  Abc123  99999 | 00000-99999 | PK  FK  FK  FK | Transactions  Product  Customer |
| Product | ProductCode  ProductLine  MSRP | VARCHAR(20)  VARCHAR(20)  INT(10) | Abc123  Abc123  99999 | 00000-99999 | PK | Shipping  Transactions |
| Customer | Customer\_ID  CustomerName  Contact\_FName  Contact\_LName  Phone  AddressLine1  AddressLine2  City  StateCode  PostalCode  Country  Territory | VARCHAR(20)  VARCHAR(20)  VARCHAR(20)  VARCHAR(20)  INT(50)  VARCHAR(20)  VARCHAR(20)  VARCHAR(20)  VARCHAR(20)  INT(50)  VARCHAR(20)  VARCHAR(20) | Abc123  Abc123  Abc123  Abc123  99999  Abc123  Abc123  Abc123  Abc123  99999  Abc123  Abc123 | 00000-99999  00000-99999 | PK | Transactions  Annual Status |
| Annual Sales | Customer\_ID  TotalSale  Year\_ID | VARCHAR(20)  INT(10)  INT(10) | Abc123  99999  99999 | 00000-99999  00000-99999 | FK | Customer |

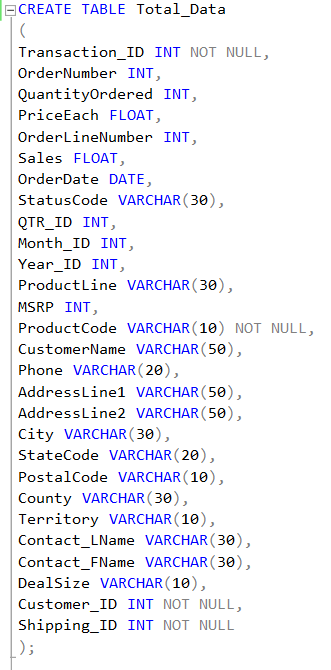
# 

# Database Creation:

The database was created using SQL Microsoft Server as the SQL program. This allows the user to easily create the database by writing scripts or importing data directly into the database via the server tools. In this project, we were required to demonstrate the SQL knowledge we have gained over the semester so we decided to write scripts for the project. The entire database can be created by running 2 scripts that were created by us. This could have been simplified to one script but for our purposes, we wanted to keep the separated for easier editing. The following figures will be screenshots of the 2 scripts that were created.

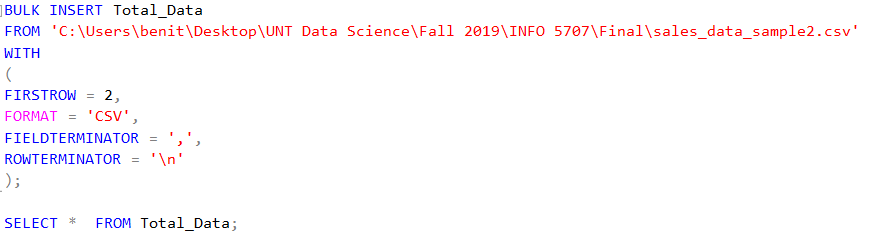


This screenshot is a small snippet of the code that is shown only to explain that these first four lines are used to create the database that is worked in called “Full\_Custom”. The “GO” command is used to further execute the statements in a consecutive manner when running the scripts. Otherwise, these statements will not run properly unless they are executed one at a time.

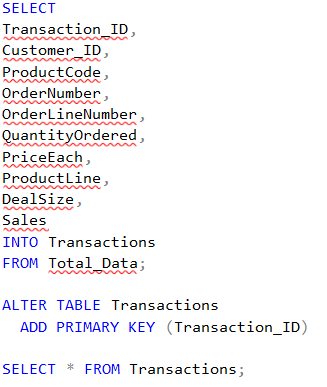


# 

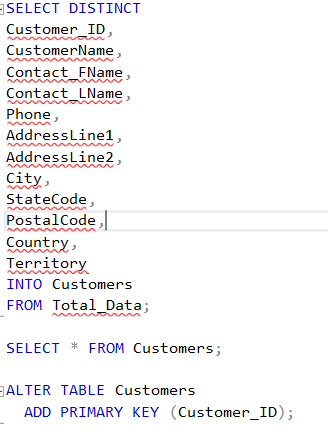
In the figures above we can see the create table statement that can be executed to create the table shell for our data. This must be created corresponding to the data in our excel file that we gathered from kaggle. The data columns must correspond to the names given for each data attribute so we do not have the wrong data and data type. If the data is not lined up correctly then the SQL server will give an error.



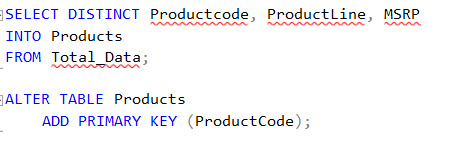
In the figure above we can see the bulk insert statement used to import the data from the local computer. We must specify the path, first row in the file since we are not using the names given, format as CSV, field terminator which indicated the attribute separator and the row terminator which indicates each data instance. Once this code has been executed in the script we have successfully imported our data file into a database and one table. This one table contains all of our data which is one way we chose to create the database.



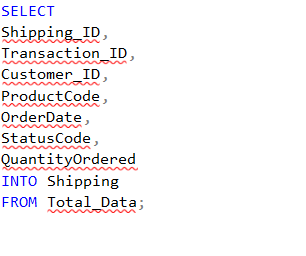
The figure to the left is the “Transactions” table comprised of the Transaction\_ID, Customer\_ID, ProductCode, OrderLineNumber, QuanitityOrdered, PriceEach, DealSize, and Sales attributes form the data. We are able to use the “SELECT INTO” query to insert the data into a new table from our original table “Total\_Data”. We then alter the table created by adding a primary key constraint to the table. We will also add foreign key constraints to the table as well later once the other tables have been made.



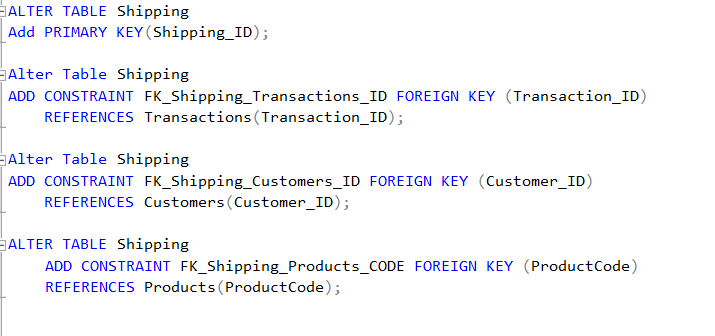
Here we use the select into statement again but with a distinct statement as well that allows us to avoid duplicates when creating this customer’s table. We can see the customer’s table is made up of the customer’s id, customer’s name, contact first name, contact last name, phone, address line 1, address line 2, city, state code, postal code, country and territory from the total data table. We then alter the table and add the constraint of a primary key to the table. This table is used to keep track of the customer's information and contact information. We can use the customer id to create a relationship with other tables.

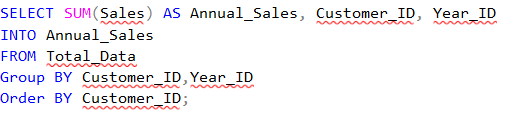


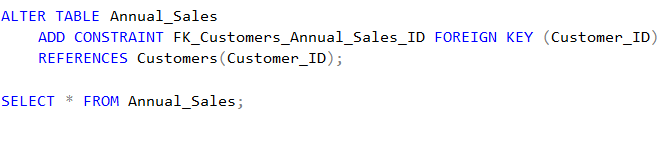
Now we create the product table made up of only three attributes the product code, product line and MSRP from the total data table. We use the distinct statement when selecting into the table because we do not want duplicates in this table as it is a reference table when looking a the products. We then alter the table and add the primary key constraint to the product code.

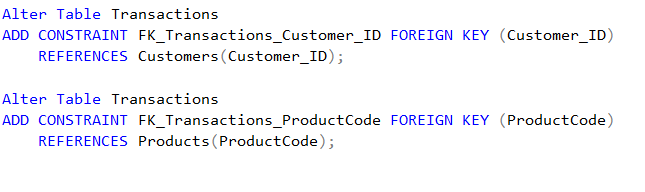


Here we are creating our shipping table that is used to properly track the status of shipping. The shipping table is also used to connect the transaction, customer and product tables for joining tables. The customer can request information about the shipping status, see the order date, and quantity ordered to make sure their order has been filled properly. Once the table is created we can then alter the table and add constraints like the primary key, and foreign keys from the transaction, products and customer tables.



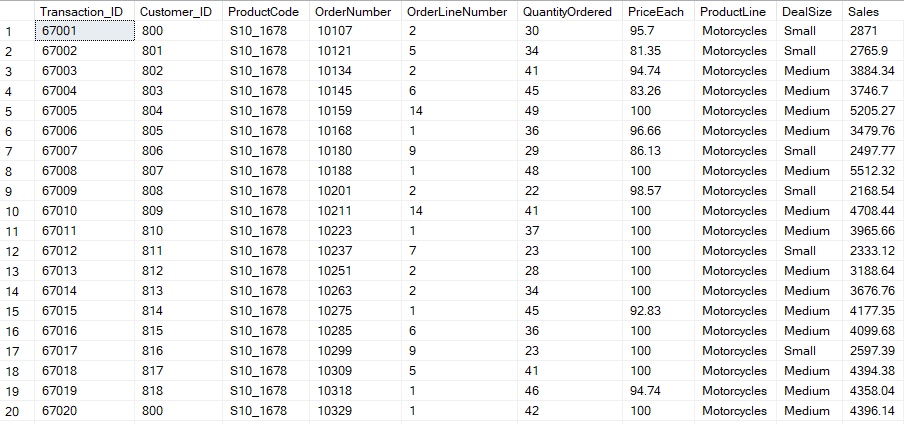


In the figure above we create the annual sales table that contains the total sales for the year by customer. This gives the sales department a quick glance at which customers generate more revenue and could give insight into how the company should make business decisions. Below we see the alterations that are made to the annual sales table. We add the foreign key constraint for the customer id key and reference the customer table. 

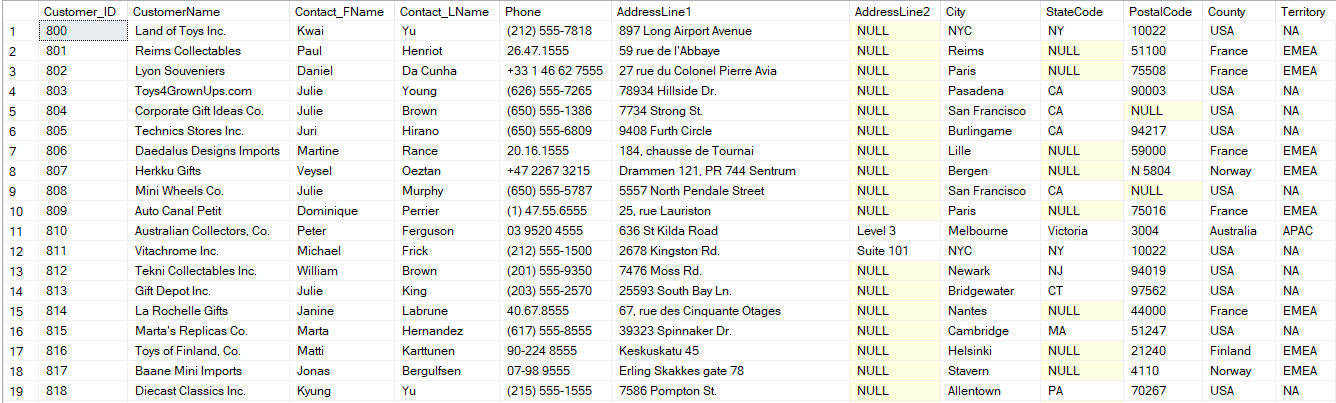
Lastly, we make the alterations to the transactions table that we could not reference before because the necessary tables had not been created yet. However, now that all the tables are created we can reference the correct tables for the added foreign keys. We see below we add the foreign keys for customer id from the customer's table and product code from the products table. We can use these keys as well when trying to complete complex queries and joining tables.

# Data Snapshots:

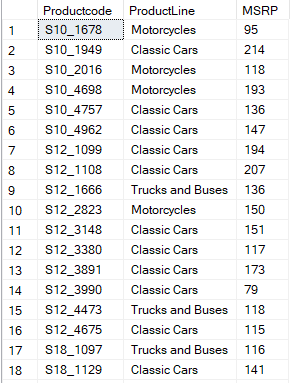
# Transaction Table: First 20 lines



# Customers Table: First 19 lines



# Products Table: First 18 Lines



# Shipping Table: First 20 lines

# 

# 

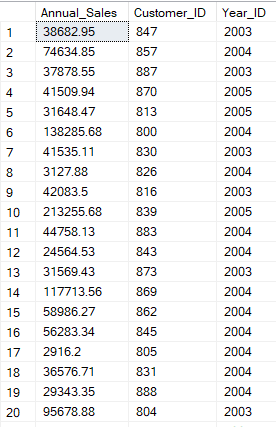
# 

# 

# 

# 

# Annual Sales Table: First 20 lines



# SQL Queries:

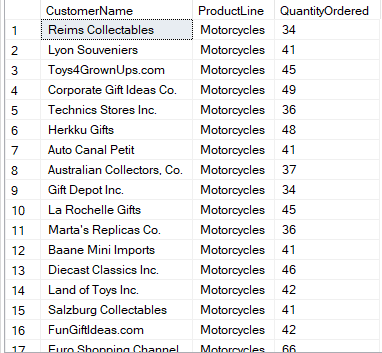
1.) What are the customer(names) who have ordered 30 or more motorcycles?

SELECT c.CustomerName, ProductLine, QuantityOrdered

FROM Transactions t

JOIN Customers c ON c.Customer\_ID = t.Customer\_ID

WHERE QuantityOrdered > 30;



2.)What are the customers who ordered a train in November of 2004?

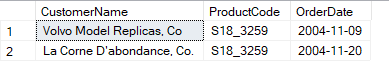
SELECT c.CustomerName, ProductCode, OrderDate

FROM Shipping s JOIN Customers c ON c.Customer\_ID = s.Customer\_ID

WHERE ProductCode = 'S18\_3259'

AND MONTH(OrderDate) = 11

AND YEAR(OrderDate) = 2004;



3.) What is the total number of planes sold in the year 2005

SELECT ROUND(SUM(t.QuantityOrdered), 0)

FROM Transactions t JOIN Shipping s ON s.Transaction\_ID = t.transaction\_ID

JOIN Products p ON p.ProductCode = s.ProductCode

WHERE YEAR(OrderDate) = '2005'

AND p.ProductLine = 'Planes';



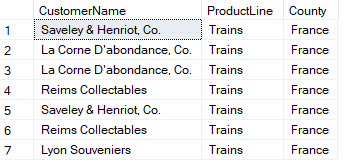
4.) What are the customer's names who bought trains and motorcycles and live in France?

SELECT c.CustomerName, p.ProductLine, Country FROM Customers c JOIN Shipping s on s.Customer\_ID = c.Customer\_ID

JOIN Products p ON p.ProductCode = s.ProductCode

WHERE p.ProductLine = 'Trains'

AND c.Country = 'France';



5.) What is the total of sales generated from each company originating in France?

SELECT SUM(Sales) AS 'Total Customer Sales', c.CustomerName FROM Transactions t JOIN Customers c ON c.Customer\_ID = t.Customer\_ID

WHERE c.Country = 'France'

Group BY c.CustomerName;

