**Unit 8 SQL JOINs**

Corey Crooks

Purdue University Global

IT163 – Kathi Nicholson

April 21, 2021

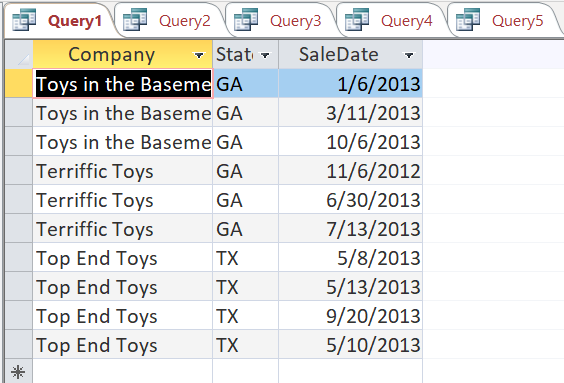
**Query Question Responses**

**2. Use a query to determine which company had sales in Texas and Georgia and when were these sales.**

SELECT Company, State, SaleDate

FROM tblCustomers INNER JOIN tblSales ON tblCustomers.CustomerID = tblSales.CustomerID

WHERE (State='tx') OR (State='GA');

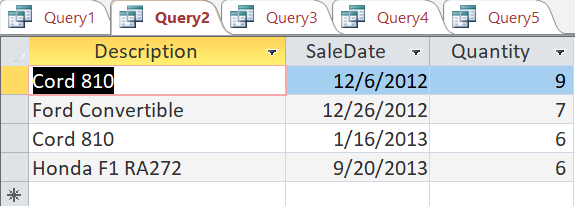


**3. Use a query to determine which items had more than five items sold and when were these sales.**

SELECT tblProducts.Description, tblSales.SaleDate, tblSalesLineItems.Quantity

FROM tblProducts INNER JOIN (tblSales INNER JOIN tblSalesLineItems ON tblSales.InvoiceNumber = tblSalesLineItems.InvoiceNumber) ON tblProducts.ProductID = tblSalesLineItems.ProductID

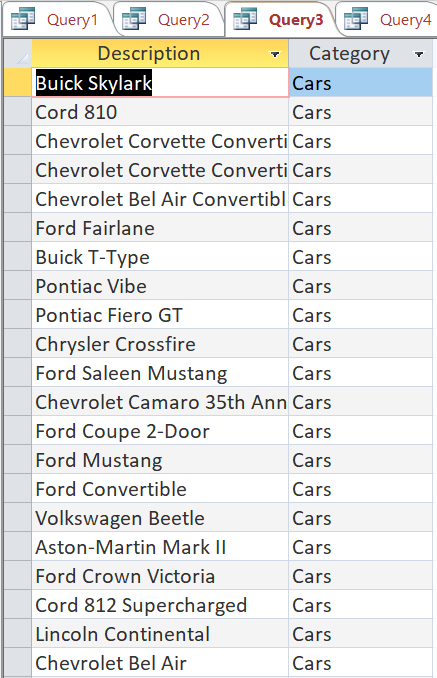
WHERE Quantity>5;



**4a. Select the description and category from the Products table and the category from the Categories table. Create an inner join using the category field that exists in each table. Save as Query3. Run the query to see the subset of data produced. Examine the results. Notice the total record count for the recordset in the results view.**

SELECT tblProducts.Description, tblCategories.Category

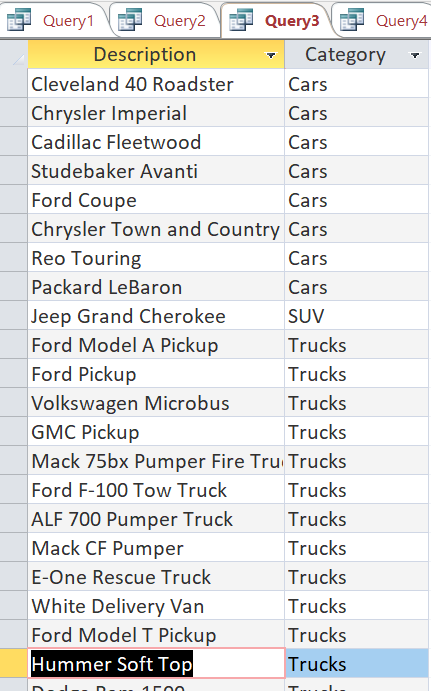
FROM tblProducts INNER JOIN tblCategories ON tblProducts.Category = tblCategories.Category;

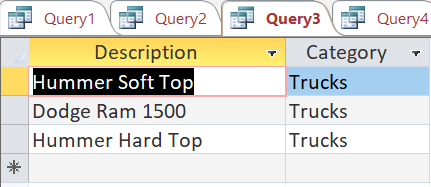








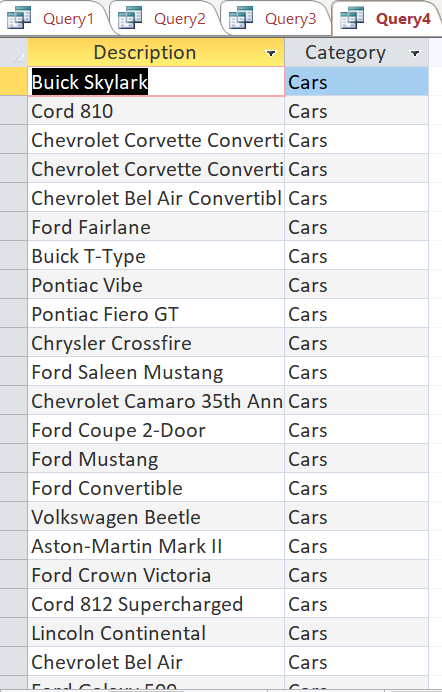


****

**b. Select the description and category from the Products table and the category from the Categories table. Create a left join using the category field that exists in each table. Save as Query4. Run the query to see the subset of data produced. Examine the results. Notice the total record count for the recordset in the results view.**

SELECT tblProducts.Description, tblCategories.Category

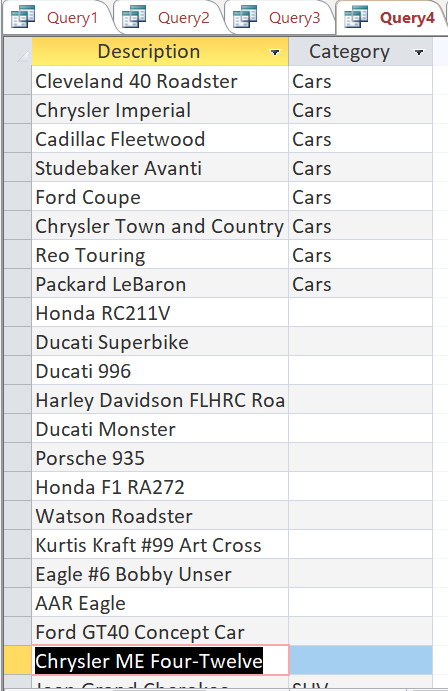
FROM tblProducts LEFT JOIN tblCategories ON tblProducts.Category = tblCategories.Category;

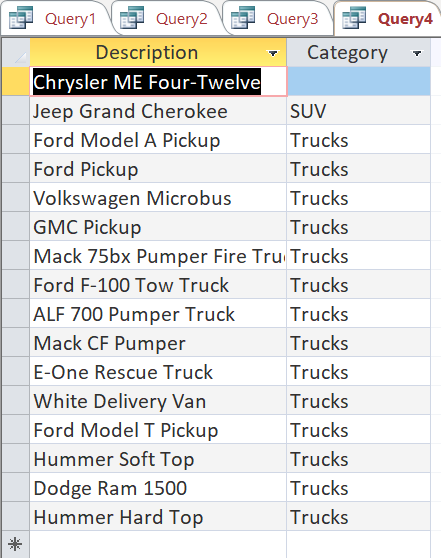








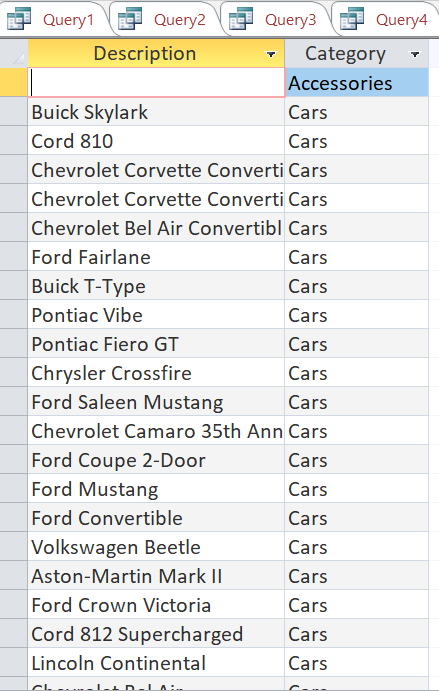


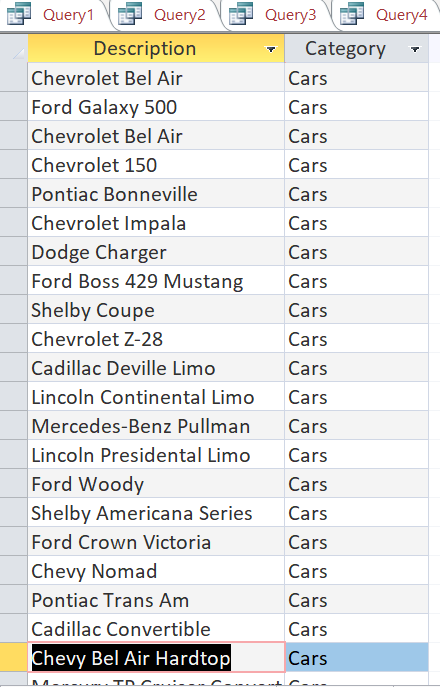


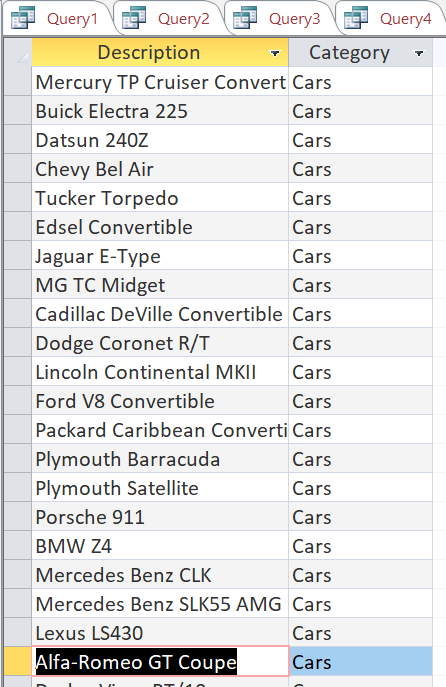
**c. Select the description and category from the Products table and the category from the Categories table. Create a right join using the category field that exists in each table. Save as Query5. Run the query to see the subset of data produced. Examine the results. Notice the total record count for the recordset in the results view.**

SELECT tblProducts.Description, tblCategories.Category

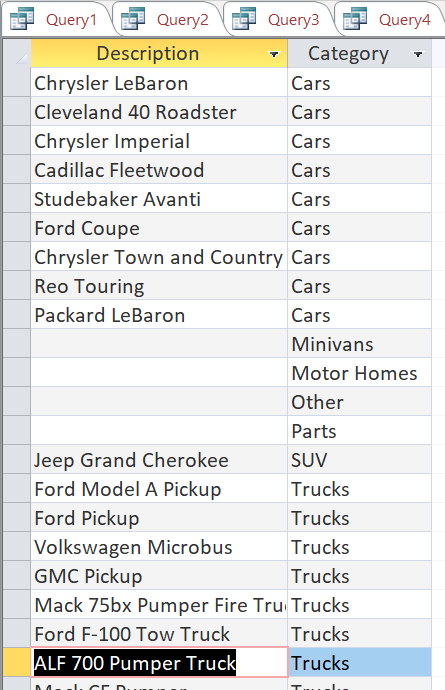
FROM tblProducts RIGHT JOIN tblCategories ON tblProducts.Category = tblCategories.Category;

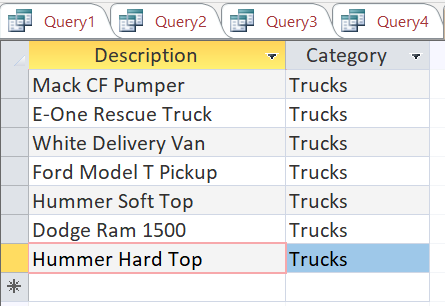












**Join Reflections**

Question 4 of this week’s assignment required running INNER JOIN, LEFT JOIN, and RIGHT JOIN on the same dataset to further understand the differences in each command. INNER JOIN produced a set that I wasn’t quite expecting at first. The dataset retrieved by the query did not include all vehicles with categories found in the Products table. Query3 only included the categories Cars, Trucks, and SUV. But when looking at the Products table, it seems that categories such as Motorcycles, and Special Interest were completely omitted from the dataset. They were non-null short-text category entries that seemed perfectly valid, so maybe something was wrong with the SQL I wrote. Upon further inspection, I noticed that tblCategories actually didn’t contain any Motorcycle or Special Interest category. Furthermore, all categories listed in the SQL results were also listed as categories in tblCategories. This was quite a realization; in prior Queries, I had been using join to match columns so it never really occurred to me that the JOIN ON clause validates per field too. It makes sense looking back on the assignment’s previous queries, and the ON clause in general. If you didn’t need the two columns to match for a relationship between the two tables, then there is something wrong with your data as a whole. A foreign key found in one table cannot have extraneous data compared to a primary key in another table, or else it will be a completely different attribute and thus ruin the relationship of the table. Query4 elaborated on the concepts I had learned in the last query by containing the same results in the report, just adding records to the report with null values. At first, this was very confusing to see. This list now only included SUV, Trucks, Cars, and a blank field in the Categories section. At first when I saw this, I thought something was obviously wrong. After all, the dataset that was retrieved included the same list from the previous query, but also included a chunk of Null fields for the category type. Why would it include Null fields, which were not in tblCategories, but not the other values? My confusion was multiplied tenfold when I took a look in tblProducts and saw that no such null category existed. How could this happen? It seemed that the query was rejecting certain category types in favour of others. Looking more closely, my confusion was put to rest when investgating the Category types of the vehicles in tblProducts that had reported blank values for the category in the query. Each vehicle that had a category that isn’t listed in tblCategories was reporting blank in the query results. This makes much more sense when combined with the information that LEFT JOIN is supposed to report all information from the left table without bias (W3Schools, 2021). The query was reporting all information from the appropriate table, but leaving the information that wasn’t consistent with tblCategories blank. Query5 provided much less confusion on the subject, as it did exactly as I thought it would after working through my confusions with the previous query. It essentially does the same effect of LEFT JOIN but mirrored in a sense. This being the case, I was surprised to find out the switching the tables in the ON clause is exactly the same as switching to another table. For instance, tblCategories RIGHT JOIN tblProducts is the same as tblProducts LEFT JOIN tblCategories, which begs the question, why do we need both joins? If one is the same as a reworded other, then why not just have one? My best guess would be that having both makes it simpler to visualize, and would take one more step out of the process involved in getting ideas transferred to code. It seems many developers have standardized these processes, favoring the more important table first, or simply putting the more generic table first (Computer Science, 2019). In these cases, you would need to utilize both JOINs, as you couldn’t simply rewrite the syntax around.

# **References**

Computer Science. (2019, June 10). SQL Inner Joins and Outer Joins [Video]. Youtube. https://www.youtube.com/watch?v=7yvB-tTHRfQ

W3Schools. (2021, April 19). *SQL Joins*. Retrieved from W3Schools.com: https://www.w3schools.com/sql/sql\_join.asp