**Name: Cameron Justice**

**CITC 1303 – SQL Lab 3 (35 points)**

1. **Download this instruction file to your class disk. Please copy and paste screen shots of all SQL Statements that you use and output that you receive into this file under the corresponding instruction as proof that you’ve completed that step. Also respond to each question under that question in this document.**
2. Type your name at the top of this document.
3. Launch **SQL Developer.**
4. Change your SQL Developer environment using the following SQL Plus commands:  
     
   set echo on

set define off

set escape **\**

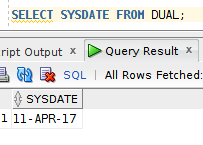
Note: The *set* commands are Oracle proprietary and NOT part of the SQL standard language. They are Oracle command line commands, otherwise known as SQL Plus commands. If you complete the lab in multiple sessions, you need to rerun these commands at the beginning of each SQL Developer session.

As you step through the exercises for this part of the lab, keep in mind that if any queries require the use of a specific date format, then you might need to use an alter session command prior to running the select statement: **alter session set nls\_date\_format=’match-string’** to match the date format. For example, to match 11/13/2003 you would type the command   
 ***alter session set nls\_date\_format=’MM/DD/YYYY’***.   
The default date that is recognized depends on how the DBA has set up the system defaults.

1. Use the command *select sysdate from dual* to display todays’ date*.*

*Sysdate* is a system function that displays the current system’s date. *Dual* is a 1-column, 1-row generic table used to perform operations that don’t require one of the user’s tables.

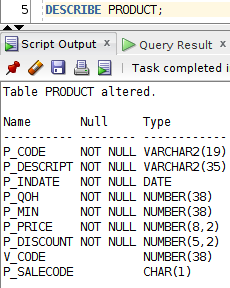
**Paste proof that it worked here:**



1. Read section 7.5.3 (p. 272) to gain an understanding of the ALTER TABLE command as it is used to modify a column’s attribute and constraints. Add a column named P\_SALECODE to the PRODUCT table as specified in the following *ALTER* command.  
    *alter table PRODUCT add(P\_SALECODE char(1));*

This is a command that adds a new column named P\_SALECODE to the PRODUCT table. If the user tries to enter a character string of 2 characters or more the row will be rejected.

**What is the data type of this new column and what is the reason for the “rejection” described in the previous paragraph? Character; Variable size restriction of 1;**

1. Run a *describe* command to verify that you’ve completed this task successfully. **Paste proof that it worked here.**
2. Read sections 7.5.4 & 7.5.5.
3. Answer the following questions:
   1. What is the difference between the ALTER statement and the UPDATE statement? **ALTER modifies table structure. UPDATE modifies table data.**
   2. Which statement (ALTER or UPDATE) requires a COMMIT before the changes will become permanent? Why? **UPDATE; It changes the data.**
   3. When you did the ALTER statement in step 6 (above), were you changing the structure of the table or the actual data itself? **Technically both, as you add a NULL value into the every record. Logically though, you’re only changing the structure.**
4. Read section 7.5.6 (pp. 275-276) describing how to insert data from one table into another table. Create a new PART table and populate it with data by using a nested query within the create statement:

**create table PART as**

**select p\_code as part\_code,**

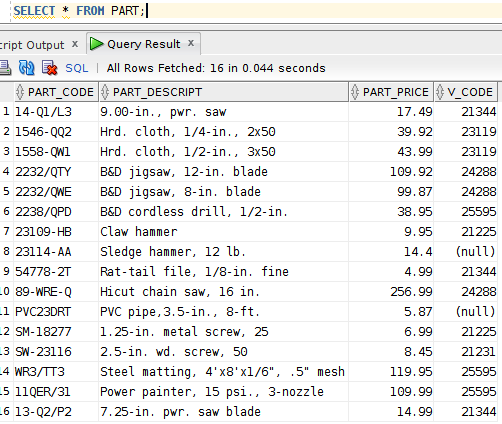
**p\_descript as part\_descript,**

**p\_price as part\_price,**

**v\_code**

**from product;**

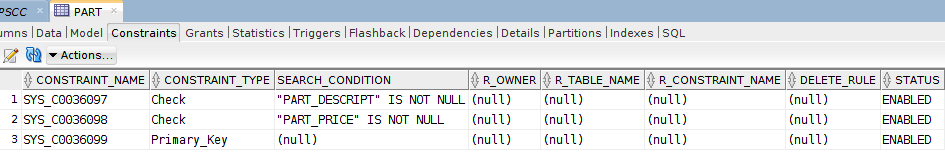
\*\*\* You will get a WARNING message. That’s OK, though. The command should still successfully create the new PART table. Just ignore the warning message.

The statement above copies PART data from the PRODUCT table to a new PART table. The fields and their data types from the PRODUCT table were copied to define the structure of PART. \*\*\* Use a *select* statement to verify that there is data in your new table.**Paste proof that it worked here:**

1. Add a primary key to your new PART table using the following command:

**alter table part add primary key(part\_code);**

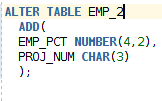
**\*\*\* In SQL Developer, open the PART table and click the [Constraints] tab. Examine the list of constraints to verify that a primary key has been added*;* paste a screen capture of the output here.**

****

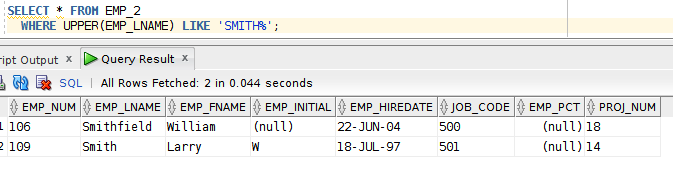
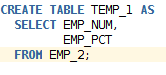
**Did you just change the structure (DDL) or data (DML) in the table?**

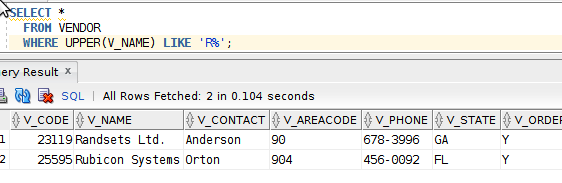
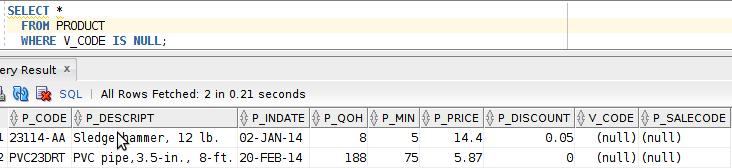
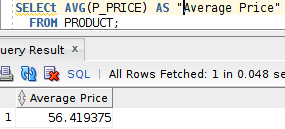
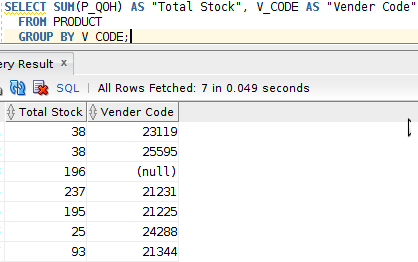
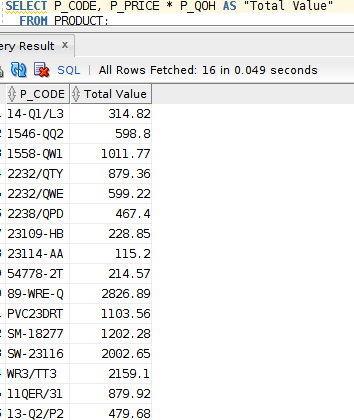
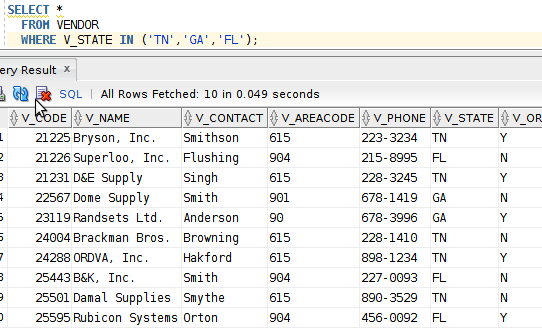
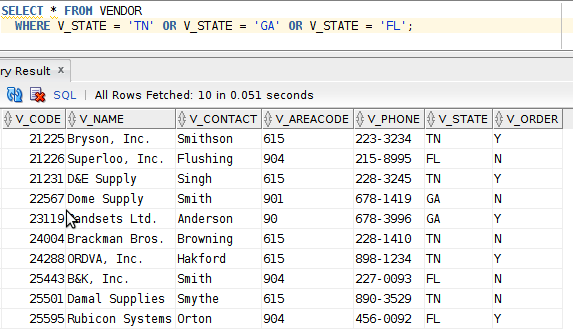
**Structure**

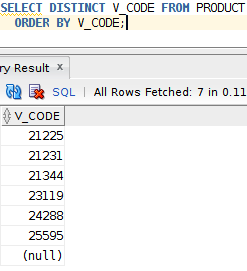
1. Use the examples in sections 7.5.\* as references to complete Problem#8 on page 295 at the end of the chapter. Paste proof that you successfully completed **all** of those tasks here.

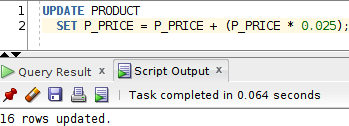
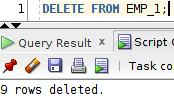


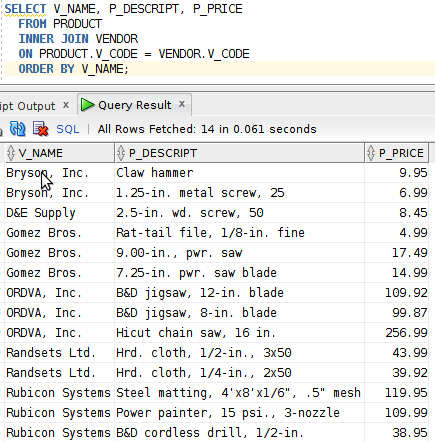
1. Complete Problems#9, 10, 11, 12, 13, 14 & 15 at the end of Chapter 7 (pp. 295-296). For all changes that you make, be sure to run a query to verify that your changes are correct. Paste proof that you completed all of these tasks here:



1. Read about the advanced queries starting at section 7.6 (pp. 277-287). Indicate your understanding of aggregate functions and related SELECT clauses by answering the following questions.
   1. What is the purpose of the GROUP BY clause in a SELECT statement? **Provide aggregated data by combining like data.**
   2. Does the GROUP BY clause sort the data in ascending order? (explain) **No, the GROUP BY tells the DBMS to grab the data and aggregate it together. It shows it as it gets it.**
   3. What is the purpose of the HAVING clause in a SELECT statement? **For use in place of WHERE when aggregate functions are required.**
   4. Explain the similarities and difference between a WHERE clause and HAVING clause. **HAVING checks the conditions AFTER AGGREGATES. WHERE checks the conditions BEFORE AGGREGATES.**
   5. Which aggregate function can we use to find a total for a numeric column? **SUM()**
   6. Which aggregate function can we use to find the largest value in a specific column? **MAX()**
   7. Which aggregate function can we use to determine how many rows match specific criteria? **COUNT()**
   8. Which aggregate function can we use to calculate the average of a group of numbers? **AVG()/AVERAGE()**
   9. In a GROUP BY clause, for example *GROUP BY xxxx*, should the ‘xxxx’ on which we’re grouping be a text field (such as *department or state*) or should it be a numeric field (such as *price* or *population*)? (explain) **It CAN be either, but logistically you should only ever group by text.**
2. Write SQL queries to display the following information. Be sure to copy the SQL statement and the resulting output under each task as proof that you got it working (or tried). The best reference for related examples is sections 7.4 & 7.6 of your textbook.
   1. All VENDORS whose vendor name begins with a ‘R’;
   2. All PRODUCTs with a NULL vendor code.
   3. The average price of the products in our PRODUCT table (one value should be displayed);
   4. The total quantity on hand (P\_QOH) grouped by vendor for products in our PRODUCT table (multiple values should be displayed);
   5. The value of our inventory (price \* quantity on hand) for each product in our PRODUCT table;
   6. All VENDORS that are in either Georgia, Florida or Tennessee;
   7. Repeat e. but use a different technique to get the exact same result set. **Do you mean F? here’s F with ORs instead of IN**
   8. A list of vendor codes from the PRODUCT table in vendor number order, but don’t list duplicated vendors more than once (i.e. a list of unique vendors).



1. Read the summary at the end of Chapter 7 (p. 292). Prepare questions to ask during lecture if there are concepts that still seem cloudy to you.
2. Write and run SQL statements to perform the following tasks. Place the statement and proof that it worked under **each question**.
   1. Write an update statement that can increase the price of all products by 2.5%.
   2. Write a SQL statement that removes all rows from your EMP\_1 table, but leaves the structure of the table intact;
   3. Rollback the change made in steps a & b.
   4. Write a SQL statement (but don’t run it) that can be used to remove a table named EMP\_2, including all data plus the entire EMP\_2 table structure. **DROP TABLE EMP\_2;**
   5. Write a SQL statement that joins the PRODUCT table to the VENDOR table. It should display only the vendor name, product description, and price for products that each vendor supplies. Be sure to add a clause that matches the PK to the FK so you won’t produce an unwanted Cartesian Product (rows x rows). Sort the output alphabetically by vendor name. All of this must be done using a single SQL statement.

**Save this file with responses included and upload one file to the SQL Lab 3 Dropbox in the online course.**