**BSQL Assignment 203**

**1. Exercise 1: Querying and Filtering Data**

**Questions to answer**:

Query 1: Write a query that retrieves the columns ProductID, Name, Color and ListPrice from the Production.Product table, with no filter. Your result set should look something like the following.

SELECT ProductID, Name, Color, ListPrice

FROM Production.Product

Query 2: Continue to work with the previous query and exclude those rows that are 0 for the column ListPrice. Your result set should look something like the following.

SELECT ProductID, Name, Color, ListPrice

FROM Production.Product

WHERE ListPrice > 0

Query 3: Use the same query, but this time you just want to see the rows that are NULL for the Color column. Your result set should look something like the following.

SELECT ProductID, Name, Color, ListPrice

FROM Production.Product

WHERE Color IS NULL

Query 4: Use the same query, but this time you just want to see the rows that are not NULL for the Color column. Your result set should look something like the following.

SELECT ProductID, Name, Color, ListPrice

FROM Production.Product

WHERE Color IS NOT NULL

Query 5: Now, combine two search arguments in the query you have been working with. You just want to see the rows that are not NULL for the column Color, and the column ListPricehas a value greater than zero. Your result set should look something like the following.

SELECT ProductID, Name, Color, ListPrice

FROM Production.Product

WHERE Color IS NOT NULL

AND ListPrice > 0

Query 6: Now we want a report that concatenates the columns Name and Color from the Production.Product table. Your result set should look something like the following. Make sure you exclude rows that are NULL for the column Color. Also notice the column name.

SELECT Name + ' : ' + Color AS 'Name And Color'

FROM Production.Product

WHERE Color IS NOT NULL

Query 7: Customize the previous query so the answer looks like the following.

SELECT Name + ' -- COLOR: ' + Color AS 'Name And Color'

FROM Production.Product

WHERE Color IS NOT NULL

Query 8: Now we would like to see the columns ProductID and Name from the Production.Product table filtered by ProductID from 400 to 500. Write a query that makes your result set look something like the following. Try to make your WHERE clause as simple and readable as possible

SELECT ProductID, Name

FROM Production.Product

WHERE ProductID BETWEEN 400 AND 500

Query 9: We would like to see the columns ProductID, Name and color from the Production.Product table restricted to the colors black and blue. Write a query that makes your result set look something like the following. Try to make your WHERE clause as simple and readable as possible.

SELECT ProductID, Name, Color

FROM Production.Product

WHERE Color IN ('Black', 'Blue')

Query 10: This exercise and the next three following will make use of wildcards in Transact-SQL. To begin with, we would like a report on products that begins with the letter S.

Write a query that retrieves the columns Name and ListPrice from the Production.Product table. Your result set should look something like the following. Order the result set by the Name column.

SELECT Name, ListPrice

FROM Production.Product

WHERE Name LIKE 's%'

ORDER BY Name

Query 11: Now we would like a report on products that begins with the letters S or A. Write a query that retrieves the columns Name and ListPrice from the Production.Product table. Your result set should look something like the following. Order the result set by the Name column.

SELECT Name, ListPrice

FROM Production.Product

WHERE Name LIKE '[sa]%'

ORDER BY Name

Query 12: Adjust your query so you retrieve rows that have a Name that begins with the letters SPO, but is then not followed by the letter K. After this zero or more letters can exists. Order the result set by the Name column.

SELECT Name, ListPrice

FROM Production.Product

WHERE Name LIKE 'spo[^k]%'

ORDER BY Name

Query 13: Write a query that retrieves unique colors from the table Production.Product. We do not want to see all the rows, just what colors that exist in the column Color. Your result set should look something like the following.

SELECT DISTINCT Color

FROM Production.Product

Query 14: Write a query that retrieves the unique combination of columns ProductSubcategoryID and Color from the Production.Product table. Format and sort so the result set accordingly to the following. We do not want any rows that are NULL.in any of the two columns in the result.

SELECT DISTINCT ProductSubcategoryID, Color

FROM Production.Product

WHERE ProductSubcategoryID IS NOT NULL

AND Color IS NOT NULL

ORDER BY ProductSubcategoryID ASC, Color DESC

Query 15: Something is “wrong” with the WHERE clause in the following query.

We do not want any Red or Black products from any SubCategory than those with the value of 1 in column ProductSubCategoryID, unless they cost between 1000 and 2000.

SELECT ProductSubCategoryID

, LEFT([Name],35) AS [Name]

, Color, ListPrice

FROM Production.Product

WHERE Color IN ('Red','Black')

AND ProductSubCategoryID = 1

OR ListPrice BETWEEN 1000 AND 2000

ORDER BY ProductID

Query 16: Use the Production.Product table to return product name, color and list price for each product. For the color column, where there is NULL, replace it with the string *Unknown*.

SELECT Name, ISNULL(Color, 'Unknown') AS Color, ListPrice

FROM Production.Product

#### ****2. Exercise 2: Grouping and Summarizing Data****

Query 1: How many products can you find in the Production.Product table? Your result set should look like the following.

SELECT COUNT(\*) FROM Production.Product

Query 2: Write a query that retrieves the number of products in the Production.Product table that are included in a subcategory. The rows that have NULL in column ProductSubcategoryID are considered to not be a part of any subcategory.

SELECT COUNT(ProductSubcategoryID) AS HasSubCategoryID

FROM Production.Product

Query 3: How many Products reside in each SubCategory?

The answer to this is retrievable if you write a query that use the COUNT aggregate function combined with a GROUP BY clause.

The column ProductSubcategoryID is a candidate for building groups of rows when querying the Production.Product table. Your result set should look something like the result below.

Notice the column alias for the second column.

SELECT ProductSubcategoryID, COUNT(Name) AS CountedProducts

FROM Production.Product

GROUP BY ProductSubcategoryID