# Lesson 2 Lab: SQL

Using the AdventureWorks2016CTP3 database, we are going to write a few basic and some advanced SQL queries. The goal is to revise some SQL that we already know and get us comfortable with the concept of tables, relationships and also with the syntax.

You might want to spend a few minutes reviewing the [Database Diagram](https://nbelyh.github.io/svgpublishdemo/AdvWorksOLTPSchemaVisio.html) of AdventureWorks database. If you are not familiar with the concept of Primary Key (PK) and Foreign Key (FK) then you can ignore the arrows in the diagram. Essentially, you should get familiar with the different table names that exist in the database. I would also strongly suggest to view the contents of a few tables using this syntax:

*-- Displays top 10 rows from a table..*

SELECT TOP 10 \* FROM [AdventureWorks2016CTP3.TABLE\_NAME]

OR

*-- Displays top 10 rows from a table..*

*-- This syntax works in Oracle DB not in SQL Server*

SELECT \* FROM [AdventureWorks2016CTP3.TABLE\_NAME] LIMIT 10

In a practical visualization or analytics project, you will have to go through these steps to get familiar with the database. And, unfortunately, you will be lucky if you can find a correct database diagram! You might actually have to make one yourself for the tables you are concerned with.

Now that you may have an idea about what AdventureWorks looks like, let's get to some of the exercises that you reinforce your understanding.

### Lab Exercises:

1. **Aggregation & Ordering:** We know that the count keyword can be used to get a count of the number of rows in a column. Such a function that summarizes data is called an Aggregation function. Some of the common aggregation functions in SQL are sum, count, count distinct, avg, min and max. Based on the flavor of SQL that you are using, you may also have some more functions like median and standard deviation. Such functions are often referred to as Analytical functions (typically by interviewers!!)

**Question 1:** From the Employee table, get a list of Title and the number of employees with that title. The titles are sorted in order of decreasing number of employees.  
*-- SQL GROUP BY clause with COUNT aggregate*  
SELECT JobTitle,  
COUNT(\*) as N\_Employees  
FROM AdventureWorks2016CTP3.HumanResources.Employee  
GROUP BY JobTitle  
ORDER BY N\_Employees DESC  
   
*--In MOST flavors of SQL, the above query WILL result in an error because we*  
*-- are using N\_Employees in the ORDER BY clause to order the output but*  
*-- N\_Employees is a derived field which doesn't yet anywhere. Hence referring*  
*-- to it will result in an error in nearly all flavors of SQL. The following*  
*-- syntax is guaranteed to work in any flavor of SQL.*

1. SELECT JobTitle,  
   COUNT(\*) as N\_Employees  
   FROM AdventureWorks2016CTP3.HumanResources.Employee  
   GROUP BY JobTitle  
   ORDER BY COUNT(\*) DESC  
   The Results look like this:  
   JobTitle                        N\_Employees  
   Production Technician - WC50    26  
   Production Technician - WC60    26  
   Production Technician - WC40    26  
   Production Technician - WC30    25  
   Production Technician - WC20    22  
   Production Technician - WC10    17  
   ....  
   ...  
   This exercise combines the usage of GROUP BY and ORDER BY keywords.
2. **Filtering:** Now let's put to practice our knowledge of where and having keywords. An easy way to remember the difference between where and having is:
   * WHERE is used to narrow the scope of the query i.e. to limit the data on which the SQL query performs operations
   * HAVING is used to narrow the results from an already executed query based on an aggregated column like count, mean, min, max etc.

**Question 2:** From the SalesOrderHeader table, count the number of unique orders where the OrderDate is in the year 2014.*-- When we use WHERE keyword, the data that the counts are being calculated on becomes smaller - i.e. the WHERE keyword narrows the scope of a query*  
SELECT COUNT(DISTINCT SalesOrderID) AS N\_ORDERS  
FROM AdventureWorks2016CTP3.Sales.SalesOrderHeader  
WHERE YEAR(OrderDate) = 2014

The Results look like this:

N\_ORDERS  
11761

**Question 3:** From the SalesOrderDetail table, count the number of unique orders that have at least 10 items in it (each item in an order is a row) and the SalesOrderID is greater than 40000.*-- When we use WHERE keyword, the data that the counts are being calculated on becomes smaller - i.e. the WHERE keyword narrows the scope of a query*  
*-- The HAVING keyword filters the final results (after all operations have executed)*  
SELECT SalesOrderID,  
COUNT(distinct SalesOrderDetailID) AS N\_ITEMS  
FROM AdventureWorks2016CTP3.Sales.SalesOrderDetail  
WHERE SalesOrderID > 40000  
GROUP BY SalesOrderID  
HAVING COUNT(distinct SalesOrderDetailID) >= 10  
*--You can try replacing the last line with HAVING N\_ITEMS >=10 to try if you can refer to the count column using its name*

The result should look like this:

SalesOrderID    N\_ITEMS  
     43659    12  
     43661    15  
     43662    22  
     43665    10  
     43668    29  
     43671    11  
     ...  
     ..

WHERE can be used with >, <, =, between, in, contains, like operators that compare two or more values hence forming a conditional statement that is either true or false. The HAVING keyword can also be used with most of the above mentioned operators.

1. **Inner Join:** Now we get into the joins. Inner join is the most frequently used type of JOIN. Let's try an inner join with this exercise. The Sales.SalesOrderHeader table contains high level information about the orders (eg. total sale, total quantity, ShiptoAddressID etc.)

**Question 4:** Our goal in this exercise is to generate a table summarizing the sales data for year 2014 by City, Number of Orders, Total Order Amount. The table that contains basic order info is Sales.SalesOrderHeader . The columns that we are interested in are SalesOrderID to count the number of orders, OrderDate to filter for the order year = 2014, ShiptoAddressID to join with the Person.Address table and get the City.  
Have a look at the SalesOrderHeader table using this code:  
SELECT TOP 10 \* FROM AdventureWorks2016CTP3.Sales.SalesOrderHeader  
Have a look at the Person.Address table using this code:  
SELECT TOP 10 \* FROM AdventureWorks2016CTP3.Person.Address  
Performing the inner join:  
SELECT b.City,  
count(distinct a.SalesOrderID) as Num\_of\_Orders,  
sum(a.TotalDue) as Total\_Order\_Amount  
FROM AdventureWorks2016CTP3.Sales.SalesOrderHeader a   
INNER JOIN AdventureWorks2016CTP3.Person.Address b  
ON a.ShipToAddressID = b.AddressID   
WHERE YEAR(a.OrderDate) = 2014  
GROUP BY b.City  
ORDER BY count(distinct a.SalesOrderID) DESC

The result should look like this:  
City    Num\_of\_Orders    Total\_Order\_Amount  
London          290       739850.344  
Paris           218       400188.4393  
Cliffside       167        67977.7145  
Beaverton       116        56870.9807  
N. Vancouver    113        43401.3535  
Burlingame      111        86853.9202  
...  
..

1. **Left Outer Join:** The second most common type of join is the LEFT OUTER JOIN where the contents of a second table are joined onto the contents of a base table. The resulting number of rows is greater than or equal to the number of rows in the base table. For rows where there is no matching key between the two tables, the base table columns are populated as is while the second table columns show up as NULL.  
   **Question 5:** Now lets try to repeat the same example as Question 4 using left outer join.  
   SELECT b.City,  
   count(distinct a.SalesOrderID) as Num\_of\_Orders,  
   sum(a.TotalDue) as Total\_Order\_Amount  
   FROM AdventureWorks2016CTP3.Sales.SalesOrderHeader a  
   LEFT OUTER JOIN AdventureWorks2016CTP3.Person.Address b   
   ON a.ShipToAddressID = b.AddressID  
   WHERE YEAR(a.OrderDate) = 2014

GROUP BY b.City  
ORDER BY count(distinct a.SalesOrderID) DESC  
  
The result should look like this:  
City    Num\_of\_Orders    Total\_Order\_Amount  
London          290       739850.344  
Paris           218       400188.4393  
Cliffside       167        67977.7145  
Beaverton       116        56870.9807  
N. Vancouver    113        43401.3535  
Burlingame      111        86853.9202  
...  
..  
This result is exactly same as that from the INNER JOIN. This is because all the rows from SalesOrderHeader were successfully matched to a single row each from the Address table. If a few records from SalesOrderHeader were not successfully matched with a record from the Address table, then the number of rows resulting from the INNER JOIN would have been less than the number of rows resulting from LEFT OUTER JOIN.