Module 4: Aggregation and GROUP BY

Aggregate Functions

Analytical queries of the sort you'll use for data analysis and reports will often involve calculating aggregate values within groups. An ***aggregate*** value is a summary value or descriptive statistic such as a sum, count, arithmetic average or mean, standard deviation, or variance.

For example, let's say we want to compute the sum of sales for cars. Almost certainly, we're going to want to use this calculation to compare some meaningful grouping of cars, such as by make, model, year, or all three.

Aggregations themselves are computed using an ***aggregate function***, such as COUNT, SUM, AVG, STDEV, and MIN. The following query simply computes the sum of SalesAmount across all rows of the result of a JOIN between dbo.Sales and dbo.Cars. Notice that the single column in the result set has no name:

```
SELECT
	SUM(s.SalesAmount)
FROM dbo.Sales s
JOIN dbo.Cars c
on s.CarID=c.CarID;
```

The query below demonstrates the COUNT function. A column alias is defined.

```
SELECT
	COUNT(SaleID) as [Total Number of Cars Sold]
from dbo.Sales;
```

In addition to SUM and COUNT, TSQL provides the AVG, CHECKSUM_AGG, GROUPING, GROUPING_ID, MAX, MIN, STDEV, STDEVP, VAR, and VARP aggregate functions. More information about the aggregate functions can be found at https://msdn.microsoft.com/en-us/library/ms173454.aspx.

4.2 GROUP BY

To create groups, we use **GROUP BY**. GROUP BY is used to organize records into groups so that an aggregate function can be applied to the records within a group. In general, the grouping column will be a column containing categorical values such as product names, product categories, student demographics, etc. The column to which aggregate functions are applied will usually be numeric columns such as sales amounts, test scores, etc.

So for example, let say we have a table named TableX, with one grouping column and one numeric column, and we wish to sum up the values of the numeric column within each group. The SELECT query would look something like this:

```
SELECT
        [Grouping Column]
        ,SUM([Numeric Column])
FROM TableX
GROUP BY [Grouping Column];
```

The rows in TableX would first be organizing into groups based on the Grouping Column value, then the SUM would be calculated across all rows within each group.

GROUP BY organizes rows into groups, so that an aggregate function can be applied to the rows within each group.

TableX

Grouping Column	Numeric Column
Group A	10
Group A	12
Group A	11
Group B	14
Group B	8
Group C	9
Group C	7
Group C	8
Group D	14
Group D	15

Result of GROUP BY

Grouping Column Value	*Aggregate (SUM)*
Group A	33
Group B	22
Group C	24
Group D	29

The following query will first organize the records by the value of the Make column of the Cars table, then compute the sum of the SalesAmount column from the Sales table for each Make group. We've assigned an alias to the column containing the group sum.

```
SELECT
        c.Make
        ,SUM(s.SalesAmount) as [Total Sales]
FROM dbo.Sales s
JOIN dbo.Cars c
ON s.CarID=c.CarID
GROUP BY c.Make;
```

Remember the order of operations. The GROUP BY is processed before the SELECT list, so once the GROUP BY is processed, we can no longer use anything in the SELECT list that refers to individual record values. For example, the following query will return an error:

```
SELECT
        c.Make
```

```
       ,c.Model
       ,SUM(s.SalesAmount) as [Total Sales]
FROM dbo.Sales s
JOIN dbo.Cars c
on s.CarID=c.CarID
GROUP BY c.Make;
```

Since the GROUP BY is performed immediately after the FROM, the Model value for each individual record within each MAKE value group are no longer available. However, if we include Model in the GROUP BY, our groups become unique Make/Model groups, so the following query will give the sum of sales for each unique Make and Model. We've included an ORDER BY so that each Model will be displayed within each Make in the result set.

```
SELECT
       c.Make
       ,c.Model
       ,SUM(s.SalesAmount) as [Total Sales]
FROM dbo.Sales s
JOIN dbo.Cars c
on s.CarID=c.CarID
GROUP BY c.Make, c.Model
ORDER BY c.Make, c.Model;
```

We can include multiple aggregate functions on the SELECT list. The following query will return the number of sales, mean sales amount, and sum of sales amounts for each make, model, and year. Also, it sorts the results first by Make, then by Model, then by Year.

```
SELECT
       c.Make
       ,c.Model
       ,c.Year
       ,SUM(s.SalesAmount) as [Total Sales]
       ,AVG(s.SalesAmount) as [Mean Sales Price]
       ,COUNT(s.SaleID) as [Number of Sales]
FROM dbo.Sales s
JOIN dbo.Cars c
on s.CarID=c.CarID
GROUP BY c.Make, c.Model, c.Year
ORDER BY c.Make, c.Model, c.Year;
```

4.3 HAVING

Sometimes we'll want to filter our result set using the calculated aggregate. Using the example above, let's assume we want to limit our results to only makes, models, and years having more than 50 sales. The below version of the query includes a HAVING clause to limit the result set to only records with a Number of Sales value greater than or equal to 50.

```
SELECT
       c.Make
```

```
       ,c.Model
       ,c.Year
       ,SUM(s.SalesAmount) as [Total Sales]
       ,AVG(s.SalesAmount) as [Mean Sales Price]
       ,COUNT(s.SaleID) as [Number of Sales]
FROM dbo.Sales s
JOIN dbo.Cars c
on s.CarID=c.CarID
GROUP BY c.Make, c.Model, c.Year
HAVING COUNT(s.SaleID) >= 50
ORDER BY c.Make, c.Model, c.Year;
```

HAVING is processed immediately after GROUP BY and before the SELECT list, so we cannot use the column alias [Number of Sales] in it.

4.4 SELECT with WHERE, JOIN, GROUP BY, HAVING and ORDER BY

Reporting queries will often involve a WHERE clause in addition to GROUP BY, HAVING, and ORDER BY. Let's say you want to find the sum of sales of each make, model, and year of car, but only for sales in the year 2012. Also, you only want to include groups with a Total Sales value over 2,000,000. The following query will produce those results, sorted by make, model, and year.

```
SELECT
       c.Make
       ,c.Model
       ,c.Year
       ,SUM(s.SalesAmount) as [Total Sales]
FROM dbo.Sales s
JOIN dbo.Cars c
ON s.CarID=c.CarID
WHERE YEAR(s.SalesDate) = 2012
GROUP BY c.Make, c.Model, c.Year
HAVING SUM(s.SalesAmount) > 2000000
ORDER BY c.Make, c.Model, c.Year;
```

Remember the order of operations happening here:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

Remember - WHERE operates on the individual records produced by the FROM clause. HAVING operates on aggregate values after individual records have been collapsed by GROUP BY.

The following query will return sales data for 2012, and will group by sales person and car make. Only groups with Total Sales over 200,000 will be displayed. The results will be sorted by last name and sum of sales in descending order.

```
SELECT
       p.LastName [Sales Person]
       ,c.Make
       ,SUM(s.SalesAmount) as [Total Sales]
FROM dbo.Sales s
JOIN dbo.Cars c
ON s.CarID=c.CarID
JOIN dbo.SalesPeople sp
ON s.SalesPersonID=sp.EmployeeID
JOIN dbo.Person p
ON sp.PersonID=p.PersonID
WHERE YEAR(s.SalesDate) = 2012
GROUP BY p.LastName, c.Make
HAVING SUM(s.SalesAmount) > 200000
ORDER BY p.LastName, [Total Sales] DESC;
```

Notice that because of the order of operations, the column alias [Total Sales] does not yet exist when the GROUP BY is processed. However, we can use it in the ORDER BY because ORDER BY is processed last.

Additional Reading

Aggregate Functions (T-SQL). Microsoft's documentation of TSQL aggregate functions.
https://msdn.microsoft.com/en-us/library/ms173454.aspx

SELECT - GROUP BY – Transact-SQL. Microsoft's documentation of GROUP BY.
https://docs.microsoft.com/en-us/sql/t-sql/queries/select-group-by-transact-sql?view=sql-server-ver16

SELECT – HAVING (Transact-SQL). Microsoft's documentation of the HAVING clause.
https://docs.microsoft.com/en-us/sql/t-sql/queries/select-having-transact-sql?view=sql-server-ver16