Table of Contents

So far we have been using Select statements to return result sets that display individual rows from out tables. We have included filters so that we return only some of the rows but the result set were always focused on individual rows from the tables.

But think about what companies such as Amazon needs to know to do business. They do need to know detail information so that they can charge me for the one book that I purchased, but they also need to know about large groups of data- how many people purchased that book? How many books do we carry that no one has purchased in the last 6 months? What is the average amount due for our orders? What was the total sales for last month?

These type of questions ask for summary information about our data SQL provides aggregate functions to answer these questions. The aggregate functions are also called multi-row functions or group functions because they return a single value for each group of multiple rows.

The SQL aggregate functions we cover here are Count, Sum, Avg, Max, and Min. T-SQL has additional aggregate functions.

First we will use **aggregate functions** applied to the entire table. We could find the most expensive (max) price of our products; we could filter for pet supplies and find the most expensive pet supplies product. We also have the ability to take a table and partition it into a set of sub-tables. With a partition, each row of the table is put into exactly one of the sub-tables- into one of the groups. We can then use an aggregate function with each of these subsets. This uses a new clause in our Select statement- the **Group By** clause. This would let us find the most expensive product in each of our product categories.

At times we may want to see aggregate values only if the aggregate itself meets a specified criterion- for this we have another new clause- the **Having** clause. For example, we might want to see customers who have more than 5 orders in our tables.

At the end of this unit we will have the following model for the Select statement.

```
select . . .
from . . .
where . . .
group by . . .
having . . .
order by . . .
```

Reminder: the data in the tables you are downloading for this semester might not match the data that I use in preparing the notes; so the actual values displayed here might not match the actual values you get when you run the demos. That is intentional. You should be focusing on the query logic not the specific numeric results.

1. Using Aggregate Functions over the Table

1.1. Avg, Sum, Max or Min

With the Avg, Sum, Max or Min functions, supply the column that you want to use for the calculated value. For Avg and Sum the column, should be numeric; Max and Min can also be used with date columns or character data or any data which has a sort order.

Demo 01: Using AVG to find the average price of all items in the table.

```
Select AVG(prod_list_price) as "AvgPrice"
From a_prd.products;
```

```
+------------+
| AvgPrice   |
+------------+
| 117.617400 |
+------------+
```

Demo 02: Using aggregates and a criterion. What is the average price of the housewares item we carry?

```
Select Avg(prod_list_price) as "AvgPrice"
From a_prd.products
Where catg_id = 'HW';
+-----------+
| AvgPrice  |
+-----------+
| 67.641000 |
+-----------+
```

Demo 03: Using more than one aggregate function

```
Select Avg(prod_list_price) as "Avg Price"
, Min(prod_list_price) as "Min Price"
, Max(prod_list_price) as "Max Price"
From a_prd.products
Where catg_id = 'HW';
+-----------+-----------+-----------+
| Avg Price | Min Price | Max Price |
+-----------+-----------+-----------+
| 67.641000 |     25.00 |    149.99 |
+-----------+-----------+-----------+
```

Demo 04: Using aggregates and a criterion. For the houseware items we have on a current order, what is the average list price, the average quoted price and the average extended cost? We can also calculate the total extended price for all HW orders

```
Select Avg(prod_list_price) as "AvgListPrice"
, Avg(quoted_price) as "AvgQuotedPrice"
, Avg(quoted_price* quantity_ordered) as "AvgExtendedCost"
, SUM(quoted_price* quantity_ordered) as "TotalExtendedCost"
From a_prd.products
Join a_oe.order_details using (prod_id)
Where catg_id = 'HW';
+--------------+----------------+-----------------+--------------------+
| AvgListPrice | AvgQuotedPrice | AvgExtendedCost | TotalExtendedCost |
+--------------+----------------+-----------------+--------------------+
|    56.395500 |      57.896250 |      101.261500 |            4050.46 |
+--------------+----------------+-----------------+--------------------+
```

Note that for each of these queries, we have used aggregate functions in the select clause- and only aggregate functions in the select clause and we have one row in the result set. An aggregate function is different than a single row function. An aggregate function takes a group (here the whole table) and produces a single answer for that group. We can include a Where clause which filters the table produced by the From clause and that filtered set of rows becomes the group. The Where clause is carried out before the aggregates are calculated.

Demo 05: You can do some additional work in the select clause such as applying a function such as round to the aggregated value or combining aggregated values.

```
Select round(Avg(prod_list_price),0) as "Avg Price"
, Max(prod_list_price) - Min(prod_list_price) as "Price Range"
```

```
From a_prd.products
;
```

1.2. Aggregate functions and non aggregated columns

You might want to see the name of the most expensive item we sell- but this type of query cannot tell you that. A function, including an aggregate function, returns a single value for its argument. We could have two or more items selling at that high price. The query is written to return one row for all of the items grouped together and so it cannot show the product id for the most expensive product.

Demo 06: What is the highest price for any item we sell ?

```
Select Max(prod_list_price) as "Max Price"
From a_prd.products;
+-----------+
| Max Price |
+-----------+
|    850.00 |
+-----------+
```

Demo 07: What is the highest price for any item we sell and what is the item- this one does not work correctly.

```
Select Max(prod_list_price) as "Max Price", prod_id
From a_prd.products;
+-----------+---------+
| Max Price | prod_id |
+-----------+---------+
|    850.00 |    1000 |
+-----------+---------+
```

But when we look at the products table:

```
select prod_id, prod_list_price From a_prd.products;
+---------+-----------------+
| prod_id | prod_list_price |
+---------+-----------------+
|    1000 |          125.00 |
|    1010 |          150.00 |
|    1020 |           12.95 |
. . .  rows omitted
```

Product ID 1000 does not sell for 850.00 . This is caused by a MySQL implementation of the aggregates. What MySQL does is aggregate the price but then it just gives you one of the product ids. This is a serious problem if you do not do the aggregates properly. I will discuss this feature in a separate document.

The output of the previous demo does not accurately reflect our data. For now - the rules will be if you do an aggregate of the entire table (or the table filtered with a where clause) then display only aggregated values.

The following does not work to find the most expensive product. You are not allowed to use an aggregate function in a Where clause this way. It looks like it should work- after all you can determine the Max (product list price) and that is a single value but the Where clause is for single row filters and aggregates work on groups.

Demo 08:

```
Select prod_id
, prod_name
, prod_list_price
From a_prd.products
Where prod_list_price = MAX(prod_list_price);
```

```
Error report:
SQL Error: Invalid use of group function
```

But we have done some simple subqueries that we can use to handle this.

Demo 09: Using a subquery; the subquery returns the amount of the highest price and the outer query displays all items with that price. With our current set of data we have only one product at that price.

```
Select prod_id
, prod_name
, prod_list_price
From a_prd.products
Where prod_list_price = (
   Select MAX(prod_list_price) as "Largest Price"
   From a_prd.products
   );
+---------+-----------------+-----------------+
| prod_id | prod_name       | prod_list_price |
+---------+-----------------+-----------------+
|    1126 | Washer          |          850.00 |
+---------+-----------------+-----------------+
```

Demo 10: Suppose we want to find the highest priced sporting goods items. This runs but the result is incorrect

```
Select prod_id
, prod_name
From a_prd.products
Where prod_list_price = (
   Select MAX(prod_list_price) as "Largest Price"
   From a_prd.products
   Where catg_id = 'SPG'
   );
+---------+------------+
| prod_id | prod_name  |
+---------+------------+
|    1040 | Treadmill  |
|    4569 | Mini Dryer |
+---------+------------+
```

This query gives us two items- but if we check, one of them is actually an appliance item. So we need to filter in the outer query for category as well.

Demo 11: Suppose we want to find the highest priced sporting goods items. Corrected

```
set @catg_id = 'SPG';

Select prod_id
, prod_name
From a_prd.products
Where catg_id = @catg_id
and prod_list_price = (
   Select MAX(prod_list_price) as "Largest Price"
   From a_prd.products
   Where catg_id = @catg_id
   );
```

Demo 12: If we change the variable to PET and run the same query, we get more than one item tied for the most expensive in that category.

```
set @catg_id = 'PET';

Select prod_id
, prod_name
From a_prd.products
Where catg_id = @catg_id
and prod_list_price = (
   Select MAX(prod_list_price) as "Largest Price"
   From a_prd.products
   Where catg_id = @catg_id
   )
;
+---------+-----------------+
| prod_id | prod_name       |
+---------+-----------------+
|    4567 | Deluxe Cat Tree |
|    4568 | Deluxe Cat Bed  |
+---------+-----------------+
```

Demo 13: What is the earliest order date we have? We can use Min with dates.

```
Select min(ord_date) as "Earliest"
From a_oe.order_headers;
+---------------------+
| Earliest            |
+---------------------+
| 2012-02-02 00:00:00 |
+---------------------+
```

Demo 14: What are the earliest and latest hire date we have? We can use Min and Max with dates.

```
Select min(hire_date) as "Earliest"
, max(hire_date)      as "Recent"
From a_emp.employees;
+------------+------------+
| Earliest   | Recent     |
+------------+------------+
| 1989-06-17 | 2012-02-29 |
+------------+------------+
```

Demo 15: We can use Max and Min with strings because they have an ordering.

```
Select min(name_last), max(name_last)
From a_emp.employees;
+----------------+----------------+
| min(name_last) | max(name_last) |
+----------------+----------------+
| Chen           | Whale          |
+----------------+----------------+
```

MySQL does not allow you to nest aggregate functions.

1.3. Count

Count(expression) will determine the number of rows where that expression is not null.

Demo 16: Using the Count aggregate function with a column attribute.

```
Select COUNT(prod_id) as "Number of Products"
, COUNT(prod_warranty_period)  as "Number with warranty"
From a_prd.products
;
+--------------------+----------------------+
| Number of Products | Number with warranty |
+--------------------+----------------------+
|                 50 |                   30 |
+--------------------+----------------------+
```

At the time this query was run, we had49 products with a product id; since that is the pk attribute we have 49 products in our table. And 30of the products have a value for the warranty attribute.

Demo 17: Using the Count aggregate to count rows in the table.

```
Select COUNT(prod_id) as "Number of Products"
, COUNT(*) as "Number of rows *"
, COUNT(1) as "Number of rows 1"
From a_prd.products
;
+--------------------+-----------------+------------------+
| Number of Products | Number of rows * | Number of rows 1 |
+--------------------+-----------------+------------------+
|                 50 |              50 |               50 |
+--------------------+-----------------+------------------+
```

To count the number of rows, we can use Count (PK_attribute) or Count(*) or Count(literal) .

1.4. Count(Distinct)

Suppose we want to know how many products we have currently on order. It is important to realize that a simple request like this could have different interpretations. Suppose we have only these rows in our order details table.

Ord_id	Line_Item_id	Prod_id	Quantity	Quoted_price
1	1	P12	2	5.00
1	2	T20	4	12.00
1	3	R67	1	25.00
2	1	P12	2	5.00
3	1	P12	8	6.50
3	2	R67	5	25.00

The question "how many products do we have currently on order ?" could be interpreted to mean "how many line items do we have ?" ; in this example we have 6 line items- we could answer this by using Count(*). The question could also be interpreted to mean "what is the total quantity of items that we have on order?"; this would be 2 + 4 + 1 + 2 + 8 +5 = 22 – there is another group function Sum (quantity) that we could use to answer that question. Another interpretation is "how many different products do we have on order "; this would be products P12, R67, and T20; we have three different products currently on order. This is answered using the Count (Distinct prod_id) function.

Demo 18: Using count distinct

```
Select Count(Distinct prod_id)
From a_oe.order_details
;
+-------------------------+
| Count(Distinct prod_id) |
+-------------------------+
|                      34 |
+-------------------------+
```

Note that the use of this keyword Distinct is different in syntax than the use of Distinct with a Select statement to return only unique rows.

When you add Distinct inside the argument list to an aggregate function, the function will determine the answer by using only unique values.

With MySQL, you can have more than one argument to the function when you use distinct.

Demo 19: How many orders do we have? We can count the order ids in the order headers table.

```
Select count( ord_id)
From a_oe.order_headers
;
+-----------------+
| count( ord_id) |
+-----------------+
|              78 |
+-----------------+
```

Demo 20: If we count the order id in the order detail tables, we count any order with multiple order lines more than once. Use distinct to count each order only once.

```
Select count( ord_id), count(distinct ord_id)
From a_oe.order_details
;
+-----------------+------------------------+
| count( ord_id) | count(distinct ord_id) |
+-----------------+------------------------+
|             144 |                     75 |
+-----------------+------------------------+
```

Demo 21: How many customers do we have with orders? We can count the distinct customer ids in the order headers table. If we count cust_id then we count customer with multiple orders multiple times.We have only 33 customers.

```
Select count( distinct cust_id), count(cust_id)
From a_oe.order_headers
;
+--------------------------+----------------+
| count( distinct cust_id) | count(cust_id) |
+--------------------------+----------------+
|                       21 |             78 |
+--------------------------+----------------+
```

Demo 22: Suppose we want to find out how many different combinations of customers and shipping modes we have. I am not going to count the orders where there is no info on the shipping modes.

```
Select count( distinct cust_id, shipping_mode)
From a_oe.order_headers
Where shipping_mode is not null;
```

```
+----------------------------------------+
| count( distinct cust_id, shipping_mode) |
+----------------------------------------+
|                                     27 |
+----------------------------------------+
```

You can use Distinct with any of the aggregates- but don't use it if you really don't need it. If you are looking for the price of the most expensive item we sell, use Max(price), not Max (Distinct price).

As always, you need to use these features thoughtfully. The following query determines the number of customer id values in the table using Count(cust_id) and the number of different customer id values in the table using Count(Distinct cust_id). But it makes no sense to use the distinct feature here. This query uses a single table- the customers table and the cust_id is the primary key for that table. By definition that means that every value of cust_id is different and therefore these two counts will always be the same. This is different than using count(distinct cust_ID) from a_oe.order_headers where we can have multiple occurrences of the customer id values.

Demo 23:

```
Select count(cust_id) as Count
, count(distinct cust_id) as CountDistinct
From a_oe.customers ;
+-------+---------------+
| Count | CountDistinct |
+-------+---------------+
|    33 |            33 |
+-------+---------------+
```

Warning: MySQL has a row level function Greatest which works on one row at a time and requires more than one argument. It is NOT the same as the Max aggregate function. people use the wrong function on the final exam (a lot) and lose points (a lot).