Table of Contents

So far we have been using Select statements to return result sets that display individual rows from our 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. The Where clauses filtered on one row at a time based on the table expression produced by the From clause.

But think about what companies such as Amazon need 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 in this class are Max, Min, Sum, Avg, and Count. MySQL has additional aggregate functions.

In this unit we will discuss using aggregate functions applied to the entire table expression as filtered by the Where clause. We could find the highest price of all of our products; we could filter for pet supplies and find the highest price for pet supplies product.

1. The Max and Min functions

We will start with some examples where we consider the entire table expression collection of rows as a single group. We use the Max function to get the largest value for a column in the table and the Min function to get the smallest value. These queries are using the tables associated with the AltgeldMart schemas.

Demo 01: max and min order id from the order headers table

```
select max(order_id) as Max_OrderID
, min(order_id) as Min_OrderID
from OrderEntry.orderHeaders;
+-------------+-------------+
| Max_OrderID | Min_OrderID |
+-------------+-------------+
|        4511 |         105 |
+-------------+-------------+
```

First notice that we get one row returned. We have almost 100 rows in the table, but we are applying the max and min functions to the entire table. The order_id column is numeric so this is just the biggest number and the smallest number for the order id values.

Demo 02: We can use Max, and Min with numeric data, with dates, and with character data. With character data, max and min use the sorting order for the data.

```
select max(order_date) as Max_OrderDate
, min(order_date) as Min_OrderDate
, max(order_mode) as Max_OrderMode
, min(order_mode) as Min_OrderMode
from OrderEntry.orderHeaders;
+---------------------+---------------------+---------------+---------------+
| Max_OrderDate       | Min_OrderDate       | Max_OrderMode | Min_OrderMode |
+---------------------+---------------------+---------------+---------------+
| 2016-05-12 00:00:00 | 2015-04-05 00:00:00 | ONLINE        | DIRECT        |
+---------------------+---------------------+---------------+---------------+
```

2. The Sum function

The Sum function adds the values in the table column. Sum is used with numeric data but notes that MySQL will allow the use of Sum with non-numeric data- but the result is not what a person should expect.

Demo 03: getting the total of the quantities sold

```
select Sum(quantity_ordered ) as SumQuantity
from OrderEntry.orderDetails;
+-------------+
| SumQuantity |
+-------------+
|         950 |
+-------------+
```

Demo 04: getting the total amount due for all orders in the table.

```
select Sum(quantity_ordered * quoted_price) as totalAmountDue
from OrderEntry.orderDetails;
+----------------+
| totalAmountDue |
+----------------+
|       84989.98 |
+----------------+
```

Demo 05: (MySQL Feature) What happens if you try to use Sum with a character type or a date type or column? For the character type columns, you get a warning message about data types for each row in the table. There is no good business rule for adding string values. You may have seen this warning message before: Warning (Code 1292): Truncated incorrect DOUBLE value: 'DIRECT' This comes when you try to treat a string as a number.

The date type is worse, you get an answer with no warnings. I do not see any point in adding up date values. Other dbms will throw errors with these queries.

```
select sum(ord_mode)
from OrderEntry.orderHeaders;
+---------------+
| sum(ord_mode) |
+---------------+
|             0 |
+---------------+
1 row in set, 97 warnings (0.00 sec)
Warning (Code 1292): Truncated incorrect DOUBLE value: 'DIRECT'
Warning (Code 1292): Truncated incorrect DOUBLE value: 'DIRECT'
Warning (Code 1292): Truncated incorrect DOUBLE value: 'ONLINE'
Warning (Code 1292): Truncated incorrect DOUBLE value: 'ONLINE'
. . . .
```

```
select sum(ord_date)
from OrderEntry.orderHeaders;
+------------------+
| sum(order_date)  |
+------------------+
| 1955013402000000 |
+------------------+
```

3. The Avg function

The average function returns the average(mean) of the values. Avg is used with numeric data. MySQL will accept date and string data but you will get the same types of problems as with Sum.

Demo 06: Examples of the avg function

```
select
  Avg(quantity_ordered) as AvgQuantity
, Avg(quantity_ordered * quoted_price) as AvgAmountDue
from OrderEntry.orderDetails;
+-------------+--------------+
| AvgQuantity | AvgAmountDue |
+-------------+--------------+
|      5.1630 |   461.902065 |
+-------------+--------------+
```

4. What about nulls?

We always need to consider nulls.

Demo 07: This is a small table I created in the a_testbed database.

```
create table a_testbed.z_tst_aggr(
  id int primary key
, col_int int null
);
```

Demo 08: If I do these aggregates on an empty table, I get nulls. But I do get a row of nulls.

```
select max(col_int) as theMax, sum(col_int) as theSum, avg(col_int) as theAvg
from a_testbed.z_tst_aggr;
+--------+--------+--------+
| theMax | theSum | theAvg |
+--------+--------+--------+
|   NULL |   NULL |   NULL |
+--------+--------+--------+
```

Now I insert 8 rows and run that aggregates again.

```
insert into a_testbed.z_tst_aggr values
  ( 1, 10),  (2, 15),   (3, null), (4, null), (5, null)
, (6, null), (7, null), (8, null);

select max(col_int) as theMax, sum(col_int) as theSum, avg(col_int) as theAvg
from a_testbed.z_tst_aggr;
+--------+--------+---------+
| theMax | theSum | theAvg  |
+--------+--------+---------+
|     15 |     25 | 12.5000 |
+--------+--------+---------+
```

What these functions do is ignore nulls and just work with the non-null data values. You would have a logical point if you argued that these function should return nulls; in this case we have 8 rows, 6 of the rows have unknown values and the average function simply returns the average of two rows. So you have to think of these functions as returning the Max, Sum, Avg of the values they know.

Some people will argue that the average function should take the sum of the values (25) and divide it by the number of rows (8) and the avg would then be 3. But this means that you are treating the unknown values as zero- which may not be a good business rule.

You can code the coalesce function call to specifically treat the nulls as zero.

```
select max(col_int) as theMax, sum(col_int) as theSum
, avg(coalesce(col_int,0)) as theAvg
from a_testbed.z_tst_aggr;
+--------+--------+--------+
| theMax | theSum | theAvg |
+--------+--------+--------+
|     15 |     25 | 3.1250 |
+--------+--------+--------+
```

Don't get mad at the average function and do not assume all unknown numeric values are zero.

Suppose we are using the avg function to find the average salary for all of our employees and some rows have a null for salary. Does the result returned by avg make business sense? This depends on the situation. Suppose we have 40,000 rows in our table and every row except for two have a value for salary. In that case you might decide that it is safe to ignore the fact that we are missing two rows of data and that the value returned by avg makes sense. SQL cannot help you with that decision; you can use sql to find out what percent of rows are missing data; you can use sql to find the average of the known values. But if those two rows were the rows for the two executives in the company who are hiding their salary values because they are paid so very much more than everyone else, then the value returned by avg does not reflect reality.

On the other hand if we have 40,000 rows in our table and 30,000 of those rows are missing a value for salary, then I don't think I would put much trust that the value returned by avg reflects reality.
Business decision always need to be made by the business experts- not the sql coder. The sql person can help supply info to help the business experts make decisions.

5. Using aggregates and a Where clause and a Join

Demo 09: Using aggregates and a criterion. What is the average list price of the houseware items we carry?

```
select  Avg (prod_list_price) as "AvgPrice"
from Product.products
where catg_id = 'HW';
+-----------+
| AvgPrice  |
+-----------+
| 67.641000 |
+-----------+
```

If we change the filter to CEQ we get a null because we do not have any rows with that category- the filtered table expression in the From and Where clause is empty. But we do get a row- it is just a row with a null.

Demo 10: Using aggregates and a join. For the houseware items we have on an existing order, what is the average list price, the average quoted price and the average extended cost? select

```
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  Product.products
```

```
join OrderEntry.orderDetails on Product.products.prod_id =
OrderEntry.orderDetails.prod_id
where catg_id = 'HW';
+--------------+----------------+-----------------+-------------------+
| AvgListPrice | AvgQuotedPrice | AvgExtendedCost | TotalExtendedCost |
+--------------+----------------+-----------------+-------------------+
|    54.302037 |      54.098889 |      154.728889 |           8355.36 |
+--------------+----------------+-----------------+-------------------+
```

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 11: 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 Product.products;
```

6. 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 arguments. 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.

Demo 12: What is the highest list price for any item we sell?

```
select Max(prod_list_price) as "Max Price"
from Product.products;
+-----------+
| Max Price |
+-----------+
|    850.00 |
+-----------+
```

It is possible to display a literal and an aggregate function.

```
select current_date as RunDate
, MAX(prod_list_price) As "Max Price"
from Product.products;
+------------+-----------+
| RunDate    | Max Price |
+------------+-----------+
| 2016-03-18 |    850.00 |
+------------+-----------+
```

6.1. Find the winner queries

Demo 13: What is the highest price for any item we sell and what is the item- this one does not work properly but we do not get an error message.

```
select Max(prod_list_price) as "Max Price"
, prod_id
from  Product.products;
+-----------+---------+
| Max Price | prod_id |
+-----------+---------+
|    850.00 |    1000 |
+-----------+---------+
```

Try running the following query; it shows that we have only one product with a price of 850 and its prod_id is 1126, not 1000

```
select prod_id
from  Product.products
where prod_list_price = 850;
+---------+
| prod_id |
+---------+
|    1126 |
+---------+
```

MySQL Feature: MySQL lets you write an aggregate function- that applies to the table as a whole- and a column expression- that belongs to a single row- in the same Select clause. The value that is returned for the prod_id will be one of the values in that column but you have no control over which value. What MySQL does is aggregate the price but then it just gives you one of the product ids. We are considering the table as a whole and we can ask for the highest value we have in the table for the list price- that is a characteristic of the table as a whole- but we cannot ask for a prod_id since each row (each product) has its own value for that column and there is no single value of prod_id that represent the entire table

The MySQL implementation is a source of a serious problem if you do not do the aggregates properly. I will discuss this feature again when we discuss groups. For now, if you are using a aggregate against the entire table, the Select clause should contain only aggregate functions and literals.

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 14: THIS DOES NOT WORK

```
select  prod_id
, prod_name
, prod_list_price
from Product.products
where prod_list_price = MAX(prod_list_price)
```

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

Demo 15: Using a subquery; the subquery returns the amount of the highest price and the outer query displays all items with that price.
With the current data set there is only one match.

```
select  prod_id
, prod_name
, prod_list_price
from Product.products
where prod_list_price = (
   select  MAX(prod_list_price) as "Largest Price"
   from Product.products);
+---------+-------------+-----------------+
| prod_id | prod_name   | prod_list_price |
+---------+-------------+-----------------+
|    1126 | WasherDryer |          850.00 |
+---------+-------------+-----------------+
```

Demo 16: Suppose we want to find the highest priced sporting goods items. This runs but the result is incorrect. The mini dryer is not a sporting goods item

```
select  prod_id
, prod_name
from Product.products
where prod_list_price = (
    select  MAX(prod_list_price) as "Largest Price"
    from Product.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 17: Suppose we want to find the highest priced sporting goods items. Corrected query

```
set  @catg_id  = 'SPG';

select prod_id
, prod_name
from Product.products
where catg_id = @catg_id
and prod_list_price = (
    select  MAX(prod_list_price)
    from Product.products
    where catg_id = @catg_id);
+---------+------------+
| prod_id | prod_name  |
+---------+------------+
|    1040 | Treadmill  |
+---------+------------+
```

Demo 18: 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 Product.products
where catg_id = @catg_id
and prod_list_price = (
    select  MAX(prod_list_price) as "Largest Price"
    from Product.products
    Where catg_id = @catg_id);
+---------+-----------------+
| prod_id | prod_name       |
+---------+-----------------+
|    4567 | Deluxe Cat Tree |
|    4568 | Deluxe Cat Bed  |
+---------+-----------------+
```

7. limit 1 versus max

If you want the highest price for any product in the products table you could use either of these queries.

```
   select prod_list_price
   from Product.products
   order by prod_list_price desc
   limit 1;

   select max(prod_list_price)
   from Product.products;
```

Some reasons for using the Max function approach

1. If you use limit, you need to remember to sort and you need to sort in the proper order (asc or desc). That seems to be something that is easy to get wrong.

3. Limit is a MySQL specific technique; max works with all dbms.

8. Mistakes you should not make

1. You cannot nest aggregates in MySQL. The following does not run.

```
   select count(max(prod_list_price))
   from Product.products
```

```
ERROR 1111 (HY000): Invalid use of group function
```

2. Some people think that if a query runs and produces results, the query is in some sense ok. This is not true. For example, the function Sum will add up numeric values. So you could run this query and get a result. But the result has no meaning in a business sense. There is no point to adding up customer id numbers.

```
   select sum(customer_id)
   from Customer.customers;
   +------------------+
   | sum(customer_id) |
   +------------------+
   |         15262819 |
   +------------------+
```

You are responsible to writing queries that are meaningful.

3.The Where clause cannot contain an aggregate function. The Where clause filters on single rows; the aggregates work with groups of rows. See the Having clause in another document..

4.These functions take an argument that is either a column in the table, or an expression based on a column. So you can write `Select Max(prod_list_price) as "Max Price" from Propduct.products;` But you cannot write `Select Max(45,90)`; to have SQL find the larger value. Use the aggregate functions with column expressions.