LAB 06

GROUP QUERIES

❖ **Main contents:**

- Group Functions: SUM, AVG, MAX, MIN, COUNT
- GROUP BY clause
- HAVING clause

1. Group Functions

SUM function

Sometimes, the information we need is not actually stored in database tables, but we can get it by handling it from the stored data. For example, we have the *OrderDetails* table, which stores information about orders. When we scan this table, we do not know what the total amount of all the products sold is. However, the `SUM` function can help us answer this question. First of all, we see the operation of the `SUM` function, the data group implementation will be presented in Part 2.

Example: Calculating the total number of goods currently in stock

```
SELECT SUM(quantityInStock)
FROM products;
```

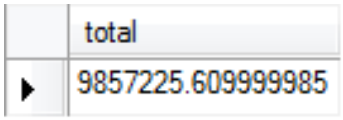
Result:

sum(quantityInStock)
555131

Or to calculate the total amount we have collected, execute the query as follows:

```
SELECT SUM(priceEach * quantityOrdered) total
FROM orderdetails;
```

Result:



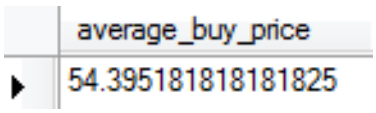
	total
▶	9857225.609999985

AVG function

`AVG` is used to calculate the average value of an expression. It does not accept NULL values. We can use `AVG` to calculate the average price of all products purchased as follows:

```
SELECT AVG(buyPrice) average_buy_price
FROM Products
```

Result:



	average_buy_price
▶	54.395181818181825

MAX and MIN function

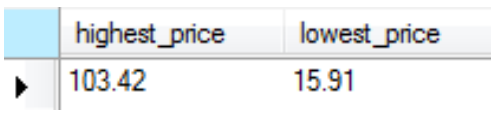
`MAX` function returns the maximum value and the `MIN` function returns the smallest value of a set of values.

```
MAX(expression)
MIN(expression)
```

Example: Use `MAX` and `MIN` to get the highest price and the lowest price of the product.

```
SELECT MAX(buyPrice) highest_price,
            MIN(buyPrice) lowest_price
FROM Products
```

Result:



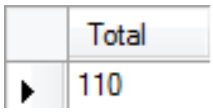
	highest_price	lowest_price
▶	103.42	15.91

COUNT function

`COUNT` is a function to count the number, such that we can count the number of products being sold are as follows:

```
SELECT COUNT(*) AS Total
FROM products
```

Result:



	Total
▶	110

Note: another version of the `COUNT` function uses a column name as a parameter. If this method is used, it will only count the rows whose value is NOT NULL.

2. GROUP BY statement

`GROUP BY` statement is used to combine records with the same value in one or more columns, into a collection. If `GROUP BY` exists, it must follow the `WHERE` or `FROM` statement. Following the `GROUP BY` keyword is a list of expressions, separated by commas.

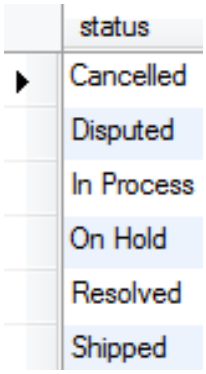
```
SELECT col1, col_2, ... col_n, group_function(expressions)
FROM table name
WHERE condition
GROUP BY col_1, col_2, ... col_n
ORDER BY column list
```

By definition, the group function allows us to perform a calculation on a record set and return a value. The group function ignores NULL values when performing calculations, except for the `COUNT` function. The group function is often used with the `GROUP BY` statement of the `SELECT` statement.

Example: Suppose we want to divide orders by groups depending on the status of the orders, we can do the following:

```
SELECT status
FROM orders
GROUP BY status;
```

Result:



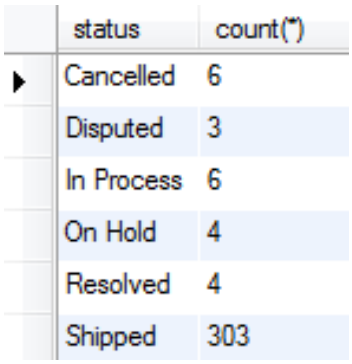
status
Cancelled
Disputed
In Process
On Hold
Resolved
Shipped

The group functions are used with `GROUP BY` *to perform calculations on each group of records and return a unique value for each row.*

Example: If we want to know how many orders are in each status group, we can use the `COUNT` function as follows:

```
SELECT status, COUNT(*)
FROM orders
GROUP BY status;
```

Result:



status	count(*)
Cancelled	6
Disputed	3
In Process	6
On Hold	4
Resolved	4
Shipped	303

Example: If we want to know how many products are in each product line:

```
SELECT productLine, COUNT(*)
FROM products
GROUP BY productline;
```

productLine	count(*)
Classic Cars	38
Motorcycles	13
Planes	12
Ships	9
Trains	3
Trucks and Buses	11
Vintage Cars	24

Example: To get the total amount for each product sold, we just need to use the `SUM` function and the product group. Here is the query:

```
SELECT productCode, SUM(priceEach * quantityOrdered) total
FROM orderdetails
GROUP by productCode;
```

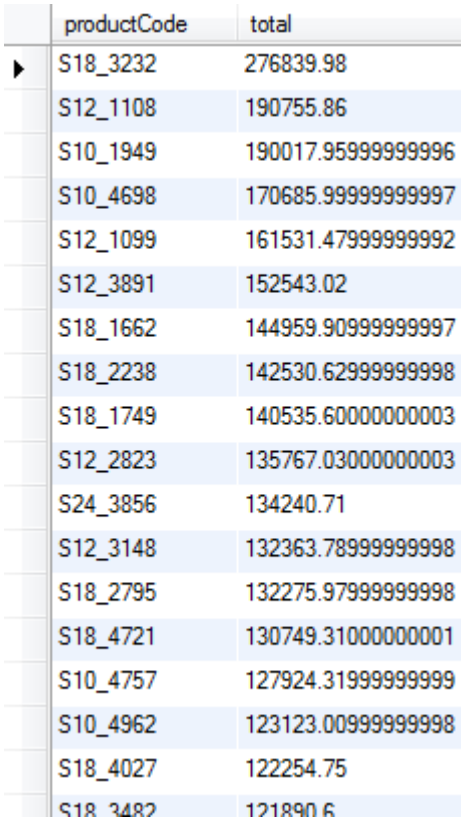
Result:

productCode	total
S10_1678	90157.77000000002
S10_1949	190017.95999999996
S10_2016	109998.81999999998
S10_4698	170685.99999999997
S10_4757	127924.31999999999
S10_4962	123123.00999999998
S12_1099	161531.47999999992
S12_1108	190755.86
S12_1666	119085.24999999999
S12_2823	135767.03000000003
S12_3148	132363.78999999998
S12_3380	98718.76000000001

Example: Suppose we want to see the results of the above query, shown in ascending order we do the following:

```
SELECT productCode, SUM(priceEach * quantityOrdered) total
FROM orderdetails
GROUP by productCode
ORDER BY total DESC;
```

Result:



productCode	total
S18_3232	276839.98
S12_1108	190755.86
S10_1949	190017.95999999996
S10_4698	170685.99999999997
S12_1099	161531.47999999992
S12_3891	152543.02
S18_1662	144959.90999999997
S18_2238	142530.62999999998
S18_1749	140535.60000000003
S12_2823	135767.03000000003
S24_3856	134240.71
S12_3148	132363.78999999998
S18_2795	132275.97999999998
S18_4721	130749.31000000001
S10_4757	127924.31999999999
S10_4962	123123.00999999998
S18_4027	122254.75
S18_3482	121890.6

Note: the difference between GROUP BY in MySQL and ANSI SQL

MySQL follows the ANSI SQL standard. However, there are 2 differences when using GROUP BY in MySQL as follows:

- In ANSI SQL, GROUP BY must be executed for all the columns that appear in the SELECT statement. MySQL does not require that, can include more columns in the SELECT statement and does not force them to appear in the GROUP BY statement.
- MySQL also allows the arrangement of groups in the order of the calculation results; the default is descending.

3. HAVING clause

`HAVING` is also a clause that may or may not appear in the `SELECT` statement. It indicates whether a filter on the data is a group of records or the result of the group function implementation. `HAVING` is often used in conjunction with `GROUP BY`, so that the filter condition is only applied on the columns that appear in the `GROUP BY` clause. If `HAVING` is not included with `GROUP BY`, then it is as meaningful as `WHERE`. Note that `HAVING` applies to groups of records, while `WHERE` applies to each individual record.

Example: Use the `GROUP BY` clause to get all orders, the number of items sold, and the total value of each order as follows:

```
SELECT ordernumber,
     SUM(quantityOrdered) AS itemsCount,
     SUM(priceEach * quantityOrdered) AS total
FROM orderdetails
GROUP BY ordernumber;
```

ordernumber	itemsCount	total
10100	151	10223.829999999998
10101	142	10549.01
10102	80	5494.78
10103	541	50218.950000000004
10104	443	40206.2
10105	545	53959.21
10106	675	52151.810000000005
10107	229	22292.620000000003
10108	561	51001.219999999994
10109	212	25833.14
10110	570	48425.69
10111	217	16537.850000000002
10112	52	7674.9400000000005
10113	143	11044.300000000001
10114	351	33383.14000000001
10115	210	21665.980000000003
10116	27	1627.56
10117	402	44380.15

Now, it is possible to request that only orders with a total value greater than $1000 be displayed using `HAVING` as follows:

```
SELECT ordernumber,
     SUM(quantityOrdered) AS itemsCount,
     SUM(priceEach * quantityOrdered) AS total
FROM orderdetails
GROUP BY ordernumber
HAVING total > 1000;
```

ordernumber	itemsCount	total
10100	151	10223.829999999998
10101	142	10549.01
10102	80	5494.78
10103	541	50218.950000000004
10104	443	40206.2
10105	545	53959.21
10106	675	52151.810000000005
10107	229	22292.620000000003
10108	561	51001.219999999994
10109	212	25833.14
10110	570	48425.69
10111	217	16537.850000000002
10112	52	7674.9400000000005
10113	143	11044.300000000001
10114	351	33383.14000000001
10115	210	21665.980000000003
10116	27	1627.56
10117	402	44380.15

We use the alias for the sum column (priceEach * quantityOrdered) as *total*, so in the `HAVING` clause, we just need to use that alias instead of Typing `SUM` (`priceEach`) again.

A combination condition can be used in the `HAVING` clause with the `OR`, `AND` operators.

Example: If we want to know that orders total greater than $1000 and have more than 600 items in them, we can use the following query:

```
SELECT ordernumber,
     sum(quantityOrdered) AS itemsCount,
     sum(priceeach) AS total
FROM orderdetails
GROUP BY ordernumber
HAVING total > 1000 AND itemsCount > 600;
```

Result:

ordernumber	itemsCount	total
10106	675	1427.2800000000002
10126	617	1623.71
10135	607	1494.86
10165	670	1794.9399999999996
10168	642	1472.5
10204	619	1619.73
10207	615	1560.08
10212	612	1541.8300000000002
10222	717	1389.51
10262	605	1217.38
10275	601	1455.4099999999999
10310	619	1656.2600000000002
10312	601	1494.1900000000003
10316	623	1375.59

❖ **Practice Exercises:**

1. Get the names of the cities and the number of customers in each city.
2. Get the number of orders in March 2005. Get the number of orders for each month in 2005.
3. Get the 10 order numbers that have the most valuable price.
4. Get the product line and the total quantity in stock of that group.
5. Get the customer number and the total amount that customer paid.