Detailed Data Selection

SQL

Other Keywords

Let’s have a look at some of the other keywords and features in MariaDB / SQL.

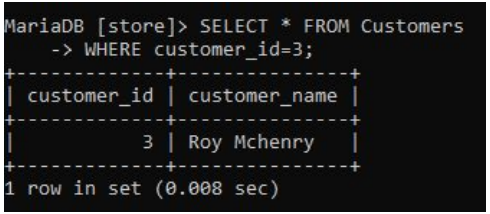
- WHERE (AND, OR, and NOT)
- NULL
- LIKE
- LIMIT
- AS
- COUNT
- AVG
- SUM
- MIN
- MAX
- GROUP BY
- HAVING

WHERE

We'd looked previously, when speaking about basic "read" of records in the database, a basic WHERE clause. Let's do a quick review. As an example let's read the name associated with customer_id number 3:

```
SELECT * FROM Customers
  WHERE customer_id=3;
```

Notice how this retrieves a record, or records, that match the specified condition.



AND

For multiple conditions, you can add “AND” and your additional expression. See the following example:

```
SELECT * FROM Customers
  WHERE customer_name=”Max Zets”;
```

```
MariaDB [store]> SELECT * FROM Customers
    -> WHERE customer_name="Max Zets";
+-------------+---------------+
| customer_id | customer_name |
+-------------+---------------+
|           1 | Max Zets      |
|           6 | Max Zets      |
+-------------+---------------+
2 rows in set (0.001 sec)
```

Here we get two entries (if you’ve followed the previous examples.) Let’s try a more specific request, making use of AND to add a second condition:

```
SELECT * FROM Customers
  WHERE customer_id=6
  AND customer_name=”Max Zets”;
```

```
MariaDB [store]> SELECT * FROM Customers
    -> WHERE customer_id=6
    -> AND customer_name="Max Zets";
+-------------+---------------+
| customer_id | customer_name |
+-------------+---------------+
|           6 | Max Zets      |
+-------------+---------------+
1 row in set (0.001 sec)
```

OR

OR and AND differ a bit in how they affect the results. AND means that **both** conditions must be met for the entry to be considered for the result. OR means that **either** of the conditions being met will allow the entry into the returned result. Notice the difference in this set of comparisons:

```
SELECT * FROM Customers
  WHERE customer_name=”Everette Bagozzi”
  AND customer_name = “Lindsey Ferderer”;
```

```
MariaDB [store]> SELECT * FROM Customers
    ->   WHERE customer_name="Everette Bagozzi"
    ->   AND customer_name = "Lindsey Ferderer";
Empty set (0.001 sec)
```

```
SELECT * FROM Customers
  WHERE customer_name=”Everette Bagozzi”
  OR customer_name=”Lindsey Ferderer”;
```

```
MariaDB [store]> SELECT * FROM Customers
    ->   WHERE customer_name="Everette Bagozzi"
    ->   OR customer_name="Lindsey Ferderer";
+-------------+------------------+
| customer_id | customer_name    |
+-------------+------------------+
|           2 | Lindsey Ferderer |
|           5 | Everette Bagozzi |
+-------------+------------------+
2 rows in set (0.000 sec)
```

NOT

Along with AND and OR, we also have the NOT keyword. NOT can be used to let the database know we want results that are **opposite** to our condition. See the following example:

```
SELECT customer_name FROM Customers
  WHERE customer_id=5;
```

```
MariaDB [store]> SELECT customer_name FROM Customers
    ->   WHERE customer_id=5;
+------------------+
| customer_name    |
+------------------+
| Everette Bagozzi |
+------------------+
1 row in set (0.001 sec)
```

```
SELECT customer_name FROM Customers
  WHERE NOT customer_id=5;
```

```
MariaDB [store]> SELECT customer_name FROM Customers
    ->   WHERE NOT customer_id=5;
+------------------+
| customer_name    |
+------------------+
| Max Zets         |
| Lindsey Ferderer |
| Roy Mchenry      |
| Ramon Artzer     |
| Max Zets         |
+------------------+
5 rows in set (0.001 sec)
```

NULL

We can ask for results that contain no (NULL) value in a column. Our Customers table doesn't allow NULL values at this time, as we'd created it with columns that have "NOT NULL" in their description. We can still run an experiment, however, by checking the difference we receive in results checking for NULL versus NOT NULL in our SELECT query.

SELECT * FROM Customers WHERE customer_name IS NULL;

```
MariaDB [store]> SELECT * FROM Customers WHERE customer_name IS NULL;
Empty set (0.017 sec)
```

SELECT * FROM Customers WHERE customer_name IS NOT NULL;

```
MariaDB [store]> SELECT * FROM Customers WHERE customer_name IS NOT NULL;
+-------------+-----------------+
| customer_id | customer_name   |
+-------------+-----------------+
|           1 | Max Zets        |
|           2 | Lindsey Ferderer |
|           3 | Roy Mchenry     |
|           4 | Ramon Artzer    |
|           5 | Everette Bagozzi |
|           6 | Max Zets        |
+-------------+-----------------+
6 rows in set (0.002 sec)
```

About LIKE

The LIKE keyword allows us to look for a pattern when retrieving results. Records found that match the pattern will be added to the list of returned rows for your query.

- _ is a **placeholder** wildcard, e.g. _s means a two letter word where the last letter is ‘s’. You can have as many underscores in a row as you like
- % is an **anything goes** wildcard, e.g. %s means any length word that ends with an s. You don’t need to use multiple percentage signs in a row because they already mean any letter, any length
- \ **escapes** a character so you can search for _ or % without it meaning wildcard e.g. LIKE ‘_s%’ would mean find a string that starts with underscore and s
- You can also escape a character by using the following syntax:

```
LIKE '%$_20%' ESCAPE '$';
```

LIKE

Notice in a query where we ask for an exact match, we have to be careful that the value matches perfectly:

```
-- In this case, the full name will find the match...
SELECT * FROM Customers
  WHERE customer_name='Ramon Artzer';

-- The partial name, in contrast, will not…
SELECT * FROM Customers
  WHERE customer_name='Ramon';
```

```
MariaDB [store]> SELECT * FROM Customers
    -> WHERE customer_name='Ramon Artzer';
+-------------+---------------+
| customer_id | customer_name |
+-------------+---------------+
|           4 | Ramon Artzer  |
+-------------+---------------+
1 row in set (0.011 sec)

MariaDB [store]> SELECT * FROM Customers
    -> WHERE customer_name='Ramon';
Empty set (0.000 sec)
```

Let’s see about using pattern matching to get some alternative results. Try out the following:

```
-- Find names that start with “ramon”.
SELECT * FROM Customers
  WHERE customer_name LIKE ‘ramon%’;

-- Find names that end with “er”.
SELECT * FROM Customers
  WHERE customer_name LIKE ‘%er’;

-- Find names that start with “M”, followed
-- by two characters, a space, and ending
-- with “s”.
SELECT * FROM Customers
  WHERE customer_name LIKE ‘M__ ___s’;
```

```
MariaDB [store]> SELECT * FROM Customers
    -> WHERE customer_name LIKE 'ramon%';
+-------------+---------------+
| customer_id | customer_name |
+-------------+---------------+
|           4 | Ramon Artzer  |
+-------------+---------------+
1 row in set (0.001 sec)

MariaDB [store]> SELECT * FROM Customers
    -> WHERE customer_name LIKE '%er';
+-------------+-----------------+
| customer_id | customer_name   |
+-------------+-----------------+
|           2 | Lindsey Ferderer |
|           4 | Ramon Artzer    |
+-------------+-----------------+
2 rows in set (0.001 sec)

MariaDB [store]> SELECT * FROM Customers
    -> WHERE customer_name LIKE 'M__ ___s';
+-------------+---------------+
| customer_id | customer_name |
+-------------+---------------+
|           1 | Max Zets      |
|           6 | Max Zets      |
+-------------+---------------+
2 rows in set (0.001 sec)
```

LIMIT

LIMIT tells your database how many records you'd like back in the response. Try the examples below:

```
-- Without LIMIT, it returns all results.
SELECT * FROM Customers;

-- With LIMIT, the number of results obey your ask.
SELECT * FROM Customers LIMIT 1;
SELECT customer_name FROM Customers LIMIT 3;
```

In large data sets it can be important to add LIMIT to queries in order to ensure faster performance. This is especially common when used in applications, as you can limit display and retrieval to only the number of records appropriate for a table or list displayed to a user.

```
MariaDB [store]> SELECT * FROM Customers;
+-------------+-----------------+
| customer_id | customer_name   |
+-------------+-----------------+
|           1 | Max Zets        |
|           2 | Lindsey Ferderer |
|           3 | Roy Mchenry     |
|           4 | Ramon Artzer    |
|           5 | Everette Bagozzi |
|           6 | Max Zets        |
+-------------+-----------------+
6 rows in set (0.013 sec)

MariaDB [store]> SELECT * FROM Customers LIMIT 1;
+-------------+---------------+
| customer_id | customer_name |
+-------------+---------------+
|           1 | Max Zets      |
+-------------+---------------+
1 row in set (0.001 sec)

MariaDB [store]> SELECT customer_name FROM Customers LIMIT 5;
+-----------------+
| customer_name   |
+-----------------+
| Max Zets        |
| Lindsey Ferderer |
| Roy Mchenry     |
| Ramon Artzer    |
| Everette Bagozzi |
+-----------------+
5 rows in set (0.001 sec)
```

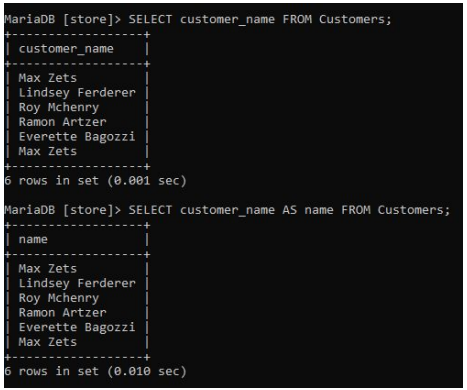
AS

Not as impressive as the previous features we've looked at but this will come in handy later as we explore JOIN and other keywords available to us: the AS keyword. It is used for aliasing a name for use in your results.

Note the difference between the following two outputs:

SELECT customer_name FROM Customers;

SELECT customer_name AS name FROM Customers;

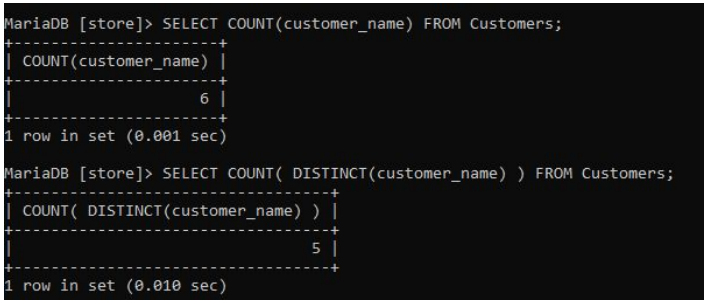


COUNT

Let’s try counting the number of results from our DISTINCT experiments:

SELECT COUNT(customer_name) FROM Customers;

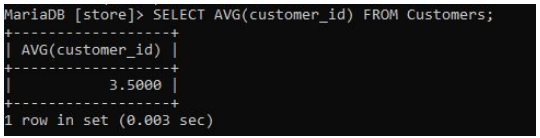
SELECT COUNT(DISTINCT(customer_name)) FROM Customers;



AVG

We can get the average of a group of numbers using AVG. Give it a try with our customers table. Since we have an integer representing our customer_id we can check the average of the entries:

```
SELECT AVG(customer_id) FROM Customers;
```



SUM

SUM can be used to add all returned entries together to calculate the sum of the numbers mathematically. Let's try, again using the customer_id, since that is a number:

SELECT SUM(customer_id) FROM Customers;

```
MariaDB [store]> SELECT SUM(customer_id) FROM Customers;
+------------------+
| SUM(customer_id) |
+------------------+
|               21 |
+------------------+
1 row in set (0.001 sec)
```

MIN and MAX

MIN can be used to find the lowest number, or string (word[s]) lowest alphabetically in your data set. Let's try one of each:

SELECT MIN(customer_id) FROM Customers;

SELECT MIN(customer_name) FROM Customers;

-- Note that we can even check for two conditions at once so long as they
-- are comma-separated...
SELECT MIN(customer_id), MIN(customer_name) FROM Customers;

```
MariaDB [store]> SELECT MIN(customer_id) FROM Customers;
+------------------+
| MIN(customer_id) |
+------------------+
|                1 |
+------------------+
1 row in set (0.010 sec)

MariaDB [store]> SELECT MIN(customer_name) FROM Customers;
+--------------------+
| MIN(customer_name) |
+--------------------+
| Everette Bagozzi   |
+--------------------+
1 row in set (0.001 sec)

MariaDB [store]> SELECT MIN(customer_id), MIN(customer_name) FROM Customers;
+------------------+--------------------+
| MIN(customer_id) | MIN(customer_name) |
+------------------+--------------------+
|                1 | Everette Bagozzi   |
+------------------+--------------------+
1 row in set (0.013 sec)
```

MAX is extremely similar, but, as its name suggests it is used to locate the highest number or the string highest alphabetically:

SELECT MAX(customer_id), MAX(customer_name) FROM Customers;

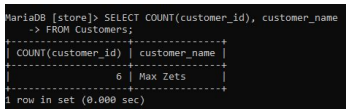
```
MariaDB [store]> SELECT MAX(customer_id), MAX(customer_name) FROM Customers;
+------------------+--------------------+
| MAX(customer_id) | MAX(customer_name) |
+------------------+--------------------+
|                6 | Roy Mchenry        |
+------------------+--------------------+
1 row in set (0.001 sec)
```

```
MariaDB [store]> SELECT * FROM Customers;
+-------------+-----------------+
| customer_id | customer_name   |
+-------------+-----------------+
|           1 | Max Zets        |
|           2 | Lindsey Ferderer |
|           3 | Roy Mchenry     |
|           4 | Ramon Artzer    |
|           5 | Everette Bagozzi |
|           6 | Max Zets        |
+-------------+-----------------+
6 rows in set (0.000 sec)
```

GROUP BY

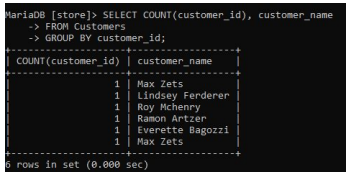
GROUP BY gives us a way to group results if a relationship exists in our selection. For instance, try the following SELECT that takes advantage of the COUNT function:

```
SELECT COUNT(customer_id), customer_name
 FROM Customers;
```



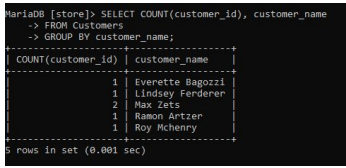
We get the total number of results, and a name… not the most useful combination. Let's try an alternative, with a grouping:

```
SELECT COUNT(customer_id), customer_name
 FROM Customers
 GROUP BY customer_id;
```



Okay, interesting… we are seeing the count, and all the names now. This is at least the full data set, and provides more total information. Let's give it one more go, grouping instead by customer_name:

```
SELECT COUNT(customer_id), customer_name
 FROM Customers
 GROUP BY customer_name;
```



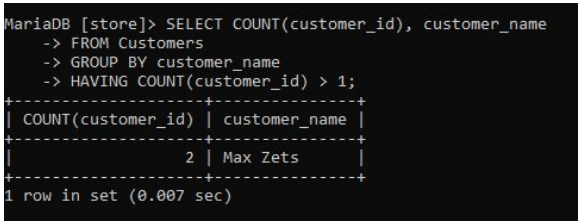
Wow! This time we get a lot more utility out of our result. The customer_name column shows only unique names, and the COUNT column gives us the number of entries with a name that matches!

Experiment with GROUP BY and you'll find many useful applications.

HAVING

To build on GROUP BY and increase its usefulness in more cases, you can add conditions for what should be included in your result set. Let's demonstrate via the following test example:

```
SELECT COUNT(customer_id), customer_name
  FROM Customers
  GROUP BY customer_name
  HAVING COUNT(customer_id) > 1;
```



This will give us a result of only the entries wherein a count of any particular name is greater than one. Consider where this could be applied. Can you think of any potential use-cases?

Imagine a table of orders. You could group by orders with matching customer IDs and an order cost over $100. The possibilities really open up as you experiment with these features.

Recommended Reading

It’s a good idea to read up more on what we’ve covered today. Have a look at the following:

- Beaulieu, A. (March 2020). Learning SQL, 3rd Edition. O’Reilly Media, Inc.
  - Chapter 3. Query Primer
  - Chapter 4. Filtering