

# MySQL Aggregate Functions Tutorial :

## SUM, AVG, MAX, MIN , COUNT, DISTINCT

Aggregate Functions are all about

- Performing calculations on multiple rows
- Of a single column of a table
- And returning a single value.

The ISO standard defines five (5) aggregate functions namely;

1) COUNT

2) SUM

3) AVG

4) MIN

5) MAX

### Why use aggregate functions?

From a business perspective, different organization levels have different information requirements. Top level managers are usually interested in knowing whole figures and not necessarily the individual details.

>Aggregate functions allow us to easily produce summarized data from our database.

For instance, from our myflix database , management may require following reports

- Least rented movies.
- Most rented movies.

- Average number that each movie is rented out in a month.

We easily produce the above reports using aggregate functions.

Let's look into aggregate functions in detail.

## COUNT Function

The COUNT function returns the total number of values in the specified field. It works on both numeric and non-numeric data types. **All aggregate functions by default exclude nulls values before working on the data.**

COUNT (\*) is a special implementation of the COUNT function that returns the count of all the rows in a specified table. COUNT (\*) also considers Nulls and duplicates.

The table shown below shows data in movierentals table

reference_ number	transactio n_date	return_d ate	membership _number	movie_ id	movie_ returne d
11	20-06-2012	NULL	1	1	0
12	22-06-2012	25-06-2012	1	2	0
13	22-06-2012	25-06-2012	3	2	0
14	21-06-2012	24-06-2012	2	2	0

---

15	23-06-2012	NULL	3	3	0
----	------------	------	---	---	---

Let's suppose that we want to get the number of times that the movie with id 2 has been rented out

```
SELECT COUNT(`movie_id`) FROM `movierentals` WHERE  
`movie_id` = 2;
```

Executing the above query in MySQL workbench against myflixdb gives us the following results.

<code>COUNT('movie_id')</code>
--------------------------------

3
---

## DISTINCT Keyword

The DISTINCT keyword that allows us to omit duplicates from our results. This is achieved by grouping similar values together .

To appreciate the concept of Distinct, lets execute a simple query

```
SELECT `movie_id` FROM `movierentals`;
```

<code>movie_id</code>
-----------------------

1
---

2
---

2
---

2

3

Now let's execute the same query with the distinct keyword -

```
SELECT DISTINCT `movie_id` FROM `movierentals`;
```

As shown below , distinct omits duplicate records from the results.

movie_id
----------

1

2

3

## MIN function

The MIN function **returns the smallest value in the specified table field.**

As an example, let's suppose we want to know the year in which the oldest movie in our library was released, we can use MySQL's MIN function to get the desired information.

The following query helps us achieve that

```
SELECT MIN(`year_released`) FROM `movies`;
```

Executing the above query in MySQL workbench against myflixdb gives us the following results.

MIN('year_released')
----------------------

---

2005

## MAX function

Just as the name suggests, the MAX function is the opposite of the MIN function. It **returns the largest value from the specified table field**.

Let's assume we want to get the year that the latest movie in our database was released. We can easily use the MAX function to achieve that.

The following example returns the latest movie year released.

```
SELECT MAX(`year_released`) FROM `movies`;
```

Executing the above query in MySQL workbench using myflixdb gives us the following results.

<b>MAX('year_released')</b>
-----------------------------

2012

## SUM function

Suppose we want a report that gives the total amount of payments made so far. We can use the MySQL **SUM** function which **returns the sum of all the values in the specified column. SUM works on numeric fields only. Null values are excluded from the result returned.**

The following table shows the data in payments table-

payment_id	membership_number	payment_date	description	amount_paid	external_reference_number
1	1	23-07-2012	Movie rental payment	2500	11
2	1	25-07-2012	Movie rental payment	2000	12
3	3	30-07-2012	Movie rental payment	6000	NULL

The query shown below gets all payments made and sums them up to return a single result.

```
SELECT SUM(`amount_paid`) FROM `payments`;
```

Executing the above query in MySQL workbench against the myflixdb gives the following results.

<b>SUM('amount_paid')</b>
---------------------------

10500

## AVG function

MySQL AVG function **returns the average of the values in a specified column**. Just like the SUM function, it **works only on numeric data types**.

Suppose we want to find the average amount paid. We can use the following query -

```
SELECT AVG(`amount_paid`) FROM `payments`;
```

Executing the above query in MySQL workbench, gives us the following results.

<b>AVG('amount_paid')</b>
---------------------------

3500

## Summary

- MySQL supports all the five (5) ISO standard aggregate functions COUNT, SUM, AVG, MIN and MAX.
- SUM and AVG functions only work on numeric data.
- If you want to exclude duplicate values from the aggregate function results, use the DISTINCT keyword. The ALL keyword includes even duplicates. If nothing is specified the ALL is assumed as the default.
- Aggregate functions can be used in conjunction with other SQL clauses such as GROUP BY

## Brain Teaser

You think aggregate functions are easy. Try this!

The following example groups members by name, counts the total number of payments, the average payment amount and the grand total of the payment amounts.

```
SELECT m.`full_names`, COUNT(p.`payment_id`) AS  
`paymentscount`, AVG(p.`amount_paid`) AS  
`averagepaymentamount`, SUM(p.`amount_paid`) AS  
`totalpayments` FROM members m, payments p WHERE  
m.`membership_number` = p.`membership_number` GROUP BY  
m.`full_names`;
```

Executing the above example in MySQL workbench gives us the following results.

	full_names	paymentscount	averagepaymentamount	totalpayments
▶	Janet Jones	2	2250	4500
	Robert Phil	1	6000	6000