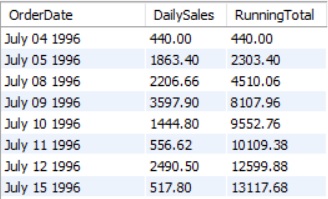
**How to generate Cumulative Sum (running total) by MySQL - Part 1 (use northwind database)**

Cumulative sum, also known as running total or Year to Date (YTD) total, is a sum of data that is related to a time sequence such as daily, monthly, or yearly. But running total could also be calculated if the dataset does not have a time sequence related column.

For example, below is a list of daily sales and running total.



For calculating running total in SQL, at a high level, the concept includes (1) rank data within a GROUP, (2) RANK BY a column, and (3) calculate running total by using SUM.

For the daily sales illustrated above, the GROUP is the entire Northwind Trader company. The RANK BY column is OrderDate. SUM is applied to DailySales. To translate the calculation in plain language, we can say that we want to RANK daily total sales BY order date within Northwind Trader and calculate running total for each order date.

It may not look too obvious what the GROUP is. To better understand the concept, consider we have 2 Northwind Traders - Northwind A and Northwind B. So for each Northwind company, we want to RANK daily total sales BY order date within each Northwind Trader and calculate running total for each order date. Then the running total is calculated within each Northwind - Northwind A and Northwind B.

To generate a running total (or cumulative sum), please consider the following guidelines.

1. Use self join to join the table (or base dataset) to itself.
2. Find a naturally available RANK value column that sequentially increments over time. If such a value does not exist, use [RANK()](https://www.geeksengine.com/database/single-row-functions/rank.php), [DENSE\_RANK()](https://www.geeksengine.com/database/single-row-functions/dense-rank.php), or [ROW\_NUMBER()](https://www.geeksengine.com/database/single-row-functions/row-number.php) function to create a RANK value.
3. The rank value must be unique within the GROUP.
4. In the self join clause, the RANK value in the left table (t1) is always coded greater than or equal to the RANK value in the right table (t2), e.g. t1.RankValue >= t2.RankValue.
5. The running total is calculated by summing the total column in the right table (t2).

Now let's use two practical examples to see how to generate running totals.

**Practice 1: rank within a single GROUP which is the entire Northwind company**

1.1 Create the basic dataset.

The result of the basic dataset is the table we want to calculate the running total for.

-- Basic set

select b.OrderID,

b.OrderDate,

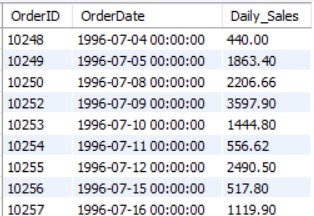
round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as Daily\_Sales

from Order\_Details as a

inner join Orders b on a.OrderID = b.OrderID

group by b.OrderDate

order by b.OrderDate;

480 records returned.  


1.2 Generate running total by self join

To calculate running total, join the above query to itself. Please note that, the join column must be a column that can uniquely rank every record by a time sequence. Here we used t1.OrderDate >= t2.OrderDate. If we use t1.OrderID >= t2.OrderID, it'll produce the same result because OrderID is sequentially incremented as time goes by.

select DATE\_FORMAT(t1.OrderDate, "%M %d %Y") as OrderDate,

t1.Daily\_Sales,

sum(t2.Daily\_Sales) as RunningTotal

from

(

select b.OrderID,

b.OrderDate,

round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as Daily\_Sales

from Order\_Details as a

inner join Orders b on a.OrderID = b.OrderID

group by b.OrderDate

) as t1

inner join -- this is self join

(

select b.OrderID,

b.OrderDate,

round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as Daily\_Sales

from Order\_Details as a

inner join Orders b on a.OrderID = b.OrderID

group by b.OrderDate

) as t2

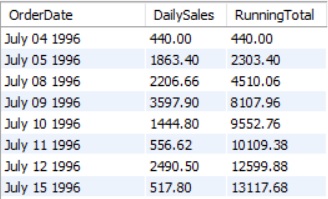
on t1.OrderDate >= t2.OrderDate

-- this should also work

-- on t1.OrderID >= t2.OrderID

group by t1.OrderDate

order by t1.OrderDate;

480 records returned.  


Below is equivalent version as above.

To simplify the two queries above, we can create a physical table by using the basic dataset, then work out the running total from this new table.

-- Create a new table from the basic set.

create table NorthwindSales

as

select b.OrderID,

b.OrderDate,

round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as Daily\_Sales

from Order\_Details as a

inner join Orders b on a.OrderID = b.OrderID

group by b.OrderDate

order by b.OrderDate;

-- Work out the running total from the new table.

select DATE\_FORMAT(t1.OrderDate, "%M %d %Y") as OrderDate,

t1.Daily\_Sales,

sum(t2.Daily\_Sales) as RunningTotal

from NorthwindSales as t1

inner join NorthwindSales as t2

on t1.OrderDate>=t2.OrderDate

group by t1.OrderDate

order by t1.OrderDate;

**Practice 2: rank within multiple GROUPs which are the product categories**

2.1 Create the basic dataset.

-- Basic set

select c.CategoryName,

year(b.OrderDate) as OrderYear,

round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as YearlySales

from Order\_Details as a

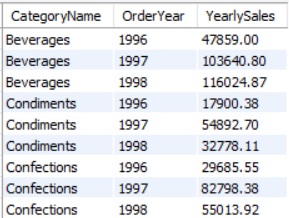
inner join Orders b on a.OrderID = b.OrderID

inner join Products as p on a.ProductID=p.ProductID

inner join Categories as c on c.CategoryID=p.CategoryID

group by c.CategoryName,year(b.OrderDate)

order by c.CategoryName, OrderYear;

24 records returned.  


2.2 Generate running total

Within each product category, calculate running total for each order year. Please note that OrderYear is a naturally available rank value that orders YearlSales for each year, so we code it in the join clause like this t1.OrderYear >= t2.OrderYear. If we need to calculate a running total where a naturally available rank value does not exist, we'll have to use MySQL RANK(), DENSE\_RANK(), or ROW\_NUMBER() function to create the rank value.

select t1.CategoryName,

t1.OrderYear,

t1.Yearly\_Sales,

sum(t2.Yearly\_Sales) as RunningTotal

from

(

select c.CategoryName,

year(b.OrderDate) as OrderYear,

round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as Yearly\_Sales

from Order\_Details as a

inner join Orders b on a.OrderID = b.OrderID

inner join Products as p on a.ProductID=p.ProductID

inner join Categories as c on c.CategoryID=p.CategoryID

group by c.CategoryName,year(b.OrderDate)

) as t1

inner join

(

select c.CategoryName,

year(b.OrderDate) as OrderYear,

round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as Yearly\_Sales

from Order\_Details as a

inner join Orders b on a.OrderID = b.OrderID

inner join Products as p on a.ProductID=p.ProductID

inner join Categories as c on c.CategoryID=p.CategoryID

group by c.CategoryName,year(b.OrderDate)

) as t2

on t1.CategoryName=t2.CategoryName and t1.OrderYear >= t2.OrderYear

group by t1.CategoryName,t1.OrderYear

order by t1.CategoryName,t1.OrderYear;

24 records returned.  


Below is equivalent version as above.

-- This is equivalent version but create a new table for the basic dataset

create table NorwindSales2

as

select c.CategoryName,

year(b.OrderDate) as OrderYear,

round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as YearlySales

from Order\_Details as a

inner join Orders b on a.OrderID = b.OrderID

inner join Products as p on a.ProductID=p.ProductID

inner join Categories as c on c.CategoryID=p.CategoryID

group by c.CategoryName,year(b.OrderDate)

order by c.CategoryName, OrderYear;

-- Calculate running total

select t1.CategoryName,

t1.OrderYear,

t1.YearlySales,

sum(t2.YearlySales) as RunningTotal

from NorwindSales2 as t1

inner join NorwindSales2 as t2

on t1.CategoryName=t2.CategoryName and t1.OrderYear >= t2.OrderYear

group by t1.CategoryName,t1.OrderYear

order by t1.CategoryName,t1.OrderYear;

**How to generate Cumulative Sum (running total) by MySQL - Part 2**

In Part 1, we have shown how to calculate running totals (aka cumulative sums) by using self join. In the join clause, we use a naturally available rank value where the left table (t1) is always coded greater than or equal to the RANK value in the right table (t2), e.g. t1.RankValue >= t2.RankValue.

In Part 2 of this tutorial, we'll show how to deliberately create a RANK value by using MySQL's ranking functions such as RANK(), DENSE\_RANK(), or ROW\_NUMBER(). Learning this technique is important because a natural rank value is not always available. In such a case, the only way is to calculate and then assign rank values to each group within which the running total is worked out.

**Practice 1: Use RANK() function to create a rank value**

The queries below are a rewrite of Practice 2 in Part 1 of this tutorial. The only difference is that RANK() function is added to create a rank value even though OrderYear can naturally act as a rank value.

1.1 Build the basis set

-- This is the basic set. Here a rank value is created within each product category

-- by using RANK() function.

select c.CategoryName,

year(b.OrderDate) as OrderYear,

round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as YearlySales,

RANK() OVER (PARTITION BY c.CategoryName ORDER BY b.OrderDate) as RankValue

from Order\_Details as a

inner join Orders b on a.OrderID = b.OrderID

inner join Products as p on a.ProductID=p.ProductID

inner join Categories as c on c.CategoryID=p.CategoryID

group by c.CategoryName,year(b.OrderDate)

order by c.CategoryName, OrderYear;

24 records returned.  


1.2 Use RANK() function to calculate running total

-- Calculate running total. Rank value is used in the join clause.

select t1.CategoryName,

t1.OrderYear,

t1.YearlySales,

sum(t2.YearlySales) as RunningTotal

from

(

select c.CategoryName,

year(b.OrderDate) as OrderYear,

round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as YearlySales,

RANK() OVER (PARTITION BY c.CategoryName ORDER BY b.OrderDate) as RankValue

from Order\_Details as a

inner join Orders b on a.OrderID = b.OrderID

inner join Products as p on a.ProductID=p.ProductID

inner join Categories as c on c.CategoryID=p.CategoryID

group by c.CategoryName,year(b.OrderDate)

) as t1 -- derived table

inner join

(

select c.CategoryName,

year(b.OrderDate) as OrderYear,

round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as YearlySales,

RANK() OVER (PARTITION BY c.CategoryName ORDER BY b.OrderDate) as RankValue

from Order\_Details as a

inner join Orders b on a.OrderID = b.OrderID

inner join Products as p on a.ProductID=p.ProductID

inner join Categories as c on c.CategoryID=p.CategoryID

group by c.CategoryName,year(b.OrderDate)

) as t2

on t1.CategoryName=t2.CategoryName and t1.RankValue >= t2.RankValue

group by t1.CategoryName,t1.OrderYear

order by t1.CategoryName,t1.OrderYear;

24 records returned.  


**Practice 2: Use RANK() function when a natural rank value is not available**

In the query below, we want to calculate sales running total based on employee names within each product. Because a natural rank value is not available, we have to use RANK() function to produce the rank value within each product (this is the GROUP) based on employee names in alphabetical order.

2.1 Load the basic set into a physical MySQL table

-- Create the base table

create table ProductSalesByEmployee

as

select p.ProductName,

e.FirstName as EmployeeName,

round(sum(a.UnitPrice\*a.Quantity\*(1-a.Discount)),2) as Sales,

RANK() OVER (PARTITION BY p.ProductName ORDER BY e.FirstName) as RankValue

from Order\_Details as a

inner join Orders b on a.OrderID = b.OrderID

inner join products p on a.ProductID=p.ProductID

inner join Employees e on e.EmployeeID=b.EmployeeID

group by p.ProductName,e.FirstName

order by p.ProductName,e.FirstName;

2.2 Use RANK() function to calculate running total because a natural ranking column is not available

-- Calculate running total

select t1.ProductName,

t1.EmployeeName,

t1.Sales,sum(t2.Sales) as RunningTotal

from ProductSalesByEmployee as t1

inner join ProductSalesByEmployee as t2 on t1.ProductName=t2.ProductName and t1.RankValue >= t2.RankValue

group by t1.ProductName,t1.EmployeeName

order by t1.ProductName,sum(t2.Sales);

589 records returned.  
