

## Rolling Sums in SQL - A Practical Example

Today, Lenz was putting together some stats on PlanetMySQL feeds added since January this year, and asked in an email whether he should include totals in the stats. I responded yes, and offered a quick SQL solution to get those numbers out of the database. I thought it might be useful for others, so here goes...

Rolling sums (or averages) are a way to include a running, or rolling, aggregate of certain columns in the output of your SQL. For instance, let's say you want to list order totals, grouped by product category, but in each row of the final output, you wish to include the running total of all rows in the output up to and including the current row. A running sum would allow you to do such a calculation.

In the specific example that came up today, Lenz had the following SQL statement which listed the month the feed was submitted, and the number of feeds in that month: Listing 1:

```
-----
SELECT
  MONTHNAME(created) AS Month
, COUNT(*) AS Added
FROM feeds
WHERE created >= '2007-01-01'
GROUP BY MONTH(created);
```

```
+-----+-----+
| Month | Added |
+-----+-----+
| January | 1 |
| February | 1 |
| March | 11 |
| April | 8 |
| May | 18 |
| June | 3 |
+-----+-----+
```

6 rows in set (0.00 sec)

Of course, this output is fine, but what if we want to show the total number of feeds in the Planet, along with the number of new feeds in that month? Here, a running sum comes into play. To collect a running sum, a technique which joins a resultset to itself via a range predicate is used. Essentially, you want to join a table (or grouped set) against itself along a great than or equals join condition. The basic setup goes like this: Listing 2:

```
-----
SELECT
  x1.key
, x1.some_column
, AGGREGATE_FN(x2.some_column) AS running_aggregate
FROM x AS x1
INNER JOIN x AS x2
ON x1.key >= x2.key
GROUP BY x1.key;
```

By joining the set against itself along the range predicate, the aggregation on the second set of some\_column (AGGREGATE\_FN(x2.some\_column)) only includes those rows up to and including the "current" row in the primary output.

To get back to our real-world PlanetMySQL example, we're going to deal with the only thing that makes running sums tricky: when you are generating running sums on already aggregated sets. In Lenz' original code, you will notice we are already aggregating by MONTH(created). The astute among you will have already seen the problem when trying to generate a running sum against the already-grouped set of information &mdash; you cannot have two GROUP BY expressions in a single SELECT. Meaning, if we are to follow the basic setup in Listing 2, we might get something like this: Listing 3:

```
-----
SELECT
  MONTHNAME(x1.created) AS Month
, COUNT(x1.*) AS Added
, SUM(COUNT(x2.*)) AS running_sum
FROM feeds x1
```

```

INNER JOIN feeds x2
ON x1.Month >= x2.Month
WHERE x1.created >= '2007-01-01'
GROUP BY MONTH(x2.created)
GROUP BY MONTH(x1.created);

```

Of course, Listing 3 doesn't make any sense from an SQL point of view. You obviously can't have more than one level of aggregation (GROUP BY) in a single SELECT expression...

So, how do we get around this problem? By using derived tables, or subqueries in the FROM clause. Using Subqueries in the FROM Clause to Do Double Aggregation

Subqueries in the FROM clause are sometimes referred to as virtual or derived tables. They represent a set of information, in its entirety, that can be joined and SELECTed from just like a normal table. To make a subquery in the FROM clause, simply surround a SELECT expression with parentheses and give an alias, or name, for the derived table. Going back to our example, this is the set of information we eventually wish to join to itself via a range predicate:

```

SELECT
  MONTHNAME(created) AS Month
, COUNT(*) AS Added
FROM feeds
WHERE created >= '2007-01-01'
GROUP BY MONTH(created);

```

However, as previously stated, since the above SELECT already contains a GROUP BY, we will have to use a derived table in order to generate a running sum. To do this, we'll build the complex SQL statement piece by piece. First, let's surround the SELECT expression above with parentheses and alias it: Listing 4:

```

-----
(
SELECT
  MONTH(created) AS MonthNo
, MONTHNAME(created) AS MonthName
, COUNT(*) AS Added
FROM feeds
WHERE created >= '2007-01-01'
GROUP BY MONTH(created)
) AS x1

```

Next, following the general formula laid out in Listing 2, we want to create a second, identical set in order to join the first set (from listing 4) to it. So, we use another derived table and alias it x2 this time: Listing 5:

```

-----
(
SELECT
  MONTH(created) AS MonthNo
, MONTHNAME(created) AS MonthName
, COUNT(*) AS Added
FROM feeds
WHERE created >= '2007-01-01'
GROUP BY MONTH(created)
) AS x2

```

Next, we want to apply the join condition. To do so, let's block out an outer SELECT expression, and a FROM clause containing the two derived tables and a join condition. In pseudo-code, something like this: Listing 6:

```

-----
SELECT ...
FROM
( x1 SELECT... ) AS x1
INNER JOIN ( x2 SELECT ) AS x2
ON x1.key >= x2.key
GROUP BY x1.key;

```

By writing the SQL in pseudo-code like Listing 6, you get a better overall idea of how all the pieces fit together, without getting bogged down in the longer, more complex SQL expression. The only things left to do from here are:

- Include the SELECT fields in the outer SELECT and replace "key" with the real field name
- Replace the pseudo-code placeholders for the derived table SELECT expressions with the real SQL

In order, let's do that. First, let's include the SELECT fields in the outer SELECT: Listing 7:

```
-----
SELECT
  x1.MonthNo
, x1.MonthName
, x1.Added
, SUM(x2.Added) AS RunningTotal
FROM
  ( x1 SELECT... ) AS x1
INNER JOIN ( x2 SELECT ) AS x2
ON x1.MonthNo >= x2.MonthNo
GROUP BY x1.MonthNo;
```

I bolded the changed code in Listing 7 to highlight that the outer SELECT field list should contain the key field from x1 (MonthNo), the Added field from x1 (which contains the number of feeds in that month), and the SUM of the x2 set's value of Added, which contains the number of feeds for all months up to and including the month in x1. In addition, I also included a dependent (on MonthNo) field, MonthName, from x1 for decorative purposes.

The final step is to replace the derived table placeholders with the actual SQL: Listing 8:

```
-----
SELECT
  x1.MonthNo
, x1.MonthName
, x1.Added
, SUM(x2.Added) AS RunningTotal
FROM
  (
  SELECT
    MONTH(created) AS MonthNo
  , MONTHNAME(created) AS MonthName
  , COUNT(*) AS Added
  FROM feeds
  WHERE created >= '2007-01-01'
  GROUP BY MONTH(created)
  ) AS x1
INNER JOIN (
  SELECT
    MONTH(created) AS MonthNo
  , MONTHNAME(created) AS MonthName
  , COUNT(*) AS Added
  FROM feeds
  WHERE created >= '2007-01-01'
  GROUP BY MONTH(created)
  ) AS x2
ON x1.MonthNo >= x2.MonthNo
GROUP BY x1.MonthNo;
```

All done. Now we have a running sum: Listing 9:

```
-----
+-----+-----+-----+-----+
| MonthNo | MonthName | Added | RunningTotal |
+-----+-----+-----+-----+
| 1 | January | 1 | 1 |
| 2 | February | 1 | 2 |
| 3 | March | 11 | 13 |
| 4 | April | 8 | 21 |
| 5 | May | 18 | 39 |
| 6 | June | 3 | 42 |
```

```
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

There's only one little problem in the query...it doesn't actually show the total number of feeds in PlanetMySQL, only the ones that were added in 2007. What if we want to show the total number of feeds? Easy. We can use a user variable containing the number of feeds not added in 2007 and add that to the running total. Here is the final SQL and result. Hope this helps someone out! Listing 10:

```
-----
SELECT COUNT(*) INTO @2006total FROM feeds WHERE created < '2007-01-01';
```

```
SELECT
  x1.MonthNo
, x1.MonthName
, x1.Added
, SUM(x2.Added) + @2006total AS RunningTotal
FROM
```

```
(
SELECT
  MONTH(created) AS MonthNo
, MONTHNAME(created) AS MonthName
, COUNT(*) AS Added
FROM feeds
WHERE created >= '2007-01-01'
GROUP BY MONTH(created)
) AS x1
INNER JOIN (
```

```
SELECT
  MONTH(created) AS MonthNo
, MONTHNAME(created) AS MonthName
, COUNT(*) AS Added
FROM feeds
WHERE created >= '2007-01-01'
GROUP BY MONTH(created)
) AS x2
ON x1.MonthNo >= x2.MonthNo
GROUP BY x1.MonthNo;
```

```
Final Results:
mysql> SELECT @2006total;
```

```
+-----+
| @2006total |
+-----+
| 159       |
+-----+
```

```
1 row in set (0.00 sec)
```

```
<snip>
```

```
+-----+-----+-----+-----+
| MonthNo | MonthName | Added | RunningTotal |
+-----+-----+-----+-----+
| 1 | January | 1 | 160 |
| 2 | February | 1 | 161 |
| 3 | March | 11 | 172 |
| 4 | April | 8 | 180 |
| 5 | May | 18 | 198 |
| 6 | June | 3 | 201 |
+-----+-----+-----+-----+
```

```
6 rows in set (0.00 sec)
```