Table of Contents

Running totals and moving totals/aggregates are two types of queries that are often used in business situations. In SQL there is generally more than one way to accomplish a task and this shows you several approaches to solve problems. Several of these use subqueries and self joins that are not the regular equality join that we have been using over the semester. You can find a variety of techniques for this posted on the internet and these techniques may be very slow for large tables. This may be a situation where, if the tables are large, that you should take the data into an application program that is more efficient for this processing.

In the database world we think of a table as a structure where the ordering of the rows is not significant; with a running total or moving average, the order of the rows is significant. So these techniques will seem somewhat contrived in SQL.

1. Running Total

A running total is a value that totals some value on its row and on the preceding rows. The first row would show the total of some attribute of row 1; the second row the total of row 1 to row 2; the third row the total of row 1 to row 3, etc

Suppose your first 6 assignment score were: 30,27, 30, 20, 30, 24; the third column is a running total of your scores; the fourth column is the running average

Assignment	Score	Running Total	Running Average
1	30	30	100%
2	27	57	95%
3	30	87	97%
4	20	107	89%
5	30	137	91%
6	24	161	89%

1.1. Using a variable: This is a MySQL trick

In these demos, we are calculating a running total of salaries for our table.

This version uses a variable to hold and calculate the total. If you use this approach, be certain to reset the variable after the query if you might run the query a second time in this session. With session level variables, you always need to consider if they have had a value set during the session- particularly when you use variables with names like @num, @x, @total.

In some situations, you will not be able to maintain the value of the variable and this technique will not work- but it is simple and shows you the interactions between the query proper and the session variables.

Demo 01: Doing a running total of salaries, ordered by the department and employee id

```
set @running_total=0;

select emp_id, dept_id, salary
     , @running_total := @running_total + salary as RunningTotal
from  a_emp.adv_emp
order by dept_id, emp_id;
```

```
+--------+---------+--------+--------------+
| emp_id | dept_id | salary | RunningTotal |
+--------+---------+--------+--------------+
|    100 |      10 |  24000 |        24000 |
|    201 |      20 |  15000 |        39000 |
|    101 |      30 |  98005 |       137005 |
|    108 |      30 |  12000 |       149005 |
|    109 |      30 |  15000 |       164005 |
|    110 |      30 |  30300 |       194305 |
|    203 |      30 |  44450 |       238755 |
|    204 |      30 |  15000 |       253755 |
|    205 |      30 |  15000 |       268755 |
|    206 |      30 |  88954 |       357709 |
|    162 |      35 |  98000 |       455709 |
|    200 |      35 |  65000 |       520709 |
|    207 |      35 |  65000 |       585709 |
|    145 |      80 |  65000 |       650709 |
|    150 |      80 |   6500 |       657209 |
|    155 |      80 |  80000 |       737209 |
|    103 |     210 |   9000 |       746209 |
|    104 |     210 |  50000 |       796209 |
|    102 |     215 |  30300 |       826509 |
|    146 |     215 |  88954 |       915463 |
|    160 |     215 |  15000 |       930463 |
|    161 |     215 |  15000 |       945463 |
+--------+---------+--------+--------------+
22 rows in set (0.38 sec)
```

Demo 02: Doing a running total of salaries, ordered by the salary

```
set @accum=0;

select  emp_id, dept_id, salary
      , @accum := @accum + salary as RunningTotal
from a_emp.adv_emp
order by salary, emp_id;
+--------+---------+--------+--------------+
| emp_id | dept_id | salary | RunningTotal |
+--------+---------+--------+--------------+
|    150 |      80 |   6500 |         6500 |
|    103 |     210 |   9000 |        15500 |
|    108 |      30 |  12000 |        27500 |
|    161 |     215 |  15000 |        42500 |
|    160 |     215 |  15000 |        57500 |
|    205 |      30 |  15000 |        72500 |
|    109 |      30 |  15000 |        87500 |
|    201 |      20 |  15000 |       102500 |
|    204 |      30 |  15000 |       117500 |
|    100 |      10 |  24000 |       141500 |
|    110 |      30 |  30300 |       171800 |
|    102 |     215 |  30300 |       202100 |
|    203 |      30 |  44450 |       246550 |
|    104 |     210 |  50000 |       296550 |
|    200 |      35 |  65000 |       361550 |
|    207 |      35 |  65000 |       426550 |
|    145 |      80 |  65000 |       491550 |
|    155 |      80 |  80000 |       571550 |
```

```
|    146 |     215 |  88954 |       660504 |
|    206 |      30 |  88954 |       749458 |
|    162 |      35 |  98000 |       847458 |
|    101 |      30 |  98005 |       945463 |
+--------+---------+--------+--------------+
22 rows in set (0.00 sec)
```

Suppose we want a running total but we want less details and we want to see only department salary totals. This uses a subquery that assembles the data to be used and an outer query to do the aggregate.

Demo 03:

```
set  @total := 0;

select dept_id, DeptTotal
,  @total := @total + DeptTotal as RunningTotal
from (select  dept_id, sum(salary)  as DeptTotal
      from a_emp.adv_emp
      group by dept_id
      ) dt
order by dept_id;
+---------+-----------+--------------+
| dept_id | DeptTotal | RunningTotal |
+---------+-----------+--------------+
|      10 |     24000 |        24000 |
|      20 |     15000 |        39000 |
|      30 |    318709 |       357709 |
|      35 |    228000 |       585709 |
|      80 |    151500 |       737209 |
|     210 |     59000 |       796209 |
|     215 |    149254 |       945463 |
+---------+-----------+--------------+
7 rows in set (0.02 sec)
```

1.2. Using a join

If we do not want to use the variable, we can do a join. Note that the table is placed in the From clause twice- this is a self join. The join uses the <= operator.

One copy of the table is used to display the first three columns and the second copy of the table is used to calculate the running total.

Demo 04: use a self join; this may be slow with large tables

```
select emp_1.emp_id
, emp_1.salary
, sum(emp_2.salary) as RunningTotal
from   a_emp.adv_emp emp_1
join   a_emp.adv_emp emp_2 on emp_2.emp_id <= emp_1.emp_id
group by emp_1.emp_id
order by emp_1.emp_id
;
+--------+--------+--------------+
| emp_id | salary | RunningTotal |
+--------+--------+--------------+
|    100 |  24000 |        24000 |
|    101 |  98005 |       122005 |
```

```
|    102 |  30300 |       152305 |
|    103 |   9000 |       161305 |
|    104 |  50000 |       211305 |
|    108 |  12000 |       223305 |
|    109 |  15000 |       238305 |
|    110 |  30300 |       268605 |
|    145 |  65000 |       333605 |
|    146 |  88954 |       422559 |
|    150 |   6500 |       429059 |
|    155 |  80000 |       509059 |
|    160 |  15000 |       524059 |
|    161 |  15000 |       539059 |
|    162 |  98000 |       637059 |
|    200 |  65000 |       702059 |
|    201 |  15000 |       717059 |
|    203 |  44450 |       761509 |
|    204 |  15000 |       776509 |
|    205 |  15000 |       791509 |
|    206 |  88954 |       880463 |
|    207 |  65000 |       945463 |
+--------+--------+--------------+
22 rows in set (0.03 sec)
```

Going for the department salary total as the basis for the running total

```
select dt_1.dept_id,  dt_1.DeptTotal
     , sum(dt_2.DeptTotal) as RunningTotal
from   (select  dept_id, sum(salary)  as DeptTotal
        from a_emp.adv_emp
        group by dept_id
       ) dt_1
join   (select  dept_id, sum(salary)  as DeptTotal
        from a_emp.adv_emp
        group by dept_id
       ) dt_2 on dt_2.dept_id <= dt_1.dept_id
group by dt_1.dept_id
order by dt_1.dept_id
;
```

1.3. Alternate approach

Demo 05: use a correlated subquery; this may be slow with large tables

```
select  emp_1.emp_id, emp_1.salary
,    (select sum(emp_2.salary)
      from a_emp.adv_emp as Emp_2
      where emp_2.emp_id <= emp_1.emp_id) as RunningTotal
from a_emp.adv_emp emp_1
group by emp_1.emp_id
order by emp_1.emp_id
;
+--------+--------+--------------+
| emp_id | salary | RunningTotal |
+--------+--------+--------------+
|    100 |  24000 |        24000 |
|    101 |  98005 |       122005 |
|    102 |  30300 |       152305 |
|    103 |   9000 |       161305 |
```

```
|    104 |  50000 |       211305 |
|    108 |  12000 |       223305 |
|    109 |  15000 |       238305 |
|    110 |  30300 |       268605 |
|    145 |  65000 |       333605 |
|    146 |  88954 |       422559 |
|    150 |   6500 |       429059 |
|    155 |  80000 |       509059 |
|    160 |  15000 |       524059 |
|    161 |  15000 |       539059 |
|    162 |  98000 |       637059 |
|    200 |  65000 |       702059 |
|    201 |  15000 |       717059 |
|    203 |  44450 |       761509 |
|    204 |  15000 |       776509 |
|    205 |  15000 |       791509 |
|    206 |  88954 |       880463 |
|    207 |  65000 |       945463 |
+--------+--------+--------------+
```

2. Moving Aggregates

This uses the adv_sales table. That table has two attributes- the first is a date that runs in sequence. This represents the sales day in some time span. The second attribute is the total sales for that day.

These are the first few rows; with rows for the last 6 days of April and for each day in May

```
+------------+-------+
| sales_day  | sales |
+------------+-------+
| 2011-04-25 |   400 |
| 2011-04-26 |   400 |
| 2011-04-27 |   400 |
| 2011-04-28 |   300 |
| 2011-04-29 |   900 |
| 2011-04-30 |   580 |
| 2011-05-01 |   425 |
| 2011-05-02 |    10 |
| 2011-05-03 |   325 |
| 2011-05-04 |   500 |
+------------+-------+
```

What we want is a three day sales total. The first total is for days 1, 2, 3 in the table = 400 + 400 + 400 = 1200; the second total is for days 2, 3, 4 = 400 + 400 + 300 = 1100; the third total is for days 3, 4, 5 = 400 + 300+ 900 = 1600. Commonly this is done for the average sales to even out ups and downs in the data that might not be significant as a trend. I am using Sum because it is easier to calculate in your head to see that this is working correctly and we don't have to worry about rounding.

This uses the self join technique similar to that used above where we have two copies of the table and we do the join- not as an equals join but as a join to get the right days linked together. We can do this as shown here so that when we calculate the Sum we get the day we are looking at in a1 and that day and the next two days in a2.

```
from a_oe.adv_sales a1
join a_oe.adv_sales a2
  on a2.sales_day between  a1.sales_day and
                           date_add(a1.sales_day ,  interval 2 day)
```

We want to display the day, its sales and the three-day sum.

Demo 06: Note where we use table alias a1 and where table alias a2

```
select   a1.sales_day
,        a1.sales
,        sum(a2.sales) as three_day_sum
from     a_oe.adv_sales a1
join     a_oe.adv_sales a2
    on   a2.sales_day between  a1.sales_day and  date_add(a1.sales_day ,
interval 2 day)
group by a1.sales_day, a1.sales
order by a1.sales_day
;
+------------+-------+---------------+
| sales_day  | sales | three_day_sum |
+------------+-------+---------------+
| 2011-04-25 |   400 |          1200 |
| 2011-04-26 |   400 |          1100 |
| 2011-04-27 |   400 |          1600 |
| 2011-04-28 |   300 |          1780 |
| 2011-04-29 |   900 |          1905 |
| 2011-04-30 |   580 |          1015 |
| 2011-05-01 |   425 |           760 |
| 2011-05-02 |    10 |           835 |
| 2011-05-03 |   325 |          1375 |
| 2011-05-04 |   500 |          1050 |
| 2011-05-05 |   550 |           550 |
| 2011-05-06 |     0 |           200 |
| 2011-05-07 |     0 |           650 |
| 2011-05-08 |   200 |          1230 |
| 2011-05-09 |   450 |          1455 |
| 2011-05-10 |   580 |          1480 |
| 2011-05-11 |   425 |          1275 |
| 2011-05-12 |   475 |          1350 |
| 2011-05-13 |   375 |          1525 |
| 2011-05-14 |   500 |          1700 |
| 2011-05-15 |   650 |          1650 |
| 2011-05-16 |   550 |          1500 |
| 2011-05-17 |   450 |          1525 |
| 2011-05-18 |   500 |          1525 |
| 2011-05-19 |   575 |          1525 |
| 2011-05-20 |   450 |          1525 |
| 2011-05-21 |   500 |          1925 |
| 2011-05-22 |   575 |          1925 |
| 2011-05-23 |   850 |          1925 |
| 2011-05-24 |   500 |          1575 |
| 2011-05-25 |   575 |          1650 |
| 2011-05-26 |   500 |          1575 |
| 2011-05-27 |   575 |          1650 |
| 2011-05-28 |   500 |          1650 |
| 2011-05-29 |   575 |          1725 |
| 2011-05-30 |   575 |          1150 |
| 2011-05-31 |   575 |           575 |
+------------+-------+---------------+
37 rows in set (0.00 sec)
```

This makes sense for the rows up to the last two- The sum for 2011-05-29 is for May 29, 30 and 31- which is three days

But the sum for 2011-05-30 is for two days only and sum for 2011-05-31 is for only one day. The table ends with May31 ; there are no rows for June 1 or 2. We might want to modify the query to skip the last two days. We do not want to write a where clause that says Where sales_day not in ('2011-05-31, '2011-05-30') since the table might have different rows at other times. We want to say "do not include the last two rows".

We can modify the join as shown here to accomplish that. We essentially ask the table for the max date value and skip it and the previous date.

Demo 07: skipping the last two rows- The end-user might not understand why the rows for May 30 and 31 are missing.

```
select   a1.sales_day
,        a1.sales
,        sum(a2.sales) as three_day_sum
from     a_oe.adv_sales a1
join     a_oe.adv_sales a2
    on   a2.sales_day between  a1.sales_day
    and  date_add(a1.sales_day ,  interval 2 day)
    and a1.sales_day <= (select date_add(max(a3.sales_day) ,  interval -2 day)
                         from a_oe.adv_sales a3)
group by a1.sales_day, a1.sales
order by a1.sales_day
;
```

Note that we can put some tests that we commonly think of as Where clause tests into the From clause.

Demo 08: Using only April 2011 dates

```
add the where clause
where month(a1.sales_day) = 4 and year (a1.sales_day) = 2011
+------------+-------+---------------+
| sales_day  | sales | three_day_sum |
+------------+-------+---------------+
| 2011-04-25 |   400 |          1200 |
| 2011-04-26 |   400 |          1100 |
| 2011-04-27 |   400 |          1600 |
| 2011-04-28 |   300 |          1780 |
| 2011-04-29 |   900 |          1905 |
| 2011-04-30 |   580 |          1015 |
+------------+-------+---------------+
```