Table of Contents

Running totals and moving totals/aggregates are two types of queries that are often used in business situations.

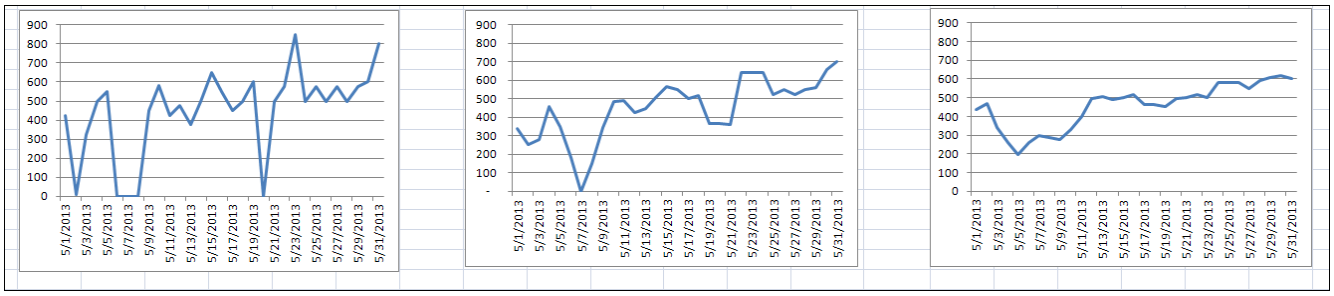
For these queries we need to have some column that is used to order the data. Remember that in standard SQL tables there is no ordering of the rows. But we can have a column such as a sales date column that we can use for ordering the data in a query.

A running total is an expression that totals a value on its current row and on the preceding rows. The following shows a Running total for the first few days in May 2013, with the total starting with May 2, 2013.

sales_date	sales	Running total
5/1/2013	425	425
5/2/2013	10	435
5/3/2013	325	760
5/4/2013	500	1260
5/5/2013	550	1810
5/6/2013	0	1810

In this class in terms of your final grade, you would be more interesting in your running total of your assignment scores than in a particular assignment score.

You probably have heard of moving averages if you listen to news about unemployment rates or other financial data. The following are three Excel charts for the sales data for May 2013. The first shows the data for each day. The second chart shows a three day moving average and the third is a 7 day moving average. The chart is smoothed out to deemphasize data that is unusually higher or lower. For some purposes that is a more meaningful value.



Very often the column used to do the smoothing is a date/time series. The aggregate is often the average but could be one of the other aggregates. In the above examples the three moving aggregate used the current date and one date before and one date after the current date; sometime people use the current date and the two previous dates. These all fall into the general category of moving aggregates.

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.

For this unit some of these are presented as techniques and I do not expect you to have a deep understanding of how these functions.

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 Aggregates

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 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 two columns and the second copy of the table is used to calculate the running total.

Demo 01: 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_testbed.adv_emp emp_1
Join a_testbed.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 |
+--------+--------+--------------+
|    101 |  45000 |        45000 |
|    102 |  20000 |        65000 |
|    103 |  28000 |        93000 |
|    104 |  25000 |       118000 |
|    105 |  25000 |       143000 |
|    106 |  28000 |       171000 |
|    107 |  45000 |       216000 |
|    108 |  28000 |       244000 |
|    109 |  32000 |       276000 |
|    110 |  45000 |       321000 |
|    111 |  45000 |       366000 |
|    112 |  30900 |       396900 |
```

```
|    113 |  45000 |       441900 |
|    114 |  32000 |       473900 |
|    115 |  24000 |       497900 |
|    116 |  28000 |       525900 |
|    117 |  28000 |       553900 |
|    118 |  25000 |       578900 |
|    119 |  25000 |       603900 |
|    120 |  22000 |       625900 |
|    121 |  28000 |       653900 |
+--------+--------+--------------+
21 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 02: 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_testbed.adv_emp
   Group by dept_id
   ) dt_1
Join (
   Select dept_id, sum(salary)  as DeptTotal
   From a_testbed.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
;
+---------+-----------+--------------+
| dept_id | DeptTotal | RunningTotal |
+---------+-----------+--------------+
|      10 |     45000 |        45000 |
|      20 |    150900 |       195900 |
|      30 |    153000 |       348900 |
|      45 |    305000 |       653900 |
+---------+-----------+--------------+
4 rows in set (0.00 sec)
```

1.2. Alternate approach

Demo 03: 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_testbed.adv_emp as Emp_2
   Where emp_2.emp_id <= emp_1.emp_id
   ) as RunningTotal
From a_testbed.adv_emp emp_1
Group by emp_1.emp_id
order by emp_1.emp_id;
```

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; the table also include a few days in June.

```
+------------+-------+
| sales_day  | sales |
+------------+-------+
| 2013-04-25 |   400 |
| 2013-04-26 |   400 |
| 2013-04-27 |   400 |
| 2013-04-28 |   300 |
| 2013-04-29 |   900 |
| 2013-04-30 |   580 |
| 2013-05-01 |   425 |
```

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 adv_sales a1
join 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 04:

```
Select a1.sales_day
, a1.sales
, sum(a2.sales) as three_day_sum
From a_testbed.adv_sales a1
Join a_testbed.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 |
+------------+-------+---------------+
| 2013-04-25 |   400 |          1200 |
| 2013-04-26 |   400 |          1100 |
| 2013-04-27 |   400 |          1600 |
| 2013-04-28 |   300 |          1780 |
| 2013-04-29 |   900 |          1905 |
| 2013-04-30 |   580 |          1015 |
| 2013-05-01 |   425 |           760 |
| 2013-05-02 |    10 |           835 |
| 2013-05-03 |   325 |          1375 |
```

```
| 2013-05-04 |   500 |          1050 |
| 2013-05-05 |   550 |           550 |
| 2013-05-06 |     0 |             0 |
| 2013-05-07 |     0 |           450 |
| 2013-05-08 |     0 |          1030 |
| 2013-05-09 |   450 |          1455 |
| 2013-05-10 |   580 |          1480 |
| 2013-05-11 |   425 |          1275 |
| 2013-05-12 |   475 |          1350 |
| 2013-05-13 |   375 |          1525 |
| 2013-05-14 |   500 |          1700 |
| 2013-05-15 |   650 |          1650 |
| 2013-05-16 |   550 |          1500 |
| 2013-05-17 |   450 |          1550 |
| 2013-05-18 |   500 |          1100 |
| 2013-05-19 |   600 |          1100 |
| 2013-05-20 |     0 |          1075 |
| 2013-05-21 |   500 |          1925 |
| 2013-05-22 |   575 |          1925 |
| 2013-05-23 |   850 |          1925 |
| 2013-05-24 |   500 |          1575 |
| 2013-05-25 |   575 |          1650 |
| 2013-05-26 |   500 |          1575 |
| 2013-05-27 |   575 |          1650 |
| 2013-05-28 |   500 |          1675 |
| 2013-05-29 |   575 |          1975 |
| 2013-05-30 |   600 |          2100 |
| 2013-05-31 |   800 |          2075 |
| 2013-06-01 |   700 |          1725 |
| 2013-06-02 |   575 |          1025 |
| 2013-06-03 |   450 |           450 |
+------------+-------+---------------+
40 rows in set (0.00 sec)
```

Demo 05: But suppose we want to use only the sales days in May 2013. Start with a view for that timespan

```
Create or replace view a_testbed.adv_sales_rest as
    Select *
    From a_testbed.adv_sales
    Where sales_day Between '2013-05-01' And '2013-05-31';
```

Demo 06:

```
Select
  a1.sales_day
, a1.sales
, SUM(a2.sales) As three_day_sum
From a_testbed.adv_sales_rest a1
Join a_testbed.adv_sales_rest 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 |
+------------+-------+---------------+
| 2013-05-01 |   425 |           760 |
```

```
| 2013-05-02 |    10 |           835 |
| 2013-05-03 |   325 |          1375 |
| 2013-05-04 |   500 |          1050 |
| 2013-05-05 |   550 |           550 |
| 2013-05-06 |     0 |             0 |
| 2013-05-07 |     0 |           450 |
| 2013-05-08 |     0 |          1030 |
| 2013-05-09 |   450 |          1455 |
| 2013-05-10 |   580 |          1480 |
| 2013-05-11 |   425 |          1275 |
| 2013-05-12 |   475 |          1350 |
| 2013-05-13 |   375 |          1525 |
| 2013-05-14 |   500 |          1700 |
| 2013-05-15 |   650 |          1650 |
| 2013-05-16 |   550 |          1500 |
| 2013-05-17 |   450 |          1550 |
| 2013-05-18 |   500 |          1100 |
| 2013-05-19 |   600 |          1100 |
| 2013-05-20 |     0 |          1075 |
| 2013-05-21 |   500 |          1925 |
| 2013-05-22 |   575 |          1925 |
| 2013-05-23 |   850 |          1925 |
| 2013-05-24 |   500 |          1575 |
| 2013-05-25 |   575 |          1650 |
| 2013-05-26 |   500 |          1575 |
| 2013-05-27 |   575 |          1650 |
| 2013-05-28 |   500 |          1675 |
| 2013-05-29 |   575 |          1975 |
| 2013-05-30 |   600 |          1400 |
| 2013-05-31 |   800 |           800 |
+------------+-------+---------------+
31 rows in set (0.00 sec)
```

The data looks the same until you get to the end of the month.

These are the rows from demo 04

```
| 2013-05-28 |   500 |          1675 |
| 2013-05-29 |   575 |          1975 |
| 2013-05-30 |   600 |          2100 |
| 2013-05-31 |   800 |          2075 |
```

These are the rows from demo 06

```
| 2013-05-28 |   500 |          1675 |
| 2013-05-29 |   575 |          1975 |
| 2013-05-30 |   600 |          1400 |
| 2013-05-31 |   800 |           800 |
+------------+-------+---------------+
```

In demo 04

the calculation for May 29 includes May 29, May 30, and May 31

the calculation for May 30 includes May 30, May 31, and June 1

the calculation for May 31 includes May 31, June 1, and June 1

In demo 06 which restricts to May dates only

the calculation for May 29 includes May 29, May 30, and May 31

the calculation for May 30 includes May 30, May 31- 2 days only

the calculation for May 31 includes May 31 - 1 days only

It is a business rule if demo 06 is a valid result set. If you truly want to include only May sales then you need to know that the last two days in the chart are not three day rows. An alternate solution would be to not display the last two rows in this result set since they are not three day sums.

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_testbed.adv_sales_rest a1
Join a_testbed.adv_sales_rest 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_testbed.adv_sales_rest a3
   )
Group By a1.sales_day, a1.sales
Order By a1.sales_day;
```

Note that we are skipping 2 rows because this is a 3 day aggregate and the last two rows are not fully 3 row aggregates.

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

You could also produce a result set which flags the last two days with a message.

```
+------------+-------+---------------+---------------+
| sales_day  | sales | three_day_sum | msg           |
+------------+-------+---------------+---------------+
| 2013-05-01 |   425 |           760 |               |
| 2013-05-02 |    10 |           835 |               |
| 2013-05-03 |   325 |          1375 |               |
. . .
| 2013-05-26 |   500 |          1575 |               |
| 2013-05-27 |   575 |          1650 |               |
| 2013-05-28 |   500 |          1675 |               |
| 2013-05-29 |   575 |          1975 |               |
| 2013-05-30 |   600 |          1400 | two days only |
| 2013-05-31 |   800 |           800 | one day only  |
+------------+-------+---------------+---------------+
31 rows in set (0.00 sec)
```