Introduction to Statistical Straight-line Regression Analysis using Oracle SQL

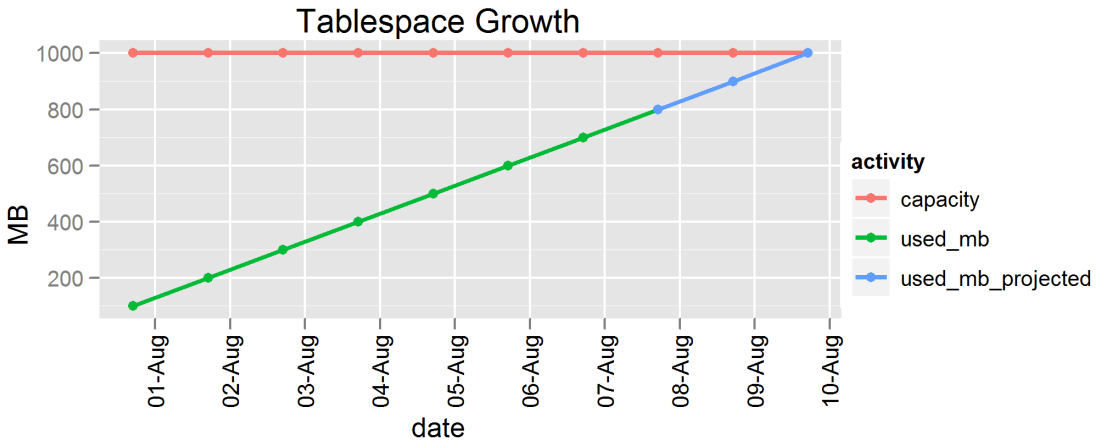
Jay Stanley at Database Specialists, Inc

2012-10-19 Fri

Statistical regression analysis

Put simply, regression analysis involves looking at set of (x,y) samples, and estimating the relationships between the variables. This addresses questions like "as X increases in value, does Y increase in value, and if so by how much?". If the relationship between the variables is known, and you can put it into an equation like `x = 2 * y`, then you can use that equation to try to predict values of x given any value of y.

It's a little more intuitive to look at a graph of these values to understand what regression analysis does. Here's a graph that shows the growth of an Oracle tablespace that we'll be using as an example:



The graph shows that from the 1st until the 8th of August, 2012, the tablespace usage grew from 100MB to 800MB. It shows that the capacity of the tablespace is 1000MB. The `used_mb_projected` value above represents

the *straight-line regression analysis*. It shows the projected growth on the 9th and 10th of August, when it fills up the tablespace.

The field of regression analysis is a large and complex; this article will *only* look at straight-line regression analysis, where there exists a graph with a relatively straight line. There are definitely more complex graph curve shapes that can be reliably estimated; the basic principles are similar.

Oracle built-in features to support regression analysis

Although it is mentioned in the Oracle documentation, the features in Oracle SQL that support regression analysis are not well-documented; especially missing are some samples of how to use it. What's really nice about these Oracle features is that they are incorporated into SQL itself, they do not require Enterprise Edition, they work 'right out of the box' from version 10 and greater, and they work very efficiently inside the database; it's not required that you export the data to a separate statistical package for analysis.

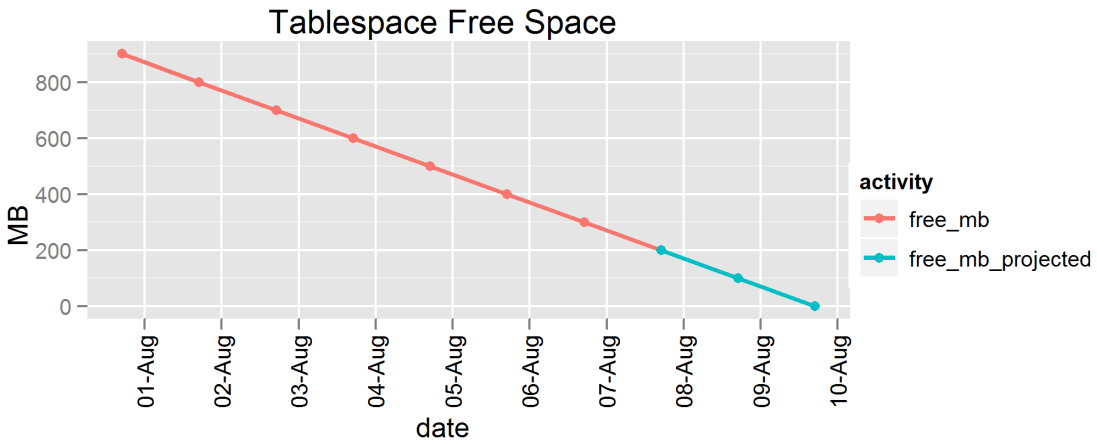
The main sql functions we'll be investigating are called `REGR_INTERCEPT` and `REGR_R2`.

The `REGR_INTERCEPT` SQL function will return when a line will intercept the Y axis of a graph, where X=0. The `REGR_R2` SQL function will return a confidence factor that can be used to understand how likely the above `REGR_INTERCEPT` function is correct.

The `REGR_INTERCEPT` linear regression function

The `REGR_INTERCEPT(expr1,expr2) over (analytic_clause)` returns the X value, where the predicted line will intercept the Y axis, after it eliminates any pairs of values with nulls in them. The `(expr1, expr2)` part of the clause is your list of X and Y values in the dataset. The `over (analytic_clause)` shows how the data should be partitioned or broken-up in to pieces that belong with each other. To use it, it is required to construct your value pairs `(expr1,expr2)` in such a way that they will intercept the Y axis.

In our example, the easiest way to think about it is not to consider about the amount of space used -vs- the capacity, but rather to consider the amount of free space over time: here is an amended graph:



If the slope of your line is positive with values climbing as the Y axis increases, then the result will be where it intercepted the Y axis on the left side of your graph.

The `REGR_R2` linear regression function

The `REGR_R2(expr1,expr2) over (analytic_clause)` function returns the confidence level – technically called a *coefficient of determination* – of the prediction made by the `REGR_INTERCEPT` function. It uses the same parameters as the `REGR_INTERCEPT` function. If the *coefficient of determination* is for example 0.9 or 0.8, then you can be pretty confident that the prediction calculated by `REGR_INTERCEPT` is highly likely. If it is 0.5 or lower, then it is far less likely to happen – it means that the pairs of (x,y) values do not suggest a straight line. The calculated value will vary from 0 (very rarely encountered) to 1 (also very rarely encountered). Higher numbers represent a higher confidence level.

An example using a time-based Y axis

Many real-world graphs involve time on the Y (horizontal) axis, with a value on the X (vertical) axis. The data that is in use in this example is not based upon pairs of numbers; it is based on a date datatype, and a number. Dates are not numbers, and unfortunately the Oracle regression functions do not work with dates. So, to make them work, we'll need to convert the dates into numbers, and then convert the result back again from a number into a

date.

One quick and easy way to convert a date to a number is to basically convert it into a Julian date, which represents an amount of time between our sample date and some constant date. In traditional SystemV Unix, the date of January 1, 1970 at midnight is used as the constant date, so that's what we'll use here. When you subtract a date from another date in Oracle, you will get a floating-point number representing the number of days, and fractions of a day between the two dates. For this example, then, we'll use the form

```
sample_date - to_date('19700101 00:00:00','YYYYMMDD HH24:MI:SS')
```

To convert the resulting floating-point number returning from the `REGR_INTERCEPT` function, we'll use the reverse:

```
to_date('19700101 00:00:00','YYYYMMDD HH24:MI:SS') + REGR_INTERCEPT() result
```

An example of capacity estimation for absolute straight-line growth

In this example, we'll be looking at predicting when (what date) the free space in a tablespace will be exhausted.

For example, if you have an Oracle tablespace, and it is growing at a very regular rate (say 100M/day), and you know the capacity of that tablespace (say 1G) and it's current size (100M), if the growth rate represented by a line on the graph is very straight, you can reasonably predict that it'll fill up in 9 days – see the graph above. You have 900M of free capacity; you're filling it up 100M/day; it'll fill up in (900M/100M)=9 days.

In the real word though, you often don't have something as predictable (will it use *exactly* 100M/day for the next 9 days? Does usage change over the weekend?). Understanding the date that it may fill up isn't enough; you'd like to know how confident you are in such an estimate. If the line jumps up and down in an unpredictable manner, you should be less confident in the conclusio n that it will indeed fill up in 9 days. Knowing that confidence level is very important to understanding how much safety you need to build into your model.

Here's a SQL script for our example; we're creating a table `TABLESPACE_USAGE`, that will have in it how much space is used, and how much space is free in the `DATA_TS` tablespace over time.

```
create table tablespace_usage(
  sample_date     date,
  tablespace_name varchar2(30),
  mbytes_used      number,
  mbytes_free      number);

rem our capacity will be mbytes_used + mbytes_free = 1000

insert into tablespace_usage(sample_date,tablespace_name,mbytes_used)
  values (to_date('20120801','YYYYMMDD'),'DATA_TS',100);
insert into tablespace_usage(sample_date,tablespace_name,mbytes_used)
  values (to_date('20120802','YYYYMMDD'),'DATA_TS',200);
insert into tablespace_usage(sample_date,tablespace_name,mbytes_used)
  values (to_date('20120803','YYYYMMDD'),'DATA_TS',300);
insert into tablespace_usage(sample_date,tablespace_name,mbytes_used)
  values (to_date('20120804','YYYYMMDD'),'DATA_TS',400);
insert into tablespace_usage(sample_date,tablespace_name,mbytes_used)
  values (to_date('20120805','YYYYMMDD'),'DATA_TS',500);
insert into tablespace_usage(sample_date,tablespace_name,mbytes_used)
  values (to_date('20120806','YYYYMMDD'),'DATA_TS',600);
insert into tablespace_usage(sample_date,tablespace_name,mbytes_used)
  values (to_date('20120807','YYYYMMDD'),'DATA_TS',700);
insert into tablespace_usage(sample_date,tablespace_name,mbytes_used)
  values (to_date('20120808','YYYYMMDD'),'DATA_TS',800);

update tablespace_usage set mbytes_free = 1000 - mbytes_used;

commit;
```

This is what our table looks like:

SAMPLE_DATE	TABLESPACE	MB_USED	MB_FREE
01-AUG-12	DATA_TS	100	900
02-AUG-12	DATA_TS	200	800
03-AUG-12	DATA_TS	300	700
04-AUG-12	DATA_TS	400	600
05-AUG-12	DATA_TS	500	500
06-AUG-12	DATA_TS	600	400
07-AUG-12	DATA_TS	700	300
08-AUG-12	DATA_TS	800	200

To figure out when the tablespace will be full, the `mbytes_used` column isn't used – it's the `mbytes_free` column that is used. The date corresponding to When the `mbytes_used` = 0 (which corresponds to when the graph line will intercept the Y axis), is the date that we wish to predict.

We need to partition this calculation by `tablespace_name`; each tablespace filling up should be an independent calculation.

Here is the query that we'll use to actually do the analysis:

```
select
  sample_date,
  sample_date - to_date('19700101 00:00:00','YYYYMMDD HH24:MI:SS') as julian_dt,
  tablespace_name,
  mbytes_used,
  mbytes_free,
  REGR_INTERCEPT(sample_date -
                 to_date('19700101 00:00:00','YYYYMMDD HH24:MI:SS'),mbytes_free)
    over (partition by tablespace_name) as regr_int,
  to_date('19700101 00:00:00','YYYYMMDD HH24:MI:SS') +
    REGR_INTERCEPT(sample_date -
                   to_date('19700101 00:00:00','YYYYMMDD HH24:MI:SS'),mbytes_free)
    over (partition by tablespace_name) as predicted_date,
  REGR_R2(sample_date - to_date('19700101 00:00:00','YYYYMMDD HH24:MI:SS'),mbytes_free
    over (partition by tablespace_name) as coeff_of_determination
from tablespace_usage
order by sample_date;
```

Here's our result! (the `COEFF_DET` is the `REGR_R2` function `Coefficient of Determination`).

SAMPLE_DATE	JULIAN_DT	TABLESPACE	MB_USED	MB_FREE	REGR_INT	PREDICTED_DATE	COEFF_DET
01-AUG-12	15553	DATA_TS	100	900	15562	10-AUG-12	1
02-AUG-12	15554	DATA_TS	200	800	15562	10-AUG-12	1
03-AUG-12	15555	DATA_TS	300	700	15562	10-AUG-12	1
04-AUG-12	15556	DATA_TS	400	600	15562	10-AUG-12	1
05-AUG-12	15557	DATA_TS	500	500	15562	10-AUG-12	1
06-AUG-12	15558	DATA_TS	600	400	15562	10-AUG-12	1
07-AUG-12	15559	DATA_TS	700	300	15562	10-AUG-12	1
08-AUG-12	15560	DATA_TS	800	200	15562	10-AUG-12	1

In this example, our data *perfectly fits* the straight line, with 0 deviation; this explains why our coefficient of determination has a value of 1. Obviously, that normally doesn't happen in the real world, especially concerning how tablespaces fill up with space.

So – let's experiment with changing our source data a bit to add a bit of 'noise', to see what happens to our coefficient of determination.

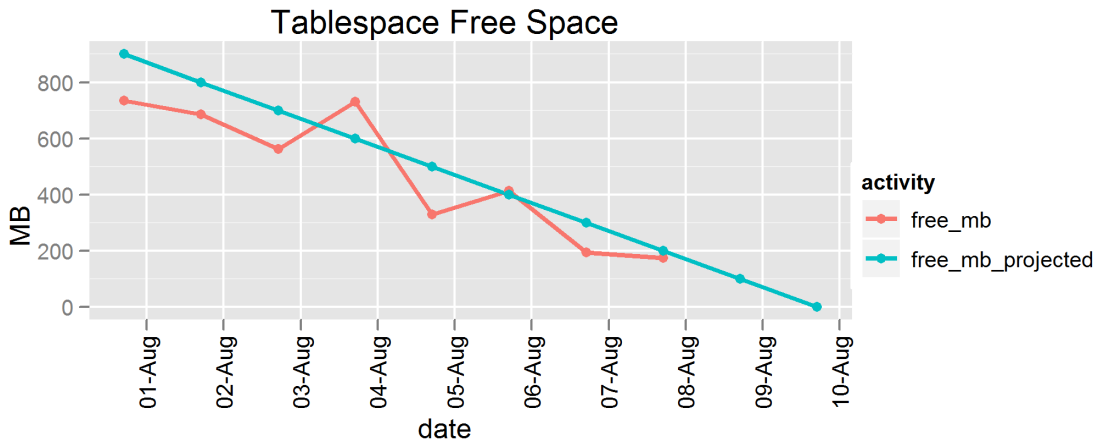
```
update tablespace_usage
set mbytes_used = mbytes_used
                  + trunc(dbms_random.value * 300)
                  - trunc(dbms_random.value * 300);

update tablespace_usage
set mbytes_free = 1000 - mbytes_used;

commit;
```

SAMPLE_DATE	JULIAN_DT	TABLESPACE	MB_USED	MB_FREE	REGR_INT	PREDICTED_DATE	COEFF_DET
01-AUG-12	15553	DATA_TS	266	734	15561.0622	09-AUG-12	.81969874
02-AUG-12	15554	DATA_TS	314	686	15561.0622	09-AUG-12	.81969874
03-AUG-12	15555	DATA_TS	438	562	15561.0622	09-AUG-12	.81969874
04-AUG-12	15556	DATA_TS	269	731	15561.0622	09-AUG-12	.81969874
05-AUG-12	15557	DATA_TS	670	330	15561.0622	09-AUG-12	.81969874
06-AUG-12	15558	DATA_TS	586	414	15561.0622	09-AUG-12	.81969874
07-AUG-12	15559	DATA_TS	806	194	15561.0622	09-AUG-12	.81969874
08-AUG-12	15560	DATA_TS	825	175	15561.0622	09-AUG-12	.81969874

Here's a graph of our values: notice how the free space is much more variable than before.



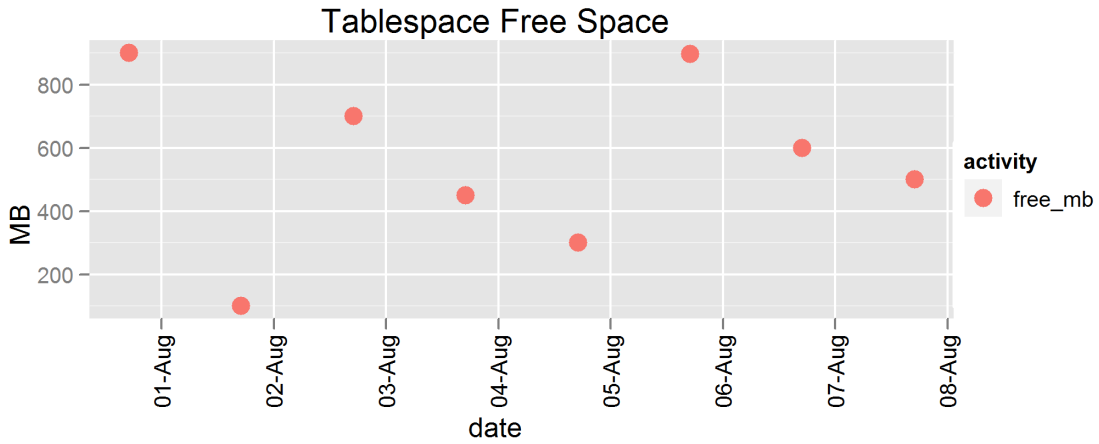
As expected, our `coefficient of determination` has decreased, as the amount of deviation from the straight-line has increased for our samples.

Let's look at a new tablespace, one that doesn't have much of a pattern of growth (free space), and see what the `coefficient of determination` becomes. The table below shows two tablespaces; the one we previously used (`DATA_TS`), as well as the new one called `RANDOM_TS`. Note how the calculation of the `predicted date` and the `coefficient of determination` are *partitioned* by the tablespace name – each is a separate calcuation – this is what

is meant by the `partition by ...` clause in both the `REGR_INTERCEPT()` and `REGR_R2()` functions.

SAMPLE_DATE	JULIAN_DT	TABLESPACE	MB_USED	MB_FREE	REGR_INT	PREDICTED_DATE	COEFF_DET
01-AUG-12	15553	DATA_TS	266	734	15561.0622	09-AUG-12	.81969874
02-AUG-12	15554	DATA_TS	314	686	15561.0622	09-AUG-12	.81969874
03-AUG-12	15555	DATA_TS	438	562	15561.0622	09-AUG-12	.81969874
04-AUG-12	15556	DATA_TS	269	731	15561.0622	09-AUG-12	.81969874
05-AUG-12	15557	DATA_TS	670	330	15561.0622	09-AUG-12	.81969874
06-AUG-12	15558	DATA_TS	586	414	15561.0622	09-AUG-12	.81969874
07-AUG-12	15559	DATA_TS	806	194	15561.0622	09-AUG-12	.81969874
08-AUG-12	15560	DATA_TS	825	175	15561.0622	09-AUG-12	.81969874
01-AUG-12	15553	RANDOM_TS	100	900	15556.4267	04-AUG-12	.000226136
02-AUG-12	15554	RANDOM_TS	900	100	15556.4267	04-AUG-12	.000226136
03-AUG-12	15555	RANDOM_TS	300	700	15556.4267	04-AUG-12	.000226136
04-AUG-12	15556	RANDOM_TS	550	450	15556.4267	04-AUG-12	.000226136
05-AUG-12	15557	RANDOM_TS	700	300	15556.4267	04-AUG-12	.000226136
06-AUG-12	15558	RANDOM_TS	102	898	15556.4267	04-AUG-12	.000226136
07-AUG-12	15559	RANDOM_TS	400	600	15556.4267	04-AUG-12	.000226136
08-AUG-12	15560	RANDOM_TS	500	500	15556.4267	04-AUG-12	.000226136

Let's see the raw data just for the `RANDOM_TS` in a graph by itself:



You'll notice immediately that the value jumps around a lot; it is much less predictable. Because it's much less predictable, the calculated `coefficient of determination` is far lower; it's basically 0 within 3 orders of magnitude. In reviewing the predicted date of the 4th of August, along with the 0.002 coefficient of determination, we can be very sure that it's not a very reliable prediction.

Conclusion (tl;dr)

- Using the `REGR_INTERCEPT()` and `REGR_R2()` functions in Oracle SQL work with all editions of Oracle from v10 or greater and can provide some solid statistically relevant results

- The functions above need to be *partitioned* for the functions to have valid results
- The `REGR_INTERCEPT()` function returns the predicted Value where the data will cross the Y axis, and so to use it the way that the query is structured needs to keep this in mind
- The `REGR_R2()` function provides value information relating to the confidence level regarding the result of the `REGR_INTERCEPT()` function

Further reading

Oracle documentation: http://docs.oracle.com/cd/B19306_01/server.102/b14200/functions132.htm
Regressionn Analysis in general http://en.wikipedia.org/wiki/Regression_analysis
Straight-line Regression Analysis http://en.wikipedia.org/wiki/Simple_linear_regression