Chapter 8

Joins

Contents

1 Joins

In this section we combine tables using the JOIN operator. There are a number of different ways to combine data, which we will go into now.

- Lets consider the following two tables, which we will use to demonstrate the different types of joins:

Table 8.1: Join Example Tables

Table 8.2: *Class1* Table

sname	grade
John	A
Jim	A
Kyle	C

Table 8.3: *Class2* Table

sname	grade
John	A
Jim	B
Ashley	F

- The first join we will consider is the LEFT JOIN, which keeps all records from the first table (the "Left Hand Side" or "LHS") and only those records that match from the second table (or the "Right Hand Side" or "RHS"):

```
select
    class1.*, class2.*
from
    cls.class1
left join
    cls.class2
on class1.sname = class2.sname;


sname    grade    sname    grade
-------  -------  -------  -------
John     A        John     A
Jim      A        Jim      B
Kyle     C
```

There are two components to the JOIN syntax. The first, within the FROM clause, specifies the type of JOIN (in this case "LEFT JOIN") to attempt and the second, the ON clause, determines how two rows are defined to match. The ON operator acts like a WHERE clause in that any boolean condition or set of conditions can be put in it by using parenthesis, AND and OR. Any type the expression within the ON operator is true, the database regards those rows as matching. Mentally, you should think of a JOIN as going through every possible combination of rows and deciding if a row matches with another based on the match criteria in the ON clause.

In this SELECT statement we choose all columns from both class1 and class2 tables. Since the columns have the same names in both tables we see that the column names are repeated in the resulting table.

This LEFT JOIN leaves the second sname and grade Null for the "Kyle" row from the first table, as there is no matching row in the second table.

- There is also a RIGHT JOIN, which, similar to the LEFT JOIN, keeps all rows from the right-hand, or second, table:

```
select
    class1.*, class2.*
from
    cls.class1
right join
    cls.class2
on class1.sname = class2.sname;


sname    grade    sname    grade
-------  -------  -------  -------
John     A        John     A
Jim      A        Jim      B
                  Ashley   F
```

- If we only want to consider rows that are in *both* tables, we use an INNER JOIN, the syntax for which is just JOIN:

```
select
    class1.*, class2.*
from
    cls.class1
join
    cls.class2
on class1.sname = class2.sname;


sname    grade    sname    grade
-------  -------  -------  -------
John     A        John     A
Jim      A        Jim      B
```

  In this case, only John and Jim are returned since they are the only individuals that are in both tables.

- A FULL JOIN (sometimes called an OUTER JOIN, or FULL OUTER JOIN) includes all rows from either table:

```
select
    class1.*, class2.*
from
    cls.class1
full join
    cls.class2
on
    class1.sname = class2.sname;


sname    grade    sname    grade
-------  -------  -------  -------
John     A        John     A
Jim      A        Jim      B
Kyle     C
                  Ashley   F
```

Because “Kyle” <> “Ashley”, these two rows are kept separate.

- In a few instances we may wish to create every possible combination of rows[1], which we call a CROSS JOIN. The syntax for a CROSS JOIN is below:

```
select
    class1.*, class2.*
from
    cls.class1
cross join
    cls.class2;


sname    grade    sname    grade
-------  -------  -------  -------
John     A        John     A
John     A        Jim      B
John     A        Ashley   F
Jim      A        John     A
Jim      A        Jim      B
[...]
```

Note that this returns every possible combination of rows. Since we selected “*” it also returns every column – which means columns with duplicated names. If we wanted to only return a few of the columns we could do the following:

[1] This is sometimes called a Cartesian product.

```
select
    class1.sname as name1
    , class2.sname as name2
    , class1.grade as grade1
    , class2.grade as grade2
from
    cls.class1
cross join
    cls.class2;


name1    name2    grade1    grade2
-------  -------  --------  --------
John     John     A         A
John     Jim      A         B
John     Ashley   A         F
Jim      John     A         A
Jim      Jim      A         B
[...]
```

- If the columns that we are matching on have the same name than we can use USING, rather than ON to specify the matching column. Doing so generates a different type of output:

```
select
    *
from
    cls.class1
full join
    cls.class2
USING( sname);


sname    grade    grade
-------  -------  -------
John     A        A
Jim      A        B
Kyle     C
Ashley            F
```

In this example, the database combined the sname column into a single column! USING tells the database that the columns represent the same data and need not be repeated. This type of “natural” join is extremely powerful when you are joining two tables which represent similar data.

- The following query demonstrates USING with multiple columns:

```
select
    *
from
    cls.class1
inner join
    cls.class2
using( sname, grade);


sname    grade
-------  -------
John     A
```

- Let's look at what the following returns, which uses both a USING with multiple columns and a FULL JOIN:

```
select
    *
from
    cls.class1
full join
    cls.class2
using( sname, grade);


sname    grade
-------  -------
John     A
Jim      A
Kyle     C
Jim      B
Ashley   F
```

  In this example, the database returns 5 rows, since the only row that matches on both sname and grade is John. People should study more.

- What does the following return?

```
select * from
    cls.class1
left join
    cls.class2
on class1.sname = class2.sname
and class1.grade >= class2.grade;


sname    grade    sname    grade
-------  -------  -------  -------
John     A        John     A
Jim      A
Kyle     C
```

Since this is a LEFT JOIN, this will include all values from the left hand table, but it will only match those which are alphabetically earlier or the same as the right hand side table. In other words, this returns only those rows from the right hand side where the person did better in class1.

```
 sname | grade | sname | grade
-------+-------+-------+-------
 Jim   | A     |       |
 John  | A     | John  | A
 Kyle  | C     |       |
(3 rows)
```

- Keep in mind that the USING clause creates a synthetic column using a similar construct as a CASE statement:

```
select
    sname as from_using
    , class1.sname as lhs
    , class2.sname as rhs
    , CASE
        when class1.sname is not null then class1.sname
        else class2.sname
      END as coal
from
    cls.class1
full join
    cls.class2
using( sname );


from_using    lhs    rhs     coal
------------  -----  ------  ------
John          John   John    John
Jim           Jim    Jim     Jim
Kyle          Kyle           Kyle
Ashley               Ashley  Ashley
```

The column “from_using” and “coal” are created as the output of the using statement and from the coalesce statement; from the above they are clearly the same. Importantly, the above statement also demonstrates that in the SELECT statement there is still access to the underlying, original columns.

- In the above examples we used ON to tell the database which columns to match. However, we can also use the WHERE clause to match. For example:

```
select
    class1.*, class2.*
from
    cls.class1
cross join
    cls.class2
where class1.sname = class2.sname;


sname    grade    sname    grade
-------  -------  -------  -------
John     A        John     A
Jim      A        Jim      B
```

generates the same output as our inner join. In this case, we used a cross join to generate all possible combinations of rows and only kept those rows where the sname was the same in both columns via a WHERE clause.

- The following syntax is also used when doing cross joins.

```
select
    *
from
    cls.class1, cls.class2
where class1.sname = class2.sname;


sname    grade    sname    grade
-------  -------  -------  -------
John     A        John     A
Jim      A        Jim      B
```

This can also be done with more than two tables.

- We can also combine ON and WHERE:

```
select
    class1.*, class2.*
from
    cls.class1
left join
    cls.class2
on
    class1.sname = class2.sname
where
    class1.grade = class2.grade;


sname    grade    sname    grade
-------  -------  -------  -------
John     A        John     A
```

The above example uses a left join and a where clause to recreate an inner join.

- Very importantly, we can use any boolean expression as our JOIN condition as in the following example where we create a table which only contains those rows, in class1, which are not the same name! **Note that this joins class1 on itself!**

```
select
    lhs.sname as lname, lhs.grade as lgrade, rhs.*
from
    cls.class1 as lhs
left join
    cls.class1 as rhs
on lhs.sname <> rhs.sname;


lname    lgrade    sname    grade
-------  --------  -------  -------
John     A         Jim      A
John     A         Kyle     C
Jim      A         John     A
Jim      A         Kyle     C
Kyle     C         John     A
[...]
```

2 UNION and UNION ALL

- Another way to combine data is using UNION and UNION ALL. While the JOIN syntax puts tables side-by-side, the UNION and UNION ALL tables stack tables vertically on each other – appending (or concatenating) the data vertically.
- The syntax for UNION and UNION ALL looks a bit different than the syntax for other SQL commands since they behave not on *tables*, but on queries. Consider the following example of the UNION ALL command:

```
select sname from cls.class1
UNION ALL
select sname from cls.class2


sname
-------
John
Jim
Kyle
John
Jim
[...]
```

- The columns have to be selected in the correct order. The following query, which switches the order of grade and sname in the second table will not return properly aligned columns.

```
select sname, grade from cls.class1
UNION ALL
select grade, sname from cls.class2


sname    grade
-------  -------
John     A
Jim      A
Kyle     C
A        John
B        Jim
[...]
```

- The column types are also defined by the first statement in the command, so all SELECT statements must generate compatible columns. For example if our first query had an integer in the first column position then the second query can't put a string in the first position.
- The difference between UNION and UNION ALL is that UNION will automatically deduplicate records. For example, consider the following query:

```
select sname from cls.class1
UNION
select sname from cls.class2


sname
-------
Kyle
John
Ashley
Jim
```

In this case only four rows are returned since John and Jim are duplicates. UNION removes whole, exact, row duplicates. Every column in the row must be the same for UNION to decide two rows are duplicates.

- Keep in mind that using UNION is very expensive as removing duplicates is a costly process.[2]
- We can use the UNION command to determine the best grade that a student received, as in the following query.

```
select sname, MIN(grade) as best_grade
from
    (select sname, grade from cls.class1
    UNION ALL
    select sname, grade from cls.class2 ) as innerQ
group by 1;


sname    best_grade
-------  ------------
Kyle     C
Jim      A
Ashley   F
John     A
```

- As we said above, both UNION and UNION ALL work on *statements* not tables, meaning that the following command will *not* complete successfully:

```
select sname, MIN(grade) as best_grade
from
    (select sname, grade from cls.class1) as lhs1
    UNION ALL
    (select sname, grade from cls.class2) as lhs2
group by 1;
```

3 Best Practices when Combining Tables

1. **Always have a Unique Side**. When you join two tables on a particular column, make sure that the column that you are joining on is unique on one side. If you join on a column with duplicates on both sides the database is going to create rows (via the cartesian product), a generally negative outcome.

    - Up to this point we only considered the case where both sides are unique. Let's assume that are tables now look like the below:

      In the tables above, there are multiple observations for the name "John". The lack of uniqueness causes problems, as we will see in the following query:

[2] On my computer, using the NYSE dataset, it took 2 seconds to count the number of rows after a UNION ALL between 2010 and 2011 while doing a UNION took over five times as long.

Table 8.4: Join Example Tables (II)

Table 8.5: *Class3* Table

sname	grade
John	A
John	B
Kyle	C

Table 8.6: *Class4* Table

sname	grade
John	A
John	B
John	C
Tim	F

```
select
    class3.sname as lname
    , class3.grade as lgrade
    , class4.sname as rname
    , class4.grade as rgrade
from
    cls.class3
left join
    cls.class4
on class3.sname = class4.sname;


lname    lgrade    rname    rgrade
-------  --------  -------  --------
John     A         John     A
John     A         John     B
John     A         John     C
John     A         John     A
John     A         John     B
[...]
```

- In this case, we can see that every pair-wise combination of the matching rows ($6 = 3 \cdot 2$ for John) was generated by the query. In other words, the "ON" clause behaved as if a "WHERE" clause; each time a row matched it was returned.
- Another way of thinking about this is that when the database encounters multiple matching rows it behaves similar to a cross-join.

2. When using JOIN, label each of the tables that you are joining on based either on:

   (a) Their location or position ("LHS", "RHS", etc.)

   (b) Their contents/the data that they contain

   Naming tables in this way leads to increased readability.

3. In terms of efficiency, joins should be undertaken in the following order:

   (i) inner

   (ii) left

   (iii) outer

   (iv) cross

There are two major reasons for this order: (1) Readability (we read left to right and combining left and right joins creates difficult to understand queries) and (2) Query optimization (which we will touch upon later).

4. Be consistent with USING/ON/WHERE. I recommend using WHERE for filtering conditions, ON for matching and USING only if it makes sense. Mixing and matching yields difficult to understand queries.
5. No Nulls in join columns. Nulls do not match each other, so joining on a null always returns false! In other words, if you do a left join, the Nulls on the left are kept while the Nulls on the right are dropped.

4 Intermediate Joins

In this section we examine two common patterns around joins: (1) using cross joins to find missing data and (2) using joins with an aggregation to create datasets.

4.1 Aggregations on-self

- Consider trying to figure out what percentage of cars use EZ pass, outbound, by hour of the day. In other words we want to calculate the total number of cars which use EZ pass outbound each hour, take the sum and then divide each row. In order to do this we use a join:

```
SELECT
    hr, perhr::float / tot as pct
FROM
    (select sum(vehiclesez) as perhr, hr from cls.mta
    where direction = 'I' group by 2) as lhs
CROSS JOIN
    (select sum(vehiclesez) as tot from cls.mta
    where direction = 'I') as rhs


  hr         pct
----  ----------
   0  0.0155521
   1  0.00869534
   2  0.00568274
   3  0.00519111
   4  0.0080607
[...]
```

- What if we calculate this percentage, but by plaza? In this case we do a similar operation, but we now we join based on plaza:

```
SELECT
    plaza, hr, perhr::float / tot as pct
FROM
    (select sum(vehiclesez) as perhr, hr, plaza from cls.mta
    where direction = 'I' group by 2,3) as lhs
JOIN
    (select sum(vehiclesez) as tot, plaza from cls.mta
    where direction = 'I'
    group by plaza) as rhs
USING(plaza);


  plaza    hr        pct
-------  ----  ---------
      1     0  0.019748
      2     0  0.0135566
      3     0  0.0181986
      4     0  0.005301
      5     0  0.0143782
[...]
```

4.2 Cross Joins for missing values

- A CROSS JOIN can be useful when looking for missing data or trying to fill-in data. Let's consider the case where we want to verify that there is no missing data within the MTC table. In order to do this we can create a synthetic table which should have all values:

```
select * from
(select distinct mtadt from cls.mta ) as lhs
cross join
(select distinct hr from cls.mta ) as rhs1
cross join
(select distinct plaza from cls.mta ) as rhs2
cross join
(select distinct direction from cls.mta) as rhs3


mtadt          hr    plaza  direction
----------   ----  -------  -----------
2016-08-06     11       11  O
2016-08-06     11       11  I
2016-08-06     11        8  O
2016-08-06     11        8  I
2016-08-06     11        9  O
[...]
```

Each of the subqueries above contains the unique values for the particular column and cross joining them creates a dataset containing every possible combination of the three columns. We can then join this back against the original data to see if there are any missing values:

```
select rhs2.plaza, count(1)
    from
(select distinct mtadt from cls.mta ) as lhs
    cross join
(select distinct hr from cls.mta ) as rhs1
    cross join
(select distinct plaza from cls.mta ) as rhs2
    cross join
(select distinct direction from cls.mta) as rhs3
    left join cls.mta
using(mtadt, hr, plaza, direction)
    where mta.mtadt is null
group by 1
order by count(1) desc;


  plaza    count
-------  -------
     11    61536
      4     2400
      8      240
      5       48
      1       48
[...]
```

From this we can see that there are a number of missing observations and the plazas which are missing!

5 Statistical Analysis in SQL

In this section we will calculate a number of different features of the stock data.

1. Calculate the variance of the closing price of each stock for the year 2010. In particular, write a query which returns one row per stock and two columns: the symbol and the estimated variance of the closing price.
   - In this case we will use the formula that the variance of a variable is equal to:

     $$\begin{aligned} VAR[X] &= E[(X - \bar{X})^2] \\ &= \frac{1}{n} \sum_{i=1}^{n} \left(X_i - \bar{X}\right)^2 \end{aligned}$$

     which we will estimate over our data.[3]
   - The difficult part of computing this value is that we need to make sure that on each row of our dataset is the appropriate $\bar{X}$, which we deal with by calculating it in a separate query and then joining back on the original stocks data, as can be seen in the query below:

[3] While some authors divide by $n-1$ rather than n in the formula, we will stick with n as it makes only a small difference in our numbers and changing to $n-1$ can be easily accomplished using the same method.

```
select
    lhs.symb, avg( (rhs.cls - avg_cls)^2) as est_var
from
    (select avg(cls) as avg_cls, symb
        from stocks.s2010 group by 2) as lhs
join
    stocks.s2010 as rhs
on lhs.symb = rhs.symb
group by 1;


symb       est_var
------  ----------
A         6.00884
AA       22.6683
AAME      0.0532238
AAN       3.29018
AAON      0.423454
[...]
```

2. Stocks can have very large order of magnitude differences in key characteristics, such as volume and price. In order to complete analysis on these values, statisticians often normalize the values. One such normalization is the Z-score, which involves taking each value, subtracting off its mean and dividing by the standard deviation:

$$Z_i = \frac{x_i - \bar{x}}{\sigma_x}$$

where $\bar{x}$ is the mean and σ_x is the standard deviation of the variable in question.

Another method of normalization is linear, where the minimum takes on the value 0 and the maximum takes on the value of 1. This linear transformation can be computed as follows:

$$L_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

- Let's calculate the linear normalization of the closing price for each stock individually. Specifically I want to return the date, symbol, closing price and normalized closing price for each stock.
- Just like in the previous example, we need to compute an aggregate (in this case, the minimum and maximum values of the closing price and join it back to the original data. Once this is complete we can then apply the formula.

```
select
    lhs.symb, retdate, rhs.cls,
        (rhs.cls - lhs.min_cls) / (lhs.max_cls - lhs.min_cls) as n_cls
from
    (select max(cls) as max_cls, min(cls) as min_cls, symb
        from stocks.s2010
        group by symb) as lhs
join
    stocks.s2010 as rhs
on lhs.symb = rhs.symb
order by 1,2,3;


symb    retdate         cls     n_cls
------  ----------  -------  --------
A       2010-01-04  22.3891  0.289632
A       2010-01-05  22.1459  0.26689
A       2010-01-06  22.0672  0.259531
A       2010-01-07  22.0386  0.256857
A       2010-01-08  22.0315  0.256193
[...]
```

- Note that running the above query will fail! Why? Because some stocks have min and max closing prices which are equal. In order to avoid this, we can remove those rows:

```
select
    lhs.symb, retdate, rhs.cls,
        (rhs.cls - lhs.min_cls) /(lhs.max_cls - lhs.min_cls) as n_cls
from
    (select max(cls) as max_cls, min(cls) as min_cls, symb
        from stocks.s2010
        group by symb) as lhs
join
    stocks.s2010 as rhs
on lhs.symb = rhs.symb
where min_cls != max_cls
order by 1,2;


symb    retdate         cls     n_cls
------  ----------  -------  --------
A       2010-01-04  22.3891  0.289632
A       2010-01-05  22.1459  0.26689
A       2010-01-06  22.0672  0.259531
A       2010-01-07  22.0386  0.256857
A       2010-01-08  22.0315  0.256193
[...]
```

3. Calculate the β and α coefficients of a simple linear regression of price on volume.

    - As a reminder, if we run a simple linear regression of the form

$$y = \alpha + \beta x$$

, then our estimated coefficients are equal to:

$$
\begin{aligned}
\hat{\beta} &= \frac{COV(X,Y)}{VAR(X)} \\
&= \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2} \\
\hat{\alpha} &= \bar{Y} - \hat{\beta}\bar{X}
\end{aligned}
$$

- We can calculate this using the following query:

```
select
    beta, acls - beta * avol as alpha
from
    (select
        avg( (cls - acls) * (vol - avol) ) / avg( (vol - avol)^2) as beta
        , max( avol ) as avol
        , max (acls) as acls
    from
        (select avg(cls) as acls, avg(vol) as avol
            from stocks.s2010 ) as lhs
    cross join
        stocks.s2010 ) as IQ;


       beta    alpha
-----------  -------
0.000754955  1571.76
```

- Alternatively, we could be a bit more clever to remove the outer query:

```
select
    avg( (cls - acls) * (vol - avol) ) / avg( (vol - avol)^2) as beta
    , max(acls) - avg( (cls - acls) * (vol - avol) ) / avg( (vol - avol)^2) * max(avol) as alpha
    , max( avol ) as avol
    , max (acls) as acls
from
    (select avg(cls) as acls, avg(vol) as avol
        from stocks.s2010 ) as lhs
    cross join
        stocks.s2010;


       beta    alpha         avol     acls
-----------  -------  -----------  -------
0.000754955  1571.76  1.49898e+06  2703.42
```

This calculation retains the same information as the previous, but avoids using an inner query.

4. We can also calculate the R^2 for this regression using the following formula:

$$
\begin{aligned}
R^2 &= 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \\
&= 1 - \frac{\sum_{i=1}^{n}(y_i - (\alpha + \beta x_i))^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}
\end{aligned}
$$

- To complete this, we start with the previous query which generated alpha and beta and modify it to keep both the average volume and average closing price. We then CROSS JOIN this single row against the original stocks data:

```
select
    max( alpha) as alpha
    , max( beta) as beta
    , 1 - avg( (cls - (alpha + beta * vol))^2) /avg( (cls - acls)^2) as r2
from
(select
    avg( (cls - acls) * (vol - avol) ) / avg( (vol - avol)^2) as beta
    , max(acls) - avg( (cls - acls) * (vol - avol) ) / avg( (vol - avol)^2) * max(avol) as alpha
    , max( avol ) as avol
    , max (acls) as acls
from
    (select avg(cls) as acls, avg(vol) as avol
        from stocks.s2010 ) as lhs
    cross join
        stocks.s2010) as lhs
cross join
    stocks.s2010 as rhs;


  alpha         beta          r2
-------  -----------  ----------
1571.76  0.000754955  0.00121957
```

- Sadly our results are quite poor. The R^2 that I get is equal to 0.00122.

5. Why don't we repeat the analysis, this time *by stock*?

   - Let's first compute our α and $\hat{\beta}$ by stock:

```
select
    lhs.symb
    , avg( (cls - acls) * (vol - avol) ) / avg( (vol - avol)^2) as beta
    , max(acls) - avg( (cls - acls) * (vol - avol) ) / avg( (vol - avol)^2) * max(avol) as alpha
    , max( avol ) as avol
    , max (acls) as acls
from
    (select symb, avg(cls) as acls, avg(vol) as avol, count(1) as ct
        from stocks.s2010 group by 1) as lhs
    left join
        stocks.s2010 as rhs
    on lhs.symb = rhs.symb
    where ct > 100
   group by 1;


symb            beta     alpha              avol      acls
------  ------------  --------  ----------------  --------
ABC     -3.18636e-08  30.316         3.63933e+06  30.2001
CLF     -1.22468e-06  67.6353        5.69978e+06  60.6549
SRDX    -4.75616e-06  16.2657   107476            15.7546
PDLI    -3.88147e-10   6.00289       2.73978e+06   6.00183
VIAB    -3.80486e-07  35.7868        4.41476e+06  34.107
[...]
```

   notice that the query above added an additional filter to remove stocks with less than 100 data points. There are about 250 total trading days in our dataset, so by adding this filter we remove those stocks which are only in the data a small number of times. This also avoid any potential divide by zero error.

   - We can also compute our r^2 by stock, as demonstrated below. Once again, we limit ourselves to stocks which have more than 100 data points.

```
select
    lhs.symb
    , max( alpha) as alpha
    , max( beta) as beta
    , 1 - avg( (cls - (alpha + beta * vol))^2) / avg( (cls - acls)^2) as r2
from
    (select
        lhs.symb
        , avg( (cls - acls) * (vol - avol) ) / avg( (vol - avol)^2) as beta
        , max(acls) - avg( (cls - acls) * (vol - avol) ) / avg( (vol - avol)^2) * max(avol) as alpha
        , max( avol ) as avol
        , max (acls) as acls
    from
        (select symb, avg(cls) as acls, avg(vol) as avol, count(1) as ct
            from stocks.s2010 group by 1) as lhs
        left join
            stocks.s2010 as rhs
        on lhs.symb = rhs.symb
        where ct > 100
    group by 1) as lhs
left join
    stocks.s2010 as rhs
on lhs.symb = rhs.symb
group by lhs.symb
order by r2 desc;


symb       alpha          beta        r2
------  --------  ------------  --------
HOV       3.4266    3.46633e-07  0.612527
MGIC      2.37857   1.78381e-06  0.582692
HPJ       3.54329   7.42622e-06  0.55797
FTK       1.36355   8.33776e-07  0.524931
NUV      10.1132   -1.03103e-06  0.516196
[...]
```

6. Why not with scaled parameters?

```
with sd as (
select
    lhs.symb, retdate
        , (rhs.cls - lhs.min_cls) / (lhs.max_cls - lhs.min_cls) as ncls
        , (rhs.vol::float - lhs.min_vol) / (lhs.max_vol - lhs.min_vol) as nvol
from
    (select max(cls) as max_cls, min(cls) as min_cls
        , symb, max(vol) as max_vol, min(vol) as min_vol
        from stocks.s2010
        group by symb) as lhs
join
    stocks.s2010 as rhs
on lhs.symb = rhs.symb
where min_cls <> max_cls and min_vol <> max_vol
)

select
    lhs.symb
    , max( alpha) as alpha
    , max( beta) as beta
    , 1 - avg( (ncls - (alpha + beta * nvol))^2) / avg( (ncls - acls)^2) as r2
from
    (select
        lhs.symb
        , avg( (ncls - acls) * (nvol - avol) ) / avg( (nvol - avol)^2) as beta
        , max(acls) - avg( (ncls - acls) * (nvol - avol) ) / avg( (nvol - avol)^2) * max(avol) as alpha
        , max( avol ) as avol
        , max (acls) as acls
    from
        (select symb, avg(ncls) as acls, avg(nvol) as avol, count(1) as ct
            from sd group by 1) as lhs
        left join
            sd as rhs
        on lhs.symb = rhs.symb
        where ct > 100
    group by 1) as lhs
left join
    sd as rhs
on lhs.symb = rhs.symb
group by lhs.symb
order by r2 desc;


symb        alpha       beta        r2
------  ---------  ---------  --------
HOV     0.0333546   0.823546  0.612527
MGIC    0.125339    1.27091   0.582692
HPJ     0.0803212   0.976241  0.55797
FTK     0.076173    1.08304   0.524931
NUV     0.858118   -1.1852    0.516196
[...]
```

7. What about Z-scaled? STOPPED HERE.

```
with sd as (
select
    lhs.symb, retdate
        , (rhs.cls - lhs.min_cls) / (lhs.max_cls - lhs.min_cls) as ncls
        , (rhs.vol::float - lhs.min_vol) / (lhs.max_vol - lhs.min_vol) as nvol
from
    (select max(cls) as max_cls, min(cls) as min_cls
        , symb, max(vol) as max_vol, min(vol) as min_vol
        from stocks.s2010
        group by symb) as lhs
join
    stocks.s2010 as rhs
on lhs.symb = rhs.symb
where min_cls <> max_cls and min_vol <> max_vol
)

select
    lhs.symb
    , max( alpha) as alpha
    , max( beta) as beta
    , 1 - avg( (ncls - (alpha + beta * nvol))^2) / avg( (ncls - acls)^2) as r2
from
    (select
        lhs.symb
        , avg( (ncls - acls) * (nvol - avol) ) / avg( (nvol - avol)^2) as beta
        , max(acls) - avg( (ncls - acls) * (nvol - avol) ) / avg( (nvol - avol)^2) * max(avol) as alpha
        , max( avol ) as avol
        , max (acls) as acls
    from
        (select symb, avg(ncls) as acls, avg(nvol) as avol, count(1) as ct
            from sd group by 1) as lhs
        left join
            sd as rhs
        on lhs.symb = rhs.symb
        where ct > 100
    group by 1) as lhs
left join
    sd as rhs
on lhs.symb = rhs.symb
group by lhs.symb
order by r2 desc;


symb        alpha       beta        r2
------  ---------  ---------  --------
HOV     0.0333546   0.823546  0.612527
MGIC    0.125339    1.27091   0.582692
HPJ     0.0803212   0.976241  0.55797
FTK     0.076173    1.08304   0.524931
NUV     0.858118   -1.1852    0.516196
[...]
```