# SQL: Aggregation and Grouping

## Computing on a column

So far, we've learned how to choose the rows and columns we want, and to compute things on individual cell values. We often want to compute something across *all* the values in a column. For example, we might wish to know the total of a column of sales figures.

SQL provides functions, such as `sum`, `avg`, `min`, `max` and `count`, that can be used in a SELECT clause to apply to a column. We call this aggregation. (My dictionary tells me that aggregation comes from root words meaning "towards" and "flock", so when you aggregate, you can imagine that you are herding your data.)

Here is an example showing the basics of aggregation.

```
-- First we use aggregation to get the average of all the grades in the Took
-- table.  Notice that the result has just one tuple, because there is
-- just one average:
csc343h-prof=> SELECT avg(grade)
csc343h-prof-> FROM Took;
         avg
---------------------
 75.8545454545454545
(1 row)

-- Notice also that SQL invented a name for this column: the name of the
-- aggregation function.  If we do some fancier calculation, it uses the
-- column name "?column?":
csc343h-prof=> SELECT max(grade) - min(grade)
csc343h-prof-> FROM Took;
 ?column?
----------
      100

-- Let's give that column a better name:
csc343h-prof=> SELECT max(grade) - min(grade) AS range
csc343h-prof-> FROM Took;
 range
-------
   100
(1 row)

-- count(grade) counts the number of values in the grade column.
csc343h-prof=> SELECT count(grade)
csc343h-prof-> FROM Took;
```

```
csc343h-prof-> FROM Took;
 count
-------
    55
(1 row)

-- If we give count a wildcard instead of a specific column name, we get
-- the number of rows in the table.
csc343h-prof=> SELECT count(*)
csc343h-prof-> FROM Took;
 count
-------
    55
(1 row)
```

`count(grade)` and `count(*)` give the same result here: there are as many values in the `grade` column as there are rows in the table. We will learn soon that SQL allows empty cells, and in that case, `count(grade)` -- or counting any other column -- and `count(*)` could report different values.

## When duplicates contribute

Duplicate values contribute to aggregations. For example, if the grade 87 occurs 12 times in the table, all 12 contribute to `avg(grade)`, as well as to `count(grade)` and all the other aggregations we can compute. Of course these duplicates make no difference to `max(grade)` -- 87 either is or is not the maximum; the same holds for `min(grade)`.

Including duplicate values makes sense when we average a column of grades, but it is not always what we want. For example, suppose we used the Offering table to find out how many departments there are (CSC, ENV, etc.):

```
csc343h-prof=> SELECT count(dept)
csc343h-prof-> FROM Offering;
 count
-------
    36
(1 row)
```

Every occurence of "CSC" contributed to the count, as did every occurrence of "HIS", and so on. We couldn't get the answer we wanted. But if we put the keyword `DISTINCT` inside the call to `count`, each different value contributes only once.

```
csc343h-prof=> SELECT count(DISTINCT dept)
csc343h-prof-> FROM Offering;
 count
-------
     6
```

```
(1 row)
```

DISTINCT can also be used with the other aggregation functions. It does not affect `min` or `max`, of course, but does affect `sum` and `avg`.

## Including multiple aggregations

We can include more than one aggregation in a query. For example:

```
csc343h-prof=> SELECT max(grade), min(grade), count(distinct oid), count(*)
csc343h-prof-> FROM Took;
 max | min | count | count
-----+-----+-------+-------
 100 |   0 |    23 |    54
(1 row)
```

It doesn't matter that the aggregations don't all pertain to the same column, and one isn't even about a single column. It *does* matter that we are dealing with like quantities: a single max(grade), a single min(grade), and so on. This allows SQL to produce a structurally sound table for our result.

Consider this:

```
csc343h-prof=> SELECT sid, avg(grade)
csc343h-prof-> FROM Took;
ERROR:  column "Took.sid" must appear in the GROUP BY clause or be used in an aggregate function
LINE 1: SELECT sid, avg(grade) FROM Took;
               ^
```

Here we are combining `sid` (of which there are many) with `avg(grade)` (of which there is one). That doesn't produce a structurally sound table, which is why we get an error. The error message itself will make sense once we understand grouping.

## Grouping

Suppose we are interested in getting a sense of each student's overall grades. We might sort the Took table to get a look:

```
csc343h-prof=> SELECT sid, grade
csc343h-prof-> FROM Took
csc343h-prof-> ORDER BY sid;

  sid  | grade
-------+-------
   157 |    99
```

```
        157 |    82
        157 |    59
        157 |    72
        157 |    89
        157 |    39
        157 |    90
        157 |    98
        157 |    59
        157 |    71
        157 |    71
        157 |    91
        157 |    82
        157 |    62
        157 |    75
      11111 |    40
      11111 |     0
      11111 |    17
      11111 |    46
      11111 |    45
      98000 |    82
      98000 |    89
      98000 |    72
      . . . etc.
```

We might then like to know each student's average. A sensible table can be formed from pairs of an sid and an average grade. Let's see how to get that.

If we follow a SELECT-FROM-WHERE expression with GROUP BY the tuples that have the same value for that attribute will be treated as a group, and **one row will be generated for each group**. For example, if we `GROUP BY sid`, all tuples about sid 157 will be collapsed down and represented as one row in the result, and the same for sid 11111 and all the others.

Let's try that. In other words, let's change the ORDER BY to GROUP BY:

```
    csc343h-prof=> SELECT sid, grade
    csc343h-prof-> FROM Took
    csc343h-prof-> GROUP BY sid;
    ERROR:  column "Took.grade" must appear in the GROUP BY clause or be used in an aggregate functi
on
    LINE 1: SELECT sid, grade
                        ^
```

What went wrong? We asked for one row per sid, and we want it to include sid and grade. There is one sid per sid (of course!), but there are potentially many grades per sid. This doesn't allow SQL to make a structurally sound, rectangular, table to hold the results. But if we ask for the average grade per student, there will be exactly one per student, and the query will make sense:

```
csc343h-prof=> SELECT sid, avg(grade)
csc343h-prof-> FROM Took
csc343h-prof-> GROUP BY sid;
  sid  |         avg
-------+---------------------
 11111 | 29.6000000000000000
 98000 | 83.2000000000000000
 99132 | 76.2857142857142857
 99999 | 84.5833333333333333
   157 | 75.9333333333333333
(5 rows)

-- We can order this table too.  Here's one order.
csc343h-prof=> SELECT sid, avg(grade)
csc343h-prof-> FROM Took GROUP BY sid
csc343h-prof-> ORDER BY sid;
  sid  |         avg
-------+---------------------
   157 | 75.9333333333333333
 11111 | 29.6000000000000000
 98000 | 83.2000000000000000
 99132 | 76.2857142857142857
 99999 | 84.5833333333333333
(5 rows)

-- Here we order by the aggregated column.
csc343h-prof=> SELECT sid, avg(grade)
csc343h-prof-> FROM Took
csc343h-prof-> GROUP BY sid
csc343h-prof-> ORDER BY avg(grade);
  sid  |         avg
-------+---------------------
 11111 | 29.6000000000000000
   157 | 75.9333333333333333
 99132 | 76.2857142857142857
 98000 | 83.2000000000000000
 99999 | 84.5833333333333333
(5 rows)
```

We can even order by something not in the SELECT clause. Here's a query that tries to do this, but breaks:

```
csc343h-prof=> SELECT sid, avg(grade)
csc343h-prof-> FROM Took
csc343h-prof-> GROUP BY sid
csc343h-prof-> ORDER BY oid;
ERROR:  column "Took.oid" must appear in the GROUP BY clause or be used in an aggregate function
LINE 2: ORDER BY oid;
                 ^
```

The problem is not that oid doesn't appear in the SELECT clause. The problem is that we grouped according to sid, so every group has one sid but it doesn't necessarily have one oid. How could we order the groups according to oid when a group may have more than one oid?

In contrast, this query works because there is exactly one count(oid) per group:

```
csc343h-prof=> SELECT sid, avg(grade)
csc343h-prof-> FROM Took
csc343h-prof-> GROUP BY sid
csc343h-prof-> ORDER BY count(oid);
  sid  |         avg
-------+---------------------
 11111 | 39.3333333333333333
 99132 | 76.2857142857142857
 99999 | 84.5833333333333333
 98000 | 83.2000000000000000
   157 | 75.9333333333333333
(5 rows)
```

Let's add `count(oid)` to the `SELECT` clause so that we can observe that the order is correct:

```
csc343h-prof=> SELECT sid, avg(grade), count(oid)
csc343h-prof-> FROM Took
csc343h-prof-> GROUP BY sid
csc343h-prof-> ORDER BY count(oid);
  sid  |         avg         | count
-------+---------------------+-------
 11111 | 29.6000000000000000 |     5
 99132 | 76.2857142857142857 |     7
 99999 | 84.5833333333333333 |    12
 98000 | 83.2000000000000000 |    15
   157 | 75.9333333333333333 |    15
(5 rows)
```

## Grouping by multiple columns

We can group by *multiple* columns. Again, let's start out with ORDER BY, as it is a good warm-up to GROUP BY.

```
-- Order by 1 attribute.
-- All ANT rows come together, then all CSC rows, and so on.
csc343h-prof=> SELECT *
csc343h-prof-> FROM Offering
csc343h-prof-> ORDER BY dept;
 oid | cnum | dept | term  | instructor
-----+------+------+-------+------------
```

```
      11 |  200 | ANT  | 20089 | Zorich
      12 |  203 | ANT  | 20089 | Davies
      31 |  203 | ANT  | 20081 | Zorich
       4 |  320 | CSC  | 20089 | Jepson
       5 |  207 | CSC  | 20089 | Craig
       6 |  207 | CSC  | 20089 | Gries
       7 |  148 | CSC  | 20089 | Jepson
       8 |  148 | CSC  | 20089 | Chechik
    etc.

    -- Order by 2 attributes.
    -- This result is the same, except that when there is a tie by dept,
    -- the rows for that dept are in ORDER BY cnum.
    csc343h-prof=> SELECT *
    csc343h-prof-> FROM Offering
    csc343h-prof-> ORDER BY dept, cnum;
     oid | cnum | dept | term  | instructor
    -----+------+------+-------+------------
      11 |  200 | ANT  | 20089 | Zorich
      31 |  203 | ANT  | 20081 | Zorich
      12 |  203 | ANT  | 20089 | Davies
       7 |  148 | CSC  | 20089 | Jepson
      27 |  148 | CSC  | 20081 | Jepson
       8 |  148 | CSC  | 20089 | Chechik
      28 |  148 | CSC  | 20081 | Miller
      25 |  207 | CSC  | 20081 | Craig
    etc.
```

Now let's try grouping by the same two attributes. This won't work:

```
    csc343h-prof=> SELECT *
    csc343h-prof-> FROM Offering
    csc343h-prof-> GROUP BY dept, cnum;
    ERROR:  column "Offering.oid" must appear in the GROUP BY clause or be used in an aggregate func
tion
    LINE 1: SELECT * FROM Offering GROUP BY dept, cnum;
```

By saying "`GROUP BY dept, cnum`", we asked for one row per dept-cnum combination, such as ENV-200. But we also asked to include columns `oid`, `term`, and `instructor`. Yet we can't expect a dept-cnum combination to have just one `oid`, `term`, and `instructor`, so the query doesn't make sense. If we were to aggregate those columns by dept and cnum, there *would* be just one value per dept-cnum combination. For example, here we ask for `count(cnum)` rather than `cnum` itself, so the query makes perfect sense:

```
    csc343h-prof=> SELECT dept, cnum, count(cnum)
    csc343h-prof-> FROM Offering
    csc343h-prof-> GROUP BY dept, cnum;
```

```
 dept | cnum | count
------+------+-------
 ENV  |  200 |     1
 ENG  |  235 |     2
 CSC  |  207 |     4
 EEB  |  216 |     1
 EEB  |  263 |     2
 HIS  |  296 |     1
 CSC  |  343 |     5
 HIS  |  220 |     2
 CSC  |  263 |     3
 ENG  |  205 |     2
 EEB  |  150 |     1
 ANT  |  203 |     2
 CSC  |  320 |     2
 CSC  |  148 |     4
 ENG  |  110 |     2
 ENV  |  320 |     1
 ANT  |  200 |     1
(17 rows)
```

If we wish to order the results of our query, the ORDER BY clause comes after the GROUP BY clause:

```
csc343h-prof=> SELECT dept, cnum, count(cnum)
csc343h-prof-> FROM Offering
csc343h-prof-> GROUP BY dept, cnum
csc343h-prof-> ORDER BY dept;
 dept | cnum | count
------+------+-------
 ANT  |  200 |     1
 ANT  |  203 |     2
 CSC  |  148 |     4
 CSC  |  207 |     4
 CSC  |  263 |     3
 CSC  |  320 |     2
 CSC  |  343 |     5
 EEB  |  150 |     1
 EEB  |  216 |     1
 EEB  |  263 |     2
 ENG  |  110 |     2
 ENG  |  205 |     2
 ENG  |  235 |     2
 ENV  |  200 |     1
 ENV  |  320 |     1
 HIS  |  220 |     2
 HIS  |  296 |     1
(17 rows)
```