CS61A Lecture 34

Wednesday, November 20, 2019

Announcements

- Scheme project due today.
- Next week will be fully online.

Aggregate Functions

So far, all SQL expressions have referred to the values in a single row at a time.

```
select [columns] from [table] where [expression] order by [expression]
```

We've only been able to refer to values from 2 separate rows if we joined tables. An **aggregate function** in the `[columns]` clause computes a value from a group of rows.

```
create table animals as
  select "dog" as kind, 4 as legs, 20 as weight union
  select "cat"        , 4        , 10           union
  select "ferret"     , 4        , 10           union
  select "parrot"     , 2        , 6            union
  select "penguin"    , 2        , 10           union
  select "t-rex"      , 2        , 12000;

select max(legs) from animals;
```

max(legs)
4

(Demo)

animals:

kind	legs	weight
dog	4	20
cat	4	10
ferret	4	10
parrot	2	6
penguin	2	10
t-rex	2	12000

We can combine expressions too:

```
$ sqlite3 -init animals.sql
SQLite
> SELECT MAX(legs) FROM animals;
4
> SELECT MAX(legs-weight) FROM animals;
-4
> SELECT MAX(legs), MIN(weight) FROM animals;
4|6
```

The values in separate columns are computed independently as displayed by the last example; there is no single animal that has 4 legs and a weight of 6.

Another kind of function is the `COUNT` function:

```
> SELECT COUNT(legs) FROM animals;
6
> SELECT COUNT(DISTINCT legs) FROM animals;
2
```

Making Aggregate Functions and Single Values

An aggregate function also selects some row in the table to supply the values of columns that are not aggregated. In the case of `MAX` or `MIN`, this row is that of the `MAX` or `MIN` value. Otherwise, it is arbitrary.

```
> SELECT max(weight), kind FROM animals
. --This selects the right "kind" (T-Rex), which will be returned ;
12000 | t-rex
```

This only applies when the maximum or minimum value is unique. If not, you won't really know which non-unique value the system will pull out. As such, try not to combine aggregated and non-aggregated values in the same statement.

> **Discussion Question**
> What are all the kinds of animals that have the maximal number of legs?
>
> **Solution**
>
> ```
> SELECT kind FROM animals WHERE legs = 4;
> ```
>
> Well, this works, but the problem is we had to hard-code the 4 into the solution. Let's fix that.
>
> ```
> > SELECT kind FROM animals WHERE legs = MAX(legs);
> ```
>
> Oops, that doesn't work! Aggregate functions must go into the column description, not anywhere else. Let's use a statement to find the maximum number of legs, then give it a name. We can't give individual names values, but we can put it inside a table and give that a name.
>
> ```
> > CREATE TABLE maxtable AS
> . SELECT max(legs) AS maxlegs FROM animals;
> > SELECT kind, legs, maxlegs from animals, maxtable;
> cat|4|4
> dog|4|4
> ferret|4|4
> parrot|2|4
> penguin|2|4
> t-rex|2|4
> > SELECT kind
> . FROM animals, maxtable
> . WHERE legs = maxlegs
> cat
> dog
> ferret
> ```

Grouping

Rows in a table can be grouped, so we can aggregate only on certain groups.

```
SELECT [column] FROM [table] GROUP BY [expression] HAVING [expression];
```

The `GROUP BY` statement lets you separate the data into groups of values with unique values of `[expression]`, and `HAVING` lets you filter the groups by those having only specific values.

```
> SELECT legs, MAX(weight) FROM animals GROUP BY legs;
4|20
2|12000
```

The `[expression]` doesn't have to be a single value. You can, for example, `GROUP BY` the weight-to-leg ratio:

```
> SELECT weight/legs, COUNT(*) FROM animals GROUP BY weight/legs;
2|2
3|1
5|2
6000|1
```

Notice this doesn't return decimal vlaues: the cat and the ferret both have a ratio of 2.5, but the table shows them as 2. We can fix this if you need to:

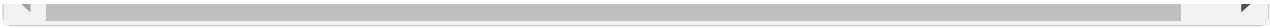
```
> SELECT weight*1.0/legs, COUNT(*) FROM animals GROUP BY weight/legs;
2.5|2
3.0|1
5.0|2
6000.0|1
```

You can put aggregation functions in the `HAVING` clause. A `HAVING` clause filters the set of groups that are aggregated:

```
SELECT weight/legs, COUNT(*) FROM aniamls GROUP BY weight/legs HAVING count(*)>1;
5|2
2|2
```

SQL performs the grouping, then filters the results for those that don't match the requirements in the `HAVING` clause.

> **Discussion Question**
> What's the maximum difference between leg count for two animals with the same weight?
>
> **Solution**
> Here's a bad way:
>
> ```
> > SELECT MAX(legs)-MIN(legs) AS diff FROM animals GROUP BY weight ORDER BY -diff
> 2
> ```
>
> But remember that we can join tables to themselves to compare data between two individual rows:
>
> ```
> > SELECT MAX(a.legs-b.legs) FROM animals AS a, animals AS b WHERE a.weight=b.weig
> 2
> ```



> This solution doesn't involve grouping at all! Sometimes, it may just be cleaner to not use it at all! Think about the best way to solve your problem.

Big Game

SQL can take a comma-separated value (CSV) file and turn it into a SQL database:

```
> CREATE table big(
.   year INTEGER,
.   location TEXT,
.   stanford INTEGER,
.   cal INTEGER);
> .mode csv
> .import big.csv big
```

And we can analyze the data from the `big` table as we usually do!