CS61A Lecture 33

Monday, November 18, 2019

Announcements

- No live lab or office hours during Thanksgiving week: 11/25 and 11/26.
- Video-only lecture Monday 11/25.
- Optional Scheme Recursive Art Contest entries are due Monday 12/2.

More SQL

Where SQL becomes interesting is when you have two tables with different information, allowing you to combine them and get info from both.

As we saw from Lecture 32, we used the example of **John the Patriotic Dog Breeder**. But what if we wanted more info on each dog, and not just which dogs are parents and children?

Joining Two Tables

Two tables `A` and `B` are joined by a comma to yield all combos of a row from `A` and a row from `B`.

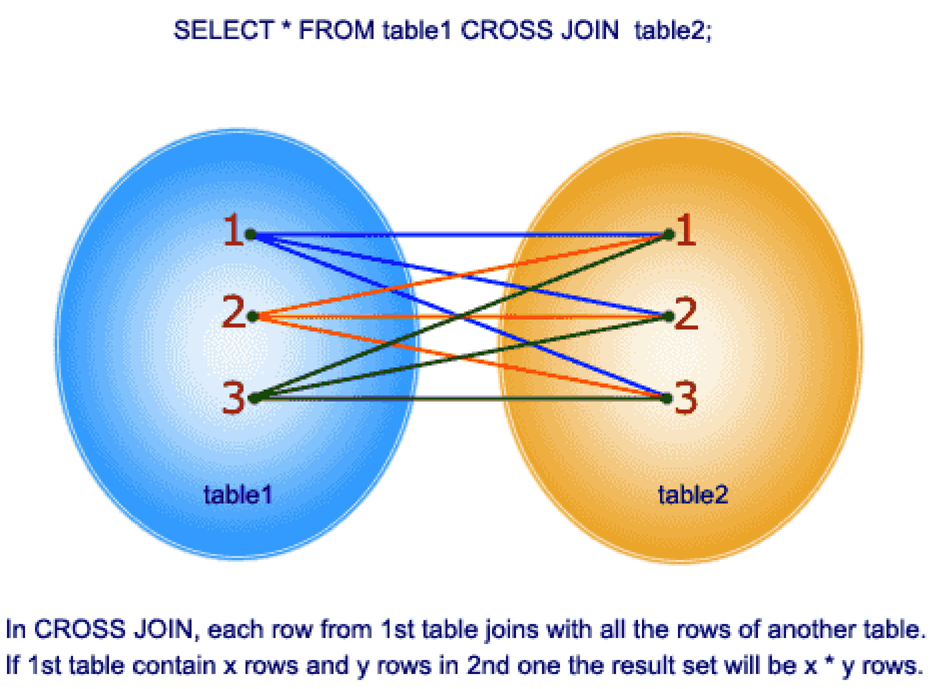
```
CREATE TABLE dogs AS
    SELECT "abraham" AS name, "long" AS fur UNION
    SELECT "barack" , "short" UNION
    SELECT "clinton" , "long" UNION
    SELECT "delano" , "long" UNION
    SELECT "eisenhower" , "short" UNION
    SELECT "fillmore" , "curly" UNION
    SELECT "grover" , "short" UNION
    SELECT "herbert" , "curly";

CREATE TABLE parents AS
    SELECT "abraham" AS parent, "barack" AS child UNION
    SELECT "abraham" , "clinton" UNION
    ...;
```

For example, if we wanted to select the parents of curly-furred dogs:

```
SELECT parent FROM parents, dogs
                WHERE child = name AND fur = "curly";
```

The process, called a cross join, works by combining every possible pairing of data points between the two tables.


SELECT * FROM table1 CROSS JOIN table2;
1
2
3
1
2
3
table1
table2
In CROSS JOIN, each row from 1st table joins with all the rows of another table.
If 1st table contain x rows and y rows in 2nd one the result set will be x * y rows.

It's very common that when you join the two tables, you use a `WHERE` statement to select only the important info we want.

SQL processes things row-by-row, so you have to join the tables first in order to filter the final table using info that's available in both tables.

Joining a Table with Itself

Why would you want to do this? Well, again, since SQL processes things row-by-row, you need to join the same table to itself if, for example, you want to filter based on info available in two separate rows.

Here's an example: select all pairs of siblings!

So far, we've used tables with unique column names. However, we can use dot expressions and aliases to disambiguate column values:

```
SELECT [columns] FROM [table] WHERE [condition] ORDER BY [order];
```

`[table]` is a comma-separated list of table names with optional aliases with the `AS` statement, which assigns a name to a table:

To select all pairs of siblings:

```
SELECT a.child as first, b.child as second
    FROM parents AS a, parents AS b
    WHERE a.parent = b.parrent AND a.child < b.child;
```

The first line builds a new table with two columns, both originally named `child`, but given the names `first` and `second`. The second line performs the join, as well as gives aliases to the tables so we can disambiguate which column names comes from which table. And the third column is a standard `WHERE` statement that filters the results; `a.child<b.child` just ensures we're always returning unique pairs, instead of for example, returning double pairs `A,D` and `D,A`.

(SQL will complain if you try to join two of the same table without giving them aliases. It will return an `ambigious column name` error, because it doesn't know how to refer to the two values)

Joining More Than Two Tables

Multiple tables can be joined to yield all combinations of rows from each. For exmaple, select all grandparents with the same fur as their grandchildren.

We need three tables, a `grandparents` table, and two `dogs` tables, one for the grandparent, one for the grandchild. First, let's formalize the `grandparents` table from before:

```
CREATE TABLE grandparents AS
    SELECT a.parent AS grandog, b.child AS granpup
    FROM parents AS a, parents as b
    WHERE b.parent = a.child;
```

And now, let's join the three tables:

```
SELECT grandog FROM grandparents, dogs AS c, dogs AS d
    WHERE grandog = c.name AND
          granpup = d.name AND
          c.fur = d.fur;
```

Here's another example: write a SQL query that selects all possible combinations of three different dogs with the same fur and lists each triple in inverse alphabetical order.

```
SELECT a.name, b.name, c.name
    FROM dogs AS a, dogs AS b, dogs A c
    WHERE a.fur = b.fur AND b.fur = c.fur
    AND a.name > b.name AND b.name > c.name;
```

Numerical Expressions

Expressions can contain function calls and arithmetic operators, such as:

- Combiners: `+`, `-`, `*`, `/`, `%`, `and`, `or`
- Transformers: `abs`, `round`, `not`, `-`
- Comparers: `<`, `<=`, `>`, `>=`, `<>`, `!=`, `=` (`<>` and `!=` both mean "not equals")

String Expressions

String values can be combined to form longer strings with the `||` operator (not `+`!):

```
> SELECT "hello," || "world";
hello, world
```

Basic string manipulation is built into SQL, but differs from Python. People use these in unique ways, using the `substr` function, but this is not required knowledge for 61A:

```
> CREATE TABLE phrase AS SELECT "hello, world" AS s;
> SELECT substr(s, 4, 2) || substr(s, instr(s, " ")+1, 1) FROM phrase;
low
```

And people have also used SQL to represent data structures like those we've seen in Python and Scheme, but don't do this!

```
> CREATE TABLE lists AS SELECT "one" AS car, "two,three,four" AS cdr;
> SELECT substr(cdr, 1, instr(cdr, ",")-1) AS cadr FROM lists;
two
```