CS 61A

Declarative Programming: SQL

Fall 2019

Lecturer: John Denero *Akilesh Praveen*

1 Declarative Languages

1.1 Why Declarative?

Systems is a general area of research that works on things like OS (the most important application), but the runner-up is a **database management system**.

A database is a collection of tables, and tables are where you put that data. A DBMS is what helps you manipulate that.

The **Structured Query Language** is the most widely used language that we use to extract information that we want to use from these database, so we can make decisions based on that information.

SQL is a declarative language.

1.2 Declarative vs. Imperative

In **declarative languages** like SQL and Prolog, a 'program' is a description of a result that you want, and it's the interpreter's job to figure out how to generate that result.

In **imperative languages** such as Python and Java, a 'program' is a description of computational processes. The interpreter simply carries out execution based on preset evaluation rules.

For this reason, **there's more flexibility in a declarative langauge interpreter**.

For example: In an imperative language, if you write a quadratic time algorithm by specifying a computational process, it'll probably just run in quadratic time. However, in a declarative language, you simply tell the interpreter what you want, and it'll decide which of the options that it has (e.g. linear, quadratic, etc.) in order to compute what you want as efficiently as possible.

> A lot of the interesting research in declarative programming involves deciding what methods to use to produce the data you want based on the queries you make, and how to make that as efficient as possible.

1.3 Structured Query Language

SQL is super common, just because lots of people do database management.

- A **select** statement creates a new table, either from scratch or by *projecting* a table. Always composed of a comma separated list of column descriptions.

  As such, the following would create a two-column table of literals.

```
select [expression] as [name], [expression] as [name}
```

  To create a multi-row table of literals, just union these statements together.

```
select "abraham" as parent, "barack" as child union
select "abraham" as parent, "clinton" as child;
```

- A **create table** statement gives a global name to a table. A select statement displays a result to the user, but if you want to give it a name and **store it**, you just use a create table command.

```
create table parents as
    select "abraham" as parent, "barack" as child union
    select "abraham" as parent, "clinton" as child;
```

All SQL statements end in semicolons.

Lots of other statements exist, but they're not too important for understanding the heart of how SQL works.

2 Projecting Tables

`select` statements can specify an *input*, using a `from` clause. For example, you can do the following:

```
select [columns] from [table] where [condition] order by [order];
```

Here, the `[columns]` label I made refers to a column description, which can be the same as the `select` statements I made above with abraham, barack and clinton.

Every select statement creates a new table with rows and columns. So you could say something like this:

```
select parents from parents where parent = "abraham";
```

3 Arithmetic

Select expressions can perform arithmetic.

When we make a select expression, we can also include arithmetic in it. This can be done when we're projecting from another table to do some neat calculations.

```
create table lift as
    select 101 as chair, 2 as single, 2 as couple union
    select 102         , 0          , 3            union
    select 103         , 4          , 1;

select chair, single + 2 * couple as total from lift;
```

We can see SQL used here to do arithmetic to produce a table of the total amount of riders.

The above SQL commands would produce the following table.

chair	total
101	6
102	6
103	6

Union works the same way as it does in set theory, where it discards duplicates.