SQL I

from berkeley cs61a

Declarative Programming

Programming Paradigms

Up until now, we've been focused (primarily) on **imperative** programming.

Imperative program contain explicit instructions to tell the computer **how** to accomplish something. The interpreter then executes those instructions

Now, we'll learn about **declarative programming**, where we can just tell the computer **what** we want, instead of how we want it done. The interpreter then figures out how to accomplish that.

Declarative programs are often specialized to perform a specific task, because they allow for repetitive computation to be abstracted away and for the interpreter to optimize its execution

Imperative vs. Declarative

Suppose you're going to a restaurant for dinner and need a table

Option 1:

"First I need to find a table, so I'll look through every available table and pick the one in the best location that has enough seats for my group, then I'll need to find someone to wait on me and make sure I have enough menus, then…"

Option 2:

"Table for two!"

SQL

SQL & Database Languages

SQL is an example of a (declarative) language with interacts with a **database management system (DBMS)** in order to make data processing easier and faster

It collects records into **tables,** or a collection of rows with a value for each column

A **row** has a value for each column

titude	Longitude	Name
38	122	Berkeley
42	71	Cambridge
45	93	Minneapolis

A **column** has a name and a type

A table has columns and rows

Tables in SQL (Sqlite, a small SQL database engine)

```
CREATE TABLE cities AS
  SELECT 38 AS latitude, 122 AS longitude, "Berkeley" AS name UNION
  SELECT 42,             71,               "Cambridge"        UNION
  SELECT 45,             93,               "Minneapolis";
```

```
SELECT "west coast" AS region, name FROM cities WHERE longitude >= 115 UNION
SELECT "other",                name FROM cities WHERE longitude < 115;
```

Cities:

Latitude	Longitude	Name
38	122	Berkeley
42	71	Cambridge
45	93	Minneapolis

Region	Name
west coast	Berkeley
other	Minneapolis
other	Cambridge

SQL Basics

The SQL language varies across implementations but we will look at some shared concepts.

- A SELECT statement creates a new table, either from scratch or by taking information from an existing table
- A CREATE TABLE statement gives a global name to a table.
- Lots of other statements exist: DELETE, INSERT, UPDATE etc…
- Most of the important action is in the SELECT statement.

Today's theme:

Full credit to John DeNero for the examples in today's lecture

Using SQL

Can download SQL at https://sqlite.org/download.html

Just want to follow along or try out some examples? Go to sql.cs61a.org. It also has all of today's examples loaded.

Selecting Value Literals

Demo_1

A SELECT statement always includes a comma-separated list of column descriptions.

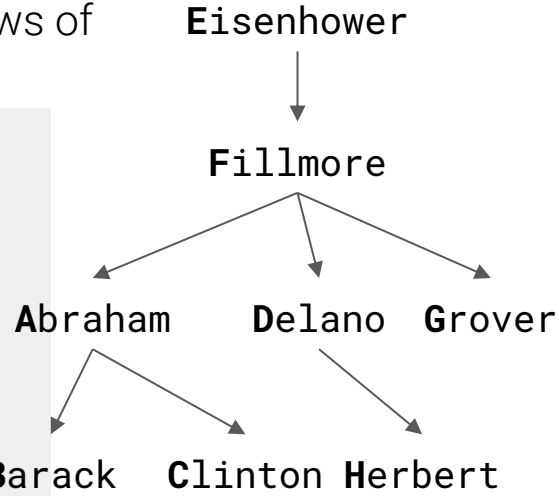
A column description is an expression, optionally followed by AS and a column name.

SELECT [expression] AS [name], [expression] AS [name], ... ;

SELECTing literals CREATEs a one-row table.

The UNION of two SELECT statements is a table containing the rows of both of their results.

```
SELECT "delano" AS parent,  "herbert" AS child UNION
SELECT “abraham”,           "barack"           UNION
SELECT "abraham",           "clinton“          UNION
SELECT "fillmore",          "abraham"          UNION
SELECT "fillmore",          "delano"           UNION
SELECT "fillmore",          “grover"           UNION
SELECT "eisenhower",        "fillmore";
```



Naming Tables

Demo_2

SQL is often used as an interactive language.

The result of a SELECT statement is displayed to the user, but not stored.

A CREATE TABLE statement gives the result a name.

CREATE TABLE [name] AS [SELECT statements];

```
CREATE TABLE parents AS
  SELECT "delano" AS parent,  "herbert" AS child UNION
  SELECT “abraham”,           "barack"           UNION
  SELECT "abraham",           "clinton“          UNION
  SELECT "fillmore",          "abraham"          UNION
  SELECT "fillmore",          "delano"           UNION
  SELECT "fillmore",          “grover"           UNION
  SELECT "eisenhower",        "fillmore";
```

Parents:

Parent	Child
delano	herbert
abraham	barack
abraham	clinton
fillmore	abraham
fillmore	delano
fillmore	grover
eisenhower	fillmore

Selecting From Tables

SELECT Statements Project Existing Tables

Demo_3

A SELECT statement can specify an input table using a FROM clause.

A subset of the rows of the input table can be selected using a WHERE clause.

Can declare the order of the remaining rows using an ORDER BY clause. Otherwise, no order

Column descriptions determine how each input row is projected to a result row:

SELECT [columns] FROM [table] WHERE [condition] ORDER BY [order] [ASC/DESC] LIMIT [number];

```
sqlite> SELECT * FROM parents ORDER BY parents DESC;
sqlite> SELECT child FROM parents WHERE parent = "abraham";
sqlite> SELECT parent FROM parents WHERE parent > child;
```

Only use ASC/DESC if there's order by

WHERE and ORDER BY are optional

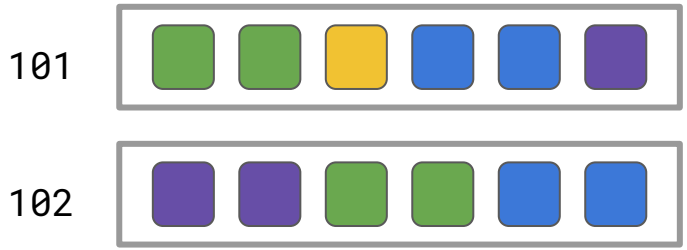
Arithmetic in SELECT Statements

In a SELECT expression, column names evaluate to row values.
Arithmetic expressions can combine row values and constants

```
CREATE TABLE restaurant AS
  SELECT 101 AS table,  2 AS single,  2 AS couple UNION
  SELECT 102          ,  0          ,  3          UNION
  SELECT 103          ,  3          ,  1;
```

```
sqlite> SELECT table, single + 2 * couple AS
total FROM restaurant;
```

table	total
101	6
102	6
103	5



Given a table ints that describes how to sum powers of 2 to form various integers.

```
CREATE TABLE ints AS
  SELECT "zero" AS word, 0 AS one, 0 AS two, 0 AS four, 0 AS eight UNION
  SELECT "one"         , 1       , 0       , 0        , 0          UNION
  SELECT "two"         , 0       , 2       , 0        , 0          UNION
  SELECT "three"       , 1       , 2       , 0        , 0          UNION
  SELECT "four"        , 0       , 0       , 4        , 0          UNION
  SELECT "five"        , 1       , 0       , 4        , 0          UNION
  SELECT "six"         , 0       , 2       , 4        , 0          UNION
  SELECT "seven"       , 1       , 2       , 4        , 0          UNION
  SELECT "eight"       , 0       , 0       , 0        , 8          UNION
  SELECT "nine"        , 1       , 0       , 0        , 8;
```

(A) Write a SELECT statement for a two-column table of the word and value for each integer

word	value
zero	0
one	1
...	...

(B) Write a SELECT statement for the word names of the powers of two

Demo_4

word
one
two
...

Joining Tables

Back To Dogs

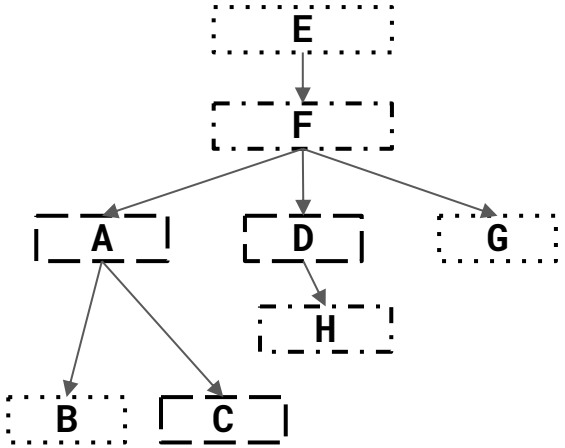
```
CREATE TABLE parents AS
  SELECT "delano" AS parent, "herbert" AS child UNION
  SELECT "abraham",          "barack"           UNION
  SELECT "abraham",          "clinton"          UNION
  SELECT "fillmore",         "abraham"          UNION
  SELECT "fillmore",         "delano"           UNION
  SELECT "fillmore           "grover"           UNION
  SELECT "eisenhower",       "fillmore";
```

Parents:

Parent	Child
delano	herbert
abraham	barack
abraham	clinton
fillmore	abraham
fillmore	delano
fillmore	grover
eisenhower	fillmore

An Example:

```
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";
```



Select the parents of curly-furred dogs

```
CREATE TABLE parents AS
  SELECT "delano" AS parent, "herbert" AS child UNION
  SELECT "abraham",          "barack"           UNION
  ...
```

Parent
eisenhower
delano

Joining Tables

Demo_5

Two tables A & B are joined by a comma to yield all combinations of a row from A & a row from B.

```
SELECT * FROM parents, dogs;
```

Selects all combinations of rows from both tables. We only want the rows for curly haired dogs.

```
SELECT * FROM parents, dogs WHERE fur = "curly";
```

This filters the 56 rows to now only have rows where the fur is curly. But this has rows that have nothing to do with each other. We only care about rows where the two dogs match.

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

The condition on which the tables are joined on is called the **join condition**.

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

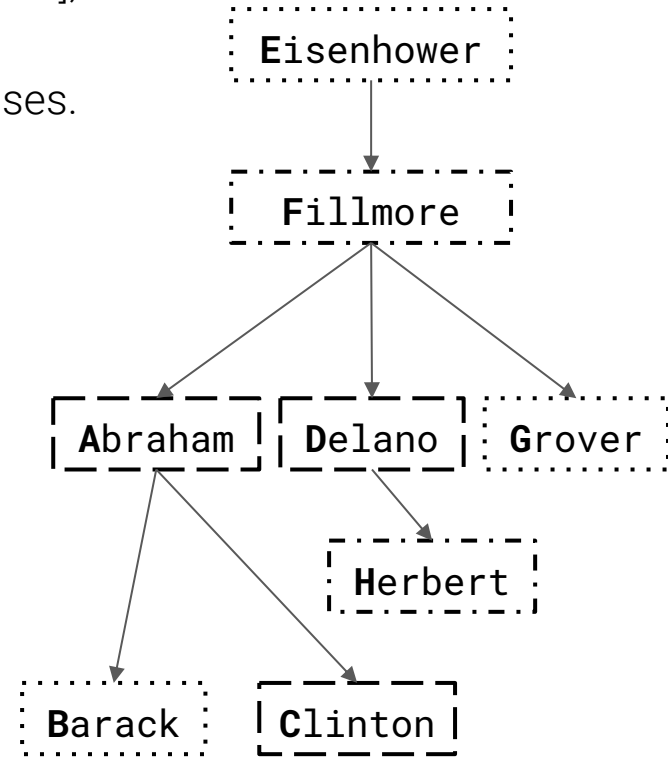
Joining A Table With Itself

Demo_6

Two tables may share a column name; dot expressions and aliases disambiguate column values.

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

[table] is a comma-separated list of table names with optional aliases.



Select all pairs of siblings. No duplicates.

First	Second
barack	clinton
abraham	delano
abraham	grover
delano	grover

Aliasing

```
SELECT * FROM parents, parents;
```

This doesn't work because the tables share a column. Let's fix that!

```
SELECT * FROM parents AS a, parents AS b;
```

This works because now SQL can tell the columns in the two tables apart. Let's now only keep rows where the children share a parent.

```
SELECT * FROM parents AS a, parents AS b WHERE a.parent = b.parent;
```

We need to get rid of duplicates because pairs of siblings appear twice. We can do this by enforcing an arbitrary ordering, a.child < b.child alphabetically. Then we get the two columns we want.

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

String Expressions (If Time)

String Expressions

Demo_7

String values can be combined to form longer strings.

```
sqlite> SELECT "hello," || " world";
hello, world
sqlite> SELECT name || " dog" FROM dogs;
abraham dog
barack dog
clinton dog
delano dog
eisenhower dog
fillmore dog
grover dog
herbert dog
```

Let’s look at an example of this in action!