61A Lecture 31

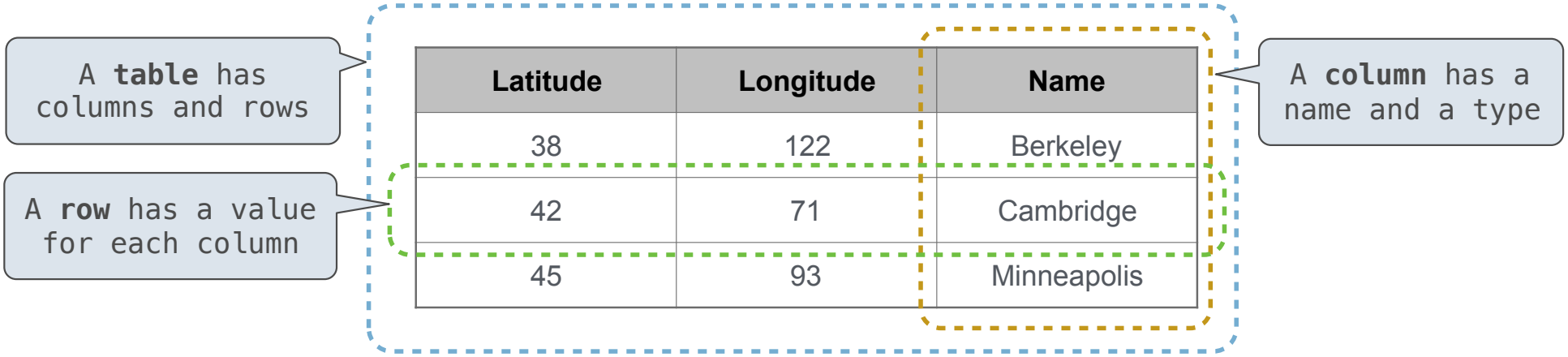
Announcements

Declarative Languages

Database Management Systems

Database management systems (DBMS) are important, heavily used, and interesting!

A table is a collection of records, which are rows that have a value for each column



The Structured Query Language (SQL) is perhaps the most widely used programming language

SQL is a *declarative* programming language

Declarative Programming

In **declarative languages** such as SQL & Prolog:

- A "program" is a description of the desired result
- The interpreter figures out how to generate the result

The interpreter chooses which way is fastest.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes
- The interpreter carries out execution/evaluation rules

Cities:

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

```
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;
```

Region	Name
west coast	Berkeley
other	Minneapolis
other	Cambridge

Structured Query Language (SQL)

SQL Overview

The SQL language is an ANSI and ISO standard, but DBMS's implement custom variants

- A **select** statement creates a new table, either from scratch or by projecting a table
- A **create table** statement gives a global name to a table
- Lots of other statements exist: **analyze**, **delete**, **explain**, **insert**, **replace**, **update**, etc.
- Most of the important action is in the **select** statement

Today's theme:

http://awhimsicalbohemian.typepad.com/.a/6a00e5538b84f3883301538dfa8f19970b-800wi

Getting Started with SQL

Install sqlite (version 3.8.3 or later): http://sqlite.org/download.html

Use sqlite online: http://kripken.github.io/sql.js/GUI/

Selecting Value Literals

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

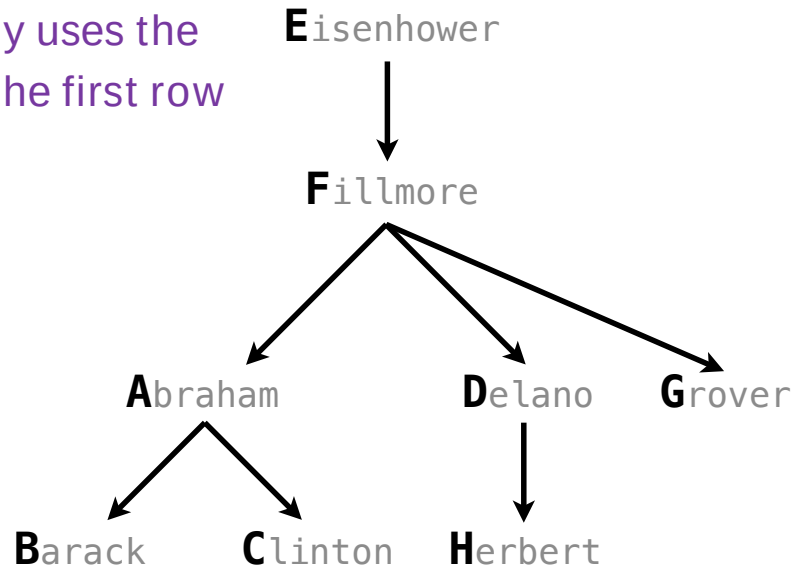
```
select [expression] as [name], [expression] as [name]; ...
```

Selecting literals creates a one-row table

The union of two select statements is a table containing the rows of both of their results

union only uses the name of the first row

```
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

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 statement];
```

```
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
abraham	barack
abraham	clinton
delano	herbert
fillmore	abraham
fillmore	delano
fillmore	grover
eisenhower	fillmore

Projecting Tables

Select Statements Project Existing Tables

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

An ordering over the remaining rows can be declared using an **order by** clause

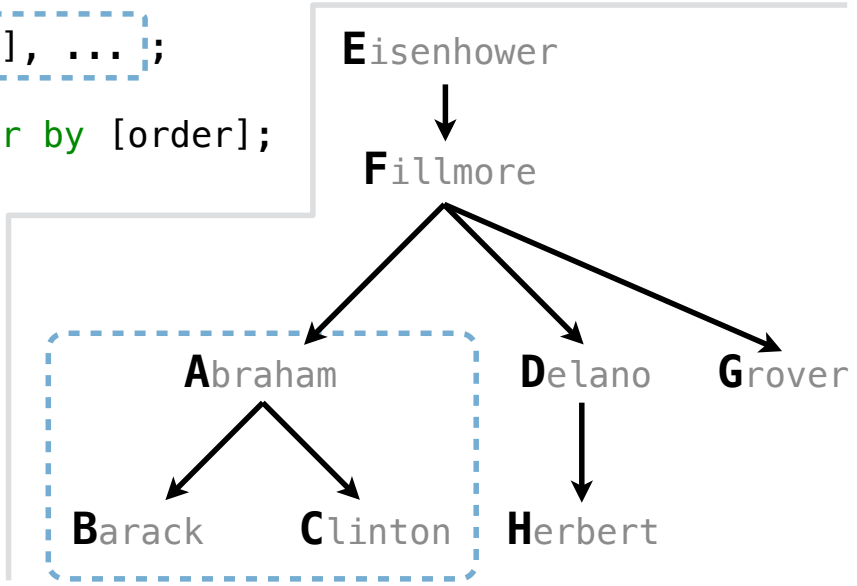
Column descriptions determine how each input row is projected to a result row

```
select [expression] as [name], [expression] as [name], ... ;
select [columns] from [table] where [condition] order by [order];
select child from parents where parent = "abraham";
select parent from parents where parent > child;
```

Child
barack
clinton

Parent
fillmore
fillmore

(Demo)



Arithmetic

Arithmetic in Select Expressions

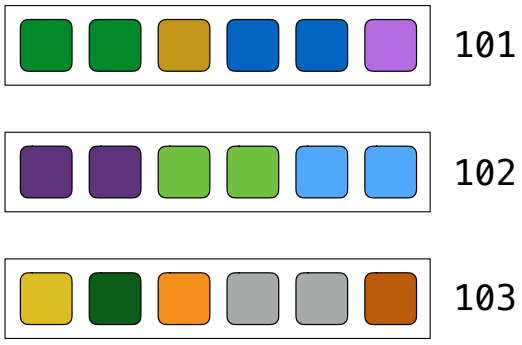
In a select expression, column names evaluate to row values

Arithmetic expressions can combine row values and constants

```
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;
```

chair	total
101	6
102	6
103	6



Discussion Question

Given the 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
two	2
three	3
...	...

(Demo)

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

word
one
two
four
eight

```
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;
```

```
~/lec$ sqlite3 -init ex.sql
-- Loading resources from ex.sql

SQLite version 3.8.10.2 2015-05-
Enter ".help" for usage hints.
sqlite> select * from ints;
eight|0|0|0|8
five|1|0|4|0
four|0|0|4|0
nine|1|0|0|8
one|1|0|0|0
seven|1|2|4|0
six|0|2|4|0
three|1|2|0|0
two|0|2|0|0
zero|0|0|0|0
sqlite>
```

Note: The order a table is presented is different from how we created it. In fact, in this case it's arranged in alphabetic order, for the reason that SQL has checked whether there are repetitions when creating the table. Thus, we can know that declarative language requires the interpreter to decide which method to choose when carrying out statements.

(A)

```
sqlite> select word, one+two+four+eight as value
   ...> from ints;
eight|8
five|5
four|4
nine|9
one|1
seven|7
six|6
three|3
two|2
zero|0
```

Note: Though "one" also exists as an element in the role, select statement only looks for "one" as a column name.

(B)

```
sqlite> select word from ints
   ...> where one + two/2 + four/4 + eight/8 = 1;
eight
four
one
two
```