

61A Lecture 32

Monday, November 17

Announcements

Announcements

- Project 4 due Friday 11/21 @ 11:59pm

Announcements

- Project 4 due Friday 11/21 @ 11:59pm
 - Project party Monday 11/17 6:30pm – 8:30pm in 10 Evans

Announcements

- Project 4 due Friday 11/21 @ 11:59pm
 - Project party Monday 11/17 6:30pm – 8:30pm in 10 Evans
 - Early submission point #2: Questions 1–16 by Tuesday 11/18 @ 11:59pm

Announcements

- Project 4 due Friday 11/21 @ 11:59pm
 - Project party Monday 11/17 6:30pm – 8:30pm in 10 Evans
 - Early submission point #2: Questions 1–16 by Tuesday 11/18 @ 11:59pm
 - Early submission point #3: Submit by Thursday 11/20 @ 11:59pm

Announcements

- Project 4 due Friday 11/21 @ 11:59pm
 - Project party Monday 11/17 6:30pm – 8:30pm in 10 Evans
 - Early submission point #2: Questions 1–16 by Tuesday 11/18 @ 11:59pm
 - Early submission point #3: Submit by Thursday 11/20 @ 11:59pm
- Homework 9 (6 pts) due Wednesday 11/26 @ 11:59pm

Declarative Languages

Database Management Systems

Database Management Systems

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

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

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

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

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

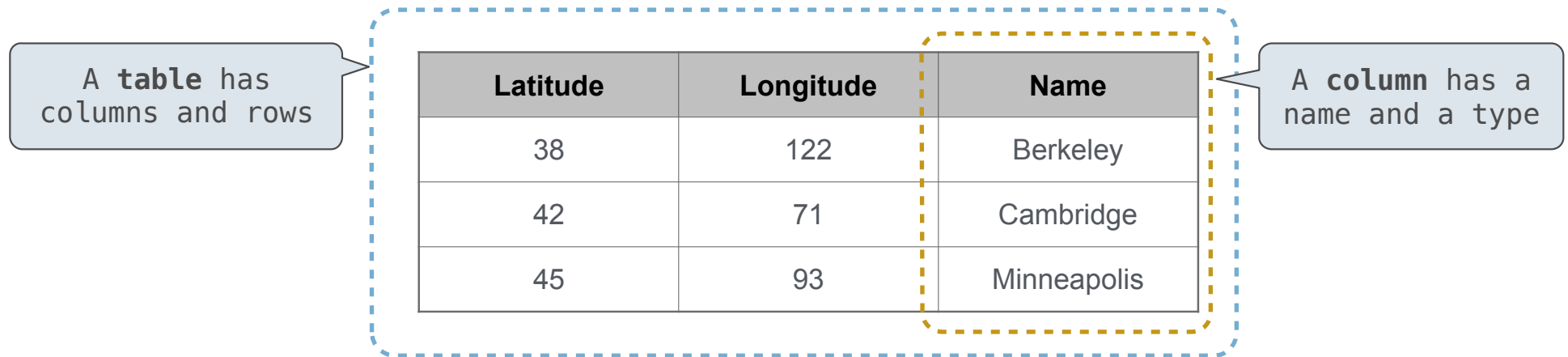
A **table** has
columns and rows

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

Database Management Systems

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

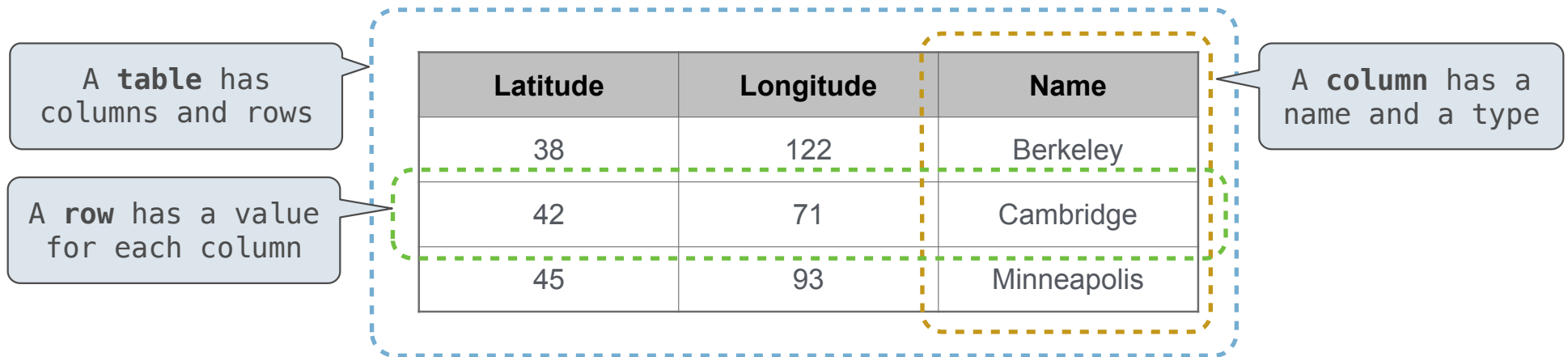
A table is a collection of records, which are rows that have a value for each column



Database Management Systems

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

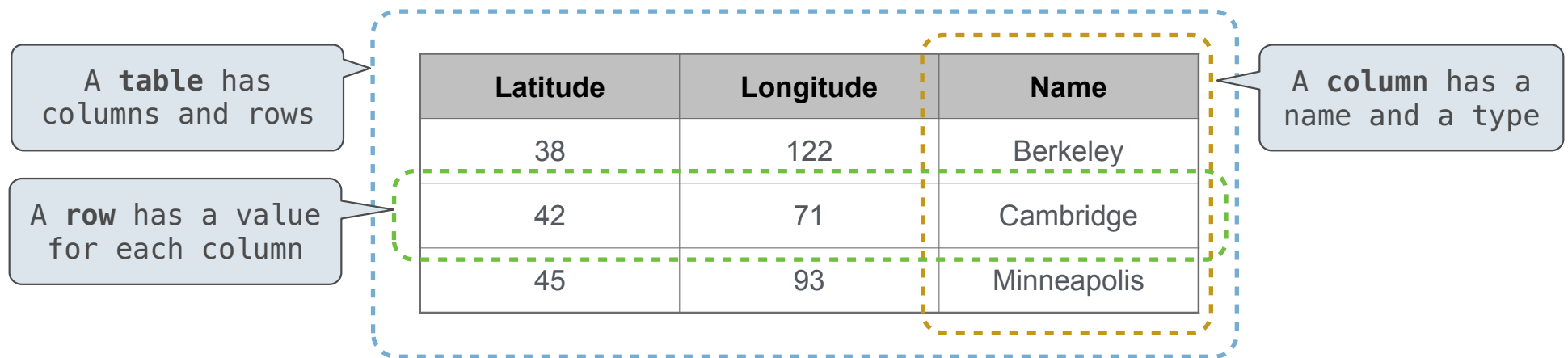
A table is a collection of records, which are rows that have a value for each column



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

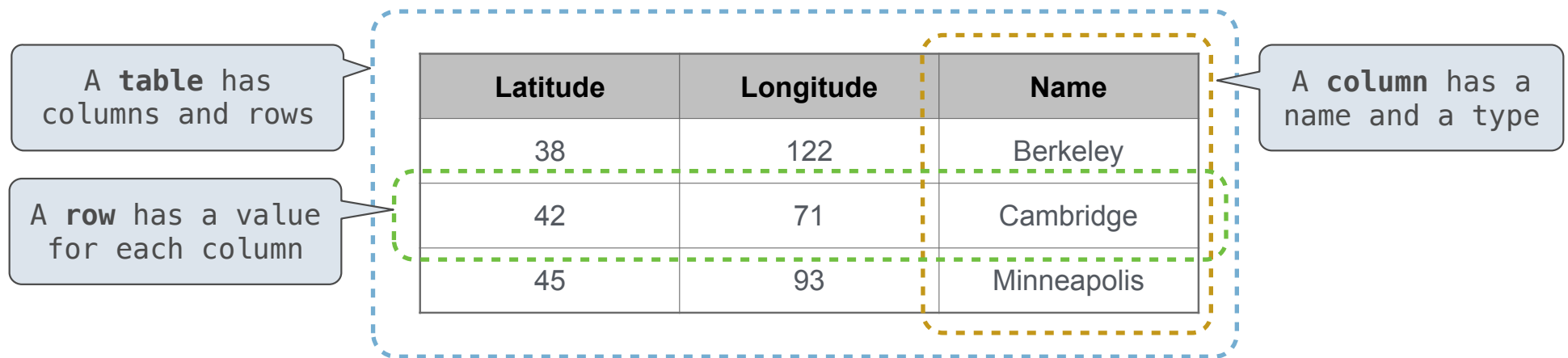


The Structured Query Language (SQL) is perhaps the most widely used programming language

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

Declarative Programming

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

Declarative Programming

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

- A "program" is a description of the desired result

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

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

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

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

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

- A "program" is a description of computational processes

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

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

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

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

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

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

`create table cities as`

Cities:

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

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

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

```
create table cities as
```

```
  select 38 as latitude, 122 as longitude, "Berkeley" as name union
```

Cities:

Latitude	Longitude	Name
38	122	Berkeley

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

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

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

`create table cities as`

```
select 38 as latitude, 122 as longitude, "Berkeley" as name union
select 42,              71,              "Cambridge"      union
```

Cities:

Latitude	Longitude	Name
38	122	Berkeley
42	71	Cambridge

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

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";
```

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

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;
```

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

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

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

```
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

Structured Query Language (SQL)

SQL Overview

SQL Overview

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

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

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

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.

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

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
- The code for executing **select** statements fits on a single sheet of paper (next lecture)

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
- The code for executing **select** statements fits on a single sheet of paper (next lecture)

Today's theme:

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
- The code for executing **select** statements fits on a single sheet of paper (next lecture)

Today's theme:



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/>

Use the SQL example from the textbook: <http://composingprograms.com/examples/sql/sql.zip>

Selecting Value Literals

Selecting Value Literals

A **select** statement always includes a comma-separated list of column descriptions

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

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]
```

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 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 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 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

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

Selecting Value Literals

A **select** statement always includes a comma-separated list of column descriptions

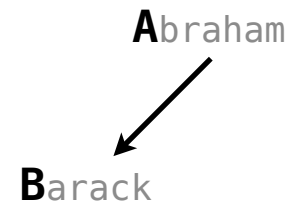
A column description is an expression, optionally followed by **as** and a column name

```
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

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



Selecting Value Literals

A **select** statement always includes a comma-separated list of column descriptions

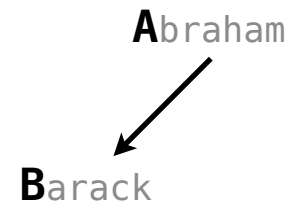
A column description is an expression, optionally followed by **as** and a column name

```
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

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



Selecting Value Literals

A **select** statement always includes a comma-separated list of column descriptions

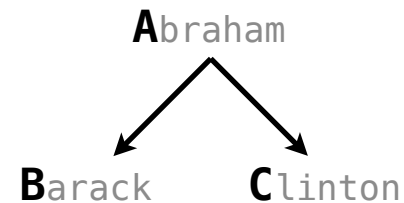
A column description is an expression, optionally followed by **as** and a column name

```
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

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



Selecting Value Literals

A **select** statement always includes a comma-separated list of column descriptions

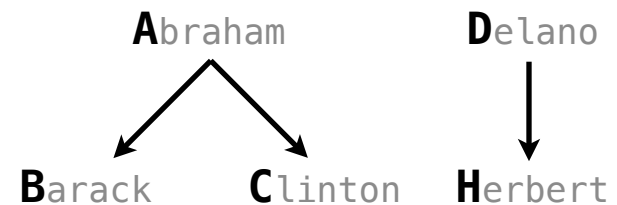
A column description is an expression, optionally followed by **as** and a column name

```
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

```
select "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"        , "herbert"      union
```



Selecting Value Literals

A **select** statement always includes a comma-separated list of column descriptions

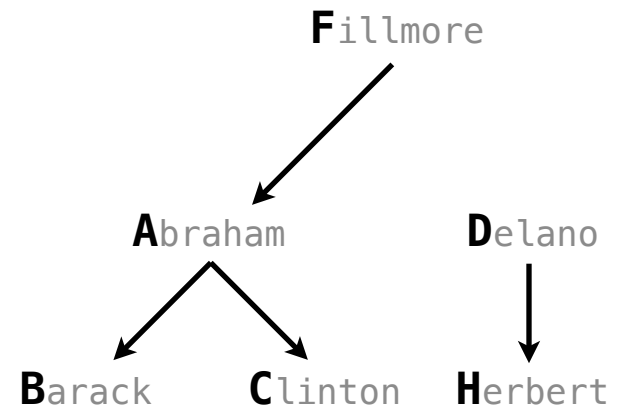
A column description is an expression, optionally followed by **as** and a column name

```
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

```
select "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"       , "herbert"      union
select "fillmore"     , "abraham"      union
```



Selecting Value Literals

A **select** statement always includes a comma-separated list of column descriptions

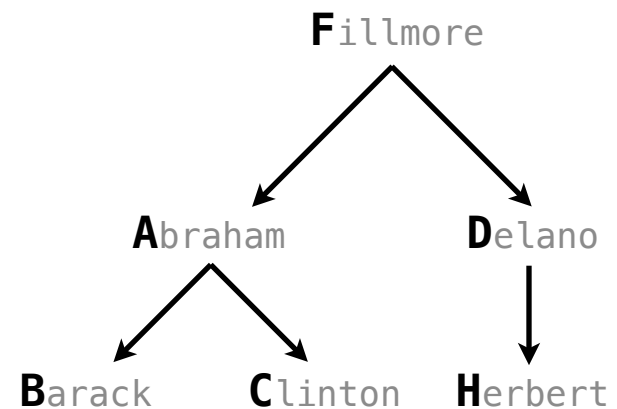
A column description is an expression, optionally followed by **as** and a column name

```
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

```
select "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"       , "herbert"      union
select "fillmore"     , "abraham"     union
select "fillmore"     , "delano"      union
```



Selecting Value Literals

A **select** statement always includes a comma-separated list of column descriptions

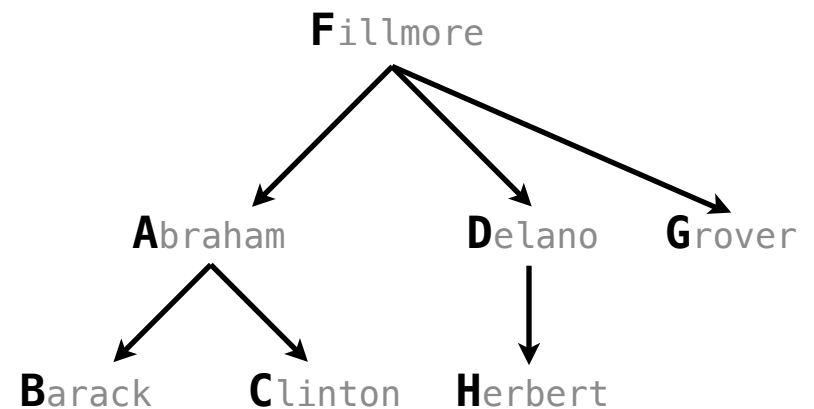
A column description is an expression, optionally followed by **as** and a column name

```
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

```
select "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"       , "herbert"      union
select "fillmore"     , "abraham"      union
select "fillmore"     , "delano"       union
select "fillmore"     , "grover"       union
```



Selecting Value Literals

A **select** statement always includes a comma-separated list of column descriptions

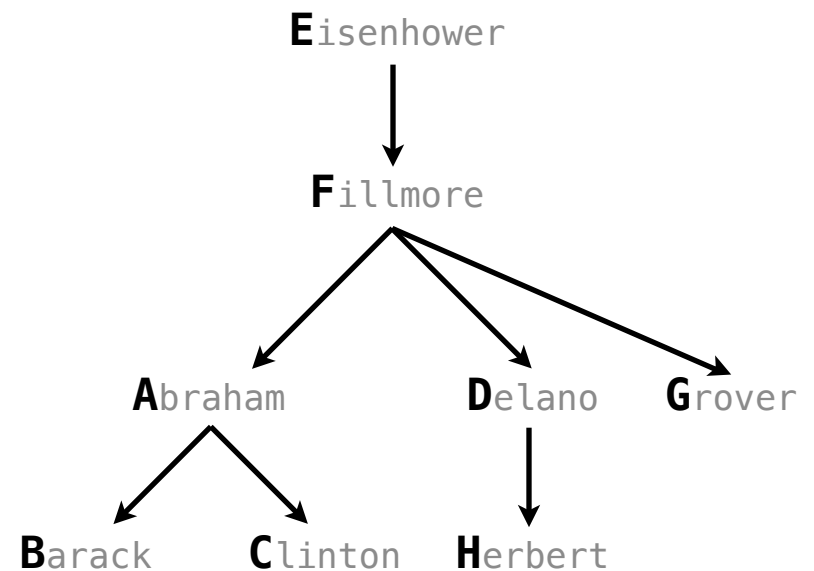
A column description is an expression, optionally followed by **as** and a column name

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

Selecting literals creates a one-row table

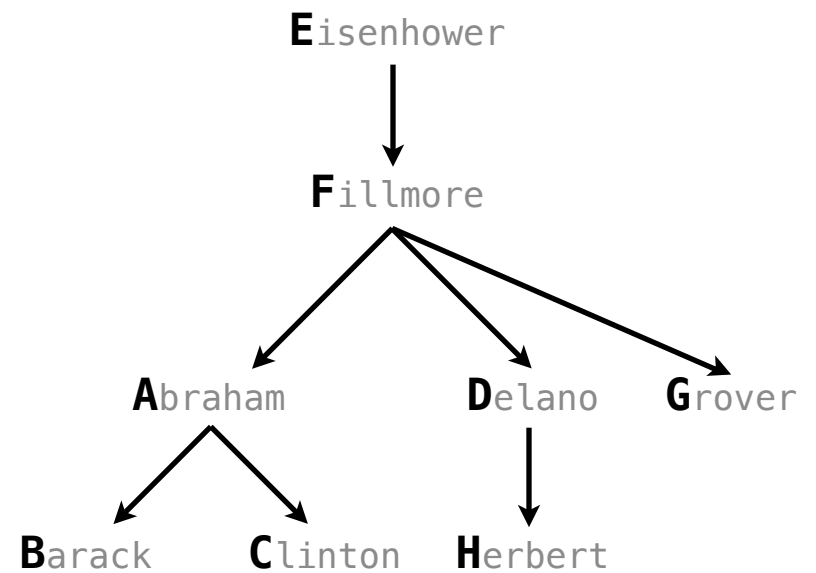
The union of two select statements is a table containing the rows of both of their results

```
select "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"       , "herbert"      union
select "fillmore"     , "abraham"      union
select "fillmore"     , "delano"       union
select "fillmore"     , "grover"       union
select "eisenhower"   , "fillmore";
```



Naming Tables

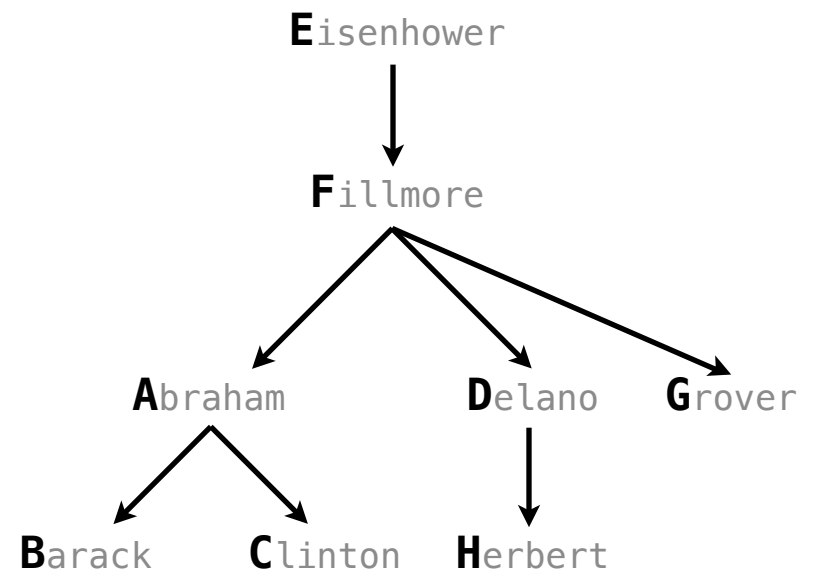
```
select "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"       , "herbert"      union
select "fillmore"     , "abraham"      union
select "fillmore"     , "delano"       union
select "fillmore"     , "grover"       union
select "eisenhower"   , "fillmore";
```



Naming Tables

SQL is often used as an interactive language

```
select "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"       , "herbert"      union
select "fillmore"     , "abraham"     union
select "fillmore"     , "delano"      union
select "fillmore"     , "grover"      union
select "eisenhower"   , "fillmore";
```

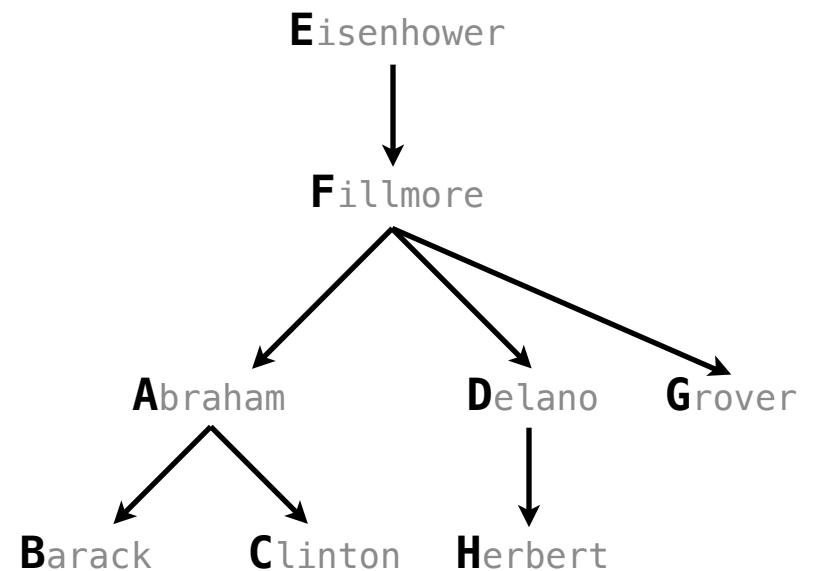


Naming Tables

SQL is often used as an interactive language

The result of a **select** statement is displayed to the user, but not stored

```
select "abraham" as parent, "barack" as child union
select "abraham"          , "clinton"          union
select "delano"           , "herbert"           union
select "fillmore"         , "abraham"           union
select "fillmore"         , "delano"           union
select "fillmore"         , "grover"           union
select "eisenhower"       , "fillmore";
```



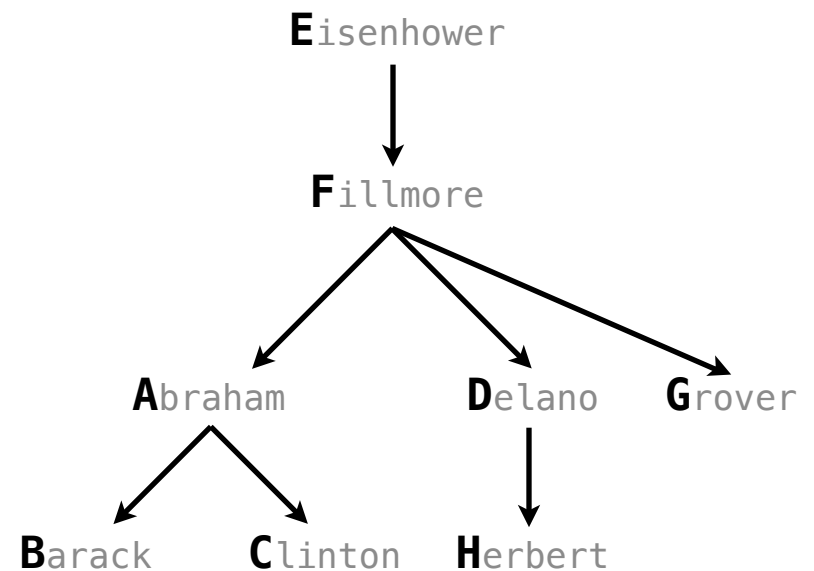
Naming Tables

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

```
select "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"       , "herbert"      union
select "fillmore"     , "abraham"      union
select "fillmore"     , "delano"       union
select "fillmore"     , "grover"       union
select "eisenhower"   , "fillmore";
```



Naming Tables

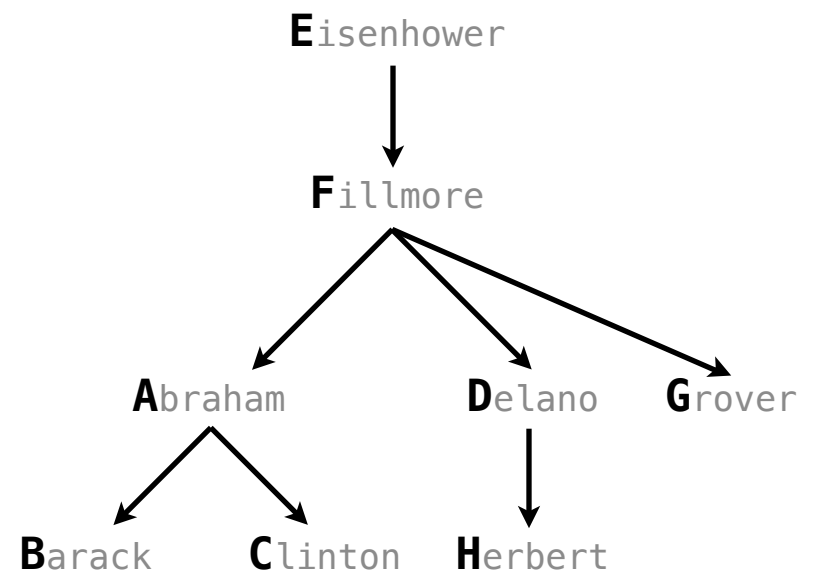
SQL is often used as an interactive language

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];
```

```
select "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"       , "herbert"      union
select "fillmore"     , "abraham"     union
select "fillmore"     , "delano"      union
select "fillmore"     , "grover"      union
select "eisenhower"   , "fillmore";
```



Naming Tables

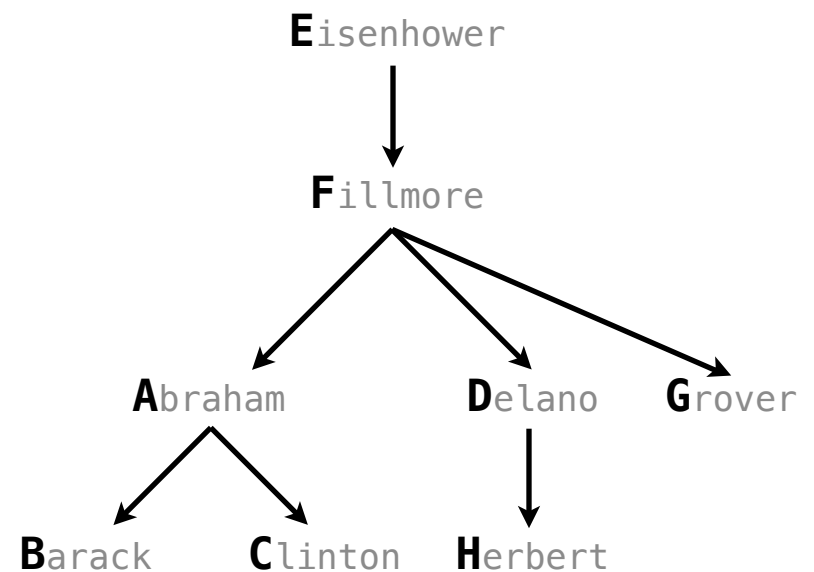
SQL is often used as an interactive language

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 "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"       , "herbert"      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

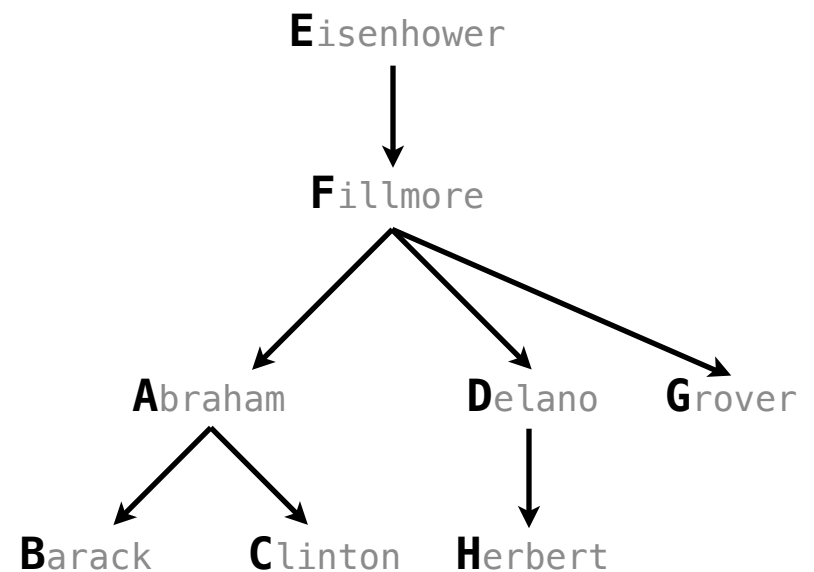
The result of a **select** statement is displayed to the user, but not stored

A **create table** statement gives the result a name

```
create table [name] as [select statement];
```

```
create table parents as
```

```
select "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"        , "herbert"      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 "abraham" as parent, "barack" as child union
select "abraham"      , "clinton"      union
select "delano"       , "herbert"      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

Select Statements Project Existing Tables

A **select** statement can specify an input table using a **from** clause

Select Statements Project Existing Tables

A **select** statement can specify an input table using a **from** clause

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

Select Statements Project Existing Tables

A **select** statement can specify an input table using a **from** clause

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

Select Statements Project Existing Tables

A **select** statement can specify an input table using a **from** clause

```
select [expression] as [name], [expression] as [name], ... ;  
select [columns] from [table] ;
```


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

```
select [expression] as [name], [expression] as [name], ... ;  
select [columns] from [table] ;
```

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

```
select [expression] as [name], [expression] as [name], ... ;  
select [columns] from [table] where [condition] ;
```

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

```
select [expression] as [name], [expression] as [name], ... ;  
select [columns] from [table] where [condition] ;
```

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

```
select [expression] as [name], [expression] as [name], ... ;  
select [columns] from [table] where [condition] order by [order];
```

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 Statements Project Existing Tables

A **select** statement can specify an input table using a **from** clause

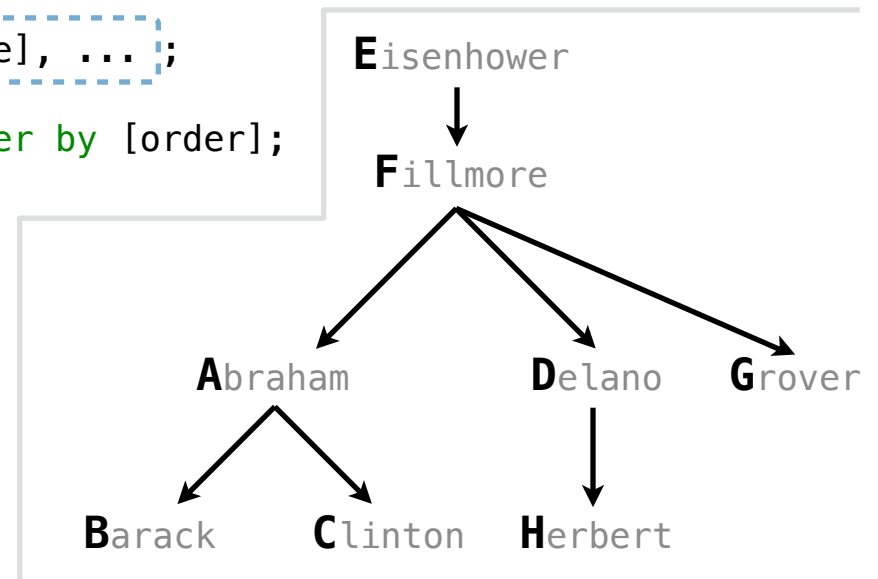
A subset of the rows of the input table can be selected using a **where** clause

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 Statements Project Existing Tables

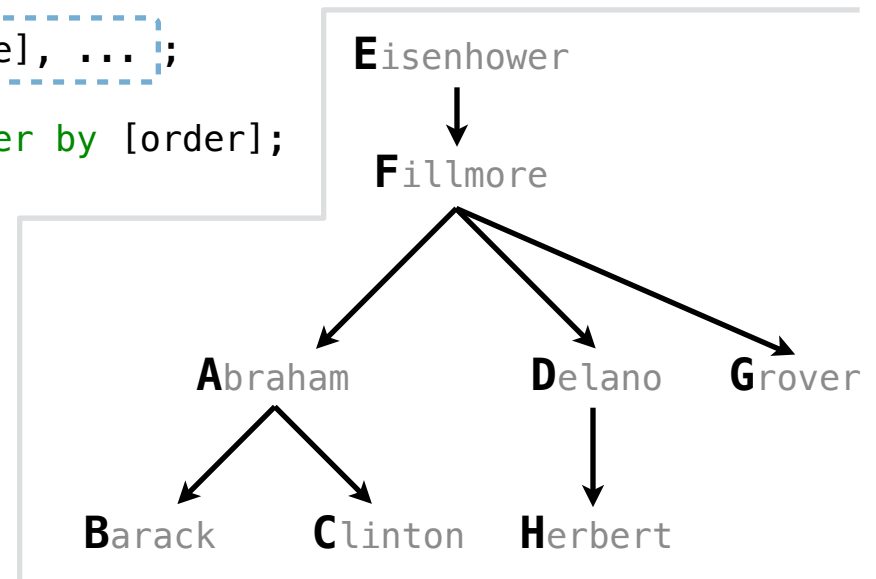
A **select** statement can specify an input table using a **from** clause

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 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

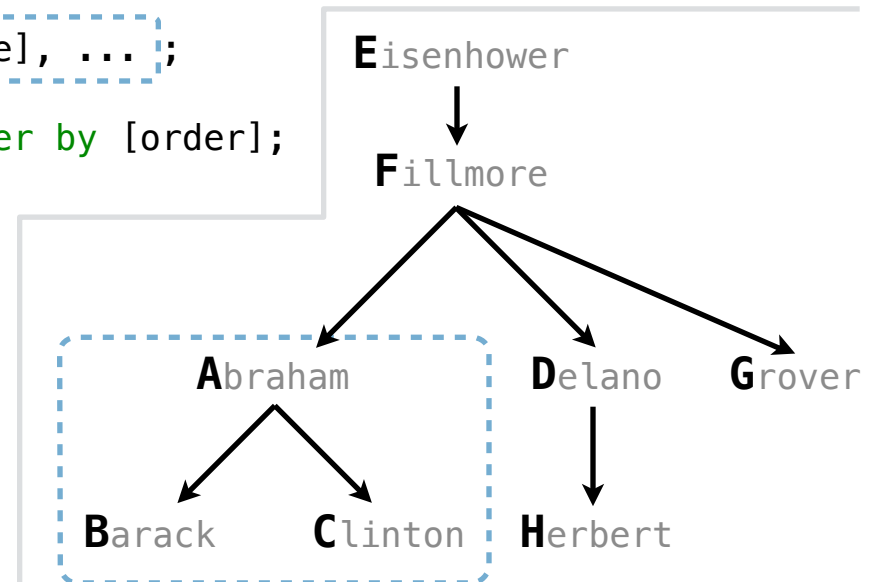
An ordering over the remaining rows can be declared using an **order by** clause

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 Statements Project Existing Tables

A **select** statement can specify an input table using a **from** clause

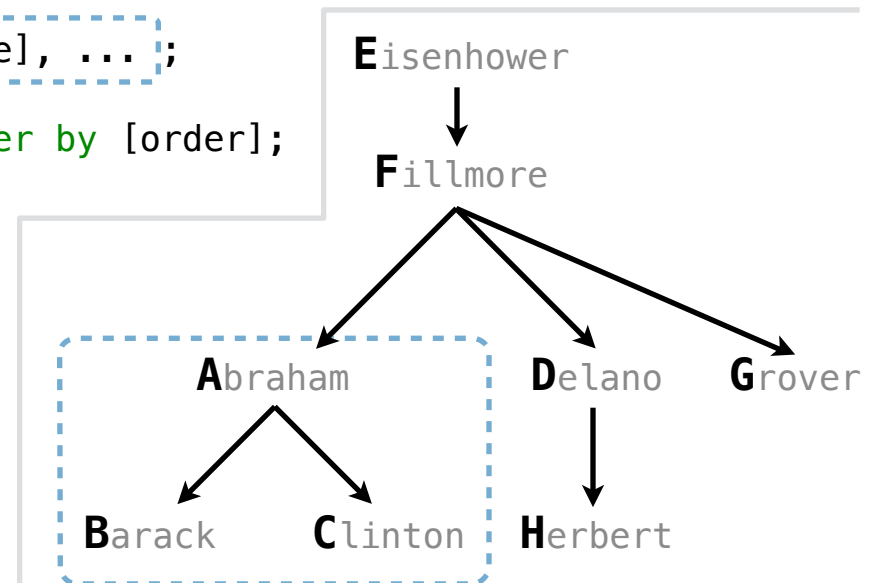
A subset of the rows of the input table can be selected using a **where** clause

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";
```

Child
barack
clinton



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

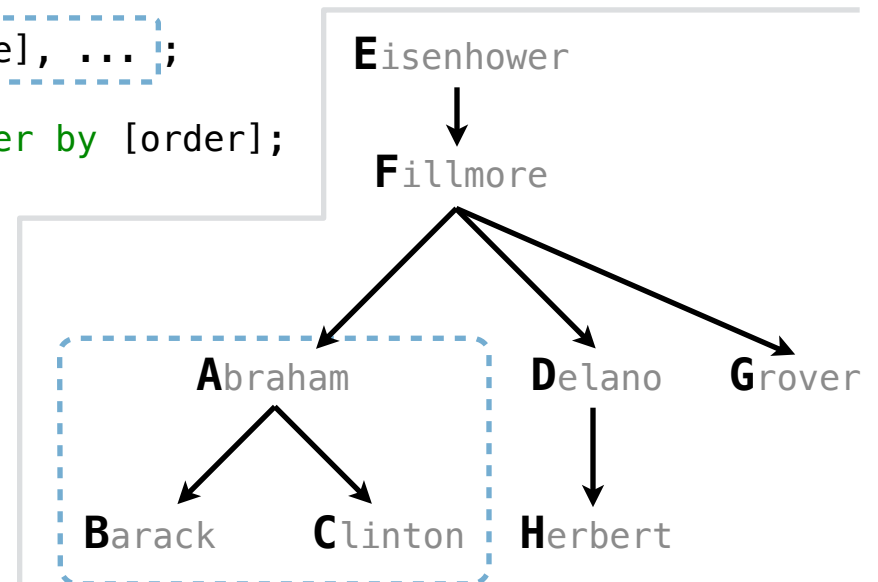
```
select [expression] as [name], [expression] as [name], ... ;
```

```
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



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], ... ;
```

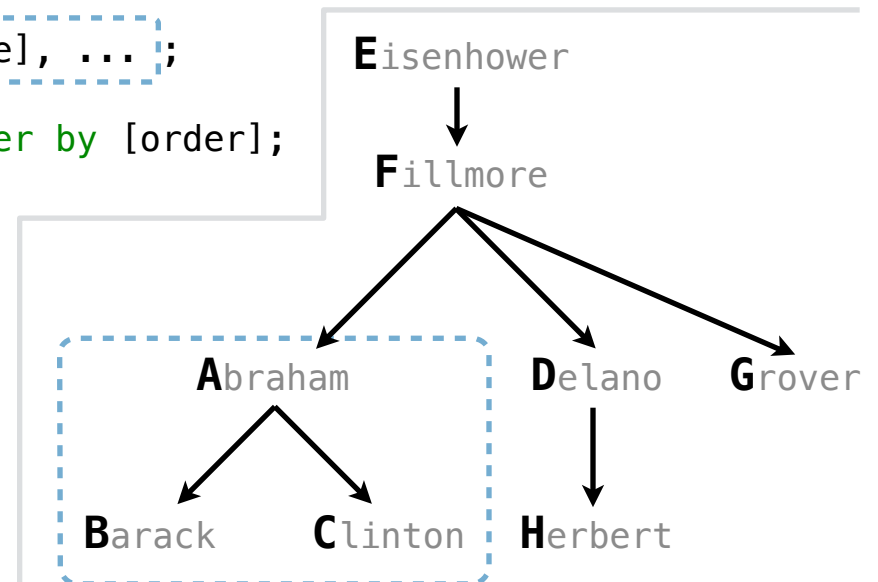
```
select [columns] from [table] where [condition] order by [order];
```

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

```
select parent from parents where parent > child;
```

Child
barack
clinton

Parent
fillmore
fillmore



Select Statements Project Existing Tables

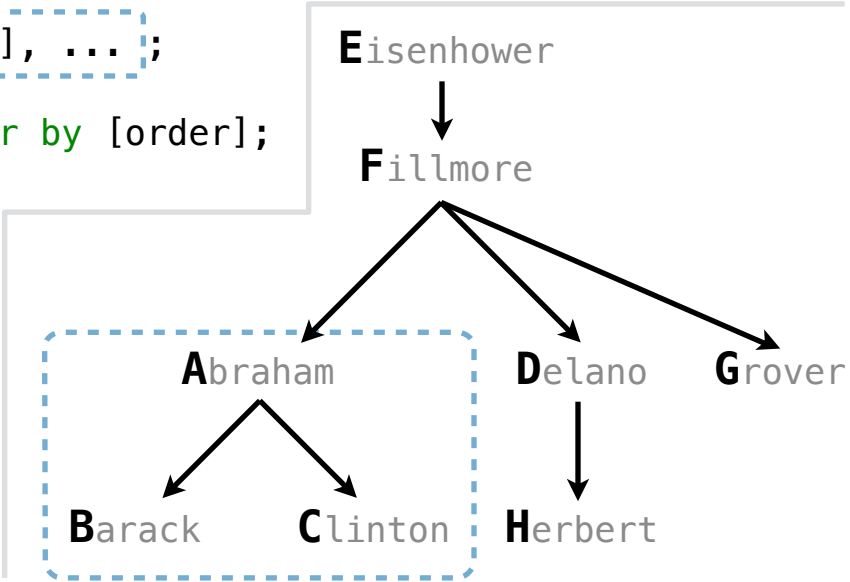
- A **select** statement can specify an input table using a **from** clause
- A subset of the rows of the input table can be selected using a **where** clause
- An ordering over the remaining rows can be declared using an **order by** clause
- Column descriptions determine how each input row is projected 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)



Joining Tables

Joining Two Tables

Joining Two Tables

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

Joining Two Tables

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

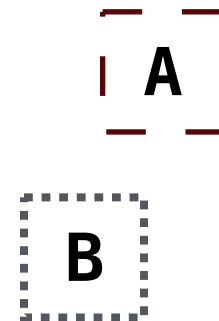
```
create table dogs as  
  select "abraham" as name, "long" as fur union
```

A

Joining Two Tables

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

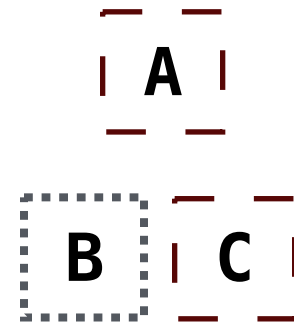
```
create table dogs as
  select "abraham" as name, "long" as fur union
  select "barack"      , "short"      union
```



Joining Two Tables

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

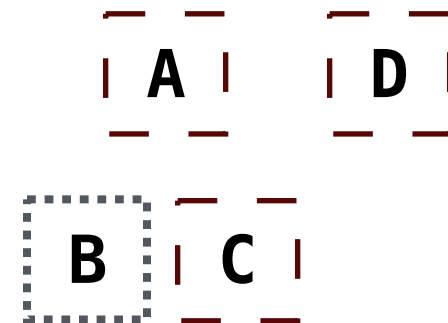
```
create table dogs as
  select "abraham" as name, "long" as fur union
  select "barack"      , "short"      union
  select "clinton"     , "long"       union
```



Joining Two Tables

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



Joining Two Tables

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

E

A **D**

B **C**

Joining Two Tables

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

E

F

A

D

B

C

Joining Two Tables

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

E

F

A

D

G

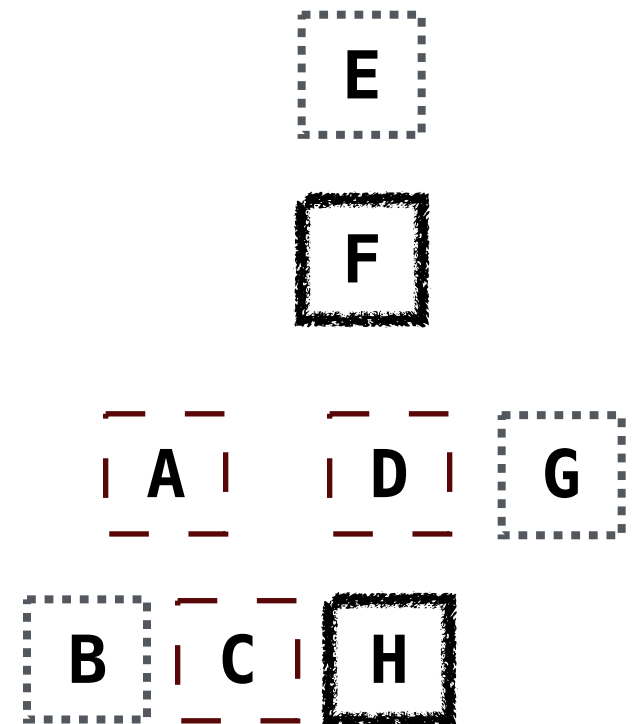
B

C

Joining Two Tables

Two tables **A** & **B** are joined by a comma to yield all combos of a row from **A** & 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";
```

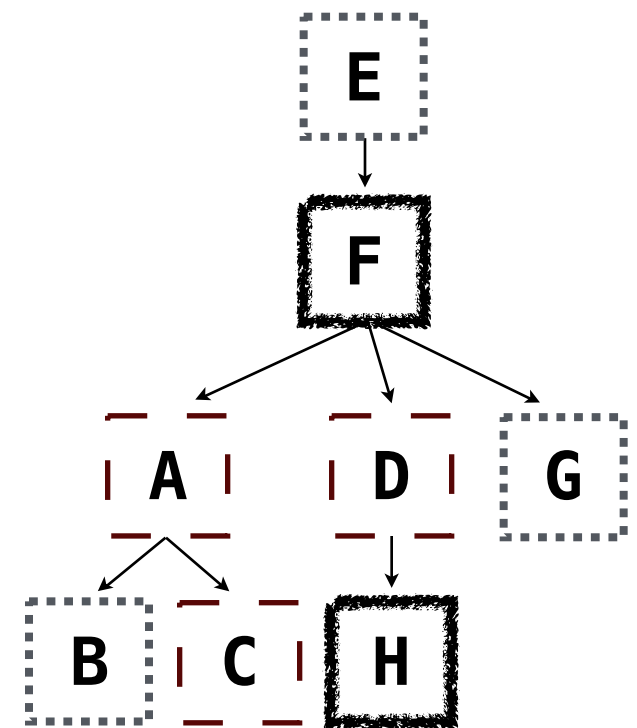


Joining Two Tables

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



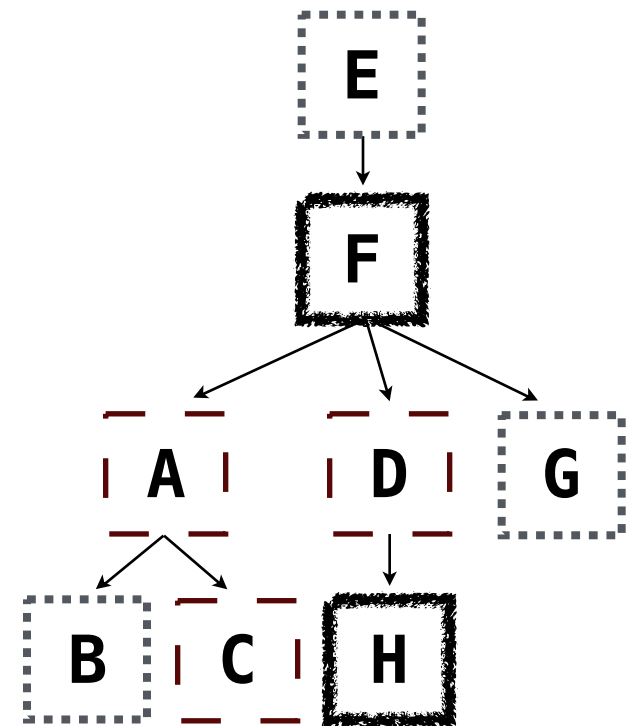
Joining Two Tables

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

Select the parents of curly-furred dogs



Joining Two Tables

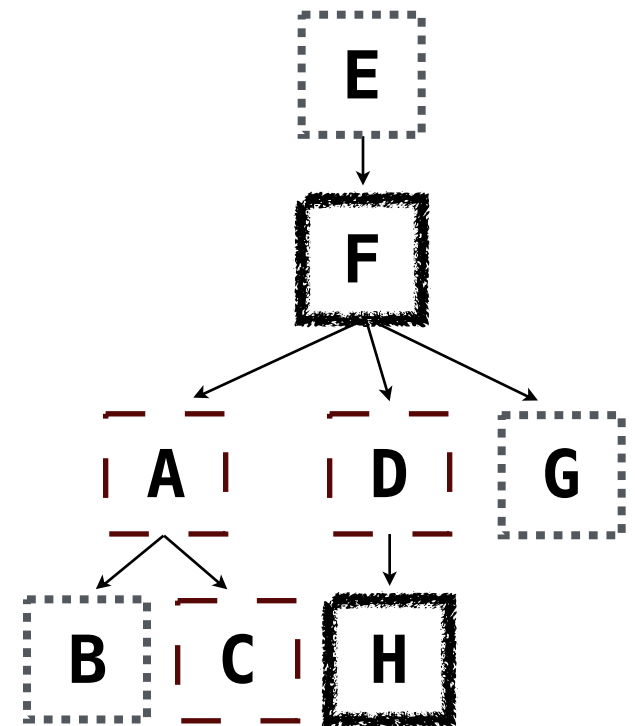
Two tables **A** & **B** are joined by a comma to yield all combos of a row from **A** & a row from **B**

```
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
  ...;
```

Select the parents of curly-furred dogs

```
select parent from parents, dogs
      where child = name and fur = "curly";
```



Joining Two Tables

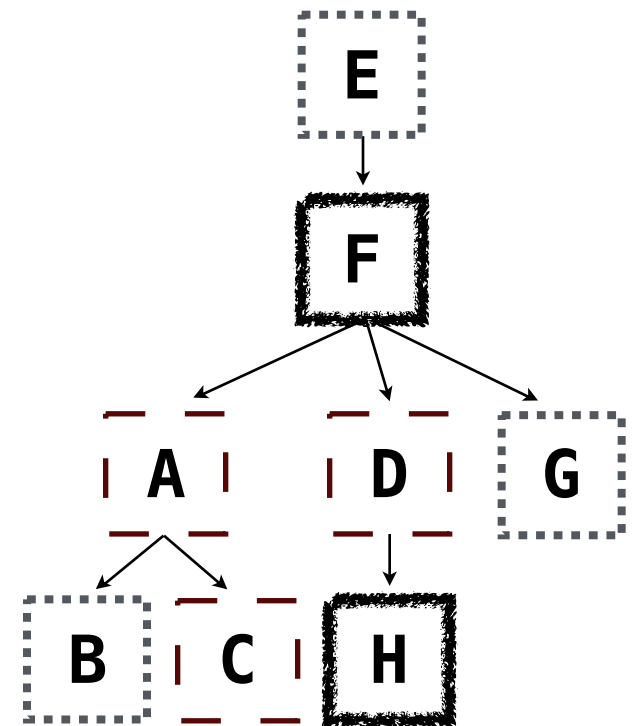
Two tables **A** & **B** are joined by a comma to yield all combos of a row from **A** & a row from **B**

```
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
  ...;
```

Select the parents of curly-furred dogs

```
select parent from parents, dogs
where child = name and fur = "curly";
```



Joining Two Tables

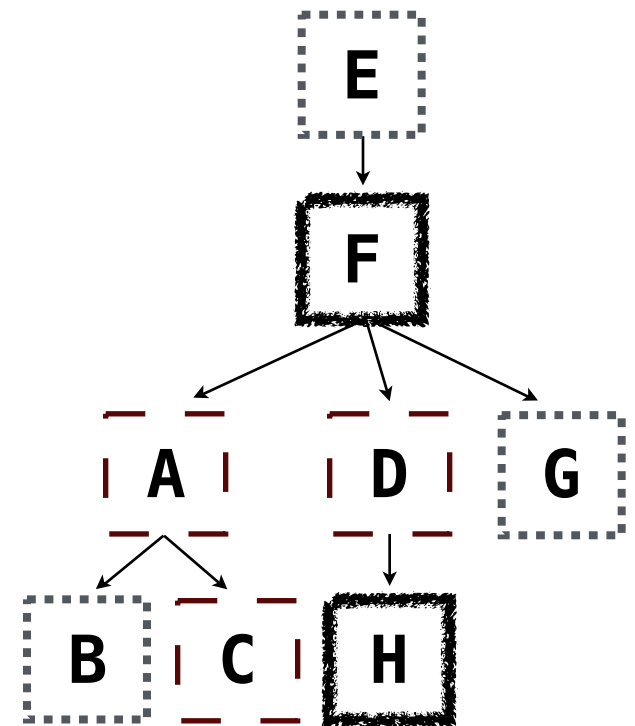
Two tables **A** & **B** are joined by a comma to yield all combos of a row from **A** & a row from **B**

```
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
  ...;
```

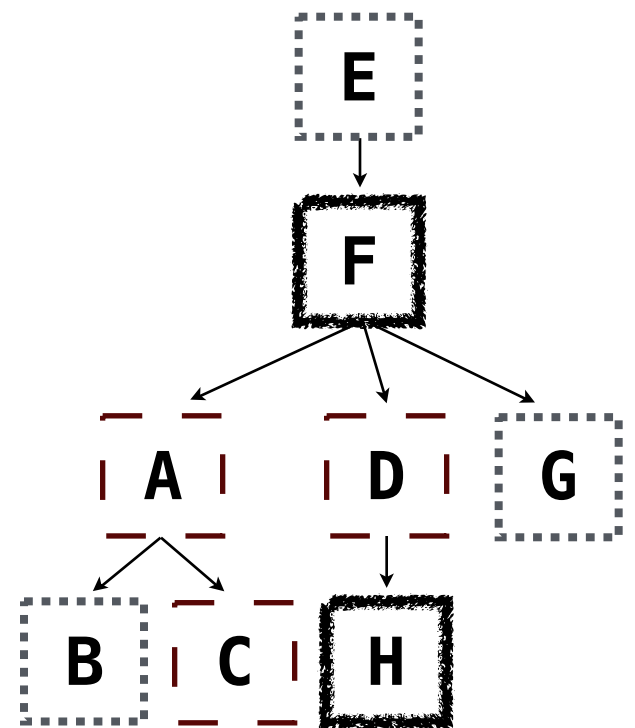
Select the parents of curly-furred dogs

```
select parent from parents, dogs
where child = name and fur = "curly";
```



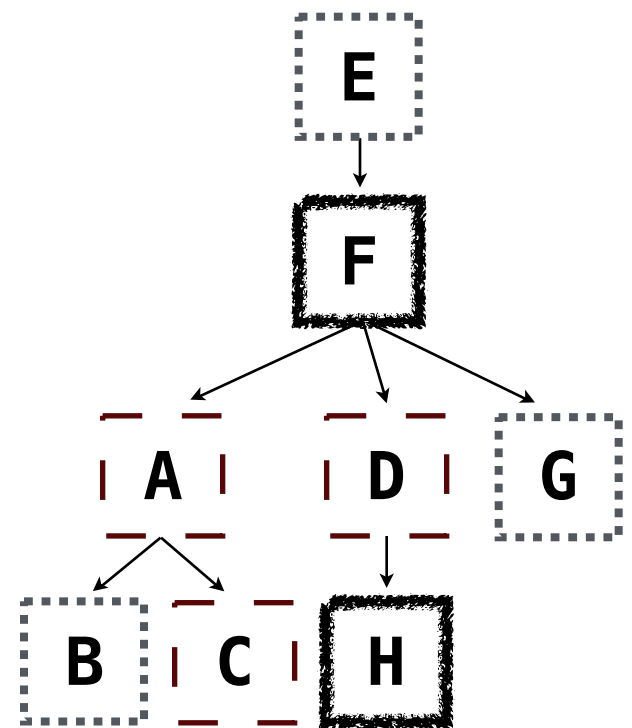
Aliases and Dot Expressions

Joining a Table with Itself



Joining a Table with Itself

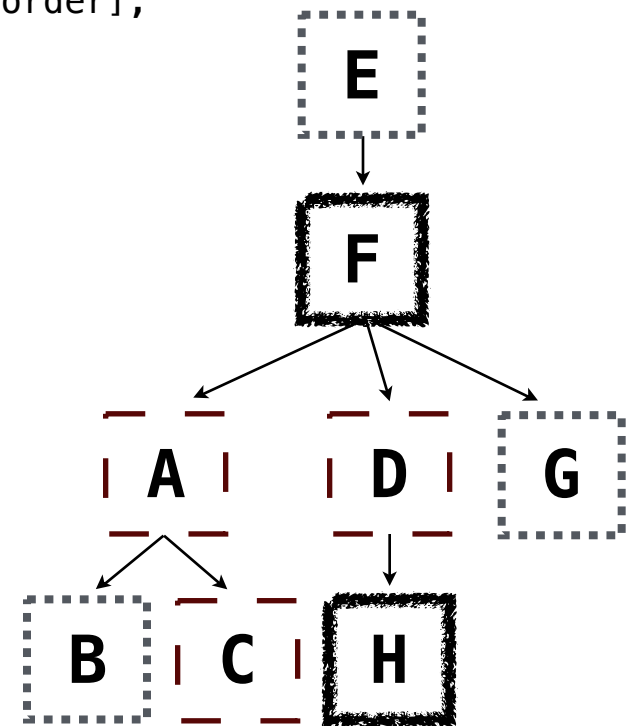
Two tables may share a column name; dot expressions and aliases disambiguate column values



Joining a Table with Itself

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

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

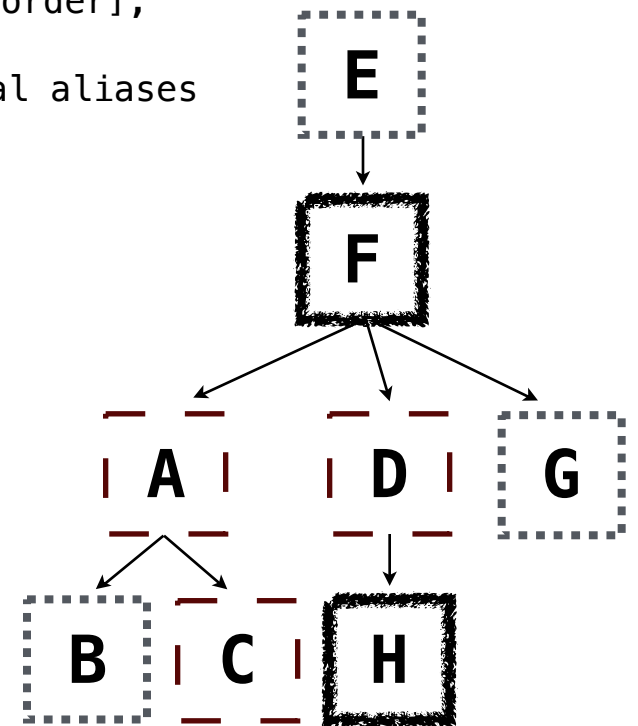


Joining a Table with Itself

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



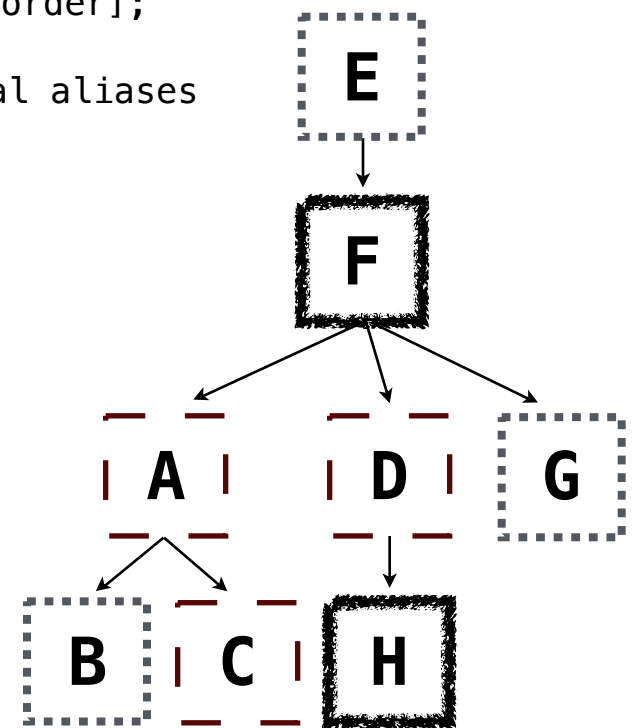
Joining a Table with Itself

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



Joining a Table with Itself

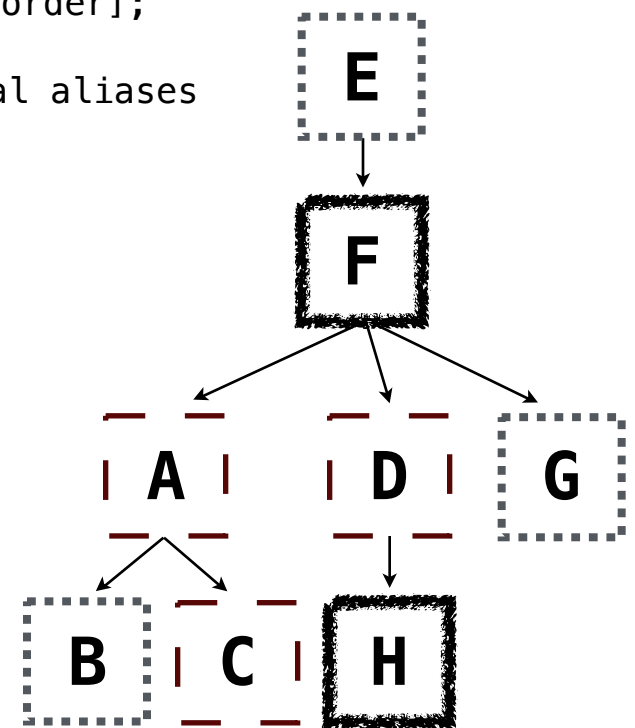
Two tables may share a column name; dot expressions and aliases disambiguate column values

```
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

```
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;
```



Joining a Table with Itself

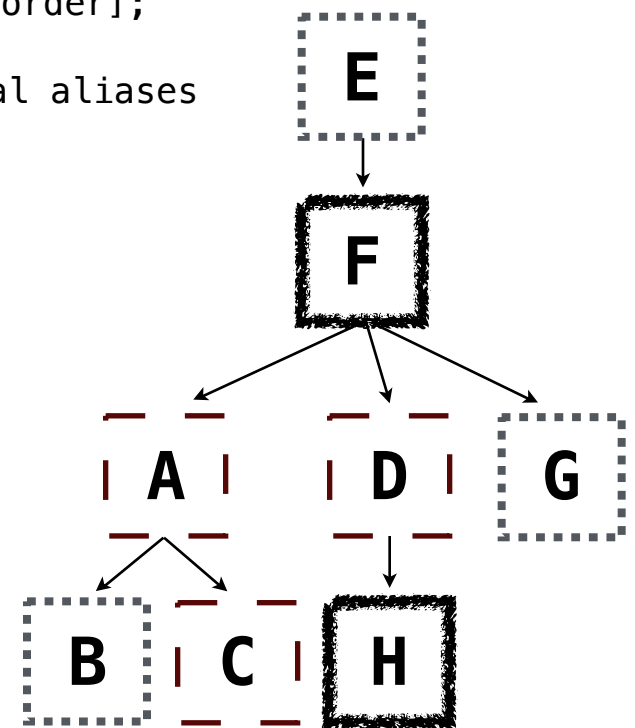
Two tables may share a column name; dot expressions and aliases disambiguate column values

```
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

```
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;
```



Joining a Table with Itself

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

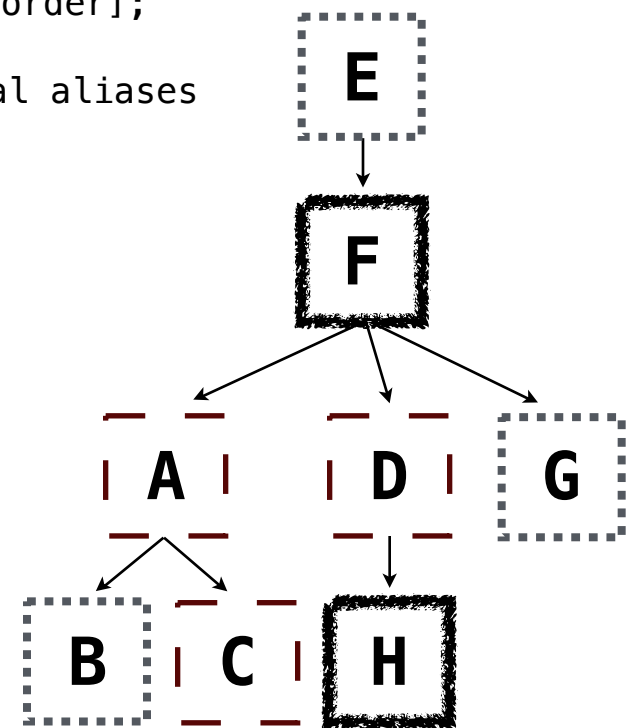
```
select [columns] from [table] where [condition] order by [order];
```

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

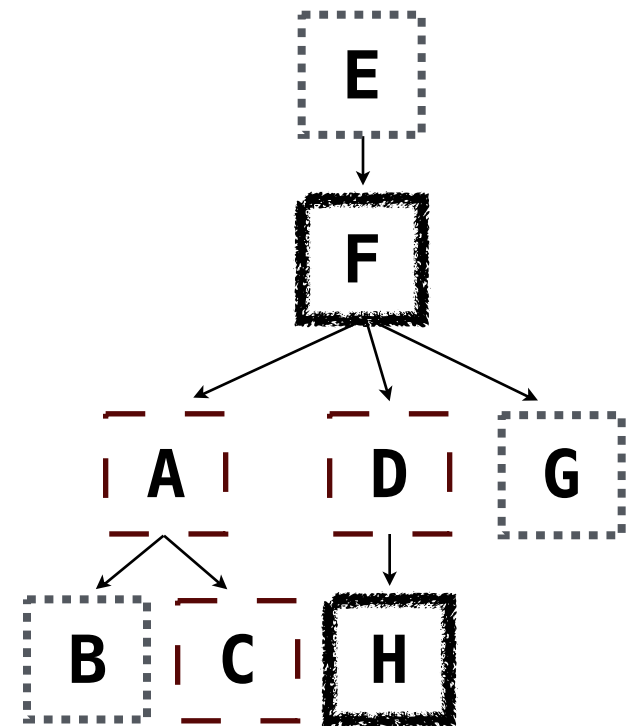
Select all pairs of siblings

```
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;
```

First	Second
barack	clinton
abraham	delano
abraham	grover
delano	grover

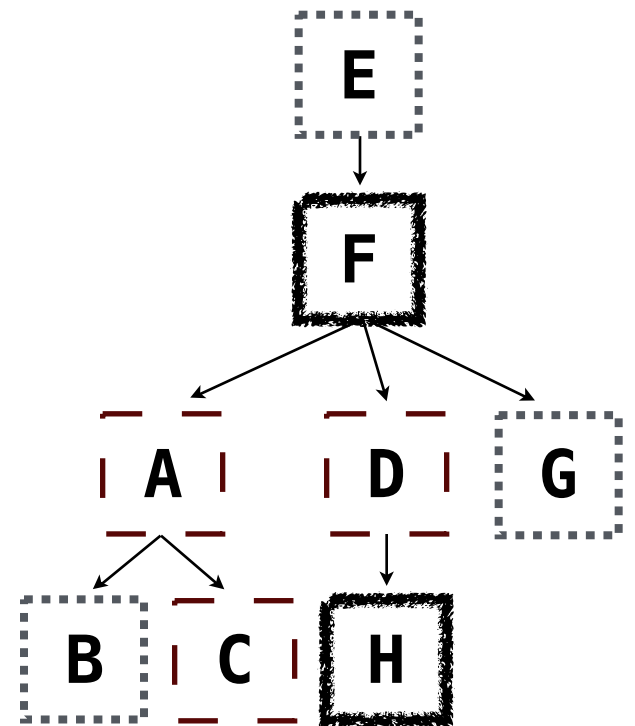


Joining Multiple Tables



Joining Multiple Tables

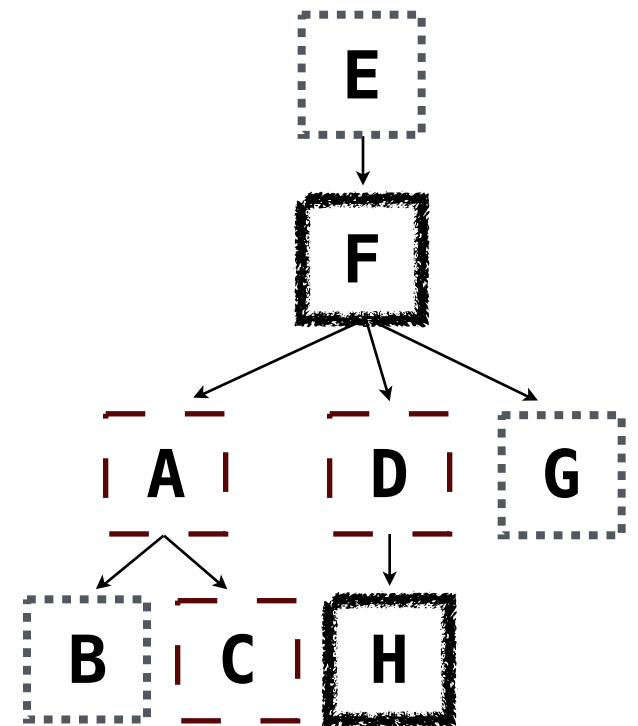
Multiple tables can be joined to yield all combinations of rows from each



Joining Multiple Tables

Multiple tables can be joined to yield all combinations of rows from each

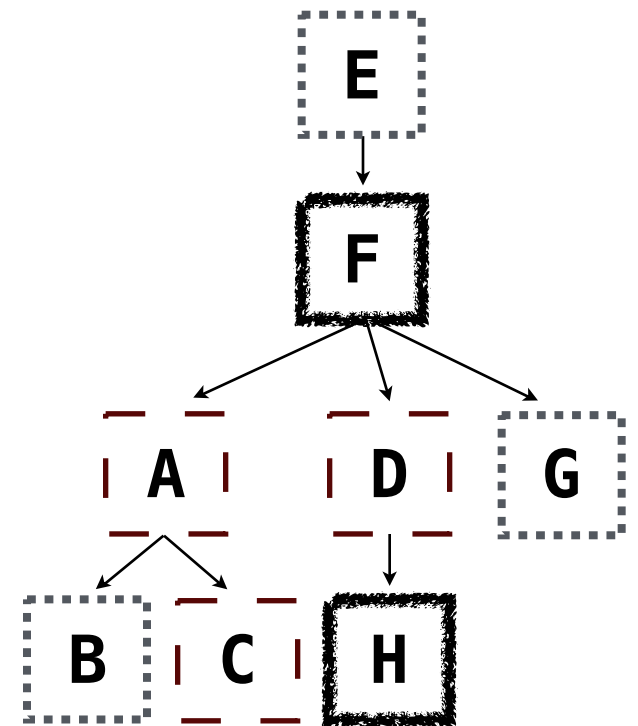
```
create table grandparents as
```



Joining Multiple Tables

Multiple tables can be joined to yield all combinations of rows from each

```
create table grandparents as
select a.parent as granddog, b.child as granpup
from parents as a, parents as b
where b.parent = a.child;
```

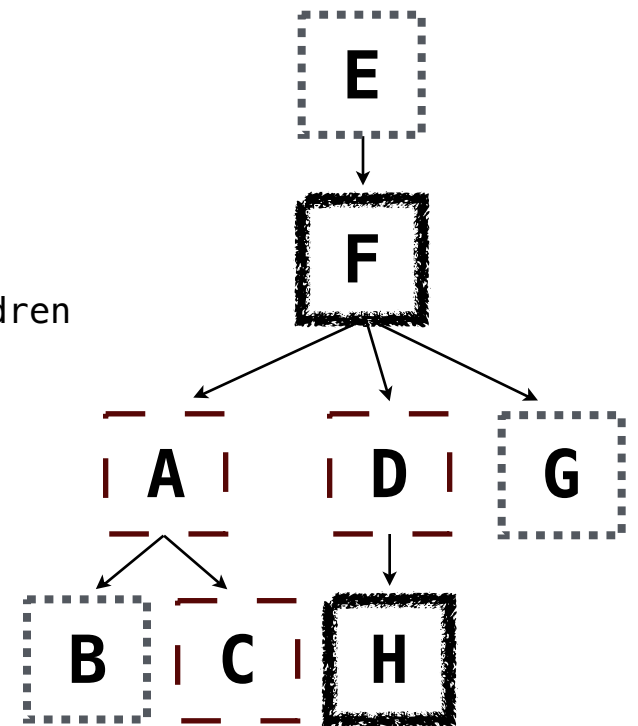


Joining Multiple Tables

Multiple tables can be joined to yield all combinations of rows from each

```
create table grandparents as
select a.parent as granddog, b.child as granpup
from parents as a, parents as b
where b.parent = a.child;
```

Select all grandparents with the same fur as their grandchildren



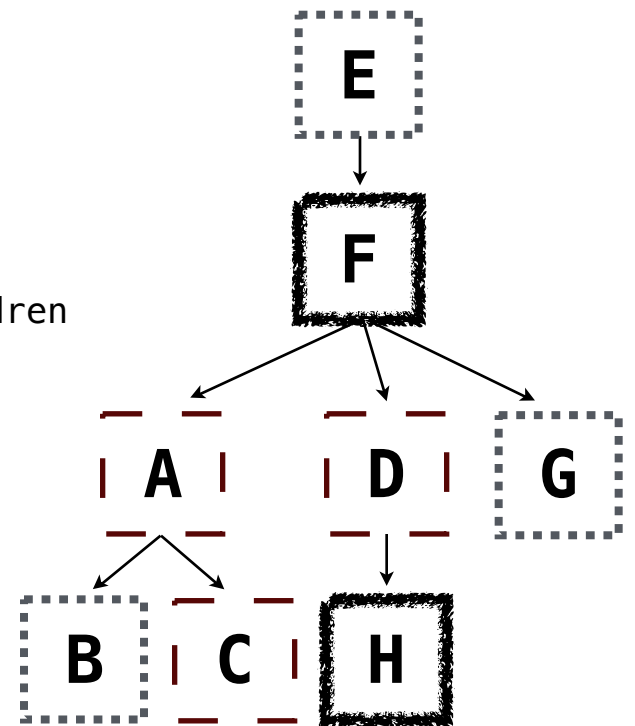
Joining Multiple Tables

Multiple tables can be joined to yield all combinations of rows from each

```
create table grandparents as
select a.parent as granddog, b.child as granpup
from parents as a, parents as b
where b.parent = a.child;
```

Select all grandparents with the same fur as their grandchildren

Which tables need to be joined together?



Joining Multiple Tables

Multiple tables can be joined to yield all combinations of rows from each

```
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;
```

Select all grandparents with the same fur as their grandchildren

Which tables need to be joined together?

```
select grandog from grandparents, dogs as c, dogs as d
where grandog = c.name and
granpup = d.name and
c.fur = d.fur;
```

