Welcome to Data C88C!

Lecture 21: Tables
Wednesday, July 30th, 2025
Week 6
Summer 2025
Instructor: Eric Kim (ekim555@berkeley.edu)

Announcements

- Mid-semester survey feedback: [link] (extended, due tonight!!!)
  - If **75%** of the class completes this form by **tonight (Wednesday July 30th 11:59 PM)**, everyone will receive 1 point of extra credit! If this goal is not met, nobody will receive the extra point.
  - As of today (July 30th, 3:07 PM PST): **65%** of the class has completed the survey
- Midterm regrades: due this Friday
  - Midterm solutions doc released: [link]
- August 1st: Change Grade Option deadline
- Ants project is ongoing! Checkpoint due Mon Aug 4th

Lecture Overview

- More SQL
  - Joins

Joining Tables

Joining tables together

- Joining tables allows you to combine rows from two (or more) tables
- Ex: suppose I have two tables, `prices` and `orders`. I'd like to compute how much money I've made per product.
- Here is a query that achieves this:

```
> select prices.name, quantity_sold * price as total_money
from prices, orders
where prices.name = orders.name;

name    total_money
burger    45.5
fries   50
hot cocoa   9.9
soda    22
```

prices

name	price
soda	1.1
burger	3.5
fries	2.0
hot cocoa	0.9
coffee	0.75

orders

name	quantity_sold
soda	20
burger	15
fries	25
hot cocoa	11
secret item	1

Joining tables together

Generates all possible pairs of rows between `prices` and `orders` (aka "Cartesian product", "cross join")

```
> select *
from prices, orders;

name    price    name   quantity_sold
 burger   3.5    burger   13
 burger   3.5    fries    25
 burger   3.5    hot cocoa    11
 burger   3.5    secret item    1
 burger   3.5    soda   20
 coffee   0.75   burger   13
 coffee   0.75   fries    25
 coffee   0.75   hot cocoa    11
 coffee   0.75   secret item    1
 coffee   0.75   soda   20
 fries    2    burger   13
 fries    2    fries    25
 fries    2    hot cocoa    11
 fries    2    secret item    1
 fries    2    soda   20
 hot cocoa    0.9    burger   13
...
```

Alternate syntax:

```
select * from prices cross join orders;
```

Joining tables together

Generates all possible pairs of rows between `prices` and `orders` (aka "Cartesian product", "cross join")

Adding this filter criterion restricts to just the rows we care about ("join criterion")

```
> select *
from prices, orders
where prices.name = orders.name;

name    price    name   quantity_sold
burger    3.5    burger   13
fries   2    fries    25
hot cocoa   0.9    hot cocoa    11
soda    1.1    soda   20
```

Alternate syntax (**much** more common in practice):

```
select * from prices join orders on prices.name = orders.name;
```

Dog Family Tree

```
CREATE TABLE parents AS
  SELECT "ace" AS parent, "bella" AS child UNION
  SELECT "ace"          , "charlie"        UNION
  SELECT "daisy"        , "hank"           UNION
  SELECT "finn"         , "ace"            UNION
  SELECT "finn"         , "daisy"          UNION
  SELECT "finn"         , "ginger"         UNION
  SELECT "ellie"        , "finn";
```

$$\begin{tikzcd}
 & & \text{Ellie} \arrow[d] & & \\
 & & \text{Finn} \arrow[dl] \arrow[dr] \arrow[drr] & & \\
 & \text{Ace} \arrow[dl] \arrow[d] & & \text{Daisy} \arrow[d] & \text{Ginger} \\
\text{Bella} & \text{Charlie} & & \text{Hank} &
\end{tikzcd}$$

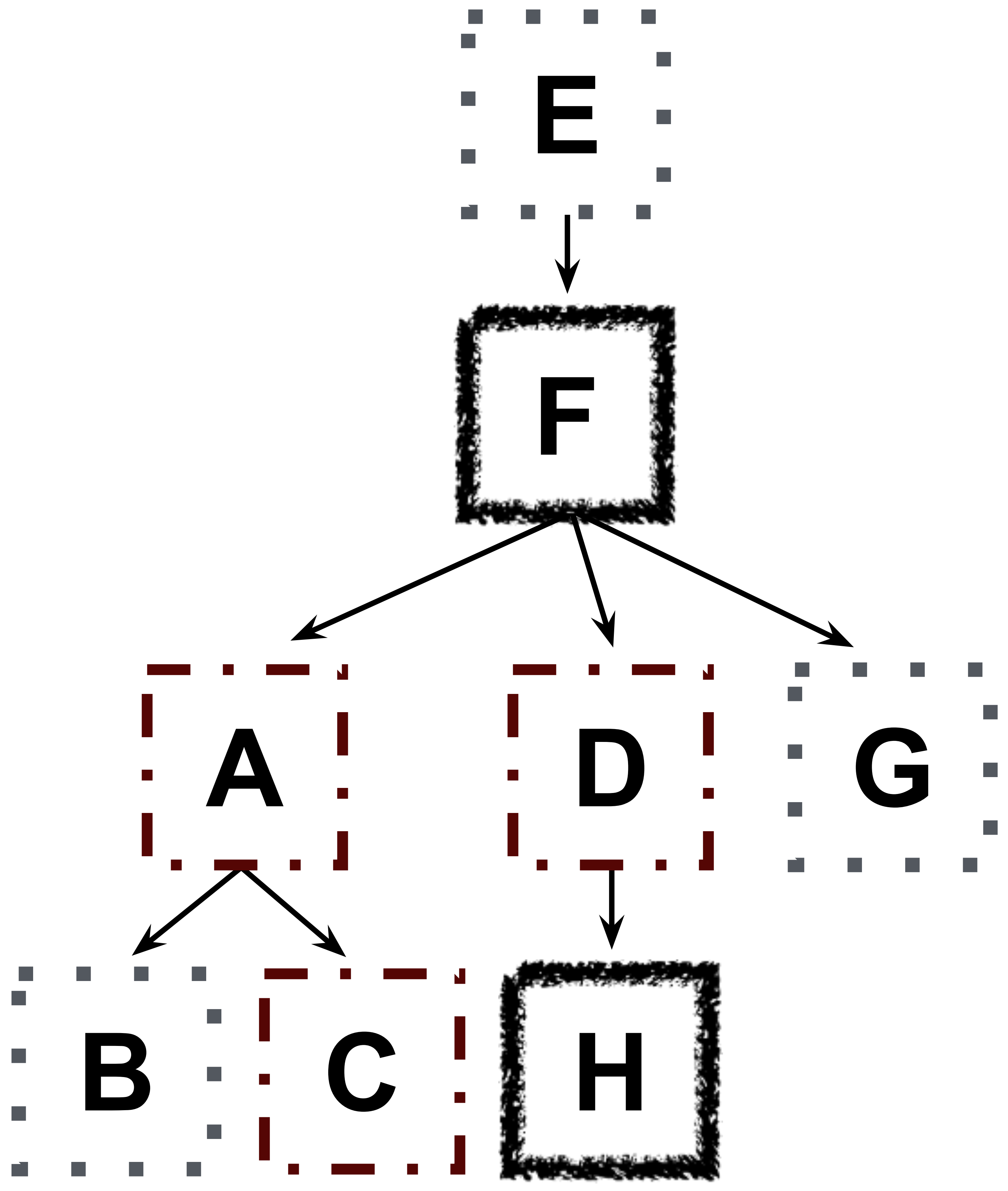
Joining Two Tables

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 "ace" AS name, "long" AS fur UNION
  SELECT "bella"      , "short"       UNION
  SELECT "charlie"    , "long"        UNION
  SELECT "daisy"      , "long"        UNION
  SELECT "ellie"      , "short"       UNION
  SELECT "finn"       , "curly"       UNION
  SELECT "ginger"     , "short"       UNION
  SELECT "hank"       , "curly";
CREATE TABLE parents AS
  SELECT "ace" AS parent, "bella" AS child UNION
  SELECT "ace"          , "charlie"        UNION
  ...;
```

Select the parents of curly-furred dogs

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

SELECT parent FROM parents JOIN dogs
              ON child = name WHERE fur = "curly";
```



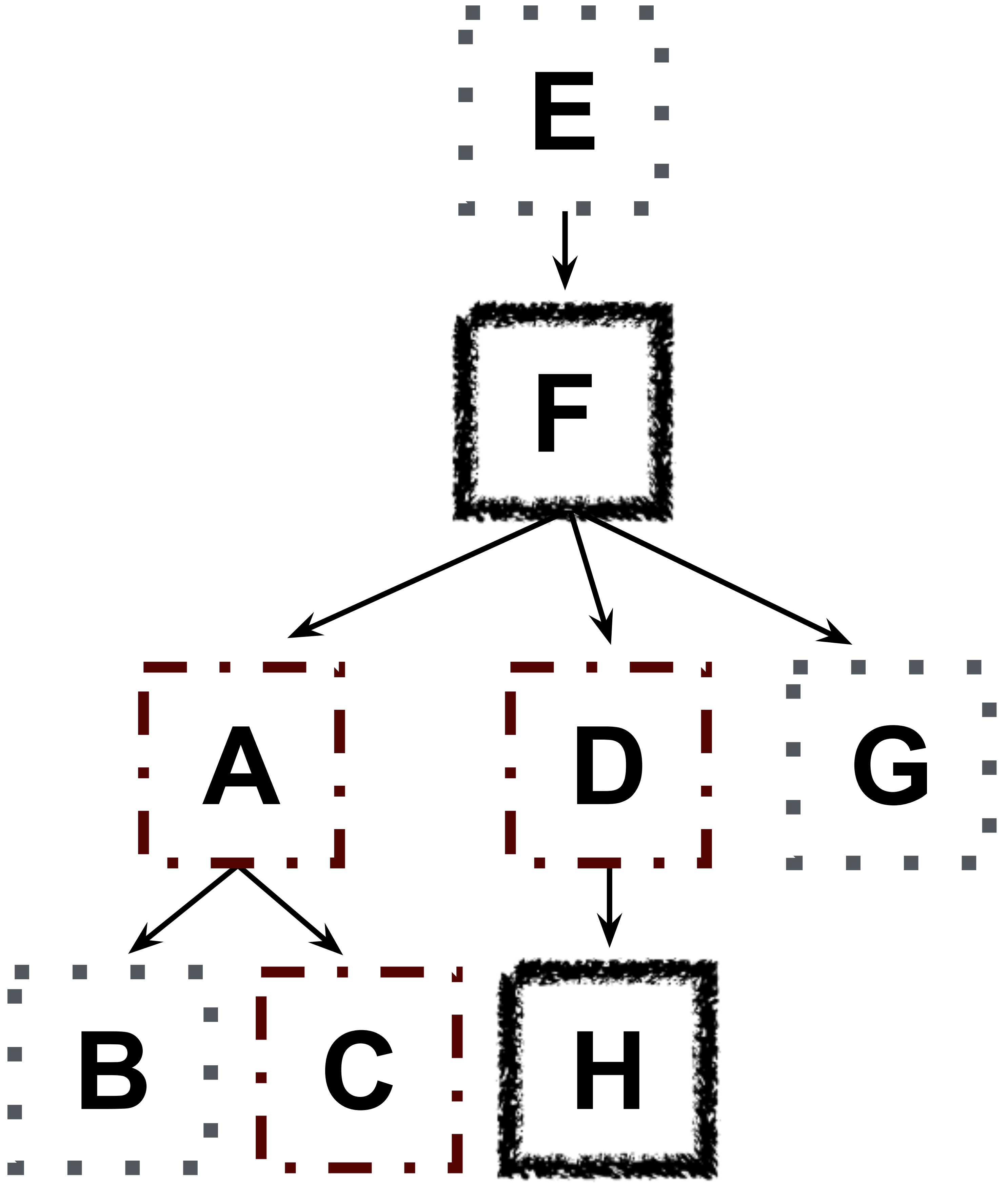
(Demo 21.sql:Demo00)

Discussion Question

```
CREATE TABLE dogs AS
  SELECT "ace" AS name, "long" AS fur UNION
  SELECT "bella"      , "short"       UNION
  SELECT "charlie"    , "long"        UNION
  SELECT "daisy"      , "long"        UNION
  SELECT "ellie"      , "short"       UNION
  SELECT "finn"       , "curly"       UNION
  SELECT "ginger"     , "short"       UNION
  SELECT "hank"       , "curly";
CREATE TABLE parents AS
  SELECT "ace" AS parent, "bella" AS child UNION
  SELECT "ace"          , "charlie"        UNION
  ...;
```

Show the name and fur of the parents of Daisy and Bella

```
SELECT name, fur FROM parents JOIN dogs ON _parent=name_

                     WHERE _child="daisy" or child="bella"_;
```



Aliases and Dot Expressions

Joining a Table with Itself

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

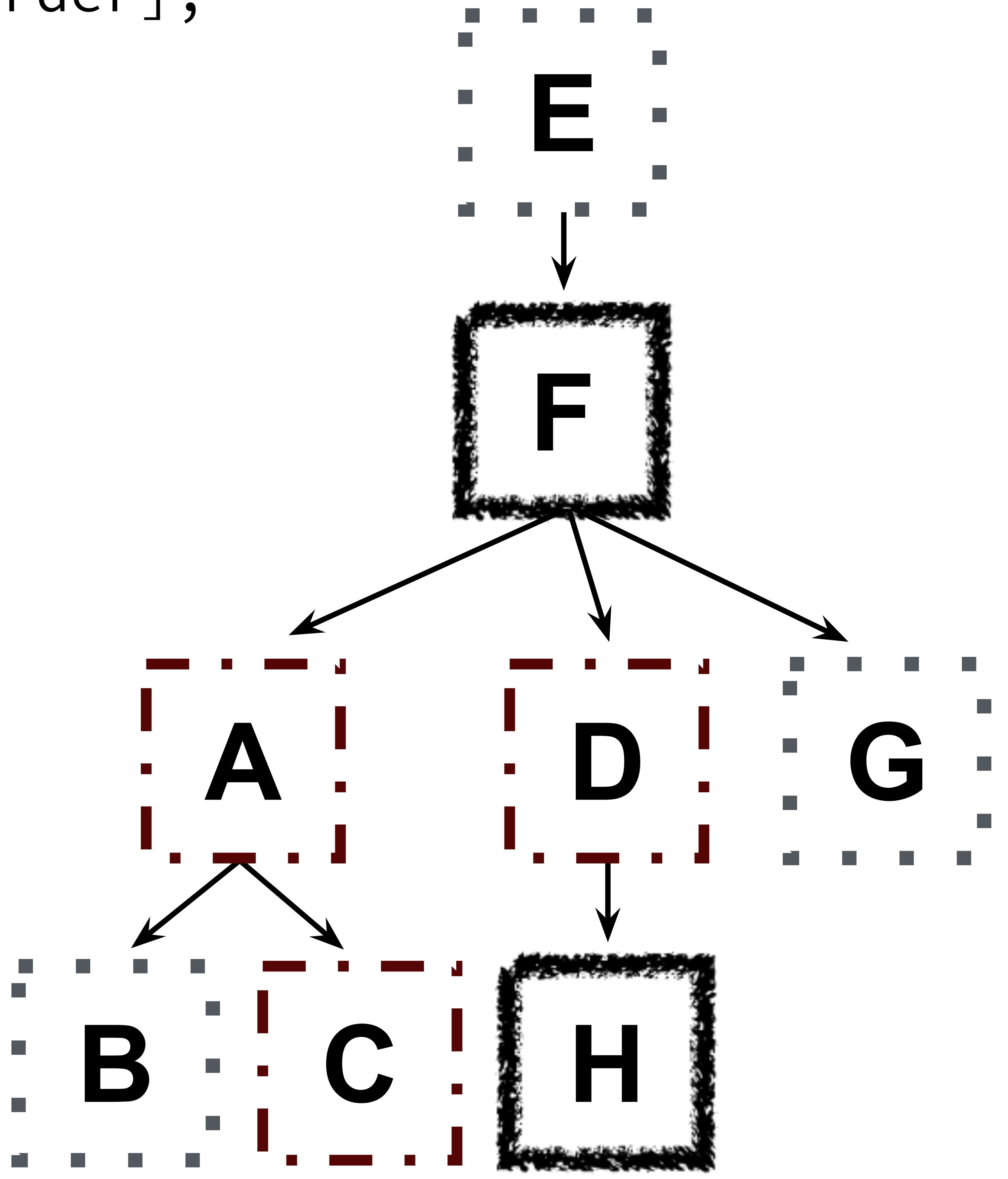
```
SELECT [columns] FROM [table] WHERE [condition] ORDER BY [order];
```

[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
bella	charlie
ace	daisy
ace	ginger
daisy	ginger



(Demo 21.sql:Demo01)

Example: Dog Triples

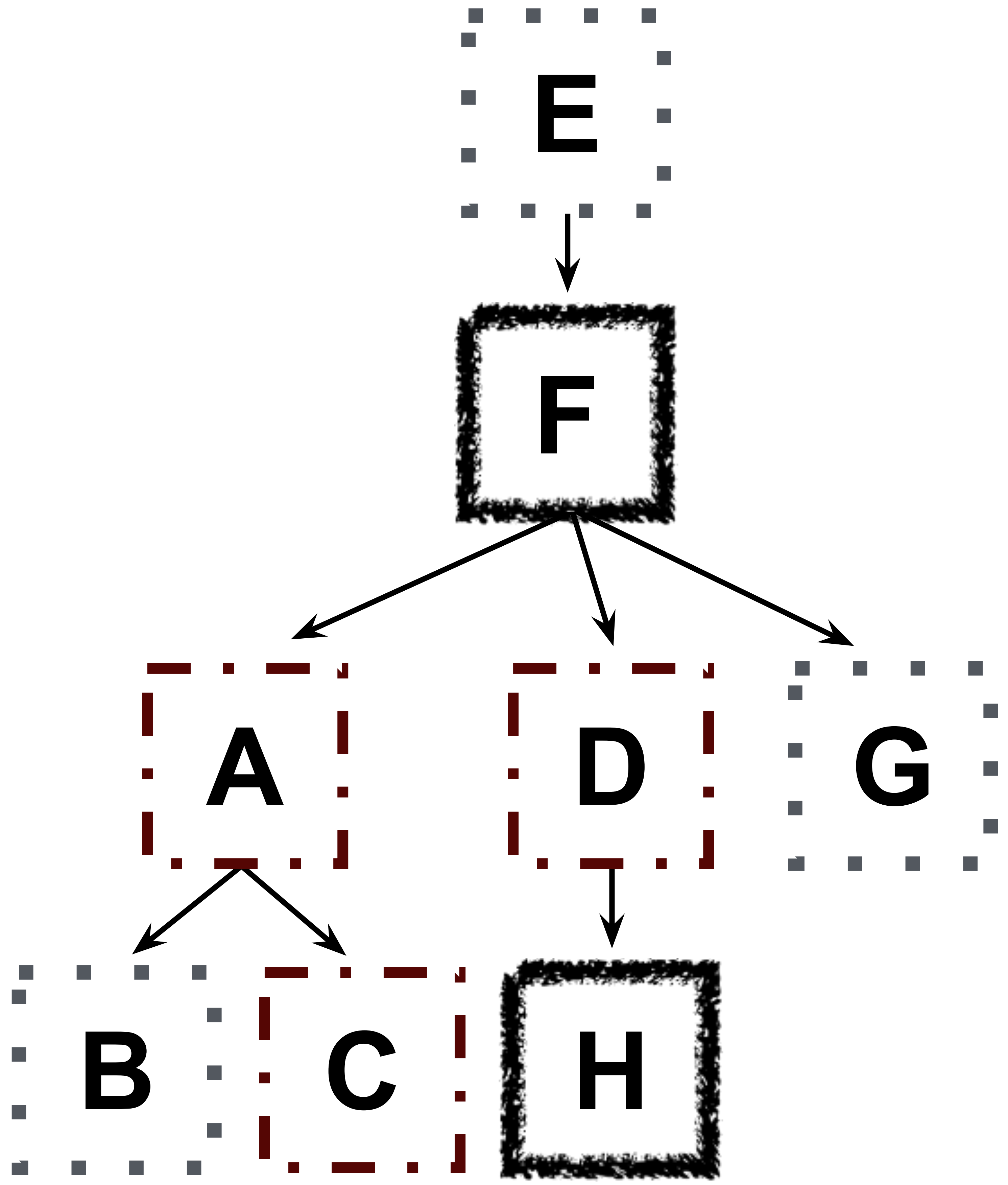
Fall 2014 Quiz Question (Slightly Modified)

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

```
CREATE TABLE dogs AS

  SELECT "ace" AS name, "long" AS fur UNION

  SELECT "bella"      , "short"       UNION

  ...;

CREATE TABLE parents AS
  SELECT "ace" AS parent, "bella" AS child UNION
  SELECT "ace"          , "charlie"        UNION
  ...;
```

Expected output:

```
daisy|charlie|ace
ginger|ellie|bella
```



(Demo 21.sql:Demo02)

SQL string concatenation: `||`

- We can concatenate strings via the `||` operator

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

> select "the price of " || prices.name || " is: " || prices.price
from prices;
the price of burger is: 3.5
the price of coffee is: 0.75
the price of fries is: 2.0
the price of hot cocoa is: 0.9
the price of soda is: 1.1
```

prices

name	**price**
soda	1.1
burger	3.5
fries	2.0
hot cocoa	0.9
coffee	0.75

orders

name	**quantity_sold**
soda	20
burger	15
fries	25
hot cocoa	11
secret item	1