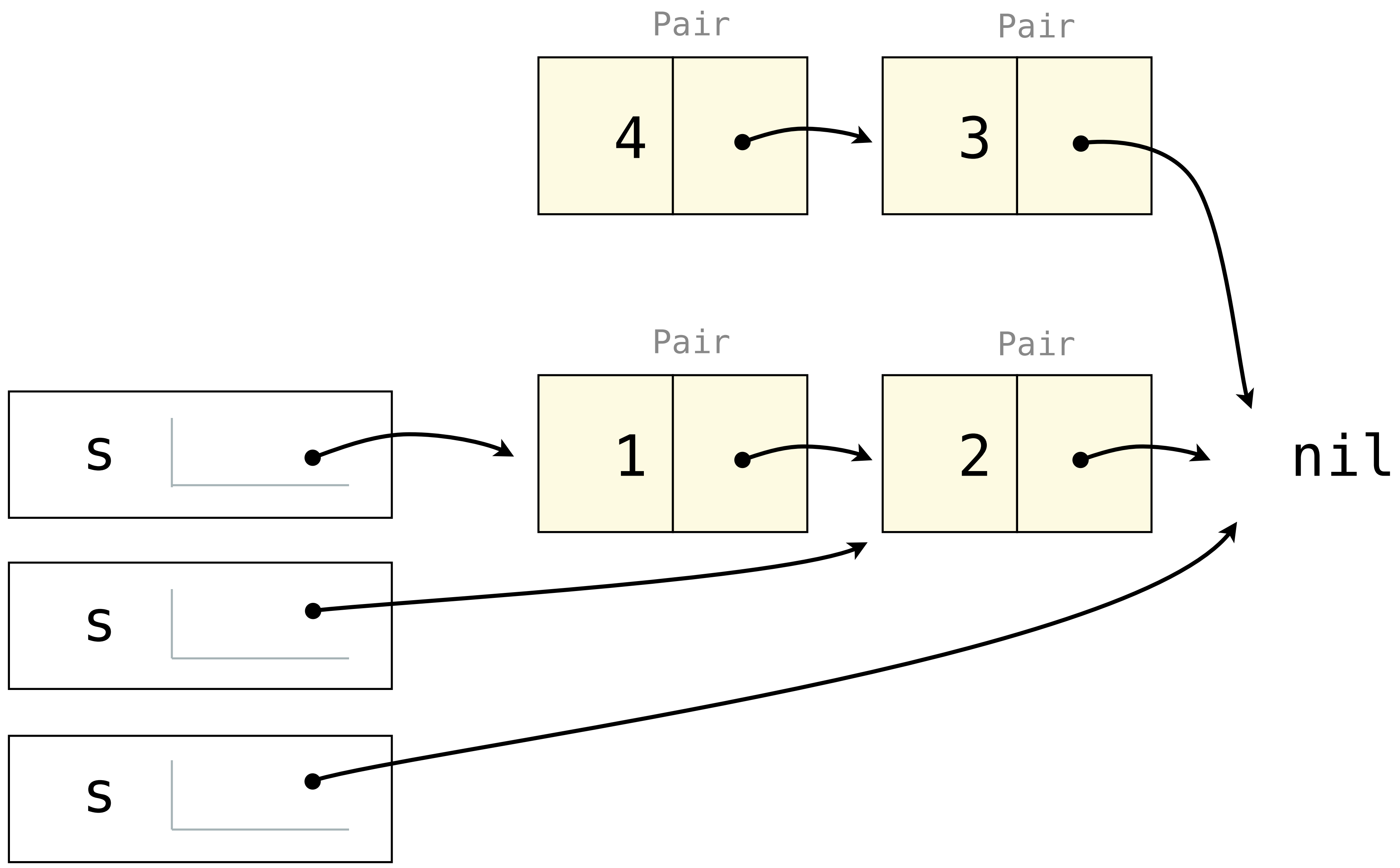
SQL and Tables

Announcements

Tail Recursion with Scheme Lists

```
(define (map procedure s)
  (if (null? s)
      nil
      (cons (procedure (car s))
            (map procedure (cdr s))) ) )
```

```
(map (lambda (x) (- 5 x)) (list 1 2))
```



```
(define (map procedure s)
  (define (map-reverse s m)
    (if (null? s)
        m
        (map-reverse (cdr s)
                     (cons (procedure (car s))
                           m))              ) )
  (reverse (map-reverse s nil)))
```

```
(define (reverse s)
  (define (reverse-iter s r)
    (if (null? s)
        r
        (reverse-iter (cdr s)
                      (cons (car s) r)) ) )
  (reverse-iter s nil))
```

Tail Recursion Techniques

Base case should return the complete answer (rather than a partial solution).

Define a helper with an extra parameter to keep track of progress so far.

Sketch an iterative solution (e.g. in Python) — names that are iteratively updated need to be tracked as function arguments in recursion.

Verify all recursive calls are tail calls.

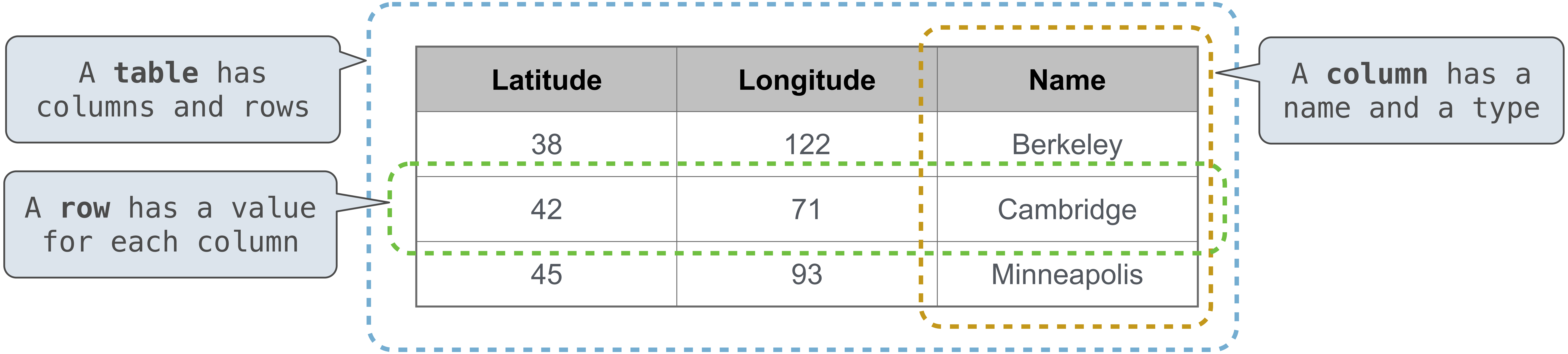
(Demo)

Databases

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

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

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

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

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

SQL is a *declarative* programming language

Declarative Programming

In **declarative programming:**

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

SQL Server Query Plan:

Results
Messages
Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT e.[BusinessEntityID], p.[Title], p.[FirstName], p.[MiddleName], p.[LastName], p.[Suffix], e.[JobTitle], pp.[PhoneNumber], pnt.[Name] AS [PhoneNumberType], ea.[E
```

SELECT
Cost: 0 %
Hash Match
(Right Outer Join)
Cost: 18 %
0.062s
290 of
505 (57%)
Compute Scalar
Cost: 0 %
Index Scan (NonClustered)
[EmailAddress].[IX_EmailAddress_Ema…
Cost: 3 %
0.003s
19972 of
19972 (100%)
Hash Match
(Right Outer Join)
Cost: 0 %
0.046s
290 of
505 (57%)
Compute Scalar
Cost: 0 %
Clustered Index Scan (Clustered)
[PhoneNumberType].[PK_PhoneNumberTy…
Cost: 0 %
0.000s
3 of
3 (100%)
Compute Scalar
Cost: 0 %
Hash Match
(Right Outer Join)
Cost: 13 %
0.046s
290 of
291 (99%)
Compute Scalar
Cost: 0 %
Index Scan (NonClustered)
[PersonPhone].[IX_PersonPhone_Phone…
Cost: 2 %
0.003s
19972 of
19972 (100%)
Hash Match
(Inner Join)
Cost: 1 %
0.030s
290 of
291 (99%)
Compute Scalar
Cost: 0 %
Index Scan (NonClustered)
[CountryRegion].[AK_CountryRegion_N…
Cost: 0 %
0.000s
238 of
238 (100%)

https://learn.microsoft.com/en-us/sql/relational-databases/performance/display-an-actual-execution-plan?view=sql-server-ver16

Structured Query Language (SQL)

Naming Tables

A **select** statement creates a new table and displays it.

A **create table** statement names the result of a **select** statement.

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

```
create table parents as
select "d" as parent, "h" as child union
select "a"          , "b"          union
select "a"          , "c"          union
select "f"          , "a"          union
select "f"          , "d"          union
select "f"          , "g"          union
select "e"          , "f";
```

Parents:

parent	child
a	b
a	c
d	h
f	a
f	d
f	g
e	f

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 = "a";

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

child
b
c

parent
f
f

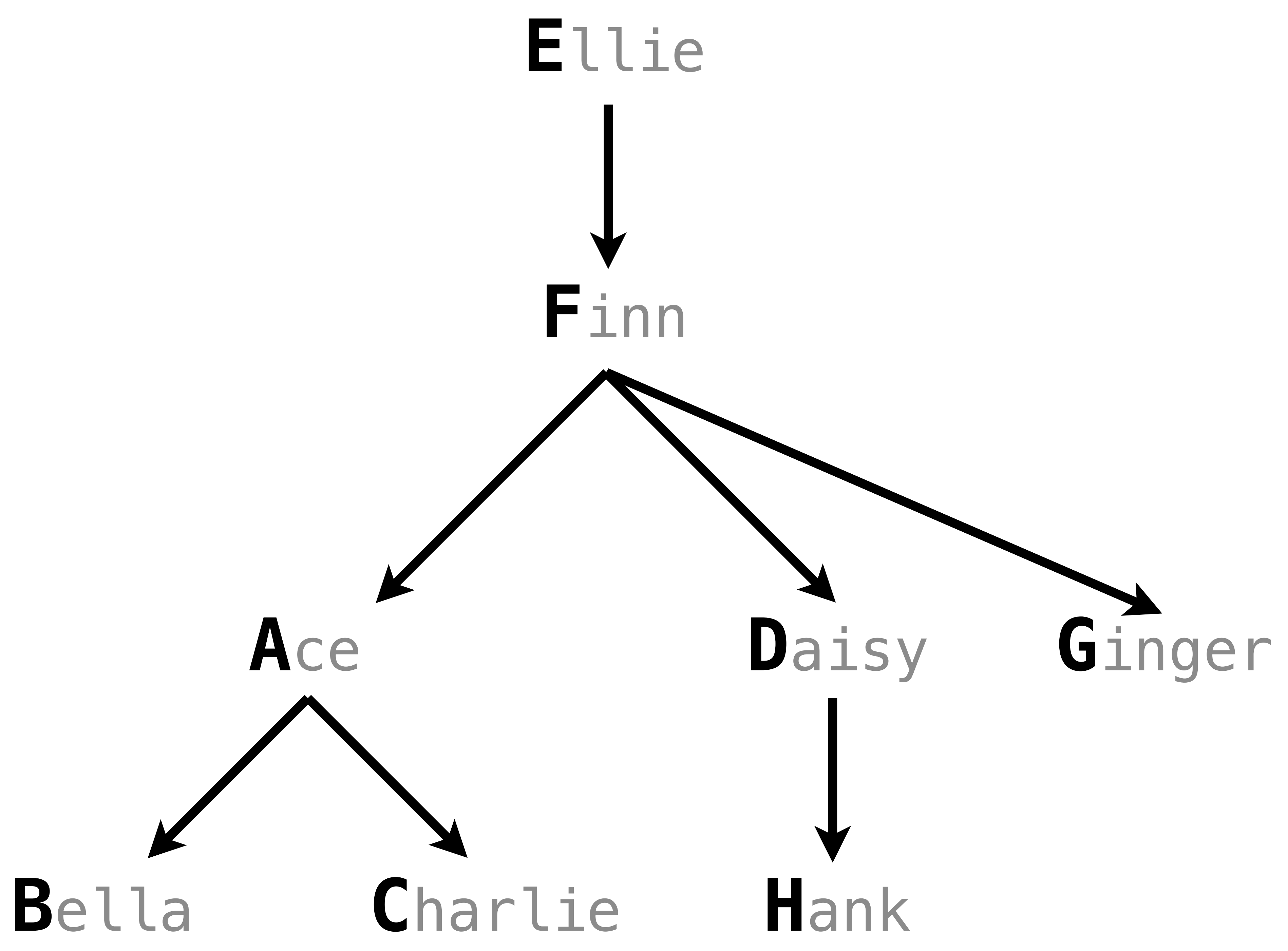
Parents:

parent	child
a	b
a	c
d	h
f	a
f	d
f	g
e	f

Joining Tables

Dog Family Tree

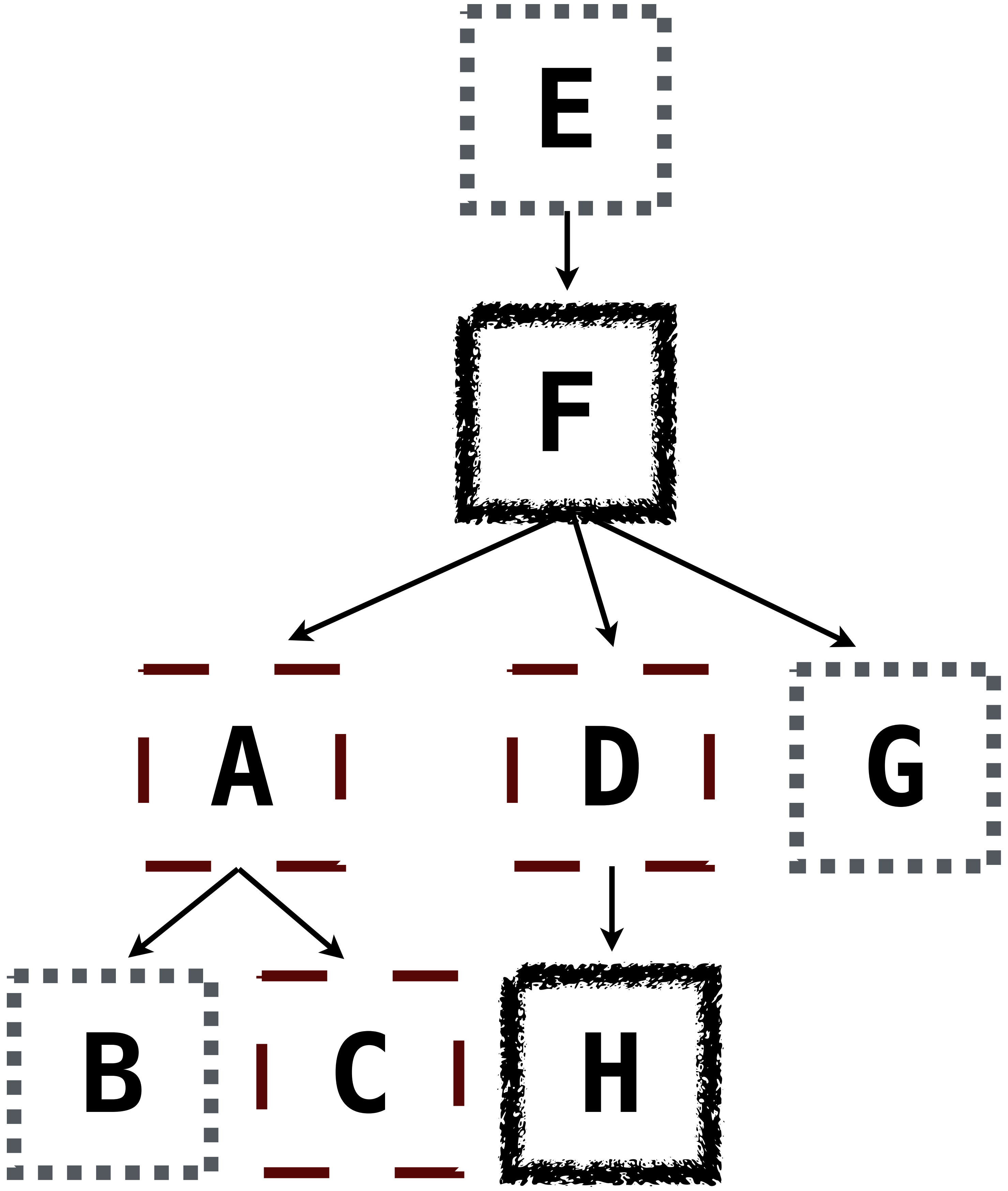
```
CREATE TABLE parents AS
  SELECT "ace" AS parent, "bella" AS child UNION
  SELECT "ace"          , "charlie"        UNION
  SELECT "daisy"        , "hank"           UNION
  SELECT "finn"         , "ace"            UNION
  SELECT "finn"         , "daisy"          UNION
  SELECT "finn"         , "ginger"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



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
  ...;
```



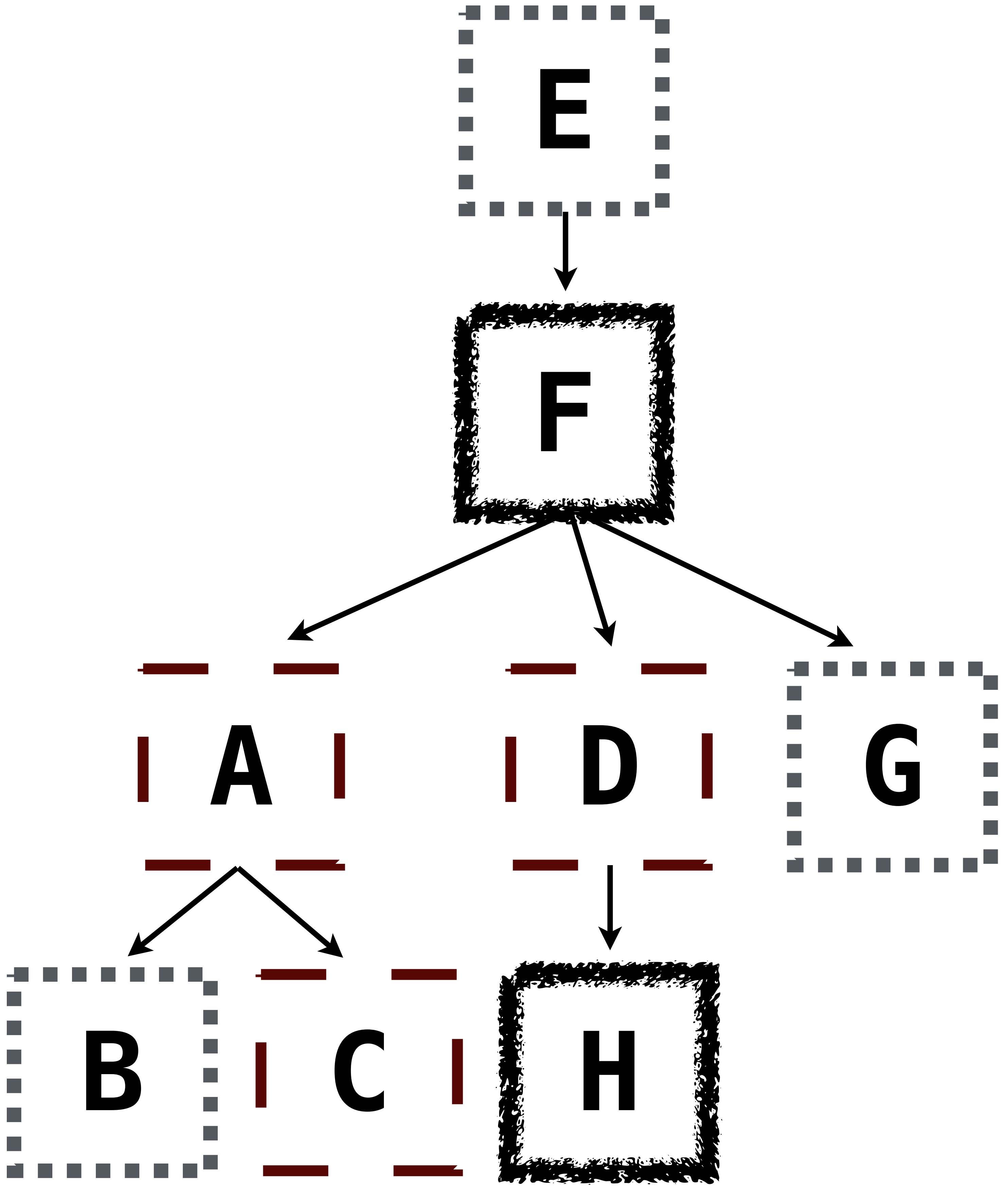
Select the parents of curly-furred dogs

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

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

(Demo)

Practice 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, dogs WHERE parent=name
                     AND 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

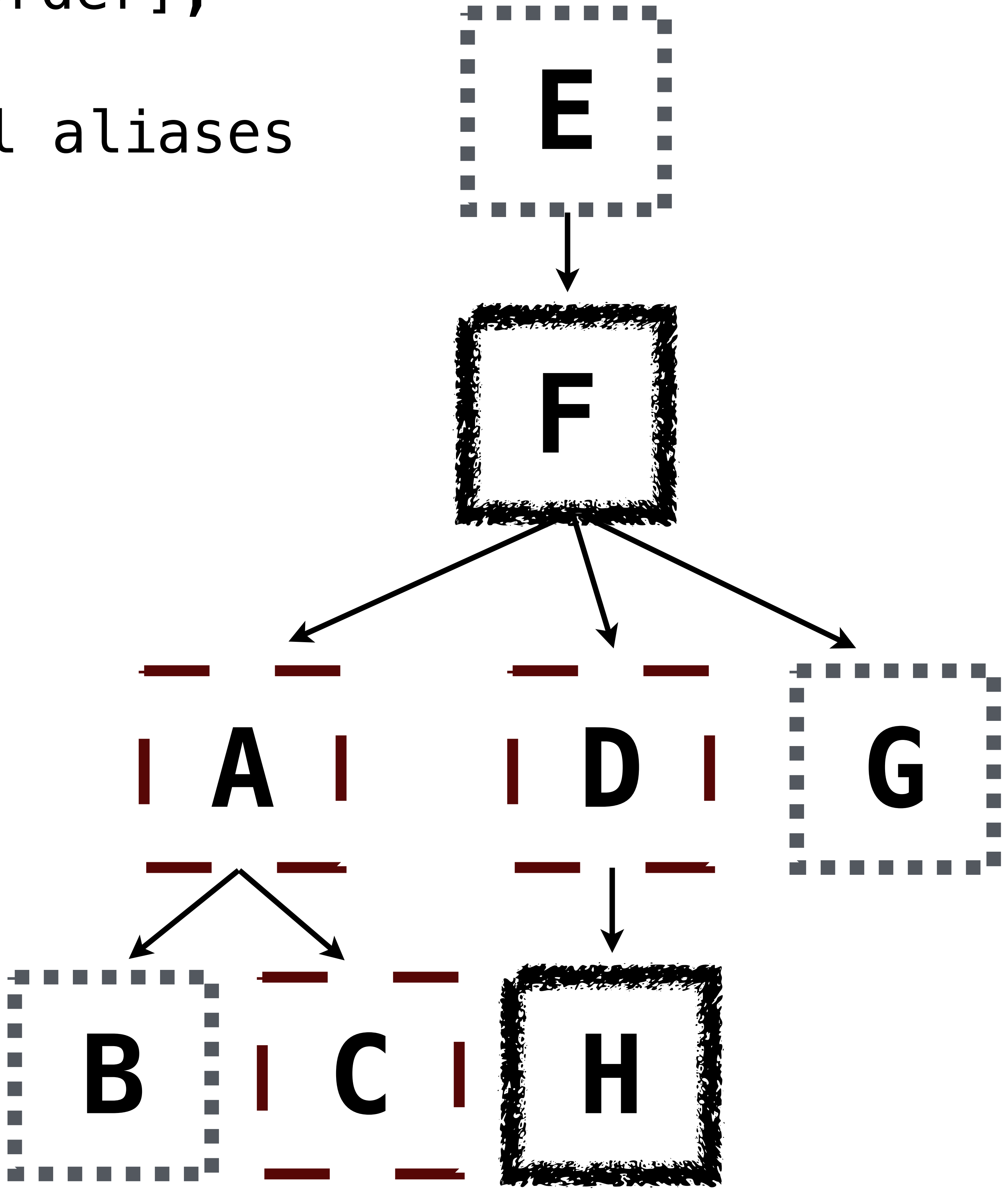
```
SELECT [columns] FROM [table] WHERE [condition] ORDER BY [order];
```

[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



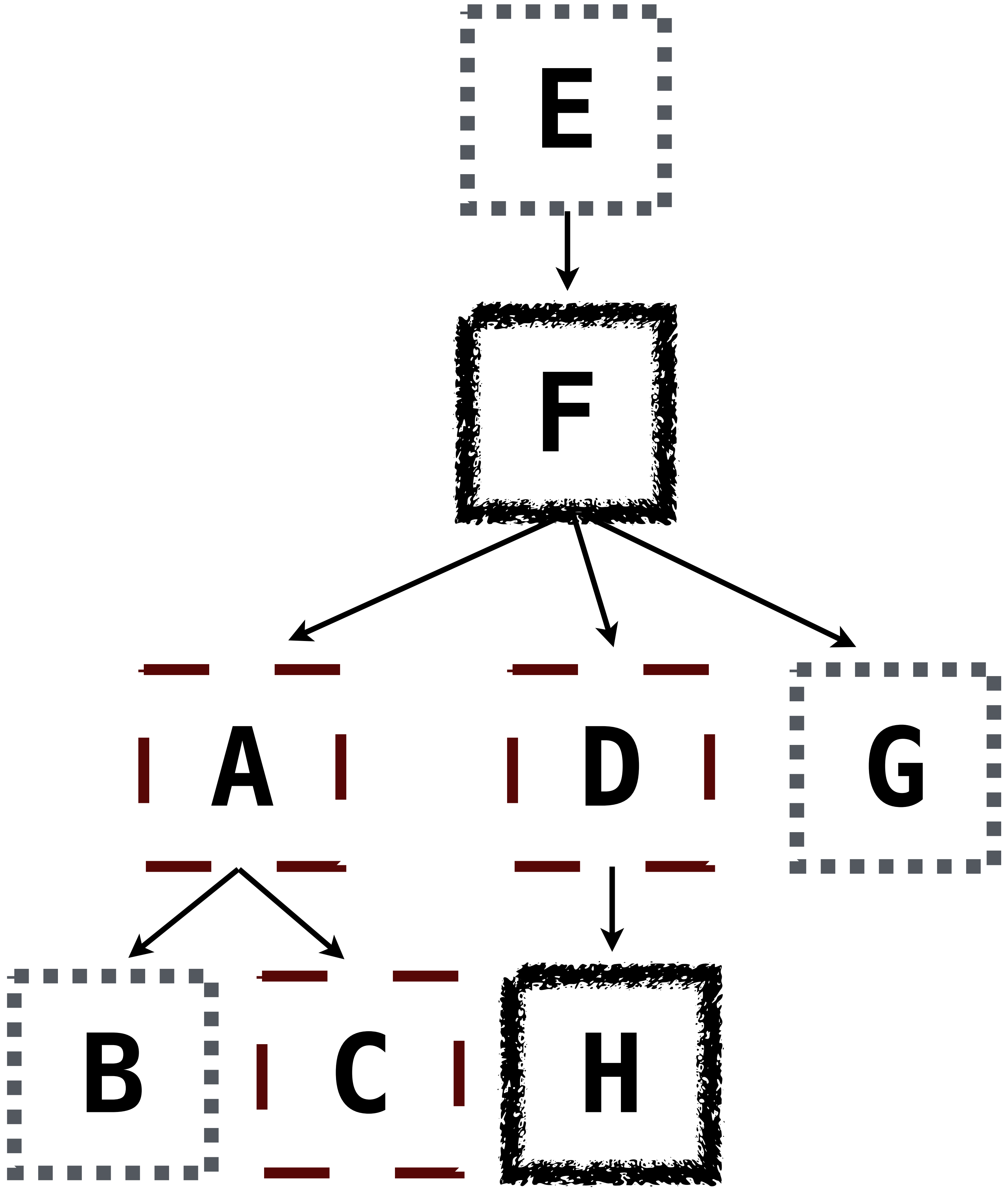
(Demo)

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)

String Expressions

(Demo)