Tables

Announcements

Review: Select Statements Project Existing Tables

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

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

A **SELECT** statement specifies an input table using `FROM [table]`

3

We can optionally use `[column] AS [name]` to rename the input column in our new table.

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

A subset of the rows can be selected (ie. filtered) using `WHERE [condition]`

An ordering can be declared using `ORDER BY [column]`

```
CREATE TABLE [name] AS [SELECT statement goes here];
```

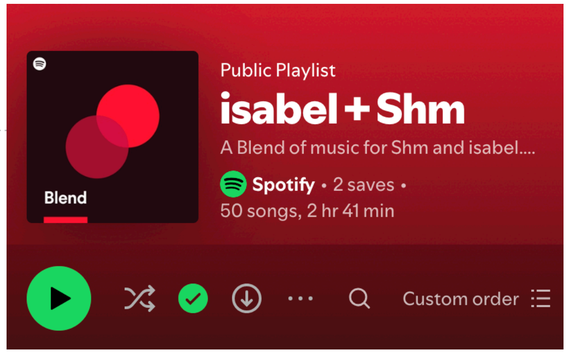
saves the result of a **SELECT** statement to your database for reuse.

SQL is not capitalization or indentation sensitive! (yay)

`;` signals the end of your SQL statement.

Joining Tables

Example: Music with Friends



Create (and save) this short table:

```
CREATE TABLE shm_tracks AS
SELECT "360" AS track, “charli" AS artist UNION
SELECT “cinderella”  , "remi"             UNION
SELECT “wildflower”  , “billie”;
```

shm_tracks:

track	artist
360	charli
cinderella	remi
wildflower	billie

Then display it with another select statement:

```
SELECT * FROM shm_tracks;
```

(You can use any SQL interpreter, ex: the one on code.cs61a.org)

Example: Music with Friends

Now create (and save) *this* short table:

```
CREATE TABLE anya_tracks AS
SELECT "apple" AS track, "charli" AS artist UNION
SELECT "taste"         , "sabrina"          UNION
SELECT "wildflower"    , "billie";
```

anya_tracks:

track	artist
apple	charli
taste	sabrina
wildflower	billie

Then display it with another select statement:

```
SELECT * FROM anya_tracks;
```

(tip: you can use the up arrow to reuse the last line of code you entered)

Example: Music with Friends

Challenge: Write a SELECT statement that will find and display ***a table of all the tracks that these two friends have in common.***

(And ideally, one that will work even if we had way more songs!)

shm_tracks:

track	artist
360	charli
cinderella	remi
wildflower	billie

anya_tracks:

track	artist
apple	charli
taste	sabrina
wildflower	billie

First: How would you (as a human) do this systematically?

Idea: Take **each row** of the first table and **compare it with every row in the second table.**

How many comparisons will we make in this case?

Joining Two Tables

Tables **A** & **B** are *joined* by a comma (or **JOIN**) to form all combos of a row from **A** & a row from **B**.
try this:

```
SELECT * FROM shm_tracks,anya_tracks;
```

shm_tracks, anya_tracks:

track	artist	track	artist
360	charli	apple	charli
360	charli	taste	sabrina
360	charli	wildflower	billie
cinderella	remi	apple	charli
cinderella	remi	taste	sabrina
cinderella	remi	wildflower	billie
wildflower	billie	apple	charli
wildflower	billie	taste	sabrina
wildflower	billie	wildflower	billie

```
SELECT track FROM shm_tracks,anya_tracks; -> Parse error: ambiguous column name: track
```

Working with our joined table will be clearer and easier if we rename the columns!

Aliases and Dot Expressions

Joining Tables that Share Column Names

Two tables may share a column name;
dot expressions help us disambiguate column values.

```
SELECT [column] FROM [table];

SELECT [table.column AS new_column_name, table.column AS new_column_name] FROM [tables];
```

comma separated list of columns with new names for each

comma-separated list of tables

```
SELECT
shm_tracks.track AS s_track,
shm_tracks.artist AS s_artist,

anya_tracks.track AS a_track,
anya_tracks.artist AS a_artist

FROM shm_tracks, anya_tracks;
```

s_track	s_artist	a_track	a_artist
360	charli	apple	charli
360	charli	taste	sabrina
360	charli	wildflower	billie
cinderella	remi	apple	charli
cinderella	remi	taste	sabrina
cinderella	remi	wildflower	billie
wildflower	billie	apple	charli
wildflower	billie	taste	sabrina
wildflower	billie	wildflower	billie

(reminder: you can use the up arrow to reuse the last line of code you entered)

Example: Music with Friends (final)

```
SELECT
shm_tracks.track AS s_track, shm_tracks.artist AS s_artist,

anya_tracks.track AS a_track, anya_tracks.artist AS a_artist

FROM shm_tracks, anya_tracks

WHERE s_track = a_track OR a.artist = b.artist ;
```

s_track	s_artist	a_track	a_artist
360	charli	apple	charli
360	charli	taste	sabrina
360	charli	wildflower	billie
cinderella	remi	apple	charli
cinderella	remi	taste	sabrina
cinderella	remi	wildflower	billie
wildflower	billie	apple	charli
wildflower	billie	taste	sabrina
wildflower	billie	wildflower	billie

s_track	s_artist	a_track	a_artist
wildflower	billie	wildflower	billie

How would you add to the WHERE condition such that the table *also* contains any tracks with shared *artists*?

Example: Adding to a table

You can insert a new row into a table like so:

```
INSERT INTO <table> VALUES (<column1>, <column2>);
    (make sure the # of values matches the # and expected order of columns!)

INSERT INTO shm_tracks VALUES ("bad guy", "billie");
INSERT INTO shm_tracks VALUES ("apple", "charli");
```

shm_tracks:

track	artist
360	charli
apple	charli
bad guy	billie
cinderella	remi
wildflower	billie

How can I create a table like this, showing **pairs of songs** from the same artist?

track1	track2	artist
360	apple	charli
bad guy	wildflower	billie

Joining a Table with Itself

Dot expressions and **aliases** help disambiguate columns from copies of the same table.

```
SELECT [columns]
FROM [table];

SELECT [alias1.column AS new_column_name, alias2.column AS new_column_name]
FROM [table AS alias1, table AS alias2];
```

```
SELECT a.track AS track1, b.track AS track2
FROM shm_tracks AS a, shm_tracks AS b;
```

How many rows and columns will there be in the table displayed by this SELECT statement?

shm_tracks: (not yet joined with itself)

track	artist
360	charli
apple	charli
bad guy	billie
cinderella	remi
wildflower	billie

Finding Pairs of Songs

How can I create a table like this, showing **pairs of songs** from the same artist?

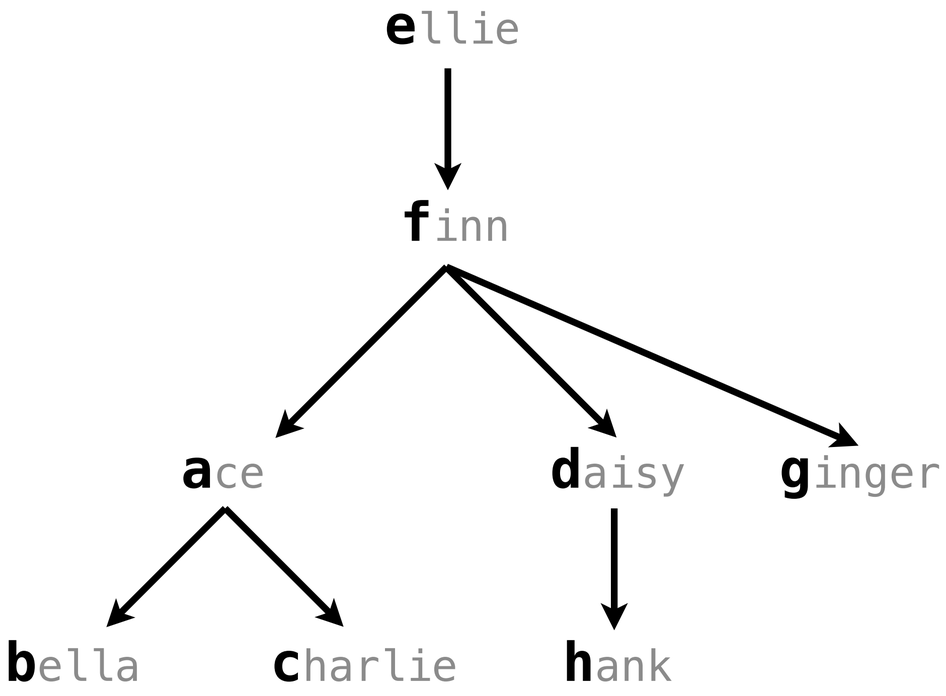
track1	track2	artist
360	apple	charli
bad guy	wildflower	billie

```
SELECT a.track AS track1, b.track AS track2
FROM shm_tracks AS a, shm_tracks AS b
WHERE a.artist = b.artist AND a.track < b.track;
```

(reminder: you can use the up arrow to reuse the last line of code you entered)

Joining Tables Example: Dog Breeder (from the videos)

These tables are built into the SQL interpreter on code.cs61a.org!

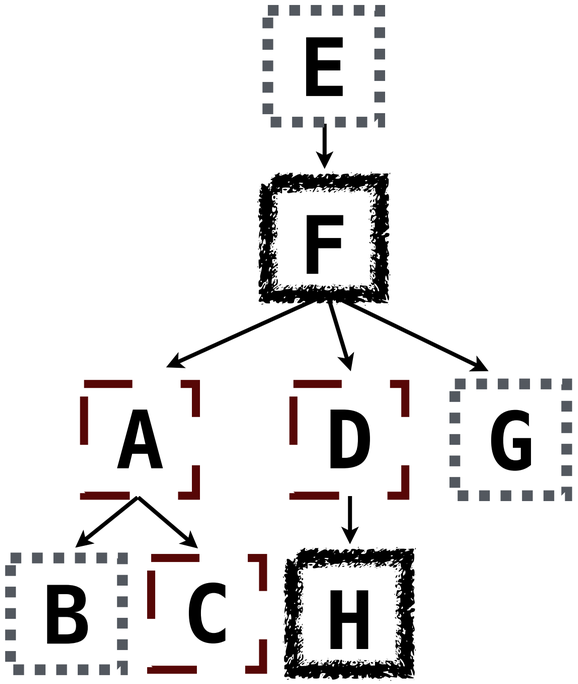


parents:

parent	child
ace	bella
ace	charlie
daisy	hank
ellie	finn
finn	ace
finn	daisy
finn	ginger

dogs:

name	fur
ace	long
bella	short
charlie	long
daisy	long
ellie	short
finn	curly
ginger	short
hank	curly



Write a SELECT statement to display a table containing **the parents of curly haired dogs.**

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

Joining a Table with Itself Example: Grandparents

Which select statement evaluates to all grandparent, grandchild pairs?

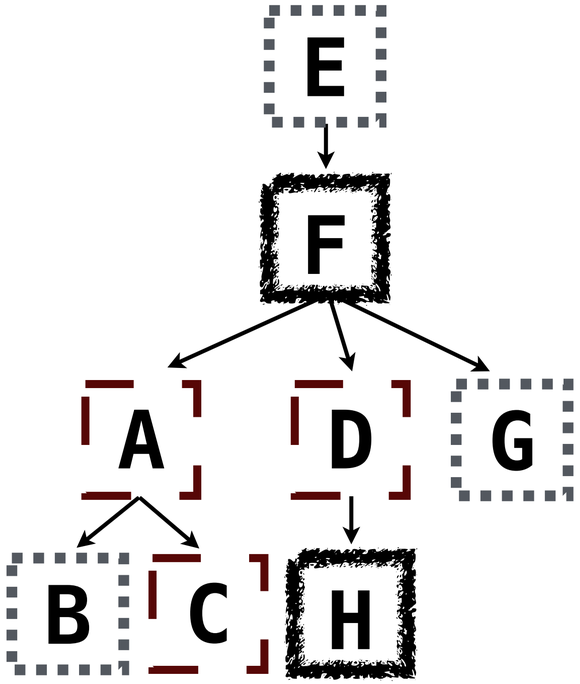
1. `SELECT a.grandparent, b.child FROM parents AS a, parents AS b WHERE b.parent = a.child;`

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

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

4. `SELECT a.grandparent, b.child FROM parents AS a, parents AS b WHERE a.parent = b.child;`

5. None of the above



Joining Multiple Tables

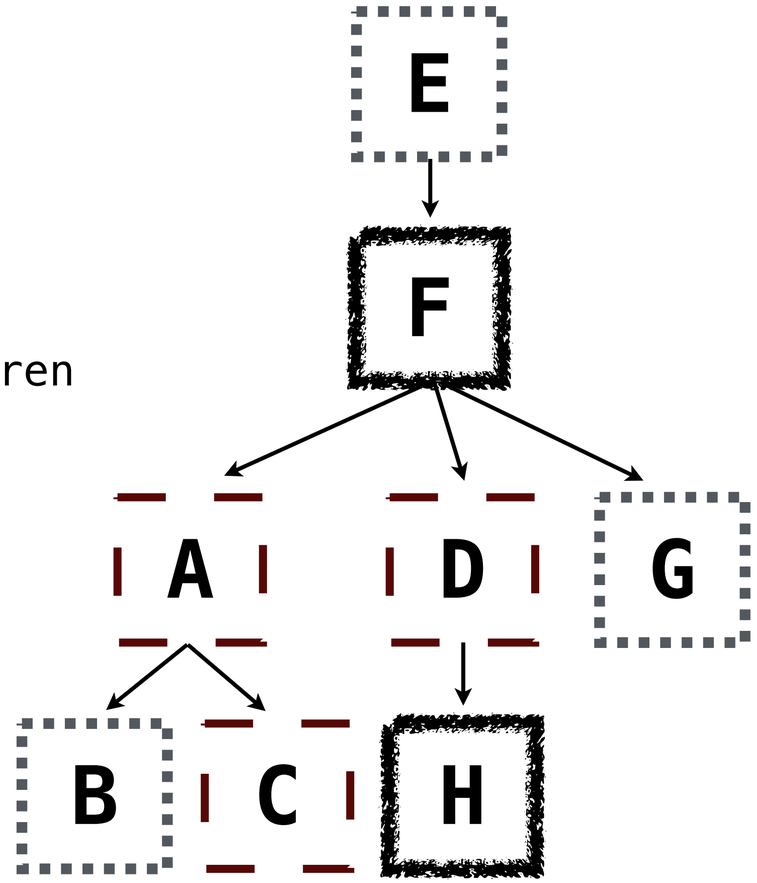
Multiple tables can be joined to yield all combinations of rows from each

```
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;
```



Numerical Expressions

Numerical Expressions

Expressions can contain function calls and arithmetic operators

```
[expression] AS [name], [expression] AS [name], ...

SELECT [columns] FROM [table] WHERE [expression] ORDER BY [expression];
```

Combine values: +, -, *, /, %, and, or

Transform values: abs, round, not, -

Compare values: <, <=, >, >=, <>, !=, =

(Demo)

String Expressions

String Expressions

String values can be combined to form longer strings

```
sqlite> SELECT "hello," || " world";
hello, world
```

Basic string manipulation is built into SQL, but differs from Python

```
sqlite> CREATE TABLE phrase AS SELECT "hello, world" AS s;
sqlite> SELECT substr(s, 4, 2) || substr(s, instr(s, " ")+1, 1) FROM phrase;
low
```

Strings can be used to represent structured values, but doing so is rarely a good idea

```
sqlite> CREATE TABLE lists AS SELECT "one" AS car, "two,three,four" AS cdr;
sqlite> SELECT substr(cdr, 1, instr(cdr, ",")-1) AS cadr FROM lists;
two
```

(Demo)