

# SQL Tables

# Class outline:

- Creating tables
- Joining tables
- Table aliases
- Numerical expressions
- String expressions

# Creating tables

# Creating tables with SELECT

The **CREATE TABLE** statement can be used to create a table in various ways.

Creating a table with the results of a **SELECT**:

```
CREATE TABLE top_music_videos AS  
    SELECT title, views FROM songs ORDER BY views DESC;
```

That limits the new table to a subset of existing data, however.

# Creating tables with UNION

It's possible to use a **SELECT** to create a row of entirely new data, and save that into a table.

```
CREATE TABLE musical_movies AS  
    SELECT "Mamma Mia" as title, 2008 as release_year;
```

# Creating tables with UNION

It's possible to use a **SELECT** to create a row of entirely new data, and save that into a table.

```
CREATE TABLE musical_movies AS
  SELECT "Mamma Mia" as title, 2008 as release_year;
```

We can use **UNION** to merge the results of multiple **SELECT** statements:

```
CREATE TABLE musical_movies AS
  SELECT "Mamma Mia" as title      , 2008 as release_year UNION
  SELECT "Olaf's Frozen Adventure", 2017  UNION
  SELECT "Across the Universe"    , 2007  UNION
  SELECT "Moana"                  , 2016  UNION
  SELECT "Moulin Rouge"           , 2001;
```

## 2-step table creation

The most common approach is to first use **CREATE** to declare the column names and types:

```
CREATE TABLE musical_movies (title TEXT, release_year INTEGER);
```

Then use **INSERT** to insert each row of data:

```
INSERT INTO musical_movies VALUES ("Mamma Mia", 2008);  
INSERT INTO musical_movies VALUES ("Olaf's Frozen Adventure", 2017);  
INSERT INTO musical_movies VALUES ("Across the Universe", 2007);  
INSERT INTO musical_movies VALUES ("Moana", 2016);  
INSERT INTO musical_movies VALUES ("Moulin Rouge", 2001);
```

# Joining related tables



# Related tables

A table is related to another table if two columns describe the same piece of information.

section			
<b>id</b>	<b>capacity</b>	<b>staff_id</b>	<b>tag_string</b>
145	35	142	Regular
146	36	188	Zoom
147	36	144	Scholars
148	45	145	Transfer
149	45	174	Regular

user			
<b>id</b>	<b>email</b>	<b>name</b>	<b>section_id</b>
192	ana_kerluke@berkeley.edu	Ana Kerluke	149
255	paige_witheiser@berkeley.edu	Paige Wintheiser	149
270	leanna.feest@berkeley.edu	Leanna Feest	149
387	marcelo35@berkeley.edu	Marcelo Gruno	149
401	baron95@berkeley.edu	Baron Weiss	149

# John's dogs

dogs

<b>name</b>	<b>fur</b>
abraham	long
barack	short
clinton	long
delano	long
eisenhower	short
fillmore	curly
grover	short
herbert	curly

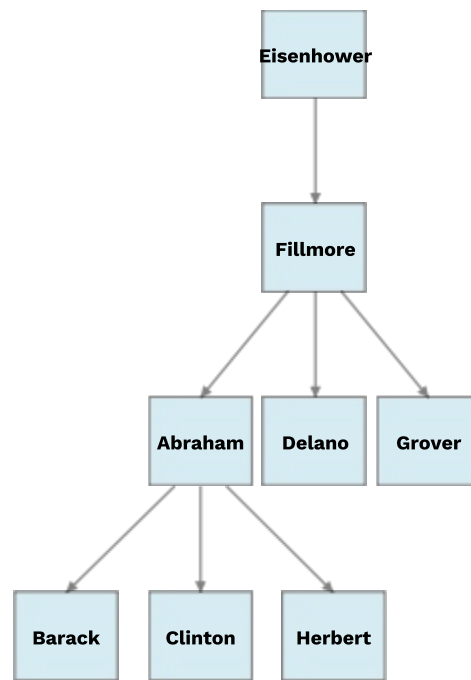
parents

<b>parent</b>	<b>child</b>
abraham	barack
abraham	clinton
delano	herbert
fillmore	abraham
fillmore	delano
fillmore	grover
fillmore	grover
eisenhower	fillmore

# Dog family tree (visualized)

parents

<b>parent</b>	<b>child</b>
abraham	barack
abraham	clinton
delano	herbert
fillmore	abraham
fillmore	delano
fillmore	grover
fillmore	grover
eisenhower	fillmore



# Joining related tables

A join on two tables A and B yields all combinations of a row from table A and a row from table B.

Select the parents of curly-furred dogs:

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

# Joining a table with itself

Two tables may share a column name (especially when they're the same table!). Dot expressions and aliases disambiguate column values.

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;
```

# Exercise: Grandparents

Which statement evaluates to all grandparent, grandchild pairs?

A:

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

B:

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

C:

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

D:

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



# Exercise: Grandparents

Which statement evaluates to all grandparent, grandchild pairs?

A: ✗

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

B:

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

C:

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

D:

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

# Exercise: Grandparents

Which statement evaluates to all grandparent, grandchild pairs?

A: ✗

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

B: ✗

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

C:

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

D:

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

# Exercise: Grandparents

Which statement evaluates to all grandparent, grandchild pairs?

A: ✗

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

B: ✗

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

C: ✓

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

D:

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

# Exercise: Grandparents

Which statement evaluates to all grandparent, grandchild pairs?

A: ✗

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

B: ✗

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

C: ✓

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

D: ✗

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

# Joining more than 2 tables

Starting with this table...

```
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 style as their grandchildren:

# Joining more than 2 tables

Starting with this table...

```
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 style as their grandchildren:

```
SELECT grandog FROM grandparents, dogs AS c, dogs AS d  
  WHERE grandog = c.name AND  
  granpup = d.name AND  
  c.fur = d.fur;
```

# Exercise: Dog Triples

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.

Expected output:

```
delano|clinton|abraham  
grover|eisenhower|barack
```

# Expressions



# Numerical expressions

Multiple parts of a `SELECT` statement can include an **expression**.

```
SELECT [result-column] FROM [table] WHERE [expr];
```

`result-column` can expand to either `expr` AS `column-alias` or `*`.

Expressions can contain function calls and arithmetic operators.

- Combine values: `+`, `-`, `*`, `/`, `%`, `and`, `or`
- Transform values: `ABS()`, `ROUND()`, `NOT`, `-`
- Compare values: `<`, `<=`, `>`, `>=`, `<>`, `!=`, `=`

See all the possibilities for `expressions`.

# String expressions

The `||` operator does string concatenation:

```
SELECT (views || "M") as total_views FROM songs;
```

There are basic functions for string manipulation as well:

```
SELECT SUBSTR(release_year, 3, 2) AS two_digit_year  
FROM songs ORDER BY two_digit_year;
```