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 SQL `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.

```sql
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:

```sql
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.

<table>
<thead>
<tr>
<th>id</th>
<th>capacity</th>
<th>staff_id</th>
<th>tag_string</th>
</tr>
</thead>
<tbody>
<tr>
<td>145</td>
<td>35</td>
<td>142</td>
<td>Regular</td>
</tr>
<tr>
<td>146</td>
<td>36</td>
<td>188</td>
<td>Zoom</td>
</tr>
<tr>
<td>147</td>
<td>36</td>
<td>144</td>
<td>Scholars</td>
</tr>
<tr>
<td>148</td>
<td>45</td>
<td>145</td>
<td>Transfer</td>
</tr>
<tr>
<td>149</td>
<td>45</td>
<td>174</td>
<td>Regular</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>email</th>
<th>name</th>
<th>section_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>192</td>
<td><a href="mailto:ana_kerlue@berkeley.edu">ana_kerlue@berkeley.edu</a></td>
<td>Ana Kerlue</td>
<td>149</td>
</tr>
<tr>
<td>255</td>
<td><a href="mailto:paige_wintheiser@berkeley.edu">paige_wintheiser@berkeley.edu</a></td>
<td>Paige Wintheiser</td>
<td>149</td>
</tr>
<tr>
<td>270</td>
<td><a href="mailto:leanna.feest@berkeley.edu">leanna.feest@berkeley.edu</a></td>
<td>Leanna Feest</td>
<td>149</td>
</tr>
<tr>
<td>387</td>
<td><a href="mailto:marcelo35@berkeley.edu">marcelo35@berkeley.edu</a></td>
<td>Marcelo Gruno</td>
<td>149</td>
</tr>
<tr>
<td>401</td>
<td><a href="mailto:baron95@berkeley.edu">baron95@berkeley.edu</a></td>
<td>Baron Weiss</td>
<td>149</td>
</tr>
</tbody>
</table>
# John's dogs

<table>
<thead>
<tr>
<th>name</th>
<th>fur</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>long</td>
</tr>
<tr>
<td>barack</td>
<td>short</td>
</tr>
<tr>
<td>clinton</td>
<td>long</td>
</tr>
<tr>
<td>delano</td>
<td>long</td>
</tr>
<tr>
<td>eisenhower</td>
<td>short</td>
</tr>
<tr>
<td>fillmore</td>
<td>curly</td>
</tr>
<tr>
<td>grover</td>
<td>short</td>
</tr>
<tr>
<td>herbert</td>
<td>curly</td>
</tr>
</tbody>
</table>

parents
<table>
<thead>
<tr>
<th>parent</th>
<th>child</th>
</tr>
</thead>
<tbody>
<tr>
<td>abraham</td>
<td>barack</td>
</tr>
<tr>
<td>abraham</td>
<td>clinton</td>
</tr>
<tr>
<td>delano</td>
<td>herbert</td>
</tr>
<tr>
<td>fillmore</td>
<td>abraham</td>
</tr>
<tr>
<td>fillmore</td>
<td>delano</td>
</tr>
<tr>
<td>fillmore</td>
<td>grover</td>
</tr>
<tr>
<td>fillmore</td>
<td>grover</td>
</tr>
<tr>
<td>eisenhower</td>
<td>fillmore</td>
</tr>
</tbody>
</table>
# Dog family tree (visualized)

<table>
<thead>
<tr>
<th>parents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>parent</strong></td>
</tr>
<tr>
<td>abraham</td>
</tr>
<tr>
<td>abraham</td>
</tr>
<tr>
<td>delano</td>
</tr>
<tr>
<td>fillmore</td>
</tr>
<tr>
<td>fillmore</td>
</tr>
<tr>
<td>fillmore</td>
</tr>
<tr>
<td>fillmore</td>
</tr>
<tr>
<td>eisenhower</td>
</tr>
</tbody>
</table>
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:

```sql
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:

```sql
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:

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

B:

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

C:

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

D:

```sql
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: ✗

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

B:

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

C:

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

D:

```sql
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: ✗

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

B: ✗

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

C:

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

D:

```sql
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: ✗

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

B: ✗

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

C: ✓

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

D:

```sql
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...

```sql
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:
Joining more than 2 tables

Starting with this table...

```sql
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:

```sql
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 \texttt{SELECT} statement can include an \texttt{expression}.

\texttt{SELECT [result-column] FROM [table] WHERE [expr];}

\texttt{result-column} can expand to either \texttt{expr AS column-alias} or \texttt{*}.

Expressions can contain function calls and arithmetic operators.

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

See all the possibilities for \texttt{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;
```