Introduction to Databases

Relational Databases with PostgreSQL

- Databases have tables to classify data
- Collections have:
  - **rows**: data defining an entire rectod, e.g. a user
  - **columns**: attributes about the record, e.g. a user's email and birthday

Example table

email	password	birthday	location
jon.miller@nycda.com	penguin	10/10/1988	Amsterdam
katie@patie.com	bubbles123	12/01/1990	Los Angeles
flurble@wurzle.com	bl0rp	01/02/1981	Antarctica

Other databases

- MySQL
- MongoDB
- HBase

Common psql commands

- `psql` is the PostgreSQL command line interface

```
\?          /* help: list available commands */
\c my_app   /* connect to database 'my_app' */
\dt+        /* list tables */
\d+ fruit   /* describe table 'fruit' */
\q          /* quit */
```

PostgresSQL commands: create

- Creates a named table with some information about each record:

```
create table hats (
  name     text,
  material text,
  height   integer,
  brim     boolean
);
```

Exercise

- Create the hats table shown previously.

PostgreSQL commands: insert into

- Adds records into a table with supplied information.

```
insert into hats values ('sun hat', 'straw', 7, true);

insert into hats (name, material, height, brim) values
  ('top hat', 'buckram', 12, true);

insert into hats (name, material, height, brim) values
  ('cloche', 'felt', 6, false),
  ('chicken', 'bwuk bwuk bwuk', 12, false);
```

PostgresSQL commands: select, where

- Retrieves information from a table, optionally given conditions.

```
select * from hats;
select * from hats where name = 'top hat';
select count(*) from hats;
```

PostgreSQL commands: delete from

- Removes data from a table that meet given conditions.

```
delete from hats where name = 'chicken';
```

PostgreSQL commands: alter

- Modify a table.

```
alter table hats add column price integer;
```

Exercise

- Add three new hats to the hats table.
- Select all the hats that are made of felt.

Primary keys

- Say we needed a specific hat - how would we get it?

Primary keys (continued)

- `serial`: auto-incremented integer
- `primary key`: 2 constraints: unique and non-null

```
drop table hats;
create table hats (
  id       serial primary key,
  name     text,
  material text,
  height   integer,
  brim     boolean
);
```

Exercise

- Recreate your hats table with the `serial primary key` added.

Adding to the database

- Notice how the `id` is set automagically.

```
insert into hats (name, material, height, brim) values
  ('cloche', 'felt', 6, false),
  ('top hat', 'buckram', 12, true);
```

Exercise

- Create a users table with three columns:
  - id (should be a serial primary key)
  - name
  - email

Exercise (continued)

- Add three sample users to your table.

"Relational" Databases

- If we had a users table, and each user had a few hats, how would we relate these two tables to each other?

"Relational" Databases (continued)

- Add a `user_id` to the hats table.
- This is called a Many-to-one relationship. (many hats, one user)

```
alter table hats add column user_id integer;
```

Constraints

- If we wanted to make sure no two hats belonged to the same user, what constraint would we add?
- If we wanted to make sure the `user_id` was required, what constraint would we add?
- Coming up: If we wanted to make sure the user that the `user_id` was pointing to actually existed, what constraint would we use?

Foreign keys

```
alter table hats drop column user_id integer;
alter table hats add column user_id integer references users;
```

Foreign keys (continued)

- Note the error message: since the `user_id` 10 does not exist, PostgreSQL will complain.

```
insert into hats (name, material, height, brim, user_id) values
    ('bowler', 'velvet', 6, false, 10);
```

Exercise

- Insert two more hats which reference valid users.

Select: Chaining Queries

- What if we wanted to get all the hats that belonged to a particular user, but we only had his email?

```
select * from hats where user_id = (select (id) from users where email = 'josh@gmail.com');
```