Lecture 4A

SQL DDL

In last lecture, we saw a bit of SQL:

```
CREATE TABLE person (
  person_id SERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL
);
```

This creates an SQL *table* that can contain data from our Person model.

Several pieces:

- The table name
- The columns, each of which hsa
    - Column name
    - Type
    - Constraints and other attributes (NOT NULL, PRIMARY KEY)

There are more pieces that we will see later.

Finish SQL DDL for our model

SQL data types

A few families:

- Textual
    - VARCHAR
    - CHAR (rarely, if ever, needed)
    - TEXT
    - In PostgreSQL, VARCHAR and TEXT are really the same thing.
- Numeric
    - INTEGER
        - Various sizes; BIGINT is of particular interest
        - SERIAL is auto-incrementing
    - REAL (or DOUBLE PRECISION): floating-point
        - **Never** store money in REAL/DOUBLE. People have embezzled money in floating-point errors.
    - DECIMAL or NUMERIC: exact decimal numbers
        - Use this for money

        - But it is less efficient than REAL
- Date/time
    - DATE
    - TIME
    - TIMESTAMP (date + time)
    - May have time zones
- BOOLEAN (newer addition; some systems don't have it, use TINYINT to fake)
- BLOB (Binary Large OBject, for storing blocks of bytes like image files)

We'll be getting back to these more later.

Defining Foreign Keys

We'll see this in the example code.

Inserting Data

The INSERT statement inserts data.

```
INSERT INTO person (name) VALUES ('Alice');
```

The primary key (person_id) is auto-generated!

PostgreSQL Extension: if you want to get the ID that was just added, put 'RETURNING person_id' at the end of the statement! Super-useful.

Writing Queries

SELECT * FROM person;

SELECT person_id FROM person; -- project!

SELECT * FROM person WHERE name = 'Alice'; -- select!