SQL for data analysis with PostgreSQL
What is SQL

- **Declarative Language**
  
  Say what you want, and the interpreter figures out how to get it

- **Designed to mimic natural language**

  ```sql
  CREATE TABLE patients ( height numeric, name text );
  INSERT INTO patients ( height, name ) VALUES ( 70, 'Nathan' );
  SELECT height FROM patients WHERE name = 'Nathan';
  UPDATE patients SET height = 71 WHERE name = 'Nathan';
  DELETE FROM patients WHERE name = 'Nathan Boley';
  DROP TABLE patients;
  ```
What is SQL

- SQL is usually interpreted by an RDBMS
- Three distinct steps in executing a query
  - Parse
    - Figure out what the query is asking for
  - Plan
    - Decide the best way to get the data
  - Execute
    - Actually get the information
Tables, Types, and Operators

- **Tables**
  - Just like R data frames
  - Store columns of data
    - Each column has a single type
  - The columns are stored together in records
    - Elements in the same row are grouped together

- **Types**
  - Standard types (int, float, char, bool, enum, temporal, etc.)
  - Exotic Types (geometric, text search, trees, hashes, …)
  - Custom Types
Operators

- Set Operators
  - Union, Difference, etc.
  - Multi-sets!
    - Union All
- Join Operators
  - Inner, Outer, Semi, Anti, Cartesian
- Aggregates
  - Sum over ( stuff ), GROUP BY
  - Customizable
An Example

(from 215a Lab 1)

- 2 sets of readings from sensors
  - 1 from logged data, 1 from network data
  - Often the data points correspond
- 2 distinct trees
- Plenty of noise
  - Voltage readings differ
  - Non-physical readings
- Associated Meta-data
An Example

Setup The Environment
An Example

Load the Data
An Example

(try to) Join the Data
Constraints

- Tell the database what you expect the data to look like.
  - I.e., you might want to make sure that every value in a height column is positive
  - Use constraints liberally! They make sure your data is, and stays, clean

- Unique Contraints
  - Makes sure that every value is unique
    - Postgres has generalized unique contraints
  - Primary Keys are a special unique constraint
- **Visibility**
  - MVCC and Postgresql

- **Syntax**
  - `BEGIN;`
  - `( do stuff )`
  - `SAVEPOINT a;`
  - `( do stuff, error or want to go back )`
  - `ROLLBACK TO SAVEPOINT a;`
  - `( do stuff )`
  - `COMMIT;`
An Example

Finish Joining the Data
An Example

Get the data into R, and plot it
Long story short, a log-log linear model fit the data very well. So, we use this to transform the data in the net_voltage relation.
An Example

Finally join all the data together
An Example

- Views
  - A saved query
    - Still relations
  - Planner treats them just like tables

- Analyze
  - See what the planner decides to do!
  - Good for measuring performance
Views Example
An Example

Aggregates
More Cool Stuff

- **Window Functions**
  - Moving 'averages' for any aggregate
    - This means you can define an aggregate, and automatically use it over series

- **Geometric Types**
  - SELECT name, loc FROM restaurants
    WHERE loc in ( SELECT bounding_box WHERE zip_code = 94707 ) as zip_box;
  - PostGis
More Cool Stuff

- Range Types (in 9.2)
  - Build interval types (i.e., genomic regions)
  - Run queries of the form
    ```sql
    SELECT * FROM features WHERE feature OVERLAPS (other features sub-query)
    ```
- Full Text Search
- Durability
- Replication
- CTE's (if time, an example)
Summary

- SQL is powerful
  - With CTE's turing complete
- Databases can make your life easier
  - Durability means your data is safe
  - Transactional isolation means parallelizing work is simple
  - Declarative model means you don't have to waste time thinking about/learning algorithms
  - Wide availability means you can keep your data in one place, and use the right tool for the job