SQL 1

August 1, 2023

Antonio Kam

Who am I?

- Hi I’m Anto (he/him/his)
- I like cats
- I like cubes

Announcements

- Scheme Project
  - Checkpoint 1 (Phase 1) due today
  - Checkpoint 2 (Phase 2 & 3) due Friday
  - Project due 8/8 (Tuesday)
  - My recommendation - get the project done by/before the weekend - OH will be more busy Monday/Tuesday
    - Project Parties
- Scheme Art Contest
- Less than 2 weeks until the final
  - Only a couple more assignments!

Declarative Programming

Declarative Programming

Today, we will dive deeper into declarative programming

- In **declarative languages** such as Regex and SQL, we just told the computer the expression and it found a match. We didn't tell it how to do that, it just figured it out
- In **imperative languages** such as Python and Scheme, we tell the computer the steps that we want it to do, and it'll evaluate the steps that you wrote as is
- Today, we will see another declarative language used primarily for working with data

Imperative vs. Declarative

- With SQL, you won’t need to worry about how it decides to handle your queries.
- All you need to do is give it instructions for *what* you want, and under the hood, SQL will find a way to give the result to you.

Poll

Imperative vs. Declarative

We are going out to dinner and we need a table

Option 1:

I’m going to look through the reservation book , see if there are any open slots currently that are available. Then confirm that these tables are for 2 people and are available for 90 minutes. Finally, ask the waitress for that table because you know it works

Option 2:

Table for 2 please.

SQL

Poll

SQL

Who here has heard of SQL?

Who here has worked with SQL?

Poll

SQL

We are going to look more at declarative programming through tables - something that you might have seen before in Google Sheets, Data 8 (through the datascience module), or somewhere else!

SQL (Structured Query Language) is a language that interacts with a database management system (DBMS)

They allow you to store data in a structured way, update it, and query it while optimizing these operations

Relational Databases

SQL deals with relational databases which are tables that have data that are often related to each other

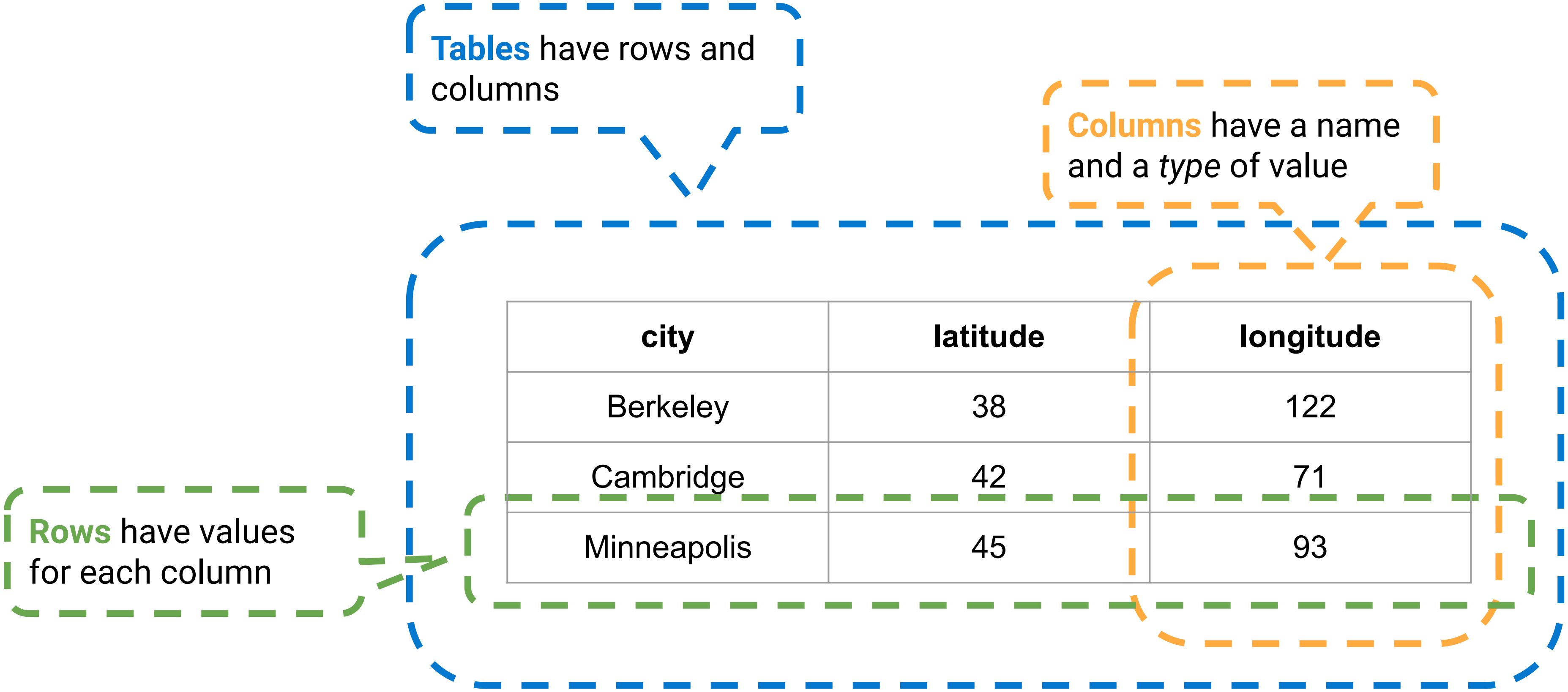
Each of these tables are made of up columns and rows of data and they tables can be related and **connected**

For example, if I'm a restaurant, I might have multiple tables:

- Table 1: Keeps track of the menu: names, prices
- Table 2: Keeps track of orders: Items ordered, total bill price

The structure is very similar to OOP (each table is a different object), but when you need to combine the data together, SQL will let you do that

Tables


Tables have rows and columns
Columns have a name and a *type* of value
Rows have values for each column

city	latitude	longitude
Berkeley	38	122
Cambridge	42	71
Minneapolis	45	93

SQL Overview

- In SQL, there are a few keywords that you'll see often. These are keywords that you'll often think of when working with data
- A select statement will either create a new table from scratch, or grab data from another table
- The create table keyword gives a global name to a table (example later)
- Some other keywords (will see later): where, order by, limit, etc.

How do I make that in SQL?

```
CREATE TABLE cities AS
 SELECT “Berkeley” AS city,    38 AS latitude, 122 AS longitude UNION
 SELECT “Cambridge”       ,    42           ,   71              UNION
 SELECT “Minneapolis”     ,    45           ,   93;
```

SELECT statements create a new table, either by creating one from scratch or taking from an existing table

CREATE TABLE assigns a table to a global name

UNION concatenates tables together to make bigger ones

All statements in SQL must end with a semicolon

Demo

How do I make that in SQL?

```
CREATE TABLE cities AS
 SELECT “Berkeley” AS city,    38 AS latitude, 122 AS longitude UNION
 SELECT “Cambridge”       ,    42          ,   71               UNION
 SELECT “Minneapolis”     ,    45          ,   93;
```

city	latitude	longitude
Berkeley	38	122
Cambridge	42	71
Minneapolis	45	93

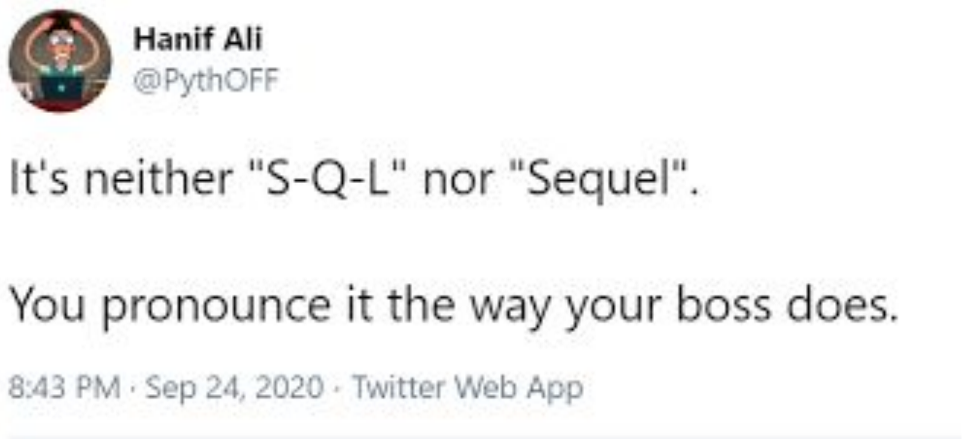
Working with SQL

If you want to experiment with SQL, code.cs61a.org has an SQL interpreter that you can use with built-in tables

Additionally, if you want to experiment more, sqlite is a Python library that lets you work with SQL within your own Python programs

In this lecture, I'll be using DataGrip to interact with my tables. It's pretty hard to set up, so please don't worry about setting it up - code.cs61a.org is more than enough for your use cases!

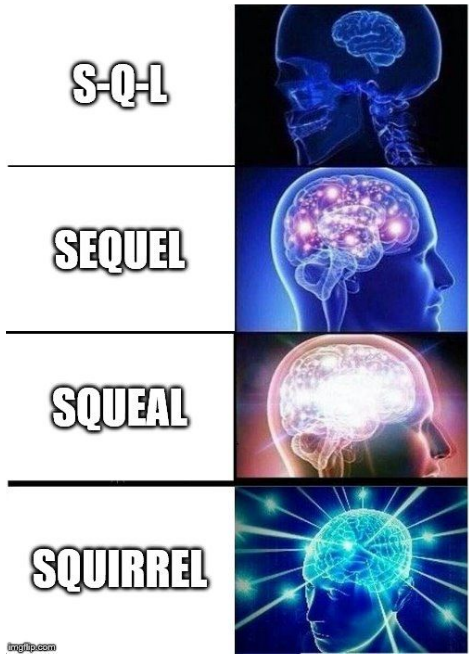
Wait… is it pronounced S Q L or Sequel?


Hanif Ali
@PythOFF

It's neither "S-Q-L" nor "Sequel".

You pronounce it the way your boss does.

8:43 PM · Sep 24, 2020 · Twitter Web App



Querying Data

Similar to what we saw with lists, creating a table is great, but we need to be able to read the data from that table

In SQL, we can just extract all the data from the table, but we can also select certain columns, do arithmetic operations on the columns, and much more!

Select Statements

SELECT statements are going to be the most powerful tool that we have in this class

We saw earlier that it can create tables but it also is how we **query** from tables

SELECT will return 0 or more rows that match our query (remember, the computer figures out which rows to return)

Demo

Code to copy over if needed

```
CREATE TABLE cities AS
 SELECT “Berkeley” AS city,    38 AS latitude, 122 AS longitude UNION
 SELECT “Cambridge”       ,    42            ,  71              UNION
 SELECT “Minneapolis”     ,    45            ,  93;
```

Demo

Examples of Select

Return all rows of the tables

```
SELECT * FROM cities;
```

(The * means get *everything* (in this case, every column))

Return the city and the latitude from all rows in cities

```
SELECT city, latitude from cities;
```

Rename columns in the returned table

```
Select latitude as lat,  longitude as lon FROM cities;
```

Manipulate the column value

```
select city, longitude * 1000 as long_lon from cities;
```

How case sensitive is SQL?

Not very! Typically people use capital letters from the keywords and lowercase for the rest.

Extending SQL

Optional Clauses for Select

So far, we have been able to get different columns from the table, but haven’t really been able to do much more.

Now, we will talk about the optional clauses for the SELECT statement. With more of these clauses, we will see more and more of the use cases of SQL!

Example Table

Table name: `menu`

item	price
Beef Brisket Noodle Soup	13.99
House Special Beef Rib	18.99
Lamb Brisket Rice Bowl	13.99
Chef Special Hand Pulled Lamb Shank	20.99
Lamb Noodle Soup	14.99

A sample of the menu from Noodle Dynasty

Code to create the table above

```sql
create table menu as

    select 'Beef Brisket Noodle Soup' as item, 13.99 as price union

    select 'House Special Beef Rib', 18.99 union

    select 'Lamb Brisket Rice Bowl', 13.99 union

    select 'Chef Special Hand Pulled Lamb Shank', 20.99 union

    select 'Lamb Noodle Soup', 14.99;
```

First Motivation: if

One thing that we might want to do is only select certain rows that match a condition.

For example, let's say I wanted to only show items from the `menu` table that cost less than $15. With what we've seen *so far* in SQL, there isn't a way to do that.

item	price
Beef Brisket Noodle Soup	13.99
~~House Special Beef Rib~~	~~18.99~~
Lamb Brisket Rice Bowl	13.99
~~Chef Special Hand Pulled Lamb Shank~~	~~20.99~~
Lamb Noodle Soup	14.99

Where

Thankfully, we have the WHERE clause!

The WHERE clause will allow you to filter rows

Output all rows and columns in the menu where the item costs less than $15

```
SELECT * FROM menu WHERE price < 15;
```

Output the name of the item if the item costs less than $15

```
SELECT item FROM menu WHERE price < 15;
```

Output the names of the items where the item costs between $14 and $20

```
SELECT item FROM menu WHERE price > 14 AND price < 20;
```

Demo

Second Motivation: sorting

One other thing that you might see with the `menu` table is that it's pretty messy. We can't take a quick glance and know what the cheapest/most expensive item on the menu is.

To fix this, we can sort our data by increasing/decreasing price.

item	price
Beef Brisket Noodle Soup	13.99
Lamb Brisket Rice Bowl	13.99
Lamb Noodle Soup	14.99
House Special Beef Rib	18.99
Chef Special Hand Pulled Lamb Shank	20.99

Demo

Order By

ORDER BY will let you specify the order of the rows

Return the rows sorted by the price (highest to lowest)

```
SELECT * FROM menu ORDER BY price DESC;
```

Return the rows sorted by the price (lowest to highest)

```
SELECT * FROM menu ORDER BY price ASC;
```

Return the rows sorted by the name and price

```
SELECT * FROM menu ORDER BY name DESC, price DESC;
```

Third Motivation: smaller output

Some tables have a lot of rows - one issue is that we might not want to display all rows (we might only want to see the top 3 items by price, for instance)

item	price
Beef Brisket Noodle Soup	13.99
Lamb Brisket Rice Bowl	13.99
Lamb Noodle Soup	14.99
~~House Special Beef Rib~~	~~18.99~~
~~Chef Special Hand Pulled Lamb Shank~~	~~20.99~~

Demo

Limit

The LIMIT clause limits the number of rows that are output.

Return the menu item with the highest price

```
SELECT item FROM menu ORDER BY price DESC LIMIT 1;
```

Return the top 3 menu item in terms of lowest price

```
SELECT item FROM menu ORDER BY price ASC LIMIT 3;
```

Demo

Combining it together

So far, we've only really seen examples of WHERE, ORDER BY, and LIMIT being used by themselves, but we haven't seen too many examples of them being used in conjunction

SQL has a specific order for each keyword:

```
SELECT ___ FROM ___ WHERE ___ ORDER BY ___ LIMIT ___
```

WHERE has to appear before ORDER BY, which has to appear before LIMIT. This is true for every single SQL query.

Remember that SQL is a declarative language - SQL will find a way to output the query that you want. All you need to do is describe what you want based on the syntax

Break

SQL Joins

Motivation

So far everything that we’ve seen can be easily done in a spreadsheet (these are operations that can be found in Google Sheets once you know your way around things)

SQL starts getting far more powerful once you get to more of the SQL-related features.

If you have two separate tables that you want to combine together, SQL ends up being very good for this task!

Combining Related Tables

Typically, you want to keep tables relatively simple and join them together, instead of just storing everything in 1 table

Storing everything in 1 table ends up taking far more space than storing them as two smaller tables. (Think about how data is stored in a table (row * columns))

For reference, some companies can have tens if not hundreds of thousands of tables in their system

Poll

Joining Tables

The way we join tables in CS 61A is to do a *complete join* of all the rows. This means every row from table 1 is matched with every row from table 2 and all the columns are kept

Table 1 has 3 columns and 50 rows. Table 2 has 5 columns and 3 rows.

How many rows does the joined table have?

8 columns, 150 rows

Joining Tables

city	latitude	longitude
Berkeley	38	122
Cambridge	42	71
Minneapolis	45	93

city	latitude	longitude
Berkeley	38	122
Cambridge	42	71
Minneapolis	45	93

Joining Tables

city	latitude	longitude	city	latitude	longitude
Berkeley	38	122	Berkeley	38	122
Cambridge	42	71	Berkeley	38	122
Minneapolis	45	93	Berkeley	38	122

Joining Tables

city	latitude	longitude	city	latitude	longitude
Berkeley	38	122	Berkeley	38	122
Cambridge	42	71	Berkeley	38	122
Minneapolis	45	93	Berkeley	38	122
Berkeley	38	122	Cambridge	42	71
Cambridge	42	71	Cambridge	42	71
Minneapolis	45	93	Cambridge	42	71
Berkeley	38	122	Minneapolis	45	93
Cambridge	42	71	Minneapolis	45	93
Minneapolis	45	93	Minneapolis	45	93

Demo

Problem with Joining

The main issue with this type of joining is that a lot of the rows that we created are not useful. If i'm trying to join the Championships and Competitions table, I will end up with a lot of unrelated rows

(demo)

This means that almost always with joins we will need a WHERE to filter out these junk rows

Column Names when Joining

Sometimes we will get lucky and have no repeating column names, but most of the time we won’t be so lucky

If there is a `id` column in each table, in the WHERE or SELECT , how do I know which one I am accessing?

In SQL, we can alias *tables*, similar to how we would rename columns, so we can refer to a column from a specific table in our statements and clauses

Demo

Aliasing

In this example, I am joining a table with itself so I will have overlapping column names. Aliasing the tables lets me specify which columns I want to filter and which columns I want in my returned table

Summary

- SQL is a declarative language that deals with tables, a structure with columns and rows
- We can create SQL tables, but we can also query from existing tables to find useful information
- SELECT statements are our best friends in both of these tasks and have a bunch of optional clauses to make them more powerful