Lecture 23: SQL

August 1, 2022

Richard Roggenkemper

Announcements

- Scheme Project
  - Checkpoint 1 due Tuesday
  - Checkpoint 2 due Friday
  - Project due 8/9
  - My recommendation - get the project done by the weekend
- Scheme Art Contest
- Less than 2 weeks until the final
  - Only a couple more assignments!

Review

Regular Expressions

Regular Expressions let us go further into the idea of declarative programming

Even more than Scheme, we told the computer what we wanted, not how we wanted it, and it figured the problem out

Specifically, given a mix of characters, groups, quantifiers, or anchors, figuring out if a given string matched

Regex Examples

Phone Numbers - r”\(\d{3}\) \d{3}-\d{4}”

- Matches
  - (408) 593-7076
- Does not Match
  - (8000) 888-8888
  - (510) 367 - 1000

Regex is case/white space sensitive

Because it is declarative, we need to be careful about our expression. Think about the difference between (haha)+ vs. haha+

SQL

Declarative Programming

Today, we will dive deeper into declarative programming

In Scheme, we talked about functional programming. Instead of having if statements, we had the if procedure, which was more declarative. We no longer had control over the logic, we just called a function

In Regex, 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

Today, we will see a similar idea, but with a very different type of data

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

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

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

DBMSes are really important, heavily used, and pretty interesting!

Popular DBMSes: MySQL, Oracle, PostgreSQL, MongoDB, Redis, SAP HANA

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 work at a Toy Store , I can have one table to keep track of the Product ID and the toy name while the other table keeps track of Product ID and price

The alternative to Relational Databases are Non-Relational Databases which do not use columns and rows (an example is MongoDB)

Tables

A **table** is a collections of rows, which have a value for each column

A **column** has a name and a type

A **row** has a value for each column

City	State	Zip Code	Population
Los Gatos	CA	95032	31943
Anaheim	CA	92802	353085
Orlando	FL	32836	284817

Demo

How do I make that in SQL?

```
CREATE TABLE cities AS
 SELECT “Los Gatos” AS city, “CA” AS state, “95032” AS zipcode, 31943 as pop UNION
 SELECT “Anaheim”          , “CA”         , “92802”           , 353085      UNION
 SELECT “Orlando”          , “FL”         , “32836”           , 284817;
```

city	state	zipcode	pop
Los Gatos	CA	95032	31943
Anaheim	CA	92802	353085
Orlando	FL	32836	284817

Creating a Table

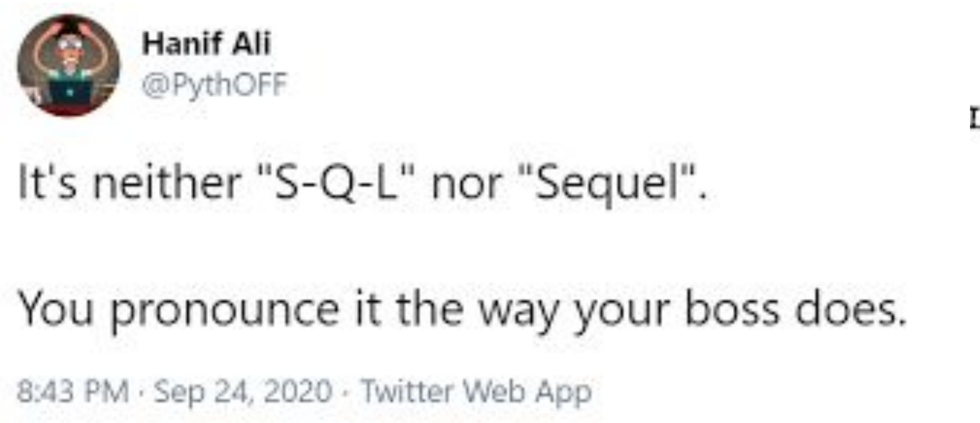
SELECT statements create a new table, either by creating one from scratch or taking from an existing table

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

Wait… is it pronounced S Q L or Sequel?


Hanif Ali
@PythOFF

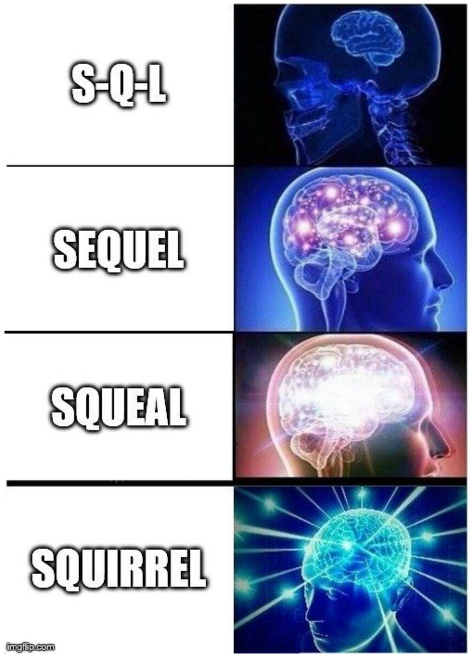
It's neither "S-Q-L" nor "Sequel".

You pronounce it the way your boss does.

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

LANGUAGE

IBM Research Laboratory
San Jose, California



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

Examples of Select

Return all rows of the tables

```
SELECT * FROM songs;
```

Return the artist and title columns from all rows

```
SELECT artist, title from songs;
```

Rename columns in the returned table

```
select artist as signer,  title as song_title FROM songs;
```

Manipulate the column value

```
select title, views * 1000000 from songs;
```

How case sensitive is SQL?

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

Break

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 that let you do a bunch of new things

Order By

ORDER BY will let you specify in what order the rows are in the returned table

Return the rows sorted by the number of views (highest to lowest)

```
SELECT title FROM songs ORDER BY views DESC;
```

Return the rows sorted by the number of views (lowest to highest)

```
SELECT title FROM songs ORDER BY views ASC;
```

Return the rows sorted by the number of views and release_year

```
SELECT title FROM songs ORDER BY views DESC, release_year DESC;
```

Where

The WHERE clause will allow you to filter rows, and it must appear before the ORDER BY

Return all Lil Nas X songs

```
SELECT title FROM songs WHERE artist = “Lil Nas X”;
```

Return all songs released after 2020

```
SELECT title FROM songs WHERE release_year > 2020;
```

Return all songs released after 2020 with 5 or more views

```
SELECT title FROM songs WHERE release_year > 2020 AND views > 5;
```

Demo

Limit

The LIMIT clause limits the number of rows that are output. Typically used with ORDER BY and it comes last in the SELECT statement

Return the Lil Nas X song with the most views

```
SELECT title FROM songs WHERE artist = “Lil Nas X” ORDER BY views
DESC LIMIT 1;
```

Return the 3 Lil Nas X song with the fewest views

```
SELECT title FROM songs WHERE artist = “Lil Nas X” ORDER BY views
ASC LIMIT 3;
```

Combining Related Tables

We briefly discussed that SQL is a relational database, but now let’s see how tables can relate to each other

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

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

Joining Tables

The way we join tables in this class 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

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 toy names and toy prices table, I will end up with unrelated rows

The Razorcrest Lego Set with ID #1776 with be joined with ID # 1999 and ID # 2022, but I only want it to be joined with the other row with ID #1776

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 Product 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

Aliasing

```
SELECT r1.salary, r2.salary FROM records as r1, records as r2 WHERE
r1.supervisor = r2.name ;
```

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