

Lecture 24

SQL 2: Electric Boogaloo

August 2nd, 2022

Cooper Bedin

Announcements

- Lab 10 due today
- HW 6 due Thursday
- First Scheme checkpoint due today
 - Second checkpoint due Friday
 - Project due a week from today
- Scheme contest due Friday !
- HW recovery tomorrow for HW 5
- This is the **last lecture** in the course that will contribute to any assignment content!
 - After this we have three special topics lectures—these are in scope for the final, but won't introduce any new coding or problem-solving techniques. Any final exam question corresponding to the special topics should be answerable so long as you watched the lecture

Review

SQL

SQL (**S**tructured **Q**uery **L**anguage) is a declarative programming language we can use to "query" information from databases

SQL assumes data is organized in rows and columns. Each row corresponds to a unique data point, and each column corresponds to a features of each data point

We could have a database on any kind of information that we want!



Databases

In this lecture, we'll primarily work with a database containing two tables:

`family`, which contains four columns:

- `first_name`
- `last_name`
- `parent_one`, the name of one of the parents
- `parent_two`, the name of the other parent

`net_worth`, which contains two columns:

- `name`
- `net_worth` (in millions)

These databases are loaded into code.cs61a.org if you want to follow along!

Querying a database

```
sql> SELECT * FROM family LIMIT 3;
```

```
Chicago|West|Kim|Kanye
```

```
Dream|Kardashian|Robert|Blac
```

```
Kendall|Jenner|Kris|Caitlyn
```

```
sql> SELECT * FROM net_worth LIMIT 3;
```

```
Caitlyn|100
```

```
Kanye|6600
```

```
Kendall|18
```

```
sql> SELECT * FROM net_worth ORDER BY net_worth DESC LIMIT 3;
```

```
Kanye|6600
```

```
Kim|1000
```

```
Kylie|700
```

```
sql> SELECT * FROM net_worth WHERE net_worth > 100;
```

```
Kanye|6600
```

```
Kim|1000
```

```
Kris|190
```

```
Kylie|700
```

Joins

```
sql> SELECT first_name, last_name, net_worth FROM family, net_worth  
WHERE first_name = name;
```

Kendall|Jenner|18

Khloe|Kardashian|40

Kim|Kardashian|1000

Kourtney|Kardashian|35

Kylie|Jenner|700

Rob|Kardashian|10

```
sql> SELECT a.first_name, b.first_name FROM family AS a, family AS b  
WHERE a.last_name = b.last_name AND a.first_name < b.first_name;
```

Chicago|North

Chicago|Psalm

Chicago|Saint

Dream|Khloe

Dream|Kim

...

Numerical and String Expressions

Expressions

We don't have to SELECT information directly from columns in SQL—we can also use operations to transform that data

For example, we can do string concatenation on information in our database, using the || operator

```
SELECT first_name AS name,  
       (parent_one || " and " || parent_two) AS parents  
FROM family LIMIT 5;
```

name	parents
Chicago	Kim and Kanye
Dream	Robert and Blac
Kendall	Kris and Caitlyn
Khloe	Kris and Robert
Kim	Kris and Robert

Expressions

We can also use function calls and arithmetic operators to create **numeric expressions** that do operations on our data

- Combining values: +, -, *, /, %, and, or,
- Transforming values: ABS(), ROUND(), NOT, -
- <, <=, >, >=, <>, !=, =

SQL also has the ability to do some operations on strings

- Concatenating strings: || (we just saw this one)
- Selecting substrings: SUBSTR()

SELECT

```
SUBSTR(first_name, 1, 1) || SUBSTR(last_name, 1, 1)  
AS initials FROM family;
```

Exercise!

Create a table with two columns:

- The name of every Kardashian in the family table
- The combined net_worth of that Kardashian's two parents
- Sort in descending order by that net worth

My solution:

SELECT

```
first_name, a.net_worth + b.net_worth AS net_worth
FROM family, net_worth AS a, net_worth AS b
WHERE parent_one = a.name AND parent_two = b.name
ORDER BY net_worth DESC;
```

Aggregation

Aggregate functions

So far, all the functions we've been able to write have only operated on single rows at a time

If we wanted to compare multiple rows across a table, we could use joins, but even then we could only really compare as many rows as joins we were willing to do

However, SQL has **aggregation functions** that allow us to compare across all the rows in a table

```
SELECT MAX(net_worth) FROM net_worth;
```

max(net_worth)
6600

Mixing aggregation and single values

If we include a non-aggregated column in an aggregation query, SQL will still fill that column with a value

In the case of MAX and MIN, that column will be filled based on the row in which the maximum or minimum value lives

For other aggregation functions (SUM, AVG, etc.) a value will be picked arbitrarily

```
SELECT name, MAX(net_worth) FROM net_worth;
```

```
SELECT name, MIN(net_worth) FROM net_worth;
```

```
SELECT name, AVG(net_worth) FROM net_worth;
```

name	max(net_worth)
Kanye	6600

name	min(net_worth)
Rob	10

name	avg(net_worth)
Caitlyn	680.576923076923

Aggregating over expressions

You can also aggregate over expressions:

```
SELECT a.name, b.name, MAX(a.net_worth - b.net_worth)
FROM net_worth AS a, net_worth AS b;
```

What do we think this query does?

a.name	b.name	max(a.net_worth - b.net_worth)
Kanye	Rob	6590

Break



Grouping

Making a helper table

What does this SQL statement do?

```
CREATE TABLE my_net_worth AS  
  SELECT first_name, last_name, net_worth  
  FROM family, net_worth  
  WHERE first_name = name;
```

first_name	last_name	net_worth
Kendall	Jenner	18
Kylie	Jenner	700
Khloe	Kardashian	40
Kim	Kardashian	1000
Kourtney	Kardashian	35
Rob	Kardashian	10

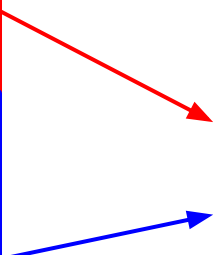
Grouping

We can divide our table into **groups**, and then aggregate within those groups, instead of aggregating across our entire table

We do this by using the GROUP BY clause:

```
SELECT first_name, MAX(net_worth) FROM my_net_worth  
GROUP BY last_name;
```

first_name	last_name	net_worth
Kendall	Jenner	18
Kylie	Jenner	700
Khloe	Kardashian	40
Kim	Kardashian	1000
Kourtney	Kardashian	35
Rob	Kardashian	10



first_name	max(net_worth)
Kylie	700
Kim	1000

Aggregation functions and grouping

These are the main aggregation functions you need to know for this class (and life in general)

- **MAX**(<col>) - Finds the maximum value of <col>, within a group
- **MIN**(<col>) - Finds the minimum value of <col>, within a group
- **SUM**(<col>) - Adds together all the values in <col>, within a group
- **AVG**(<col>) - Finds the average of all the values in <col>, within a group
- **COUNT**(*) - Counts the number of elements in a group

first_name	max(net_worth)	first_name	min(net_worth)	first_name	sum(net_worth)
Kylie	700	Kendall	18	Kendall	710
Kim	1000	Rob	10	Khloe	1085

first_name	avg(net_worth)	first_name	count(*)
Kylie	359.0	Kendall	2
Kim	271.25	Khloe	5

Exercise!

Create a table with two columns:

- The name of each distinct parent in the parent_one column of the family table
- The number of children that parent has

My solution:

```
SELECT parent_one, COUNT(*) FROM family GROUP BY parent_one;
```

parent_one	count(*)
Khloe	1
Kim	4
Kourtney	3
Kris	6
Kylie	1
Rob	1

Grouping by multiple columns

We don't have to just group by single columns—we can also group by multiple columns

```
SELECT parent_one, last_name, COUNT(*) FROM family  
GROUP BY parent_one, last_name;
```

parent_one	last_name	count(*)
Khloe	Thompson	1
Kim	West	4
Kourtney	Disick	3
Kris	Jenner	2
Kris	Kardashian	4
Kylie	Webster	1
Rob	Kardashian	1

Grouping by expressions

We can also group by expressions!

SELECT

```
SUBSTR(first_name, 1, 1) || SUBSTR(last_name, 1, 1),  
COUNT(*) FROM family GROUP BY  
SUBSTR(first_name, 1, 1) || SUBSTR(last_name, 1, 1);
```

CW	1
DK	1
KJ	2
KK	3
MD	1
NW	1
PD	1
...	...

Filtering groups

We can also filter groups based on criteria using the HAVING clause

```
SELECT parent_one COUNT(*) FROM family GROUP BY parent_one  
HAVING COUNT(*) > 1;
```

parent_one	count(*)
Kim	4
Kourtney	3
Kris	6

HAVING is similar to WHERE, but specifically for filtering by aggregate functions. If you're using an aggregate function in your filter clause, you should use HAVING. Otherwise, you should use WHERE

You can also use both together!

Database connections

Creating, dropping, and modifying tables

```
CREATE TABLE [table]([column-defs]);
```

```
DROP TABLE [table];
```

```
INSERT INTO [table] VALUES ([exprs]);
```

```
UPDATE [table] SET [column-name] = [expr] WHERE  
[expr];
```

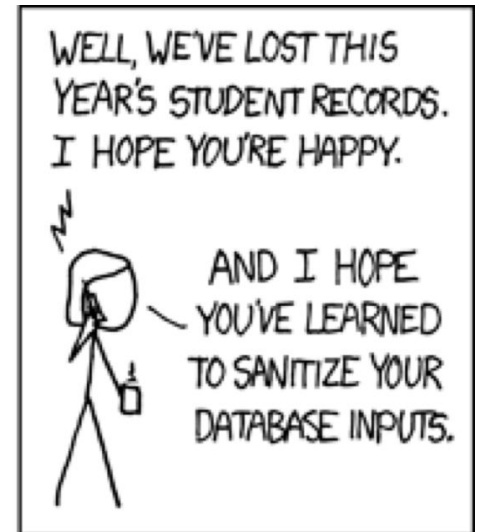
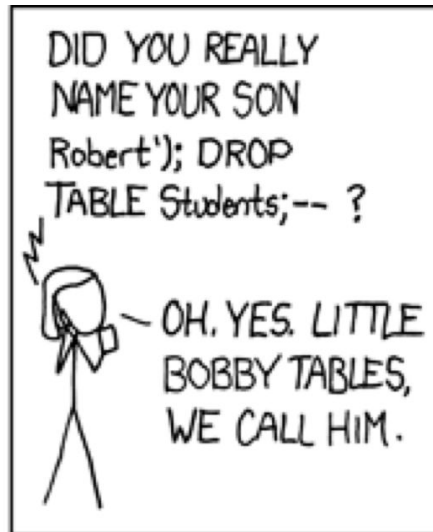
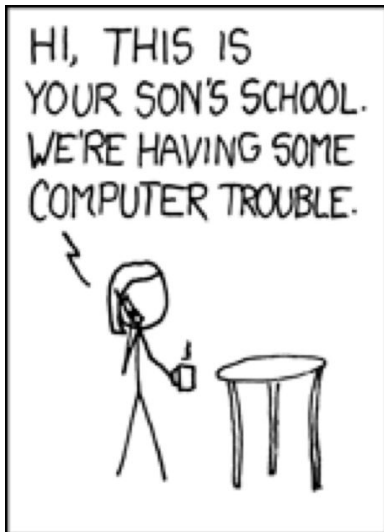
```
DELETE FROM [table] WHERE [expr];
```

None of these functions are in-scope for assignments or exams in this class, but the ability to modify tables is crucial if you're ever doing a real database project

Python and SQL

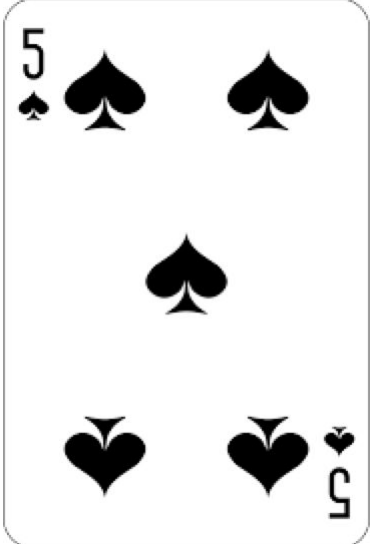
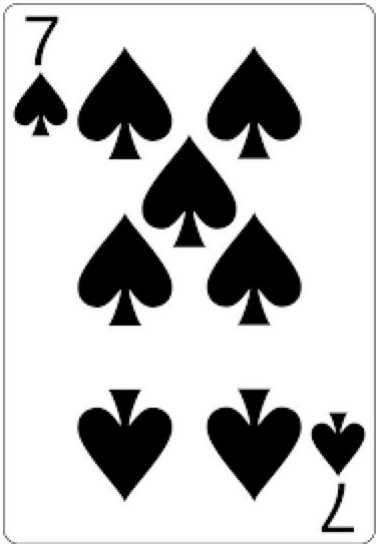
Python has a module called `sqlite3` that lets us interact with SQL databases! (truly modules for everything)

```
db = sqlite3.Connection('cards.db')
sql = db.execute
sql('DROP TABLE IF EXISTS cards;')
sql('CREATE TABLE cards(card, place);')
def play(card, place):
    sql('INSERT INTO cards VALUES (?, ?)', (card, place))
    db.commit()
```



Casino Blackjack

Player



Demo

Dealer

