Discussion 11:

SQL

Caroline Lemieux (clemieux@berkeley.edu)

April 25th, 2019

Review:
SQL Basics

SQL (*declarative*): “give me all the children who have the same fur as their parents”

```
SELECT c.name
FROM parents, dogs as p, dogs as c
WHERE p.name = parent and
        c.name = child and
        p.fur = c.fur;
```

Python (*imperative*): “store child-parent relations and fur info as dictionaries. Go through each of the keys in fur, see if child/parent match, …”

```
parents = {“Delano”: “Filmore”, ...}
fur = {“Delano”: “curly”,
       “Filmore”: “smooth”, ...}

names = []
for key in fur:
    child_fur = fur[key]
    parent_fur = fur[parents[key]]
    if child_fur == parent_fur:
        names.append(key)
```

Declarative Programming

- SQL is a *declarative language*.
  - Your code describes what you want the final result to look like
  - SQL takes care of how it works underneath; does optimizations that you don't have to understand or deal with
  - Cleaner, more readable, and faster if you're just trying to store/manipulate large amounts of data

Syntax: Creating a table

```
CREATE TABLE records AS
    SELECT “Ben” AS first, “Bitdiddle” AS last, 12 AS age UNION
    SELECT “Louis”,        “Reasoner”,          25 UNION
    SELECT “Oliver”,       “Warbucks”,          19;
```

- Column names are optional after first row (can’t change them)
- Our convention: all-caps for keywords (but lowercase works too)
- Indentations and line breaks don’t matter
- Don’t forget the semicolon at the end!

Syntax: Creating a table

You'll mostly operate on pre-existing tables

```
CREATE TABLE records AS
    SELECT "Ben" AS first, "Bitdiddle" AS last, 12 AS age UNION
    SELECT "Louis",        "Reasoner",          25 UNION
    SELECT "Oliver",       "Warbucks",          19;
```

- Column names are optional after first row (can't change them)
- Our convention: all caps for keywords (but lowercase works too)
- Indentations and line breaks don't matter
- Don't forget the semicolon at the end!

Syntax: Querying

```
SELECT first, age FROM records WHERE age > 15 ORDER BY age DESC;
```

```
Louis|25
Oliver|19
```

- Only `SELECT` and `FROM` are actually required -- rest are optional!
- Default order is ascending
  - Can do `ORDER BY age DESC` or `ORDER BY -age` for descending order

Try 2.1-2.3!

records

Name	Division	Title	Salary	Supervisor
Ben Bitdiddle	Computer	Wizard	60000	Oliver Warbucks
Alyssa P Hacker	Computer	Programmer	40000	Ben Bitdiddle
Cy D Fect	Computer	Programmer	35000	Ben Bitdiddle
Lem E Tweakit	Computer	Technician	25000	Ben Bitdiddle
Louis Reasoner	Computer	Programmer Trainee	30000	Alyssa P Hacker
Oliver Warbucks	Administration	Big Wheel	150000	Oliver Warbucks
Eben Scrooge	Accounting	Chief Accountant	75000	Oliver Warbucks
Robert Cratchet	Accounting	Scrivener	18000	Eben Scrooge

Joins/Combination

Disclaimer

There are many different types of joins in SQL!

In 61A, we look at what we can do by taking a *(cartesian) product* + filtering

Disclaimer

There are many different types of joins in SQL!

In 61A, we look at what we can do by taking a *(cartesian) product* + filtering

If you’re curious about the other things, ask me after :)

Joining: Intuition

Table **color**

fruit	color
apple	red
banana	yellow
watermelon	green

Table **radiation**

fruit	rads
apple	0
banana	3250

We want a table with the color + radiation of each fruit.

Joining: Intuition

Table **color**

fruit	color
apple	red
banana	yellow
watermelon	green

Table **radiation**

fruit	rads
apple	0
banana	3250

We want a table with the color + radiation of each fruit.

fruit	color	fruit	rads
apple	red	apple	0
banana	yellow	banana	3250

Joining: Intuition

Table **color**

fruit	color
apple	red
banana	yellow
watermelon	green

Table **radiation**

fruit	rads
apple	0
banana	3250

What rows should we pair up to get this?

fruit	color	fruit	rads
apple	red	apple	0
banana	yellow	banana	3250

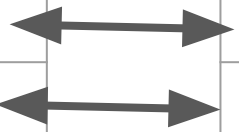
Joining: Intuition

Table **color**

fruit	color
apple	red
banana	yellow
watermelon	green

Table **radiation**

fruit	rads
apple	0
banana	3250



What rows should we pair up to get this?

fruit	color	fruit	rads
apple	red	apple	0
banana	yellow	banana	3250

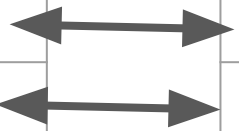
Joining: Intuition

Table **color**

fruit	color
apple	red
banana	yellow
watermelon	green

Table **radiation**

fruit	rads
apple	0
banana	3250



What is the relationship between these matched rows?

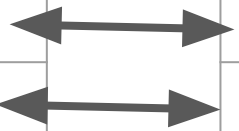
Joining: Intuition

Table **color**

fruit	color
apple	red
banana	yellow
watermelon	green

Table **radiation**

fruit	rads
apple	0
banana	3250



What is the relationship between these matched rows?

```
color.fruit = radiation.fruit
```

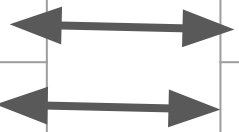
Joining: With SELECT ... FROM ...

Table **color**

fruit	color
apple	red
banana	yellow
watermelon	green

Table **radiation**

fruit	rads
apple	0
banana	3250



What is the relationship between these matched rows?

```
color.fruit = radiation.fruit
```

How do we get only those rows in SQL?

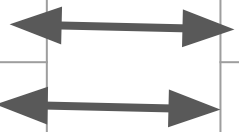
Joining: With SELECT ... FROM ...

Table **color**

fruit	color
apple	red
banana	yellow
watermelon	green

Table **radiation**

fruit	rads
apple	0
banana	3250



What is the relationship between these matched rows?

```
color.fruit = radiation.fruit
```

How do we get only those rows in SQL?

```
SELECT * FROM color, radiation WHERE color.fruit = radiation.fruit
```

What's happening?

```
SELECT * FROM color, radiation
```

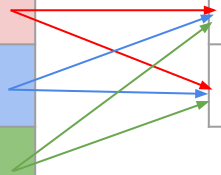
→ *takes the product of the two tables*

Table **color**

fruit	color
apple	red
banana	yellow
watermelon	green

Table **radiation**

fruit	rads
apple	0
banana	3250



What's happening?

```
SELECT * FROM color, radiation
```

→ *takes the product of the two tables*

fruit	color	fruit	rads
apple	red	apple	0
apple	red	banana	3250
banana	yellow	apple	0
banana	yellow	banana	3250
watermelon	green	apple	0
watermelon	green	banana	3250

What's happening?

```
SELECT * FROM color, radiation
```

→ *takes the product of the two tables*

We only want these!

fruit	color	fruit	rads
apple	red	apple	0
apple	red	banana	3250
banana	yellow	apple	0
banana	yellow	banana	3250
watermelon	green	apple	0
watermelon	green	banana	3250

What's happening?

```
SELECT * FROM color, radiation WHERE color.fruit = radiation.fruit
```

→ *takes the product of the two tables + filters*

fruit	color	fruit	rads
apple	red	apple	0
banana	yellow	banana	3250

What's happening?

This part of the WHERE is your "join condition"

```
SELECT * FROM color, radiation WHERE color.fruit = radiation.fruit
```

→ *takes the product of the two tables + filters*

fruit	color	fruit	rads
apple	red	apple	0
banana	yellow	banana	3250

Solving SQL Problems

1) Note down what information you want (read the problem)
2) Figure out which tables give you that information
    a) Write **FROM** clause (*joining*)
3) Find one (or more) columns from each table that link them together
    a) Write **WHERE** clause (*filtering*)
4) Write the **SELECT** clause (using step 1 as reference)
    a) Optionally add some ordering or limit # of rows

Try 3.1-3.4!

records

Name	Division	Title	Salary	Supervisor
Ben Bitdiddle	Computer	Wizard	60000	Oliver Warbucks
Alyssa P Hacker	Computer	Programmer	40000	Ben Bitdiddle
Cy D Fect	Computer	Programmer	35000	Ben Bitdiddle
Lem E Tweakit	Computer	Technician	25000	Ben Bitdiddle
Louis Reasoner	Computer	Programmer Trainee	30000	Alyssa P Hacker
Oliver Warbucks	Administration	Big Wheel	150000	Oliver Warbucks
Eben Scrooge	Accounting	Chief Accountant	75000	Oliver Warbucks
Robert Cratchet	Accounting	Scrivener	18000	Eben Scrooge

meetings

Division	Day	Time
Accounting	Monday	9am
Computer	Wednesday	4pm
Administration	Monday	11am
Administration	Wednesday	4pm

Attendance

links.cs61a.org/caro-disc

Aggregation

Aggregator Operations

- Does a calculation on all your rows to produce a single result
  - MAX, MIN, COUNT, SUM
- **Important:** squashes all your rows into a *single* row!
  - Only column values which you can access: what you calculate over (+ with a GROUP BY, what you aggregate over)

Example

SELECT SUM(salary) FROM records

name	division	salary
Alyssa P Hacker	Programming	75,000
Cy D Fect	Programming	53,600
John Denero	Programming	500,000
Eben Scrooge	Accounting	24,000
Robert Cratchet	Accounting	78,000
Oliver Warbucks	Administration	5,000,000

Example

SELECT SUM(salary) FROM records

salary
5,730,600

Group By

- Separates your table into several temporary “mini-tables”, each of which share the same column value
  - Ex. `GROUP BY division` - one group for all rows where division = “Programmer”, one where division = “Accountant”, etc.
- Aggregation happens over each individual group, not the whole table at once. More flexibility!
- `HAVING` lets you filter entire groups out, as opposed to rows (`WHERE`)

IMPORTANT

GROUP BY does not keep all the rows.

Each group will get squashed into a single row!!

When you GROUP BY, you are saying that you want to do things to **entire groups of rows at once**, and don't care that each group will get cut down to one row in the end.

Max salary in a division with more than one person!

name	division	salary
Alyssa P Hacker	Programming	75,000
Cy D Fect	Programming	53,600
John Denero	Programming	500,000
Eben Scrooge	Accounting	24,000
Robert Cratchet	Accounting	78,000
Oliver Warbucks	Administration	5,000,000

```
SELECT _____  FROM _____employees_____
GROUP BY _____  HAVING _____
```

name	division	salary
Alyssa P Hacker	Programming	75,000
Cy D Fect	Programming	53,600
John Denero	Programming	500,000
Eben Scrooge	Accounting	24,000
Robert Cratchet	Accounting	78,000
Oliver Warbucks	Administration	5,000,000

```
SELECT _____________________   FROM _____employees_____
GROUP BY ____division_____   HAVING _____________________
```

name	division	salary
Alyssa P Hacker	Programming	75,000
Cy D Fect	Programming	53,600
John Denero	Programming	500,000
Eben Scrooge	Accounting	24,000
Robert Cratchet	Accounting	78,000
Oliver Warbucks	Administration	5,000,000

```
SELECT _____________________   FROM _____employees_____
GROUP BY ____division_____   HAVING ___COUNT(*) > 1____
```

name	division	salary
Alyssa P Hacker	Programming	75,000
Cy D Fect	Programming	53,600
John Denero	Programming	500,000
Eben Scrooge	Accounting	24,000
Robert Cratchet	Accounting	78,000

```
SELECT division, MAX(salary)  FROM     employees
GROUP BY    division          HAVING   COUNT(*) > 1
```

name	division	salary
Alyssa P Hacker	Programming	75,000
Cy D Fect	Programming	53,600
John Denero	**Programming**	**500,000**
Eben Scrooge	Accounting	24,000
Robert Cratchet	**Accounting**	**78,000**

```
SELECT division, MAX(salary)  FROM  employees
GROUP BY  division  HAVING  COUNT(*) > 1
```

division	salary
Programming	500,000
Accounting	78,000

Query Order

- It can be helpful to think about SQL queries in this order:
  1) Join tables to create a big table (FROM clause)
  2) Filter the result (WHERE clause)
  3) Split filtered result into groups (GROUP BY clause)
  4) Filter the aggregated groups (HAVING clause)
  5) Write out the column values you want (SELECT clause)
  6) Rearrange rows to follow a certain order (ORDER BY clause)
  7) Throw away extra rows (LIMIT clause)

Try 4.1-4.3!

records

Name	Division	Title	Salary	Supervisor
Ben Bitdiddle	Computer	Wizard	60000	Oliver Warbucks
Alyssa P Hacker	Computer	Programmer	40000	Ben Bitdiddle
Cy D Fect	Computer	Programmer	35000	Ben Bitdiddle
Lem E Tweakit	Computer	Technician	25000	Ben Bitdiddle
Louis Reasoner	Computer	Programmer Trainee	30000	Alyssa P Hacker
Oliver Warbucks	Administration	Big Wheel	150000	Oliver Warbucks
Eben Scrooge	Accounting	Chief Accountant	75000	Oliver Warbucks
Robert Cratchet	Accounting	Scrivener	18000	Eben Scrooge

meetings

Division	Day	Time
Accounting	Monday	9am
Computer	Wednesday	4pm
Administration	Monday	11am
Administration	Wednesday	4pm

Modifying Tables

Creating Empty Tables

```
CREATE TABLE dogs(name, age, phrase DEFAULT “woof”);
```

- List out column names
  - Can have default values for future rows

Adding to a table

```
INSERT INTO dogs(name, age) VALUES (“Fido”, 1), (“Sparky”, 2);
```

- Specify which columns you will provide, and give values row-by-row as tuples

Adding to a table

Rules:

- If you don’t specify which columns, you must provide all values (even default ones)

```
INSERT INTO dogs VALUES (“Fido”, 1, “bark”), (“Sparky”, 2, “woof”);
```

- All tuples must be the same length - *default arguments don’t count!*

```
INSERT INTO dogs VALUES (“Fido”, 1, “bark”), (“Sparky”, 2); ERROR
```

Updating a table

```
UPDATE dogs SET age = age + 1 WHERE age < 5;
```

- WHERE clause is optional - if not provided, will update all rows

Deleting from a table

```
DELETE FROM dogs WHERE age < 5;
```

- WHERE clause is optional - if not provided, **will delete all rows**
  - However, empty table still exists

```
DROP TABLE dogs;
```

- Deletes table entirely

Try 5.1!

Bonus Questions?
Aggregation?