SOME SLIGHT ALTERATIONS

Jed Rembold

Wednesday, October 9, 2024

ANNOUNCEMENTS

- Homework 4 feedback is coming!
- Homework 5 due tomorrow
  - Homework 6 will be posted tomorrow
- In my grade system I'd missed bumping back the HW2 deadline by a day. So that has been remedied and will be reflected on the next grade report.

THE SHOWDOWN RETURNS

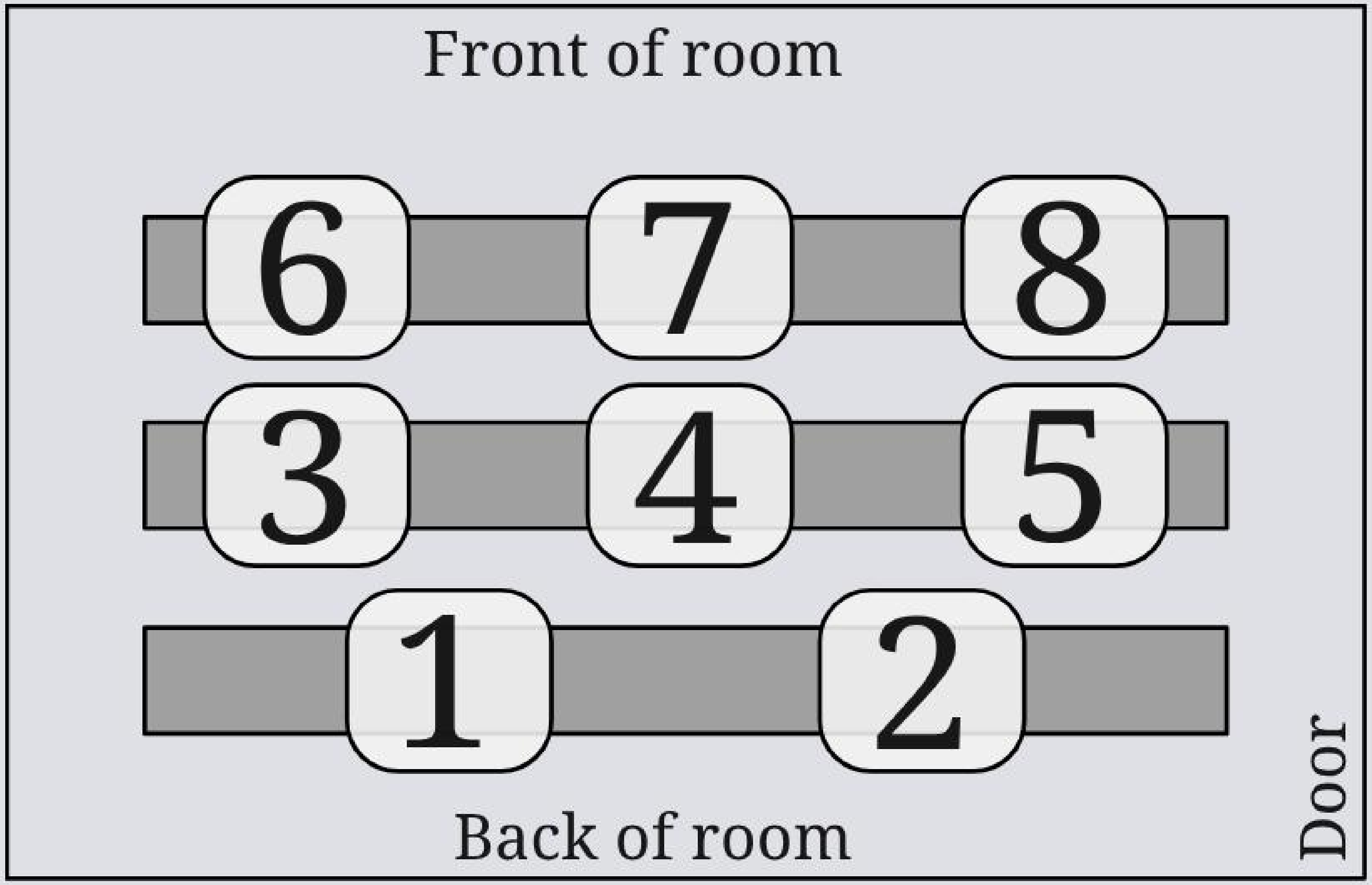
SHOWDOWN TIME!

- Divide into groups based on the next slide. It is time for another SQL showdown!
- Each group should have:
  - one person designated as the answer submitter
  - one person designated as the SQL typer (must rotate each question)
  - one computer that has the superheroes tables from HW4 on it
- Navigate to pollev.com/jedrembold441 and come up with a fun group name
- You'll have 3-5 minutes to answer each question. Submitting an answer faster gets you more points!

GROUPS

- Group 1:
  - Tippy, Marcus, Haley
- Group 2:
  - Aurora, Myles, Greg, Tiffany
- Group 3:
  - Jack, Dayton, Mallory
- Group 4:
  - Nick, Jordan, AJ
- Group 5:
  - Matthew, Sam H, Harleen
- Group 6:
  - Evan, Sergio, Finn
- Group 7:
  - Connor, Michael, Sam J
- Group 8:
  - Hannah, Jerrick, Grace

LOCATIONS!



CHAPTER 9

OVERVIEW

- Chapter 9 is essentially a tale in two parts, which are closely entwined
  - Data is likely going to be messy. How do you go about cleaning up data for further analysis. What sorts of anomolies should you keep an eye out for?
  - Tables are usually not static entities, but stores of data that might be changing. How you can make adjustments not only to the contents of a table but to the design of the table itself?

DATA CLEANING

CLEANING: CHECKING FOR DUPLICATES

- We've already seen in homework sample sets that there are sometimes duplicate row entries
- Duplicate entries across *many* columns is usually the sign
- `GROUP BY` and `HAVING` can be very useful here!
    - `GROUP BY` all the columns that you want to check for repetition
    - Use `HAVING` to only grab those groups that have more than one element (and thus a repetition)

```
SELECT col1, col2, col3, col4, COUNT(*)
FROM table_name
GROUP BY col1, col2, col3, col4
HAVING COUNT(*) > 1;
```

CLEANING: CHECKING FOR MISSING

- Data sets can sometimes (often even) be missing data that really should be present
- Fixing this generally requires some knowledge of what the data is representing
- Finding missing values can still tell you important things about the quality of your data though
- A nice way to count the number of nulls in different columns is:

```
SELECT
  COUNT(*) - COUNT(col1) as col1,
  COUNT(*) - COUNT(col2) as col2
FROM table_name;
```

CLEANING: INCONSISTENT DATA

- Especially for textual fields, there can be variation in how data is entered
  - Typos happen, or people just refer to the same thing in different ways
- To use `GROUP BY` effectively, you really need categories to be consistently named across the data set
- Several possible approaches to identify:
  - Scanning over a distinct column for quasi-duplications: phrases that are just a bit distinct
    - `GROUP BY` could do similar but add counts
  - Use pattern matching to look for duplicates matching a single approximate pattern
  - Use the fuzzystrmatch module to check Levershtein distances

CLEANING: CHECKING TEXT LENGTH

- For certain fields, you'd expect text of a certain number of characters
  - 2 for state abbreviations
  - 5 for zip codes
  - 13 for isbn13
- Can be a good idea to check these using the `LENGTH` string function
  - `LENGTH(str)` just returns the number of characters in said string

```
SELECT *
FROM table_name
WHERE LENGTH(col1) != 5;
```

DAMAGE CONTROL

- It can be a lot of work to clean up a table
- Evaluate whether it is worth it!
  - Maybe a better, cleaner data set exists?
- Sometimes, data will be missing that you simply can't fill in
  - Can your analysis work around those columns?

TABLE TWEAKING

MAKING ADJUSTMENTS

- Changing existing tables can generally be broken down into two categories:
  - Changing the structure of the table itself
    - Uses keywords `ALTER TABLE`
  - Changing the row content within the table
    - Uses keyword `UPDATE`

TABLE ALTERING: PART 1

- `ALTER TABLE` is generally followed by the table name and then another keyword command, depending on what you want to do

```
ALTER TABLE table_name ...
```

- Adding columns:

```
... ADD COLUMN col_name data_type;
```

- Removing columns

```
... DROP COLUMN col_name;
```

TABLE ALTERING: PART 2

```
ALTER TABLE table_name ...
```

- Changing columns

```
... ALTER COLUMN col_name SET DATA TYPE data_type;
... ALTER COLUMN col_name SET NOT NULL;
```

- Renaming columns

```
... RENAME col_name TO new_col_name;
```

- Rename entire table

```
... RENAME TO new_table_name;
```

UPDATING TABLES

- If you want to change the values in a particular row (or many rows), you don't want to alter the table, you want to `UPDATE` it
- `UPDATE` sets particular columns to a particular value
  - BE CAREFUL! If you do not specify *which* rows, then ALL of the rows will have that column changed to that value
    - This is partly why having primary keys is so nice: it gives you a method to update just a single row should you need

```
UPDATE table_name
SET col_name = new_value;
```

- You can specify which rows should be changed by filtering with `WHERE`

TABLE TO TABLE

- In some cases, you'll want to update information across tables
  - Maybe one table has newer values that you want to use to update the original table
- In core SQL, you'd need to use subqueries, which we'll be talking about in a few chapters
- In Postgres, you can use `FROM`:

```
UPDATE table_name
SET col_name = table_name2.col_name
FROM table_name2
WHERE table_name.col_name = table_name2.col_name
```