GROUP UP!

Jed Rembold

Monday, October 7, 2024

ANNOUNCEMENTS

- Homework 5 Thursday!
  - Working with constraints, so you have everything you need to complete it already
- Grade reports out!
  - Today is the last day to choose the credit/no credit option if that is something you were thinking about
- Polling today: polling.jedrembold.prof

REVIEW QUESTION

All of the following statements about SQL indexes are true except for which?

A) Indexes take additional processing each time a change is made to a column

B) Indexes are automatically created for any primary or foreign keys

C) Indexes can speed up the rate at which searches can be performed.

D) Indexes can be placed on any column.

DATA MODELING

ACTIVITY

Suppose you wanted to track your Spotify playlist information in your own database. Questions you may want to be able to answer might include:

- What is the total playtime of this playlist?
- What is the most common band in a particular playlist?
- Of all the artists who perform songs in any playlist, which artists collaborate with others the most?
- Songs from how many different albums are in a particular playlist?

In groups of two or three, sketch out ER diagrams that would allow all these questions to be answered. Your tables should include primary keys and foreign keys where appropriate, and be normalized as best as you are able.

Online sketching resources include: drawsql, visual-paradigm, or DrawIO

GROUPING

BIRDS OF A FEATHER

- It is frequently the case that values in a particular table column all belong to a smaller subset of categories or options
  - Think months of the year or political parties
- With current methods, if you want to compare some sort of aggregate between those categories or options, you need to do it in multiple queries:

```
SELECT AVG(age)
FROM voters
WHERE party = 'D'
```

```
SELECT AVG(age)
FROM voters
WHERE party = 'R'
```

- This rapidly becomes intractable if you want to compare across many categories

THE FIX

- SQL has one last (mostly) trick up its sleave with the `GROUP BY` command
- `GROUP BY` gathers together all rows with matching values from a particular column
  - By itself, this is basically the same as running `DISTINCT` on a column
  - Note that once you make a grouping, you can only select entire columns that are a part of the grouping

```
SELECT grouped_column
FROM table
GROUP BY grouped_column;
```

- Can visualize as if many smaller tables were created, one for each grouping, which are then joined back together with each table contributing one row

AGGREGATING GROUPS

- Groups by themselves are not that useful, since we already had `DISTINCT`
- The prime use-case of `GROUP BY` is to be able to run aggregates across *all* potential groups *simultaneously* for comparison

```
SELECT
  col_name,
  min(col_name),
  avg(col_name2)
FROM table
GROUP BY col_name;
```

- Causes any aggregate function to just aggregate over the smaller, group tables

GROUP EXAMPLE

Looking back at the cereal table, how can we answer:

- Which manufacturer sells the most sugar-laden cereal?
- What shelf has the most cereals placed on it? What about the greatest sum of calories?

MULTIPLE GROUPS

- Like with `DISTINCT` or `UNIQUE`, you can group by multiple columns
- Essentially splits the sub-tables into even smaller tables, one for each combination

```
SELECT
  col_name,
  min(col_name),
  avg(col_name2)
FROM table
GROUP BY col_name, col_name2;
```

COMBINING WITH JOINS

- `GROUP BY` can act on *whatever* our current table structure is, so can also work seamlessly on joined tables
- You may just need to be more attentive to your column identifiers

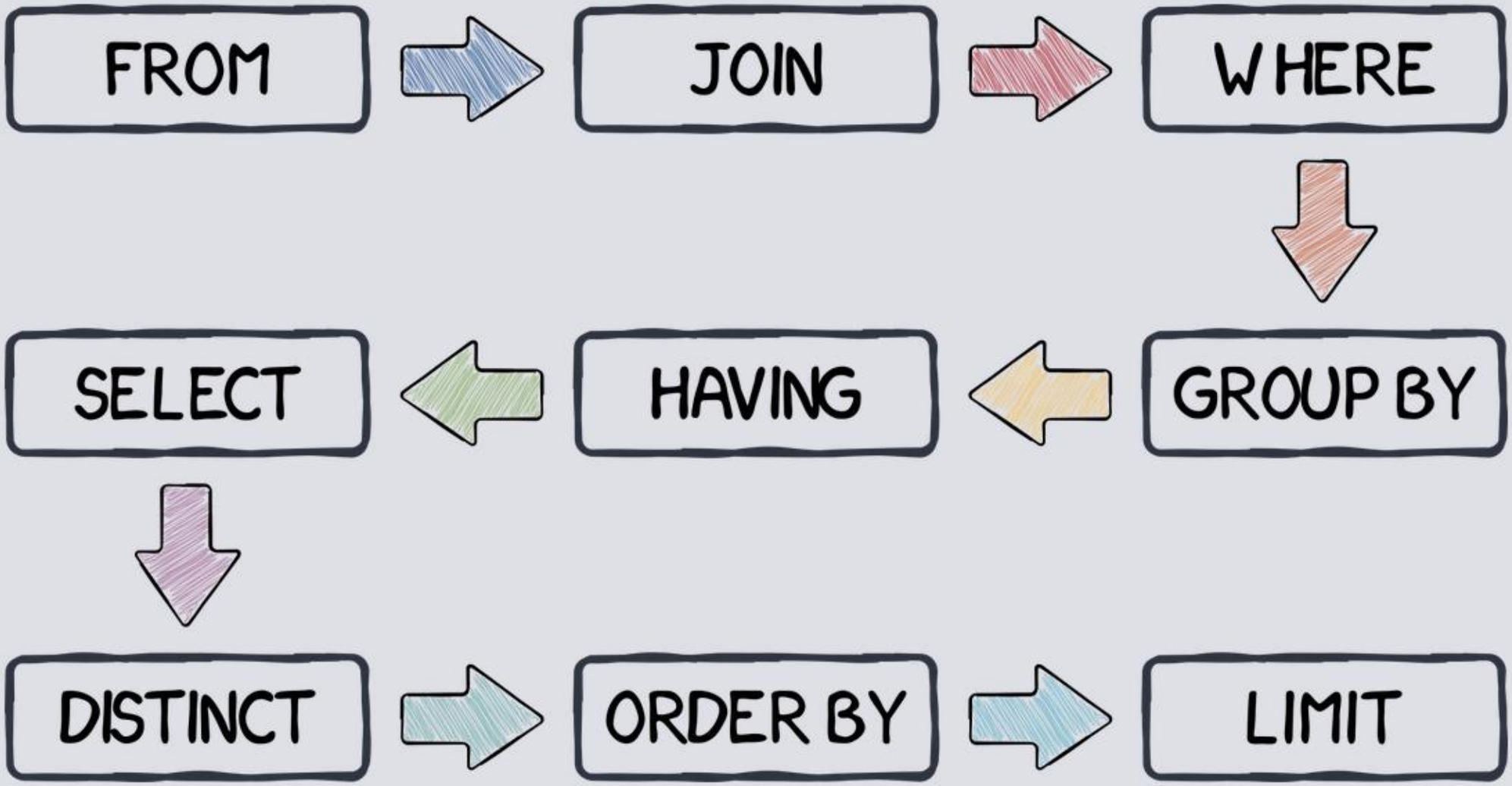
```
SELECT
  t1.col_name,
  COUNT(t2.col_name3),
FROM table_name1 AS t1
JOIN table_name2 AS t2
  ON t1.col_name = t2.col_name2
GROUP BY t1.col_name;
```

HAVE OR HAVE NOT

- You can not filter based on aggregates using `WHERE`
  - `WHERE` actions happen before any aggregates can be computed
- If you want to filter your grouped results, you need to use `HAVING`
  - `HAVING` filters take place after groups and aggregates can be computed

```
SELECT
  col_name,
  min(col_name),
  avg(col_name2)
FROM table
GROUP BY col_name
HAVING min(col_name) > 50;
```

SERVER ORDERING



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

LOCATIONS!