SQL Meta

COMPUTER SCIENCE MENTORS 61A

April 28 – May 2, 2025

Recommended Timeline

- Q1 (Circus): 20 minutes
- Q2 (Fish): 20 minutes
- Q3 (Outfits): 20 minutes

1 SQL

Last week we went over SQL and all the main keywords in SQL. This week we are going to go over more SQL problems.

SQl Clauses

Clause	Function
SELECT	Specifies the columns to retrieve or calculate or used in creation of a new row
UNION	Joins rows together
CREATE TABLE	Creates a new named table
FROM	Specifies the table from which to retrieve data
WHERE	Chooses the rows that meet the condition
GROUP BY	Groups rows that share a property
HAVING	Filters groups based on conditions, applied after grouping
ORDER BY	Sorts the result set by specified columns
LIMIT	Specifies the maximum number of rows to return

In SQL, the order of execution is as follows:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT

Created by Jim Fang, Bryan Adolfo Ventura Benitez, Rajoshi Basu Worksheet feedback: links.jimfang.dev/feedback

1. After more than 100 years of operation, the Ringling Bros. circus is closing. A victory for animal rights advocates, the circus' closure poses a challenge for the zoologists tasked with moving the circus' animals to more suitable habitats.

   The zoologists must first take the animals in a freight elevator with a weight limit of 2000. In order to speed up the process, the zoologists prefer to take groups of animals of the same species in the elevator, rather than one animal at a time.

   Assume the zoologists will only put all of the animals of a particular species in the elevator, or take animals of that particular species one at a time.

   You have access to the table `animals`, with columns containing the animals' names, heights, weights, and species.

Name	Height	Weight	Species
Wilbur	4.1	150	pig
Tigress	4.4	700	tiger
Phil	3.3	79	pig
Dug	3.5	40	dog
Buddy	4	51	dog
Marty	4.9	300	zebra
Richard Parker	5.2	918	tiger

   (a) Write a query that returns the collective weight and species of animals in a group where there is more than one animal of a particular species in a group, and the collective weight of the animals in the group is less than 2000.

   Your query should yield the following result.

```
91 dog
229 pig
1618 tiger
```

```
SELECT SUM(weight), species FROM animals
    GROUP BY species HAVING COUNT(*) > 1 and SUM(weight) < 2000;
```

(b) To take the animals to their new habitats, the zoologists load the animals into trucks. The zoologists would like to reduce the number of trips taken by pairing up the animals for the trip (species does not matter). However, the trucks have two restrictions:

1. The maximum height of any animal is 5.0.
2. The total weight of both animals cannot exceed 300.

Write a query which will yield all possible pairings in a table (which is basically the table below). To match the table below, make sure that the left column animals' names come earlier in the alphabet than their partners.

```
Buddy     Dug
Buddy     Phil
Buddy     Wilbur
Dug       Phil
Dug       Wilbur
Phil      Wilbur
```

```
SELECT a.name, b.name
    FROM animals AS a, animals AS b
    WHERE a.name < b.name AND a.weight + b.weight <= 300 AND a.height
       <= 5 AND b.height <= 5;
```

2. CS 61A wants to start a fish hatchery, and we need your help to analyze the data we've collected for the fish populations! Running a hatchery is expensive—we'd like to make some money on the side by selling some seafood (only older fish of course) to make delicious sushi.

   The table `fish` contains a subset of the data that has been collected. The SQL column names are listed in brackets.

`fish`

Species [species]	Population [pop]	Breeding Rate [rate]	$/piece [price]	# of pieces per fish [pieces]
Salmon	500	3.3	4	30
Eel	100	1.3	4	15
Yellowtail	700	2.0	3	30
Tuna	600	1.1	3	20

`competitor`

Species [species]	$/piece [price]
Salmon	2
Eel	3.4
Yellowtail	3.2
Tuna	2.6

   (a) Write a query to find the three most populated fish species.

```
SELECT species
FROM fish
ORDER BY pop DESC
LIMIT 3;
```

   (b) Write a query to find the total number of fish in the ocean. Additionally, include the number of species we summed. Your output should have the number of species and the total population and it should yield in a table with a single row.

```
SELECT COUNT(species), SUM(pop)
FROM fish;
```

   (c) Profit is good, but more profit is better. Write a query to select the species that yields the greatest number of pieces for each price point. For example, if "Trout" and "Catfish" both cost 100 dollars/-piece, you should output the species that has the greater number of pieces. Your output should include the species, price, and number of pieces per fish.

```
SELECT species, price, MAX(pieces)
FROM fish
GROUP BY price;
```

   Alternate solution

```
SELECT species, price, pieces
FROM fish
ORDER BY pieces / price DESC
LIMIT 2;
```

Show them that when you aggregate over a group and then select a column which you neither group over or aggregate over (in this case species), then the value on a particular row for that column which you selected is going to correspond to the value which is on the same row as the value of the aggregated result in the actual table. For instance, here in the group where price is 3 since max of pieces in that group is going to correspond to 30 thus yellowtail will be yielded in the species column with price 3 and pieces 30. Notice when we do this we cannot use aggregation functions like sum and etc.

(d) Write a query which is going to yield a table which has a row for each species, the species's piece in our hatchery, and then the species's price/piece in our competitors' hatchery.

```
SELECT fish.species, fish.price, competitor.price
FROM fish, competitor
WHERE fish.species = competitor.species;
```

(e) Write a query that returns, for each species, the difference between our hatchery's revenue versus the competitor's revenue for one whole fish. Assume that number of pieces per one whole fish of a certain species of fish stays the same across both our hatchery and our comptitor's hatchery. The revenue formula is price per unit * number of units.

```
SELECT fish.species, (fish.price - competitor.price) * pieces
    FROM fish, competitor
    WHERE fish.species = competitor.species;
```

Teaching Tips

- Make sure students know the basics of understanding/looking through a table
  - It may help to write a basic example using SELECT, FROM, and WHERE
- Have students clearly define which columns they will need from the table (species, price, pieces) before coding
- Remind students they are able to do arithmetic on cells they select
- While not in the solution, you can use this problem to explain the benefits of aliasing
  - Show students they can write `FROM fish as __, competitor as __`

3. In this question, you have access to two tables.

   Grades, which contains three columns: **`day`**, `class`, and `score`. Each row represents the `score` you got on a midterm for some `class` that you took on some **`day`**.

   Outfits, which contains two columns: **`day`** and `color`. Each row represents the `color` of the shirt you wore on some **`day`**. Assume you have a row for each possible day.

`grades`

Day	Class	Score
10/31	Music 70	88
9/20	Math 1A	72

`outfits`

Day	Color
11/5	Blue
9/13	Red
10/31	Orange

(a) Instead of actually studying for your finals, you decide it would be the best use of your time to determine what your "lucky shirt" is. Suppose you're pretty happy with your exam scores this semester, so you define your lucky shirt as the shirt you wore to the most exams.

Write a query that will output the color of your lucky shirt and how many times you wore it.

```
SELECT color, count(g.day) AS cnt
    FROM outfits AS o, grades AS g
    WHERE o.day = g.day
    GROUP BY color
    ORDER BY cnt DESC
    LIMIT 1;
```

Teaching Tips

- Ensure students know all potential parts of a SQL query and how they work:
  `SELECT [ ] FROM [ ] WHERE [ ] GROUP BY [ ] HAVING [ ] ORDER BY [ ] LIMIT [ ];`
- Remind students about the **SUM**, **MIN**, and **COUNT** functions
- Students may struggle with the variable to use in **ORDER BY**
  - Teach students they can alias created columns in **SELECT**
  - In SQL **SELECT** occurs before **ORDER BY**, so `cnt` can be used

(b) You want to find out which classes you need to prepare for the most by determining how many points you have so far. However, you only want to do so for classes where you did relatively poorly.

Write a query that will output the sum of your midterm scores for each class along with the corresponding class, but only for classes in which you scored less than 80 points on at least one midterm. List the output from highest to lowest total score.

```
SELECT SUM(score), class
    FROM grades GROUP BY class
    HAVING MIN(score) < 80 ORDER BY SUM(score) DESC;
```

Teaching Tips

- Remind students that the '**HAVING**' clause is used to filter out groups (you cannot use **WHERE** on the groups).
- Remind students that aggregation works on each group individually; this means we want to **GROUP BY** the things we want to **SUM** over.

(c) You want to find out how you did on the midterms to which you wore your blue shirt to. So write a query which is going to yield a table where there is a row for each day you wore a blue shirt to a midterm (or multiple midterms) and then sum of midterm scores for that day.

```
SELECT grades.day, sum(score)
FROM grades, outfits
WHERE outfits.color = "blue" and outfits.day = grades.day
GROUP BY grades.day;
```