CS 61:
Database Systems

Aggregation

Some advice about crafting SELECT commands

- Know your data
  - The importance of understanding the data model that you are working in cannot be overstated
  - Real-world databases are messy; many systems remain in service in an organization for decades

- Know the problem
  - Understand the question you are attempting to answer
  - Information reporting requests will come from a range of sources; may be one-time events or ongoing operations within an application

Adapted from Coronel and Morris

Some advice about crafting SELECT commands

Build query in this order

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

Where does the data come from? Could be one or more tables, could be subquery
Think of building one large relation

Filter out unwanted rows in relation from step 1

Which attributes do we want?

Create subgroups

Filter out unneeded subgroups

Finally, sort the results

Adapted from Coronel and Morris

Some advice about crafting SELECT commands

Build query in this order	Write SQL command in this order
1. FROM	SELECT columnlist
2. WHERE	FROM tablelist
3. SELECT	[WHERE conditionlist]
4. GROUP BY	[GROUP BY columnlist]
5. HAVING	[HAVING conditionlist]
6. ORDER BY	[ORDER BY columnlist [ASC | DESC]];

Adapted from Coronel and Morris

Agenda

1. Aggregate functions and NULL
2. Group by and having
3. Nested queries

Aggregate function provide a scalar value for an attribute

Aggregate functions

Use in the SELECT clause (e.g., SELECT MIN(score) AS MinScore FROM …)

- **AVG:** average value
- **MIN:** minimum value
- **MAX:** maximum value
- **SUM:** sum of values
- **COUNT:** number of values

AVG and SUM must be numeric attributes, others need not be numeric

Practice

use nyc_data;

- Find the min and max restaurant name
- What is the average score of all inspections scores?
- How many restaurants inspection scores were recorded?
- Try to answer the last two questions with one SELECT query

NULL means the value is missing or unknown; can cause unexpected problems

Theoretically, these two queries should be the same!

```
SELECT AVG(score) AS AvgScore
FROM restaurant_inspections; -- 20.41

SELECT SUM(score)/COUNT(*) AS AvgScore
FROM restaurant_inspections; -- 19.56
```

Practice:

First query returns 20.41, the second 19.56. Why are they different?

How can we make them the same?

Remember, NULL not considered in aggregate functions

NULL in an arithmetic operation is NULL (e.g., 5 + NULL = NULL)

Agenda

1. Aggregate functions and NULL
2. Group by and having
3. Nested queries

GROUP BY creates subgroups of tuples, you can perform aggregation over subgroups

```
SELECT ID, name, dept_name,
       FORMAT(salary,0) AS Salary
FROM instructor
ORDER BY dept_name;
```

Get avg salary by dept

```
SELECT dept_name,
       FORMAT(AVG(salary),0) AS AvgSalary
FROM instructor
GROUP BY dept_name
ORDER BY AvgSalary DESC;
```

ID	name	dept_name	Salary
76766	Crick	Biology	72,000
10101	Srinivasan	Comp. Sci.	65,000
45565	Katz	Comp. Sci.	75,000
83821	Brandt	Comp. Sci.	92,000
98345	Kim	Elec. Eng.	80,000
12121	Wu	Finance	90,000
76543	Singh	Finance	80,000
32343	El Said	History	60,000
58583	Califieri	History	62,000
15151	Mozart	Music	40,000
22222	Einstein	Physics	95,000
33456	Gold	Physics	87,000

dept_name	AvgSalary
Physics	91,000
Finance	85,000
Elec. Eng.	80,000
Comp. Sci.	77,333
Biology	72,000
History	61,000
Music	40,000

Selected attributes (e.g. dept_name and AvgSalary) must be in aggregate functions or *group by* list

Adding ID would cause query to fail!

- **Without grouping, AVG would return a single number for all departments**
- **Grouping allows aggregation of tuples with the same value for the GROUP BY attributes (e.g. dept_name)**

HAVING works with GROUP BY to filter subgroups

```
SELECT dept_name,
       FORMAT(AVG(salary),0) AS AvgSalary
FROM instructor
GROUP BY dept_name
ORDER BY AvgSalary DESC;
```

dept_name	AvgSalary
Physics	91,000
Finance	85,000
Elec. Eng.	80,000
Comp. Sci.	77,333
Biology	72,000
History	61,000
Music	40,000

```
SELECT dept_name,
       FORMAT(AVG(salary),0) AS AvgSalary
FROM instructor
GROUP BY dept_name
HAVING AVG(salary) > 65000
ORDER BY AvgSalary DESC;
```

dept_name	AvgSalary
Physics	91,000
Finance	85,000
Elec. Eng.	80,000
Comp. Sci.	77,333
Biology	72,000

- **HAVING works with GROUP BY to filter results similar to how WHERE works with SELECT**
- **Note: predicates in the HAVING clause are applied *after* the formation of groups whereas predicates in the WHERE clause are applied before forming groups**

SQL evaluation proceeds start with FROM and proceeds to LIMIT

FROM
Create a relation based on tables listed

More on this soon

SELECT
Get attributes listed

HAVING
Remove non-matching subgroups

LIMIT
Return top k items

WHERE
Remove tuples not matching criteria

GROUP BY
Aggregate tuples into subgroups

ORDER BY
Sort resulting tuples

https://www.mysqltutorial.org/mysql-group-by.aspx

Practice

use nyc_data

- In one query, find the average health inspection score and number of inspections by boro (e.g., Manhattan, Bronx, …)
- Which is better a low score or a high score? (Hint: consider the critical flag)
- In one query, find the average health inspection score and number of inspections by boro and by cuisine type. Sort by boro then by cuisine type
- For restaurants in Queens, find the average score and number of inspection scores where the restaurant has at least 5 inspection scores; sort by avg score, best first

Agenda

1. Aggregate functions and NULL
2. Group by and having
3. Nested queries

Nested queries have a subquery inside another query

Nested queries

Nesting can be done in the SELECT, FROM or WHERE clauses

SELECT $A_1, A_2, ..., A_n$
FROM $r_1, r_2, ..., r_m$
WHERE P

- **SELECT clause:**

  A_i can be replaced be a subquery that generates a single (scalar) value

- **FROM clause:** r_i can be replaced by any valid subquery because SELECT returns a relation

  More on this soon

- **WHERE clause:** P can be replaced with an expression of the form:

  A <operation> (subquery)

  A is an attribute and <operation> is <,>,IN, NOT IN, etc

Subqueries in the SELECT clause return a scalar value

Subquery in SELECT clause

SELECT
FROM
WHERE

- You can use a subquery in the SELECT clause in SQL
- Generally returns a scalar value (could be Null)

```
-- compare each restaurant score with this restaurant's max score
SELECT dba AS RestaurantID, Score,
    (SELECT MAX(Score)
        FROM restaurant_inspections r2
        WHERE r2.camis = r1.camis) AS MaxScore
FROM restaurant_inspections r1
WHERE r1.camis < 30080000;
```

Select RestaurantID and Score for each row in table

- **Find max score for this restaurant**
- **This is sometimes called a correlated subquery because camis from inner query using r2 is correlated with camis from outer query using r1**

Limit search to shorten query runtime
More on this when we get to query optimization

Subqueries can also be used in the WHERE clause

Subquery in WHERE clause

SELECT
FROM
WHERE

```
-- find scores for restaurant with min camis id
    SELECT camis AS RestaurantID, dba AS RestaurantName, Score
    FROM restaurant_inspections
    WHERE camis = (SELECT MIN(CAMIS)
                   FROM restaurant_inspections);

-- find inspections with max score from any inspection
    SELECT * FROM restaurant_inspections
    WHERE score = (SELECT MAX(Score)
                   FROM restaurant_inspections);
```

The WITH clause is also a subquery, but creates a queryable temporary relation

Subquery in WHERE clause

The **WITH** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs

```
WITH TempRelationName (ColumnName1, columnName2...) AS
     (SELECT ...)
SELECT ...
```

Practice

use nyc_data

1. For all each restaurant **not in Manhattan or Queens** return
   - RestaurantID, RestaurantName, Boro, and average score for that restaurant on one row
   - Sort the restaurants by average score descending
   - What is ironic about the name of the first restaurant returned?
2. Use a WITH clause to calculate a temporary relation with a column for the average score of all inspections, then use that temporary table to return all rows with a score greater than average
3. Do the same as 2, but without using a WITH clause