More SQL Aggregate Queries, Null Values

CS430/630
Lecture 8

Slides based on “Database Management Systems” 3rd ed, Ramakrishnan and Gehrke
### SQL Lab Query 5

For each boat color, list the number of reservations for those boats.

- **Analysis**: number of reservations: aggregate, number of rows of reserves, \( \text{count(*)} \)
- For each...number of... Need groups, GROUP BY
- Involves boats, reserves: need join
For each boat color, list the number reservations for those boats

select b.color, count(*) from reserves r, boats b
where r.bid = b.bid
group by b.color
For each boat color, list the number of reservations for those boats, but only if that number exceeds 2.

```
select b.color, count(*) from reserves r, boats b
where r.bid = b.bid
group by b.color
having count(*) > 2  -- qualification on group
```
SQL Lab Query 8

suppliers(sid, sname, address)
parts(pid, pname, color)
catalog(sid, pid, cost)

Find the number of parts each supplier provides. List sid, sname, and count of parts.

Analysis: …each supplier… , can be reworded to “For each supplier …”. So need GROUP BY.

Consider a supplier. The rows in catalog for that sid show all the parts that supplier provides, so count(*) for the group. We need to join to suppliers to pull in the sname.

select s.sid, s.sname, count(*) from suppliers s, catalog c
where s.sid = c.sid
group by s.sid, s.sname
SQL Lab Query 9

suppliers(sid, sname, address)
parts(pid, pname, color)
catalog(sid, pid, cost)

Find the name of the cheapest supplier of part 4.

Remember: to use min, need to use “select min(…)...” to run the loop to calculate the min.

Analysis: Consider part 4. The rows in catalog for that pid show all the suppliers that part has, and the cost. The min cost of these is

```
select min(c1.cost) from catalog c1
where c1.pid = 4
```

We just need to find the suppliers of part 4 with this specific cost, and we need to join to suppliers to pull in the sname.

```
select s.sname from suppliers s, catalog c
where s.sid = c.sid and c.pid = 4 and
  c.cost = (select min(c1.cost) from catalog c1
           where c1.pid = 4)
```
Aircraft Example 1

aircraft(aid, aname, cruisingrange)
employees(eid, ename, salary)
certified(eid, aid)

“Find the minimum salary of employees certified for some airplane with cruisingrange over 2000 miles”
Example 1

```
aircraft(aid, aname, cruisingrange)
employees(eid, ename, salary)
certified(eid, aid)

“Find the minimum salary of employees certified for some airplane with cruisingrange over 2000 miles”

SELECT MIN (e.salary)
FROM employees e, certified c, aircraft a
WHERE e.eid = c.eid AND c.aid = a.aid AND a.cruisingrange > 2000;
```
Aircraft Example 2

aircraft(aid, aname, cruisingrange)
employees(eid, ename, salary)
certified(eid, aid)

“Find the average salary of employees certified for some airplane with cruisingrange over 2000 miles”
Example 2

```
aircraft(aid, aname, cruisingrange)
employees(eid, ename, salary)
certified(eid, aid)

“Find the average salary of employees certified for some airplane with cruisingrange over 2000 miles”

SELECT AVG(e.salary)
FROM employees e, certified c, aircraft a
WHERE e.eid = c.eid AND c.aid = a.aid AND a.cruisingrange > 2000;

But this is the average over the certifications, and one employee can have multiple 2000+ certifications, so this is not a proper average over employees. We saw this problem before with ages of students enrolled in courses.
```
Aircraft Example 2, fixed

aircraft(aid, aname, cruisingrange)
employees(eid, ename, salary)
certified(eid, aid)

“Find the average salary of employees certified for some airplane with cruisingrange over 2000 miles”

SELECT avg (e.salary)
FROM employees e where e.eid in
  (select eid from certified c, aircraft a
   WHERE e.eid = c.eid AND c.aid = a.aid AND a.cruisingrange > 2000);
Aircraft Example 3

aircraft((aid, aname, cruisingrange))
employees((eid, ename, salary))
certified((eid, aid))

“For each aircraft name, find the minimum salary of employees certified for that airplane”
Aircraft Example 3

aircraft(aid, aname, cruisingrange)
employees(eid, ename, salary)
certified(eid, aid)

“For each aircraft name, find the minimum salary of employees certified for that airplane”

SELECT a.aname, MIN(e.salary)
FROM employees e, certified c, aircraft a
WHERE e.eid = c.eid AND c.aid = a.aid
GROUP BY a.aname
Aircraft Example 4

aircraft(aid, aname, cruisingrange)
employees(eid, ename, salary)
certified(eid, aid)

“For each aircraft aid, find the average salary of employees certified for that airplane”
Example 4

For each aircraft aid, find the average salary of employees certified for that airplane.

```
SELECT a.aid, AVG (e.salary)
FROM employees e, certified c, aircraft a
WHERE e.eid = c.eid AND c.aid = a.aid
GROUP BY a.aid
```

Here there is no problem with AVG: a particular aircraft is connected only once per employee through the certified relationship.
Aircraft Example 5

“For each aircraft aid, find the count of employees certified for that airplane”

SELECT a.aid, count(*)
FROM employees e, certified c, aircraft a
WHERE e.eid = c.eid AND c.aid = a.aid
GROUP BY a.aid
Aircraft Example 6

aircraft(aid, aname, cruisingrange)
employees(eid, ename, salary)
certified(eid, aid)

“Find aircraft by aid that have fewer than 3 employees certified for that airplane. In these cases, report the number of such employees.”
“Find aircraft by aid that have fewer than 3 employees certified for that airplane. In these cases, report the number of such employees.”

```
SELECT a.aid, count(*)
FROM employees e, certified c, aircraft a
WHERE e.eid = c.eid AND c.aid = a.aid
GROUP BY a.aid
HAVING count(*) < 3
```
Yelpdb Example 1

business(id, name, neighborhood, city, state, latitude, longitude, stars, …)
yuser(id, name, useful, funny, cool, …)
review(id, stars, review_date, text, useful, funny, cool, business_id, user_id)

“Find businesses by name in Las Vegas that have fewer than 3 stars”
### Yelpdb Example 1

```
```

**business**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>neighborhood</th>
<th>city</th>
<th>state</th>
<th>latitude</th>
<th>longitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**yuser**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>useful</th>
<th>funny</th>
<th>cool</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**review**

<table>
<thead>
<tr>
<th>id</th>
<th>stars</th>
<th>review_date</th>
<th>text</th>
<th>useful</th>
<th>funny</th>
<th>cool</th>
<th>business_id</th>
<th>user_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**SQL Query**

```
SELECT b.id, b.name
FROM business b
WHERE b.city = 'Las Vegas' and b.state = 'NV' and b.stars < 3
```

```
...  
| _WVPws8wHGOxPtGvKPr3vA | 3 Kings Hookah Lounge
| __3qOwWFBUE8mdOToI7YrQ | Custom Kings
| __IsqCZAF9YTcvKPKj2dZg | Z Gallerie
```

4425 rows in set (0.15 sec)
Yelpdb Example 2

Check on '3 Kings Hookah Lounge', id _WVPws8wHG0xPtGvKPr3vA
How did the reviews of this business break down by number of stars?

Note: "break down by": another way of saying "for each"
**Yelpdb Example 2**

```sql
business(id, name, neighborhood, city, state, latitude, longitude, stars, ...) 
yuser(id, name, useful, funny, cool, ...) 
review(id, stars, review_date, text, useful, funny, cool, business_id, user_id)

Check on '3 Kings Hookah Lounge', id _WVPws8wHGOxPtGvKPr3vA
How did the reviews of this business break down by number of stars?

```sql
select stars, count(stars) from review 
where business_id = '_WVPws8wHGOxPtGvKPr3vA' 
group by stars;

+-----------------+-------------------+
| stars | count(stars) |
+-----------------+-------------------+
| 1   | 7 | avg is 2.0, value in business.stars 
| 2   | 2 |
| 3   | 2 |
| 4   | 1 |
| 5   | 1 |
+-----------------+-------------------+
5 rows in set (0.00 sec)
Notes on business.stars

- We see that the stars value for a business is obtained by computing the average stars assigned by users in their reviews of that business.
- Thus business.stars is a derived value, not actually needed for full information in the database.
- The app could just compute this average from the review table data, but that would take a little time on each query.
- So for performance sake, the database keeps this average in the business table.
- That means that to add a review to the system, the app has to adjust the target business's stars value.
- This duplication of data means the database is not "fully normalized", a concept we will study later.
- All the databases in the book have no derived values like this case.
Histograms, i.e., counts by intervals of values

- Suppose we want to summarize the costs of parts more finely than just average, min and max.
- Cost values: 0.55, 7.95, 11.7, 15.3, 16.5, 20.5, 20.5, 22.2, 36.1, 42.3, 48.6, 57.3, 75.2, 124.23, 124.23
- We make can a histogram like this, with “buckets” of values:

<table>
<thead>
<tr>
<th>Cost</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-10</td>
<td>2</td>
</tr>
<tr>
<td>10-20</td>
<td>3</td>
</tr>
<tr>
<td>20-30</td>
<td>3</td>
</tr>
</tbody>
</table>

...  

- How can we get this data using SQL?
- Note: Bucket-number = floor(cost/10)
  - For example cost = 22 has bucket-number floor(2.2) = 2
- Clearly somehow related to GROUP BY, but GROUP BY works with column values directly, not expressions like “floor(cost/10)”, according to R&G, and the SQL standards (SQL-92 and SQL-2003)
- But first try GROUP BY floor(cost/10) in Oracle...
Histogram in Oracle

select floor(c.cost/10)*10 as "range-start", count(*) from catalog c
    group by floor(c.cost/10)  -- not OK by SQL 92 or 2003
    order by floor(c.cost/10);  --OK by SQL 2003, but not SQL 92

<table>
<thead>
<tr>
<th>range-start</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>120</td>
<td>2</td>
</tr>
</tbody>
</table>

8 rows selected.

This shows that Oracle allows GROUP BY <expression>, at least simplest enough expressions.
How about mysql?

```sql
select floor(c.cost/10)*10 as "range-start",
       count(*) from catalog c
    group by floor(c.cost/10)
    order by floor(c.cost/10);
```

ERROR 1055 (42000): Expression #1 of SELECT list is not in GROUP BY clause and contains nonaggregated column 'eoneill1db.c.cost' which is not functionally dependent on columns in GROUP BY clause; this is incompatible with sql_mode=only_full_group_by

Hmm, mysql refuses to go beyond the standard in this particular case?? What if we change the sql_mode?
How about mysql?

Turn off SQL modes for the session, and the query works!

```
SET sql_mode = '';
select floor(c.cost/10)*10 as "range-start", count(*) from catalog c
    group by floor(c.cost/10)
    order by floor(c.cost/10);
```

<table>
<thead>
<tr>
<th>range-start</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>120</td>
<td>2</td>
</tr>
</tbody>
</table>

8 rows in set (0.01 sec)
What if we are restricted to GROUP BY <column>?

- We can use a query in the FROM clause to generate the bucket numbers, the floor(cost/10) values:

```sql
select bucket*10 as "range-start",
       count(*) from catalog c,
          (select distinct floor(cost/10) as bucket from catalog) b
where floor(c.cost/10) = b.bucket
group by b.bucket;
```

This provides the same output. We have made the computed bucket numbers show up in a column in the table-query.
Null Values

- Field values in a tuple may sometimes be
  - unknown: e.g., a rating has not been assigned, or a new column is added to the table
  - inapplicable: e.g., CEO has no manager, single person has no spouse

- SQL provides a special value **NULL** for such situations
  - Special operators **IS NULL, IS NOT NULL**
  - SELECT * FROM Sailors WHERE rating **IS NOT NULL**
  - Note: **NULL** must not be used as constant in expressions!
  - A field can be declared as **NOT NULL**, means NULL values are not allowed (by default, PK fields are NOT NULL)
Try it out...

SQL> select * from class;
---unreadable mess---
SQL> col name format a20;
SQL> select * from class;

<table>
<thead>
<tr>
<th>NAME</th>
<th>MEETS_AT</th>
<th>ROOM</th>
<th>FID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structures</td>
<td>MWF 10</td>
<td>R128</td>
<td>489456522</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Introduction to Math</td>
<td>TuTh 8-9:30</td>
<td>R128</td>
<td>489221823</td>
</tr>
<tr>
<td>Artificial Intelligence</td>
<td></td>
<td>UP328</td>
<td></td>
</tr>
</tbody>
</table>

look like null column values
Try out “is null”

SQL> select * from class where meets_at is null;

<table>
<thead>
<tr>
<th>NAME</th>
<th>MEETS_AT</th>
<th>ROOM</th>
<th>FID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Psychology</td>
<td></td>
<td></td>
<td>619023588</td>
</tr>
<tr>
<td>Artificial Intellige</td>
<td></td>
<td>UP328</td>
<td></td>
</tr>
<tr>
<td>nce</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SQL> select * from class where room is null;

<table>
<thead>
<tr>
<th>NAME</th>
<th>MEETS_AT</th>
<th>ROOM</th>
<th>FID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Psychology</td>
<td></td>
<td></td>
<td>619023588</td>
</tr>
</tbody>
</table>
Dealing with Null Values

- The presence of **NULL** complicates some issues
  - **NULL op value** has as result **NULL** (op is +,-,*,/)
  - What does \( \text{rating} > 8 \) evaluate to if \( \text{rating} \) is equal to **NULL**?
  - Answer: **unknown**

- **3-valued logic**: true, false and **unknown**
  - Recall that WHERE eliminates rows that don’t evaluate to true
  - What about **AND**, **OR** and **NOT** connectives?
    - unknown AND true = unknown
    - unknown OR false = unknown
    - NOT unknown = unknown
  - Also, \(<\text{NULL\_value}> = <\text{NULL\_value}>\) is unknown!
Null Values and Aggregates

- The COUNT(*) result includes tuples with NULL.

- \( \text{COUNT}(A) \) only counts tuples where value of attribute \( A \) is not NULL.

- All other aggregates skip NULL values (if aggregate is on the field that is NULL).
  - If all values are NULL on the aggregated field, the result of aggregate is also NULL (except \( \text{COUNT}(A) \) which returns 0).
Null Values and Aggregates

Following two queries **DO NOT RETURN SAME RESULT** if there are **NULLs** (in field *name*):

SELECT COUNT(*) FROM Sailors S

SELECT COUNT(S.name) FROM Sailors S

Following two queries **DO NOT RETURN SAME RESULT** if there are **NULLs** (in field *rating*):

SELECT COUNT(*) FROM Sailors S

SELECT COUNT(*) FROM Sailors WHERE (rating > 8) OR (rating <= 8)
Table class has a row with a null room count.

`count(room)` counts non-null room values.

```
SQL> select count(*) from class;

    COUNT (*)
----------
         23

SQL> select count(room) from class;

    COUNT (ROOM)
------------
          22
```
WHERE clause evaluation with nulls
Simpler Sailors instance, with a null rating:

<table>
<thead>
<tr>
<th>SID</th>
<th>SNAME</th>
<th>RATING</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>jones</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>41</td>
<td>jonah</td>
<td>6</td>
<td>56</td>
</tr>
<tr>
<td>22</td>
<td>ahab</td>
<td>7</td>
<td>44</td>
</tr>
<tr>
<td>63</td>
<td>moby</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

SQL> select count(*) from sailors
WHERE (rating > 8) OR (rating <= 8)

<table>
<thead>
<tr>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

- Rating = null row: WHERE (null > 8) or (null <= 8)
  unknown or unknown
- Result: unknown
- so not qualified by WHERE clause
- WHERE only qualifies TRUE rows, not UNKNOWN or FALSE ones
Null Values and Duplicates

- Comparing two NULL values gives as result unknown

- But this rule does not hold when checking for duplicates!
  - NULL values are considered equal in this case!
  - Two tuples are duplicates if they match in all non-NULL attributes (of both) and have nulls in the other attributes
    - (1, null) and (1,null) are dups, but (1, null) and (1,2) are not dups

- Implications for DISTINCT, UNIQUE subqueries, set operations!
  - Tuples with NULL in some group-by attributes placed in same group if all non-NULL group-by attributes match!
    - In one-col case: all null values put in just one group-by group
    - (in applications, best to use non-null group-by columns)
  - DISTINCT: if two tuples have equal values in all non-NULL attributes and nulls matching otherwise, only one of them is output