By Marina Barsky

Structured Query Language SQL

Lecture 7

GROUP BY and AGGREGATION

Aggregation Operators

- They apply to entire columns of a table and produce a single result.
- The most important examples:
  - SUM
  - AVG
  - COUNT
  - MIN
  - MAX

Example: Aggregation

R =

A	B
1	3
3	4
3	2

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
MIN(B) = 2
AVG(B) = 3

```
SELECT SUM(A), COUNT(A), MAX(B), MIN(B), AVG(B)
FROM R;
```

Remark

- We can also use COUNT(*) which counts the number of tuples in the relation constructed from the FROM and WHERE clauses of the query.

GROUP BY clause

Grouping Operator

$R_1 := \gamma_L (R_2)$

L is a list of elements that are either:

1. Individual (*grouping*) attributes.
2. AGG(*A*), where AGG is one of the aggregation operators and *A* is an attribute.

Example: Grouping/Aggregation

R =

A	B	C
1	2	3
4	5	6
1	2	5

```
SELECT A,B,AVG(C)
FROM R
GROUP BY A,B;
```

$\gamma_{A,B,\text{AVG}(C)}(R) = ??$

First, group R :

A	B	C
1	**2**	3
1	**2**	5
4	**5**	6

Then, average C within groups:

A	B	AVG(C)
1	2	4
4	5	6

$\gamma_L(R)$ - Formally

- Group *R* according to all the grouping attributes on list *L*.
  - That is, form one group **for each distinct list** of values for those attributes in ***R***.

- Within each group, compute AGG(*A*) for each aggregation on list *L*.

- Result has grouping attributes and aggregations as attributes:

One tuple for each list of values for the grouping attributes **and** their group's aggregations.

GROUP BY

- The GROUP BY clause follows a SELECT-FROM-WHERE expression

- The result of the SELECT-FROM-WHERE query
  - is grouped according to the values of all the listed attributes in GROUP BY, and
  - any aggregation is applied only within each group and gives a single value per group

Restriction on SELECT Lists With Aggregation

- If any aggregation is used, then each element of the SELECT list must be either:
  1. Aggregated, or
  2. An attribute on the GROUP BY list.

Examples 1

```
SELECT country, MIN(GPA) AS minGPA
FROM Student
GROUP BY country
```

```
SELECT MAX(grade)
FROM took
GROUP BY course;
```

Student		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5
Maria	Mexico	4

Took		
Name	Course	Grade
Bob	Algo	55
John	Algo	90
Tom	DB	85
Maria	HCI	100

Examples 2

```
SELECT COUNT (course)
FROM took;

SELECT COUNT (DISTINCT course)
FROM took;

SELECT COUNT (*)
FROM took;

SELECT MAX(grade), MIN(grade),
COUNT (DISTINCT course), COUNT (*)
FROM took;
```

Took		
Name	Course	Grade
Bob	Algo	55
John	Algo	90
Tom	DB	85
Maria	HCI	100

Examples 3

~~SELECT name, AVG (grade)~~
FROM took;

SELECT name, AVG (grade)
FROM took
GROUP BY name;

SELECT name, AVG (grade)
FROM took
GROUP BY name
ORDER BY 2 DESC;

Took		
Name	Course	Grade
Bob	Algo	55
John	Algo	90
Tom	DB	85
Maria	HCI	100

HAVING clause

HAVING

- **HAVING <condition>** may follow a GROUP BY clause.

  If so, the condition applies to each group, and groups not satisfying the condition are eliminated.
- WHERE let's you decide which tuples to keep.
- Similarly, you can decide which *groups* to keep.
- Syntax:

  ...
  GROUP BY *«attributes»*
  HAVING *«condition»*
- HAVING refer to attributes of any relation in FROM clause, as long as the attribute makes sense within a group; i.e., it is either:
  - A grouping attribute, or
  - Aggregated attribute.

Example 1

```
SELECT country, MIN(GPA) AS minGPA
FROM Student
GROUP BY country
HAVING COUNT(country)>=2;
```

Student		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5
Maria	Mexico	4

Example 2

- For each student who took at least three courses give his maximum grade
  - First we group, using *Name* as a grouping attribute.
  - Then, we compute the Max(*grade*) for each group.
  - Also, we need to compute the COUNT(*name*) aggregate for each group, for filtering out those students with less than three courses.

Took		
Name	Course	grade
Bob	Algorithms	78
John	Algorithms	60
Tom	Algorithms	88
Bob	Python	100
Tom	Python	99
Bob	Databases	89
John	Databases	95
Maria	Databases	67
John	GUI	56
Maria	GUI	90

Example 2

- For each student who took at least three courses give his maximum grade

```
SELECT name, MAX(grade) AS maxGrade
FROM Took
GROUP BY name
HAVING COUNT(name)>=3;
```

Took		
Name	Course	grade
Bob	Algorithms	78
John	Algorithms	60
Tom	Algorithms	88
Bob	Python	100
Tom	Python	99
Bob	Databases	89
John	Databases	95
Maria	Databases	67
John	GUI	56
Maria	GUI	90

“Having” is a special kind of σ

- The previous query can also be written using sub-query as:

```
SELECT name, maxGrade
FROM
  (SELECT name, MAX(grade) AS maxGrade, COUNT(name) AS ctName
  FROM took
  GROUP BY name)
WHERE ctName >= 3
```

Impact of null values on aggregation

- Aggregation ignores NULL.
  - NULL never contributes to a sum, average, or count, and
  - can never be the minimum or maximum of a column (unless *every* value is NULL).

- If there are no *non*-NULL values in a column, then the result of the aggregation is NULL.
  - Exception: COUNT of an empty set is 0.

Example: Effect of NULL's

```
SELECT count(*)
FROM Student
WHERE gpa <= 3;
```

vs.

```
SELECT count(country)
FROM Student
WHERE gpa <= 3;
```

Student		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5
Carry		2.8
Maria	Mexico	4

Duplicate Elimination

```
SELECT DISTINCT name
FROM Took;
```

- Without DISTINCT, a name would be listed as many times as there were repetitions of student name

Took		
Name	Course	grade
Bob	Algorithms	78
John	Algorithms	60
Tom	Algorithms	88
Bob	Python	100
Tom	Python	99
Bob	Databases	89
John	Databases	95
Maria	Databases	67
John	GUI	56
Maria	GUI	90

Duplicate Elimination in aggregation

- **DISTINCT** inside an aggregation causes duplicates to be eliminated **before** the aggregation.

```
SELECT COUNT (DISTINCT name)
FROM Took;
```

vs.

```
SELECT DISTINCT COUNT (name)
FROM Took;
```

??

```
SELECT AVG(DISTINCT grade)
FROM Took;
```

??

Took		
Name	Course	grade
Bob	Algorithms	78
John	Algorithms	60
Tom	Algorithms	88
Bob	Python	100
Tom	Python	99
Bob	Databases	89
John	Databases	95
Maria	Databases	67
John	GUI	56
Maria	GUI	90

Correlated Subqueries

- Find the students who got the grade greater than the average grade for a given course.

```
SELECT name, course
FROM Took X
WHERE grade >
        (SELECT AVG(grade)
         FROM Took Y
         WHERE Y.course = X.course)
```

Remarks

1. Outer query cannot reference any columns in the subquery.
2. Subquery references the tuple in the outer query.
3. Value of the tuple changes by row of the outer query, so the database must rerun the subquery for each row comparison.

Another Solution (Subquery in FROM)

- Find the students who got the grade greater than the average grade for a given course.

```
SELECT X.name, X.course
FROM Took X, (SELECT course, AVG(grade) AS avgGrade
                    FROM Took
                    GROUP BY course) Y
WHERE X.grade>Y.avgGrade AND
X.name=Y.name;
```

CASE control structure: groups

Count how many students in each grade group per course

```
SELECT course,
  CASE WHEN grade BETWEEN 0 AND 25 THEN ‘G1'
       WHEN grade BETWEEN 25 AND 50 THEN ‘G2'
       WHEN grade BETWEEN 50 AND 75 THEN ‘G3'
       WHEN grade BETWEEN 75 AND 100 THEN ‘G4'
  END CASE AS grade_group,
  COUNT(*) AS num_students
FROM took
GROUP BY course, grade_group;
```

CASE with above average

- Find how many students got the grade above the average grade in a given course

```
SELECT  course,
COUNT(*) AS total_count_percourse,
AVG(grade) as avg_grade,
SUM (CASE WHEN grade > X.course_average THEN 1 ELSE 0 END CASE)
            AS count_aboveaverage
FROM  ( SELECT course, AVG(grade) AS course_average FROM Took
                        GROUP BY course ) as X,
            Took
WHERE
      Took.course = X.course

GROUP BY course;
```