CS411
Database Systems

06b: SQL-2

Grouping and Aggregation

Aggregations

- SUM, AVG, COUNT, MIN, and MAX can be applied to a column in a SELECT clause to produce that aggregation on the column.
- Also, COUNT(*) counts the number of tuples.

Example: Aggregation

- From Sells(bar, beer, price), find the average price of Bud:

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'Bud';
```

Example: Aggregation

- From Sells (bar, beer, price), find the number of bars selling Bud:

```
SELECT COUNT(*)
FROM Sells
WHERE beer = 'Bud';
```

Eliminating Duplicates in an Aggregation

- DISTINCT inside an aggregation causes duplicates to be eliminated before the aggregation.
- Example: find the number of different prices charged for Bud:

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = 'Bud';
```

5

NULL's Ignored in Aggregation

- NULL never contributes to a sum, average, or count, and can never be the minimum or maximum of a column.
- But if there are no non-NULL values in a column, then the result of the aggregation is NULL.

6

Example: Effect of NULL's

```
SELECT count(*)
FROM Sells
WHERE beer = 'Bud';
```

The number of bars that sell Bud.

```
SELECT count(price)
FROM Sells
WHERE beer = 'Bud';
```

The number of bars that sell Bud at a known price.

7

Grouping

- We may follow a SELECT-FROM-WHERE expression by GROUP BY and a list of attributes.
- The relation that results from the SELECT-FROM-WHERE is grouped according to the values of all those attributes, and any aggregation is applied only within each group.

8

Example: Grouping

- From Sells(bar, beer, price), find the average price for each beer:

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer;
```

9

Example: Grouping

- From Sells(bar, beer, price) and Frequents(drinker, bar), find for each drinker the average price of Bud at the bars they frequent:

```
SELECT drinker, AVG(price)
FROM Frequents, Sells
WHERE beer = 'Bud' AND
      Frequents.bar = Sells.bar
GROUP BY drinker;
```

Compute drinker-bar-price of Bud tuples first, then group by drinker.

10

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.

11

Q: How about this query?

```
SELECT bar, MIN(price)
FROM Sells
WHERE beer = 'Bud';
```

12

Q: How to do it right, then?

```
SELECT bar, MIN(price)
FROM Sells
WHERE beer = 'Bud';
```

```
SELECT bar FROM Sells
WHERE beer = 'Bud' AND price =
(SELECT MIN(price) FROM Sells
 WHERE beer = 'Bud')
```

13

HAVING Clauses

- HAVING <condition> may follow a GROUP BY clause.
- If so, the condition applies to each group, and groups not satisfying the condition are eliminated.

14

The HAVING clause: Example

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
HAVING COUNT(bar) >= 3
```

15

Requirements on HAVING Conditions

- These conditions may refer to any relation or tuple-variable in the FROM clause.
- They may refer to attributes of those relations, as long as the attribute makes sense within a group; i.e., it is either:
  1. A grouping attribute, or
  2. Aggregated.

16

General form of Grouping and Aggregation

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

S = may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but NO OTHER ATTRIBUTES
C1 = is any condition on the attributes in $R_1,\ldots,R_n$
C2 = is any condition on aggregate expressions or grouping attributes

17

General form of Grouping and Aggregation

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

Evaluation steps:

1. Compute the FROM-WHERE part, obtain a table with all attributes in $R_1,\ldots,R_n$
2. Group by the attributes $a_1,\ldots,a_k$
3. Compute the aggregates in C2 and keep only groups satisfying C2
4. Compute aggregates in S and return the result

18