

# CS411 Database Systems

## 06: SQL

Kazuhiro Minami

## Join Expressions

### Join Expressions

- SQL provides a number of expression forms that act like varieties of join in relational algebra.
  - But using bag semantics, not set semantics.
- These expressions can be stand-alone queries or used in place of relations in a FROM clause.

### Products and Natural Joins

- Natural join is obtained by:  
`R NATURAL JOIN S;`
- Cartesian product is obtained by:  
`R CROSS JOIN S;`
- Example:  
`Likes NATURAL JOIN Serves;`
- Relations can be parenthesized subexpressions, as well.

## Theta Join

- $R \text{ JOIN } S \text{ ON } \langle \text{condition} \rangle$  is a theta-join, using  $\langle \text{condition} \rangle$  for selection.
- Example: using Drinkers(name, addr) and Frequents(drinker, bar):

```
Drinkers JOIN Frequents ON  
    name = drinker;
```

gives us all  $(d, a, d, b)$  quadruples such that drinker  $d$  lives at address  $a$  and frequents bar  $b$ .

## 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';
```

## 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';
```

## NULLs are ignored in aggregations of a column

```
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 non-null price*

If there are no non-NULL values in a column, then the result of the aggregation is NULL

## 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 partitioned according to the values of all those attributes, and any aggregation is applied only within each group.

## Example: Grouping

**Sells(bar, beer, price)**

Q: find the average price for each beer:

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

## Example: Grouping

**Frequents(drinker, bar), Sells(bar, beer, price)**

Q: find for each drinker the average price of Bud at the bars they frequent:

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

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

## 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.

## Illegal Query Example

You might think you could find the bar that sells Bud the cheapest by:

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

But this query is illegal in SQL.

- Why? Note bar is neither aggregated nor on the GROUP BY list.

## 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.

## 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.

## Example: HAVING

**Sells(bar, beer, price)**

**Q: Find the average price of those beers that are served in at least three bars**

## Solution

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

Beer groups with at least 3 non-NULL bars and also beer groups where the manufacturer is Pete's.

## General form of Grouping and Aggregation

```
SELECT S
FROM R1,...,Rn
WHERE C1
GROUP BY a1,...,ak
HAVING C2
```

S = may contain attributes  $a_1, \dots, a_k$  and/or any aggregates but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in  $R_1, \dots, R_n$

C2 = is any condition on aggregate expressions or grouping attributes

## General form of Grouping and Aggregation

```
SELECT S
FROM R1,...,Rn
WHERE C1
GROUP BY a1,...,ak
HAVING C2
```

Evaluation steps:

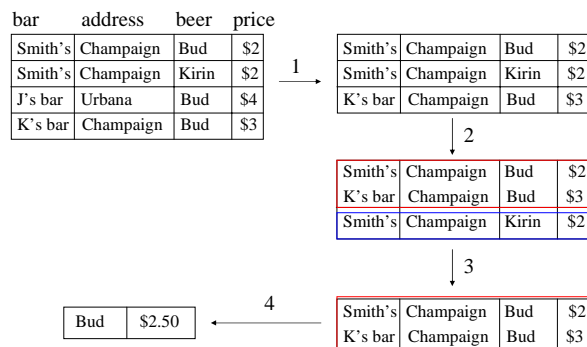
1. Compute the FROM-WHERE part, obtain a table with all attributes in  $R_1, \dots, R_n$
2. Group by the attributes  $a_1, \dots, a_k$
3. Compute the aggregates in  $C_2$  and keep only groups satisfying  $C_2$
4. Compute aggregates in  $S$  and return the result

## Example

- From Sells(bar, beer, price), find the average price for each beer that is sold by more than one bar in Champaign:

```
SELECT beer, AVG(price)
FROM Sells
Where address = 'Champaign'
GROUP BY beer
Having COUNT(bar) > 1
```

## Example



## Exercise 3: online bookstore

**Book(isbn, title, publisher, price)**

**Author(assn, aname, isbn)**

**Customer(cid, cname, state, city, zipcode)**

**Buy(tid, cid, isbn, year, month, day)**

Q3: Make a list of the names of customers who live in Illinois and spent more than \$5,000 in year 2000.