Subqueries Boolean Operators IN, EXISTS, ANY, ALL

Announcement . MT Tutorial Session Monday 3/7. Fpm. 1304 SC Green se Bor - a
Beet - BOOK Red St
Lib,
Bar - D Lib prinker - D Scholan green ted Joe Bar ~ 706

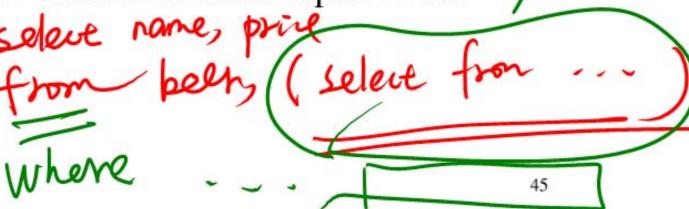
Subqueries

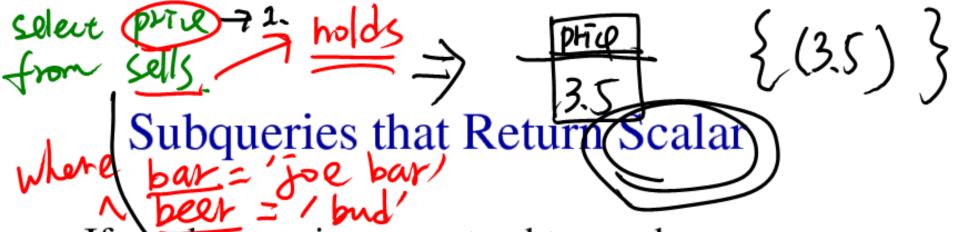
 A parenthesized SELECT-FROM-WHERE statement (subquery) can be used as a value in a number of places, including FROM and WHERE clauses.

• Example: in place of a relation in the FROM clause, we can place another query, and then query its result.

Better use a tuple-variable to name tuples of the

result.





• If a subquery is guaranteed to produce one tuple with one component, then the subquery can be used as a value.

- "Single" tuple often guaranteed by key constraint.

 A run-time error occurs if there is no tuple or more than one tuple.

when price > 3.5

Example

- From Sells(<u>bar</u>, <u>beer</u>, price), find the bars that serve Miller for the same price Joe charges for Bud.
- Two queries would surely work:
 - 1. Find the price Joe charges for Bud.
 - 2. Find the bars that serve Miller at that price.

Query + Subquery Solution

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
price = (SELECT price
FROM Sells
WHERE bar = 'Joe Bar'
AND beer = 'Bud');
```

The IN Operator

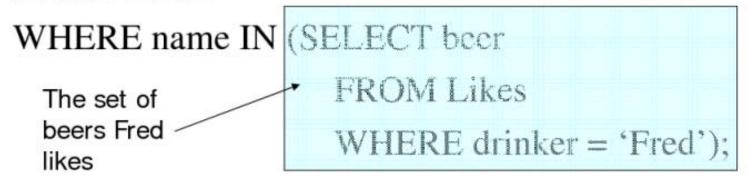
- <tuple>(IN < relation> is true if and only if the tuple is a member of the relation.
 - <tuple> NOT IN <relation> means the opposite.
- IN-expressions can appear in WHERE clauses.
- The <relation> is often a subquery.

Example

 From Beers(name, manf) and Likes(drinker, beer), find the name and manufacturer of each beer that Fred likes.

SELECT *

FROM Beers



The Exists Operator

- EXISTS(<relation>) is true if and only if the <relation> is not empty.
- Being a boolean-valued operator, EXISTS can appear in WHERE clauses.
- Example: From Beers(name, manf), find those beers that are the only beer by their manufacturer.

Example Query with EXISTS

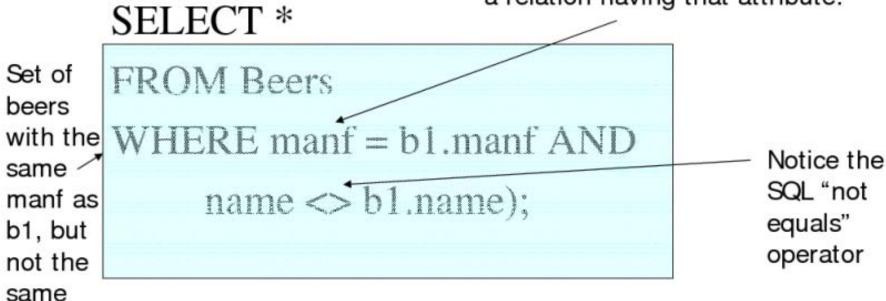
SELECT name

FROM Beers b1

beer

WHERE NOT EXISTS(

Notice scope rule: manf refers to closest nested FROM with a relation having that attribute.



The Operator ANY

- $x = ANY(\langle relation \rangle)$ is a boolean condition meaning that x equals at least one tuple in the relation.
- Similarly, = can be replaced by any of the comparison operators.
- Example: $x \ge ANY(\langle relation \rangle)$ means x is not smaller than all tuples in the relation.
 - Note tuples must have one component only.

The Operator ALL

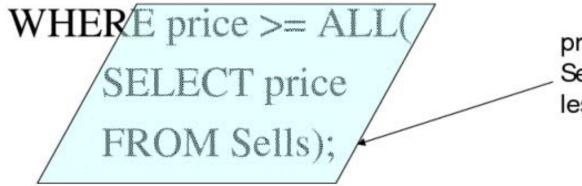
- Similarly, x <> ALL(<relation>) is true if and only if for every tuple t in the relation, x is not equal to t.
 - That is, x is not a member of the relation.
- The <> can be replaced by any comparison operator.
- Example: x >= ALL(<relation>) means there is no tuple larger than x in the relation.

Example

 From Sells(bar, beer, price), find the beer(s) sold for the highest price.

SELECT beer

FROM Sells



price from the outer Sells must not be less than any price.

CS411 Database Systems

06b: SQL-2

Grouping and Aggregation

Why Do We Learn This?

- Organize (grouping) - Gen. Report/Stat. Q: What is "aggregate"? Avg, Max. Sum, ...

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';
```

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.

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.

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.

Example: Grouping

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

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

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)

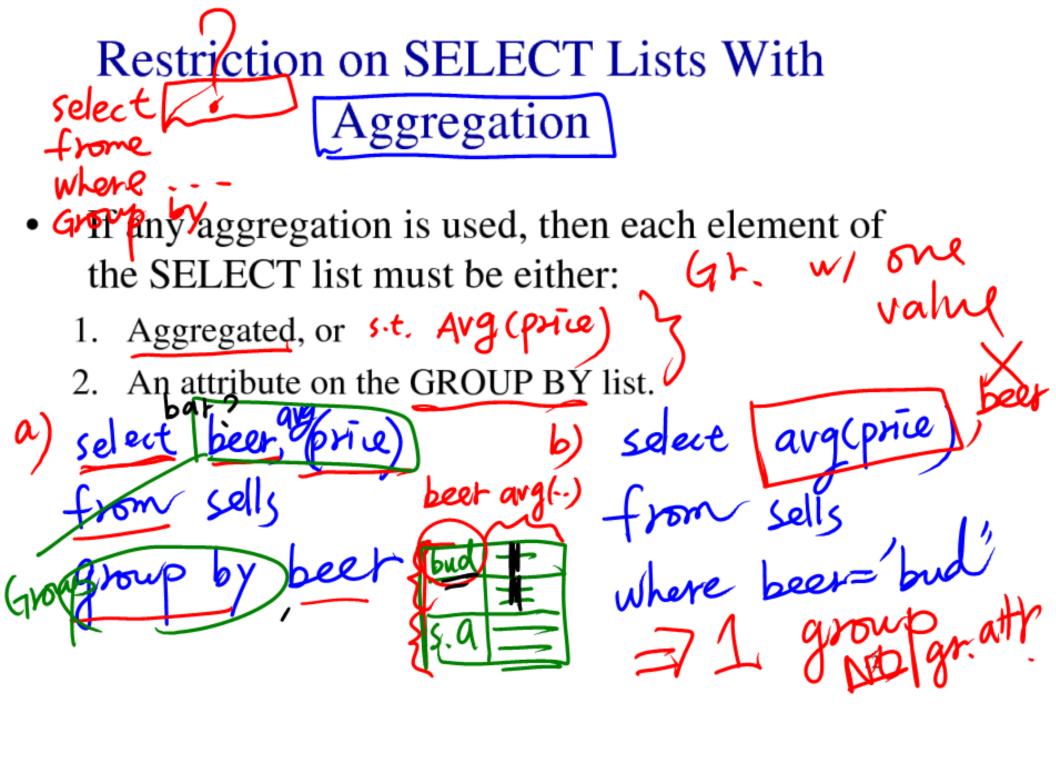
FROM Frequents, Sells

WHERE beer = 'Bud' AND

Frequents.bar = Sells.bar

GROUP BY drinker;

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



Q: How about this query? FROM Sells WHERE beer = 'Bud'; with min price, the price)

Q: How to do it right, then?

