CS411
Database Systems

06: SQL

Kazuhiro Minami

Multi-Relation Queries

Find the beers liked by at least one person who frequents Murphy's Pub

Likes(drinker, beer)   Frequents(drinker, bar)

```
SELECT   beer AS beersWorthKeeping
FROM     Likes, Frequents
WHERE Likes.drinker = Frequents.drinker
         AND bar = 'Murphy''s Pub';
```

Likes

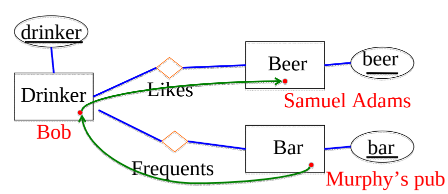
Likes. drinker	beer
Bob	Samuel Adams
Alice	Bud

Frequents

Frequesnts . drinker	bar
Bob	Murphy's Pub
Alice	Blind Pig

beersWorthKeeping
Samuel Adams

If you are confused, go back to E/R model



Likes

Likes. drinker	beer
Bob	Samuel Adams
Alice	Bud

Frequents

Frequesnts . drinker	bar
Bob	Murphy's Pub
Alice	Blind Pig

Exercise 2: online bookstore

Book(<u>isbn</u>, title, publisher, price)
Author(<u>assn</u>, aname, <u>isbn</u>)
Customer(<u>cid</u>, cname, state, city, zipcode)
Buy(<u>tid</u>, cid, isbn, year, month, day)

Q2: Make a list of the CIDs and customer names who bought books written by 'Barack Obama'?

```
SELECT Customer.cid, Customer.cname
FROM    Author, Buy, Customer
WHERE Customer.cid = Buy.cid AND Buy.isbn = Author.ibn
           AND Author.name = `Barack Obama' ;
```

Who bought Obama's book?

Author

Assn	Aname	isbn
1	Barack Obama	0307455874
1	Barack Obama	1400082773
2	Dan Brown	0385504225

Customer

cid	cname
1	John Carlson
2	Allen Huang
3	Ryan Mathews

Buy

tid	cid	isbn
1	1	0307455874
2	2	0385504225
3	3	1400082773

Unfortunately, DBMS cannot look at multiple tables at the same time

Author. assn	Author. aname	Author. isbn	Customer. cid	Customer. cname	Buy. tid	Buy. cid	Buy. isbn
1	Barack Obama	0307455874	1	John Carlson	1	1	0307455874
1	Barack Obama	0307455874	1	John Carlson	2	2	0385504225
1	Barack Obama	0307455874	1	John Carlson	3	3	1400082773
1	Barack Obama	0307455874	2	Allen Huang	1	1	0307455874
1	Barack Obama	0307455874	2	Allen Huang	2	2	0385504225
1	Barack Obama	0307455874	2	Allen Huang	3	3	1400082773
1	Barack Obama	0307455874	3	Ryan Mathews	1	1	0307455874
1	Barack Obama	0307455874	3	Ryan Mathews	2	2	0385504225
1	Barack Obama	0307455874	3	Ryan Mathews	3	3	1400082773

I cannot include all the combined tuples. Too big!

Check conditions in Where clause

Customer.cid = Buy.cid AND Buy.isbn = Author.ibn
AND Author.name = `Barack Obama'

Author. assn	Author. aname	Author. isbn	Customer. cid	Customer. cname	Buy. tid	Buy. cid	Buy. isbn
1	Barack Obama	0307455874	1	John Carlson	1	1	0307455874
1	Barack Obama	1400082773	3	Ryan Mathews	3	3	1400082773

SELECT Customer.cid, Customer.cname

What if a query needs to use two copies of the same relation?

Beers(name, manf)

Find all pairs of beers by the same manufacturer.

- Do not produce pairs like (Bud, Bud).
- Produce pairs in alphabetic order, e.g. (Bud, Miller), not (Miller, Bud).

```
SELECT  b1.name, b2.name
FROM    Beers b1, Beers b2
WHERE   b1.manf = b2.manf
    AND b1.name < b2.name
```

Beers1(name1, manf1) = Beers.

Then (almost): π[name, name1](Beers $\bowtie_{\text{manf = manf1 \& name} \neq \text{name1}}$ Beers2)

What if a query needs to use two copies of the same relation?

b1

name	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Super Dry	Asahi

b2

name	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Super Dry	Asahi

B1. name	B1.manf	B2.name	B2.manf
Bud	Anheuser-Busch	Bud	Anheuser-Bush
Bud	Anheuser-Busch	Bud Lite	Anheuser-Bush
Bud	Anheuser-Busch	Super Dry	Asahi
Bud Lite	Anheuser-Busch	Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch	Bud Lite	Anheuser-Busch
Bud Lite	Anheuser-Busch	Super Dry	Asahi
Super Dry	Asahi	Bud	Anheuser-Busch
Super Dry	Asahi	Bud Lite	Anheuser-Bush
Super Dry	Asahi	Super Dry	Asahi

Usually, you want your query results to appear in a meaningful order

SELECT *attributes1*
FROM *relations*
WHERE *condition1*
GROUP BY *attributes*
HAVING *condition2*
ORDER BY attr1, …, attrk

Sort the output on attr1.
Within each value for attr1,
sort the output on attr2.
Within each value for attr1&attr2, …

List the students in CS411.

```
SELECT studentLastName, studentFirstName, netID
FROM    courseEnrollments
WHERE  course = 'cs411'
ORDER BY studentLastName, studentFirstName ;
```

DESC ASC

Meaning (Semantics) of SQL Queries

SELECT A1, …, Ak
FROM R1, …, Rn
WHERE *conditions*

1. Nested loops:

Answer = {}
for x_1 **in** R1 **do**
 …..
  for x_n **in** Rn **do**
   if *conditions* $(x_1,..,x_n)$
    then Answer = Answer $\cup$ {$(a_1,\ldots,a_k)$}
return Answer

Projection onto A1,…,Ak

Meaning (Semantics) of SQL Queries

SELECT A1, …, Ak
FROM R1 AS x1, …, Rn AS xn
WHERE *conditions*

3. Translation to relational algebra:

$\Pi_{A1,\ldots,Ak} (\sigma_{conditions} (R1 \times \ldots \times Rn))$

Select-From-Where queries are relational algebra's Select-Project-Join queries

Unintuitive Consequence of SQL Semantics

Suppose that we have relations R(A), S(A), T(A) and that we want to compute $R \cap (S \cup T)$.

```
SELECT R.A
FROM   R, S, T
WHERE R.A = S.A OR R.A = T.A
```

Q: What would be a result of the query if T is empty?

Sub-queries

Subqueries

- A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as part of the main SELECT-FROM-WHERE statement
- Subqueries can be nested in an arbitrary depth

Usage of Subqueries

- Subquires can return a constant, and this constant can be compared with another value in a WHERE clause
- Subquries can return a relation that can be used to define conditions in a WHERE clauses
  - IN, ALL, ANY, EXISTS
- In place of a relation in the FROM clause, we can place another query
  - Better use a tuple-variable to name tuples of the result.

Because the result of a query is always a relation, you can query the result of a query

List all restaurants that opened in 2000.

subquery

```
SELECT name
FROM
    (SELECT name, yearOpened
      FROM   Restaurants)  R
WHERE R.yearOpened = 2000;
```

From Sells(bar, beer, price), find the bars that serve Miller for the same price Joe charges for Bud.

Two queries would surely work (if we save the intermediate results):

1. Find the price Joe charges for Bud.
2. Find the bars that serve Miller at that price.

If a subquery is guaranteed to produce one tuple, then the subquery can be used like an ordinary value

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

The price at which Joe sells Bud

But be careful when you use subqueries as values!

- Usually the tuple has one attribute.
- You'd better be sure only one tuple will be returned (e.g., keys guarantee it).
- A run-time error occurs if there is no tuple or more than one tuple.

The IN and NOT IN operators allow you to test whether a value or tuple is in a relation

Beers(name, manf)   Likes(drinker, beer)

Find the name and manufacturer of each beer that Fred likes.

```
SELECT *
FROM Beers
WHERE name IN (SELECT beer
               FROM Likes
               WHERE drinker = 'Fred');
```

The set of beers Fred likes

Q: Find the name and manufacturer of each beer that Fred likes.

Beers(name, manf)   Likes(drinker, beer)

name	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Super Dry	Asahi

drinker	beer
Fred	Bud
~~Kazu~~	~~Super Dry~~
Fred	Bud Lite

```
SELECT *
FROM Beers
WH Super Dry IN (
```

beer
✓ Bud
✓ Bud Lite

d');

Subquery

Output relation:

name	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch

You can test whether a relation is empty, using EXISTS and NOT EXISTS

Beers(name, manf)

Find the beers that are the only one made by their manufacturer

```
SELECT name
FROM Beers b1
WHERE NOT EXISTS(
    SELECT *
    FROM Beers
    WHERE manf = b1.manf AND
        name <> b1.name);
```

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

Q: Find the beers that are the only one made by their manufacturer

Beers b1

name	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Super Dry	Asahi

Beers (in subquery)

name	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Super Dry	Asahi

```
SELECT name
FROM Beers b1
WHERE NOT EXISTS(
  SELECT *
  FROM Beers
```

Anheuser Asahi = Anheuser-Busch

Super Dry <> Bud ame);

Temporary relation of the subquery

beer	manf
Bud Lite	Anheuser-Busch

Output relation

name
Super Dry

Use ANY/ALL to check whether a condition is true at least once/always

x = ANY(*subquery*) ⟸ true if *x* is in the (single column) answer to *subquery*

x >= ANY(*subquery*) ⟸ true if *x* is as big as one or more answers to *subquery*

x = ALL(*subquery*) ⟸ true if *x* is the only answer to *subquery*

x >= ALL(*subquery*) ⟸ true if *x* is as big as all the answers to *subquery*

Find the beer(s) sold for the highest price

Sells(bar, beer, price)

```
SELECT beer
FROM   Sells
WHERE price >= ALL(
       SELECT price
       FROM   Sells);
```

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

Relations as Bags

Bag Semantics in SQL

- The SELECT-FROM-WHERE statement uses bag semantics
  - Selection: preserve the number of occurrences
  - Projection: preserve the number of occurrences (no duplicate elimination)
  - Cartesian product, join: no duplicate elimination

Exceptions: Union, Intersection, and Difference

- Union, intersection, and difference of relations are expressed by the following forms, each involving subqueries:
  - (subquery) UNION (subquery)
  - (subquery) INTERSECT (subquery)
  - (subquery) EXCEPT (subquery)

Motivation: Efficiency

- When doing projection in relational algebra, it is easier to avoid eliminating duplicates.
  - Just work tuple-at-a-time.
- When doing intersection or difference, it is most efficient to sort the relations first.
  - At that point you may as well eliminate the duplicates anyway.

You can control whether duplicates are eliminated

- Force the result to be a **set** by SELECT DISTINCT . . .
- Force the result to be a **bag** (i.e., don't eliminate duplicates) by ALL, as in . . . UNION ALL . . .

From Sells(bar, beer, price), find all the different prices charged for beers:

```
SELECT DISTINCT price
FROM Sells;
```

Without DISTINCT, each price would be listed as many times as there were bar/beer pairs at that price.

Example: ALL

- Using relations Frequents(drinker, bar) and Likes(drinker, beer):

```
(SELECT drinker FROM Frequents)
    EXCEPT ALL
(SELECT drinker FROM Likes);
```

- Lists drinkers who frequent more bars than they like beers, and does so as many times as the difference of those counts.