Introduction to Database Systems
CSE 414

Lecture 8: Nested Queries in SQL

Announcements

- Homework 2 due tonight
- Webquiz 3 due Friday – 2 questions on nested queries (i.e., today's stuff)
- Homework 1 solution on the web now
- Homework 3 …

Homework 3

- More IMDB queries using SQL Azure
- Login details in the assignment (out today)
  - Userid = your UW netid; password on whiteboard
  - Change password on first login
- Demo in sections tomorrow
- Software (tested on these)
  - SQL Server 2008 R2 or later (free to cse414 students)
  - SQL Server 2012 Express (free download)
  - Web browser (uses silverlight)
- SQL server 20xx downloads are huge (~1GB) and can take an hour to install
  - Only need SQL Server Management Studio part (still huge)

Lecture Goals

- Today we will learn how to write more powerful SQL queries
- They are needed in Homework 3
- Reading: Ch. 6.3

Subqueries

- A subquery is a SQL query nested inside a larger query
- Such inner-outer queries are called nested queries
- A subquery may occur in:
  - A SELECT clause
  - A FROM clause
  - A WHERE clause
- Rule of thumb: avoid writing nested queries when possible; keep in mind that sometimes it's impossible

Subqueries…

- Can return a single constant and this constant can be compared with another value in a WHERE clause
- Can return relations that can be used in various ways in WHERE clauses
- Can appear in FROM clauses, followed by a tuple variable that represents the tuples in the result of the subquery
- Can appear as computed values in a SELECT clause

1. Subqueries in SELECT

Product (pname, price, cid)
Company(cid, cname, city)

For each product return the city where it is manufactured

```
SELECT X.pname, (SELECT Y.city
                 FROM Company Y
                 WHERE Y.cid=X.cid) as City
FROM  Product X
```

"correlated subquery"

What happens if the subquery returns more than one city ?

We get a runtime error
(SQLite simply ignores the extra values)

Product (pname, price, cid)
Company(cid, cname, city)

1. Subqueries in SELECT

Whenever possible, don't use a nested queries:

```
SELECT X.pname, (SELECT Y.city
                 FROM Company Y
                 WHERE Y.cid=X.cid) as City
FROM  Product X
```

=

```
SELECT X.pname, Y.city
FROM   Product X, Company Y
WHERE X.cid=Y.cid
```

We have "unnested" the query

Product (pname, price, cid)
Company(cid, cname, city)

1. Subqueries in SELECT

Compute the number of products made by each company

```
SELECT DISTINCT C.cname, (SELECT count(*)
                          FROM Product P
                          WHERE P.cid=C.cid)
FROM  Company C
```

Better: we can unnest by using a GROUP BY

```
SELECT C.cname, count(*)
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY C.cname
```

Product (pname, price, cid)
Company(cid, cname, city)

1. Subqueries in SELECT

But are these really equivalent?

```
SELECT DISTINCT C.cname, (SELECT count(*)
                          FROM Product P
                          WHERE P.cid=C.cid)
FROM  Company C
```

```
SELECT C.cname, count(*)
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY C.cname
```

No! Different results if a company has no products

```
SELECT C.cname, count(pname)
FROM Company C LEFT OUTER JOIN Product P
ON C.cid=P.cid
GROUP BY C.cname
```

Product (pname, price, cid)
Company(cid, cname, city)

2. Subqueries in FROM

Find all products whose prices is > 20 and < 500

```
SELECT X.pname
FROM (SELECT * FROM Product AS Y WHERE price > 20) as X
WHERE X.price < 500
```

Unnest this query !

2. Subqueries in FROM

- Later we will see that sometimes we really need a subquery and one option will be to put it in the FROM clause (see “finding witnesses”)

Product (pname, price, cid)
Company(cid, cname, city)

3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Using EXISTS:

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE  EXISTS (SELECT *
                        FROM Product P
                       WHERE C.cid = P.cid and P.price < 200)
```

Product (pname, price, cid)
Company(cid, cname, city)

3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Using IN

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE C.cid IN (SELECT P.cid
                FROM Product P
                WHERE P.price < 200)
```

Product (pname, price, cid)
Company(cid, cname, city)

3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Using ANY:

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE 200 > ANY (SELECT price
                 FROM Product P
                 WHERE P.cid = C.cid)
```

Product (pname, price, cid)
Company(cid, cname, city)

3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Now let's unnest it:

```
SELECT DISTINCT  C.cname
FROM     Company C, Product P
WHERE  C.cid= P.cid and P.price < 200
```

Existential quantifiers are easy ! ☺

Product (pname, price, cid)
Company(cid, cname, city)

3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200

Universal quantifiers

Universal quantifiers are hard ! ☹

Product (pname, price, cid)
Company(cid, cname, city)

3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

1. Find *the other* companies: i.e. s.t. some product ≥ 200

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE  C.cid IN (SELECT P.cid
                 FROM Product P
                 WHERE P.price >= 200)
```

2. Find all companies s.t. all their products have price < 200

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE  C.cid NOT IN (SELECT P.cid
                     FROM Product P
                     WHERE P.price >= 200)
```

Product (pname, price, cid)
Company(cid, cname, city)

3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Universal quantifiers

Using EXISTS:

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE NOT EXISTS (SELECT *
                  FROM Product P
                  WHERE P.cid = C.cid and P.price >= 200)
```

Product (pname, price, cid)
Company(cid, cname, city)

3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Universal quantifiers

Using ALL:

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE 200 > ALL  (SELECT price
                  FROM Product P
                  WHERE P.cid = C.cid)
```

Question for Database Fans and their Friends

- Can we unnest the *universal quantifier* query ?

Product (pname, price, cid)
Company(cid, cname, city)

Monotone Queries

- Definition A query Q is monotone if:
  - Whenever we add tuples to one or more input tables, the answer to the query will not lose any of of the tuples

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c003
Camera	149.99	c001

Company

cid	cname	city
c001	Sunworks	Bonn
c002	DB Inc.	Lyon
c003	Builder	Lodtz

Q

A	B
149.99	Lodtz
19.99	Lyon

Is the mystery query monotone?

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c003
Camera	149.99	c001
iPad	499.99	c001

Company

cid	cname	city
c001	Sunworks	Bonn
c002	DB Inc.	Lyon
c003	Builder	Lodtz

Q

A	B
149.99	Lyon
19.99	Lyon
19.99	Bonn
149.99	Bonn

Monotone Queries

- Theorem: If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

- Proof. We use the nested loop semantics: if we insert a tuple in a relation R_i, this will not remove any tuples from the answer

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

```
for x1 in R1 do
    for x2 in R2 do
        …..
            for xn in Rn do
                if Conditions
                    output (a1,…,ak)
```

Product (pname, price, cid)
Company(cid, cname, city)

Monotone Queries

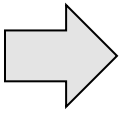
- The query:

Find all companies s.t. all their products have price < 200

is not monotone

pname	price	cid
Gizmo	19.99	c001

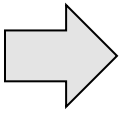
cid	cname	city
c001	Sunworks	Bonn



cname
Sunworks

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c001

cid	cname	city
c001	Sunworks	Bonn



cname

- Consequence: we cannot write it as a SELECT-FROM-WHERE query without nested subqueries

Queries that must be nested

- Queries with universal quantifiers or with negation
- The drinkers-bars-beers example next time
  - This is a famous example from textbook on databases by Ullman

Practice these queries in SQL

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

Ullman's drinkers-bars-beers example

Find drinkers that frequent some bar that serves some beer they like.

x: $\exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$

Find drinkers that frequent only bars that serves some beer they like.

x: $\forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$

Find drinkers that frequent some bar that serves only beers they like.

x: $\exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$

Find drinkers that frequent only bars that serves only beer they like.

x: $\forall y.\ \text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$