Database Systems
CSE 414

Lecture 6: Nested Queries in SQL

Announcements

- WQ2 is due on Sunday 11pm
  - no late days
- HW2 is due on Tuesday 11pm

Lecture Goals

- Today we will learn how to write (even) more powerful SQL queries
- 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 nested queries when possible; keep in mind that sometimes it's impossible
  - (though use in FROM is often not as bad)

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
```

What happens if the subquery returns more than one city ?

We get a runtime error

- (SQLite simply ignores the extra values)

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)

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
```

DBMS also does this…

We have "unnested" the query

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
```


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
```


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 !

```
SELECT pname
FROM Product
WHERE price > 20 AND price < 500
```


2. Subqueries in FROM

- We will see that sometimes we really need a subquery
  - will see most compelling examples next lecture
  - in that case, we can put it in the FROM clause

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

3. Subqueries in WHERE

Find all companies that make some products with price < 100

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 < 100)
```


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

3. Subqueries in WHERE

Find all companies that make some products with price < 100

Existential quantifiers

Using IN

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE C.cid IN (SELECT P.cid
                    FROM Product P
                    WHERE P.price < 100)
```


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

3. Subqueries in WHERE

Find all companies that make some products with price < 100

Existential quantifiers

Using ANY:

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE 100 > ANY (SELECT price
                    FROM Product P
                    WHERE P.cid = C.cid)
```

Not supported in sqlite

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

3. Subqueries in WHERE

Find all companies that make some products with price < 100

Existential quantifiers

Now let's unnest it:

```
SELECT DISTINCT C.cname
FROM    Company C, Product P
WHERE C.cid= P.cid and P.price < 100
```

Existential quantifiers are easy ! ☺

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

3. Subqueries in WHERE

Find all companies where all their products have price < 100

same as:

Find all companies that make only products with price < 100

Universal quantifiers

Universal quantifiers are hard ! ☹

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

3. Subqueries in WHERE

Find all companies where all their products have price < 100

1. Find *the other* companies: i.e. with some product >= 100

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE C.cid IN (SELECT P.cid
                    FROM Product P
                    WHERE P.price >= 100)
```

2. Find all companies where all their products have price < 100

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE C.cid NOT IN (SELECT P.cid
                    FROM Product P
                    WHERE P.price >= 100)
```

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

3. Subqueries in WHERE

Find all companies where all their products have price < 100

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 >= 100)
```


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

3. Subqueries in WHERE

Find all companies where all their products have price < 100

Universal quantifiers

Using ALL:

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE 100 >= ALL  (SELECT price
                   FROM Product P
                   WHERE P.cid = C.cid)
```

Not supported in sqlite

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 the tuples

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003

Company

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

Q

A	B
Gizmo	Lyon
Camera	Lodtz

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003
iPad	499.99	c001

Company

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

Q

A	B
Gizmo	Lyon
Camera	Lodtz
iPad	Lyon

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 where all their products have price < 100

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

(that is, cannot be SFW queries)

- Queries with universal quantifiers or negation
- Queries that use aggregates in certain ways
  - Note: sum(..) etc. are NOT monotone
  - **select count(*) from R** is not monotone!