Introduction to Database Systems
CSE 414

Lecture 7: SQL Wrapup

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
  - But sometimes it's impossible, as we will see

FWGHOS

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

(and SQLite simply ignores the extra values…)

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
```

Simple Aggregations

Five basic aggregate operations in SQL

```
select count(*) from Purchase
select sum(quantity) from Purchase
select avg(price) from Purchase
select max(quantity) from Purchase
select min(quantity) from Purchase
```

count of an empty table is 0

Except count, all aggregations apply to a single attribute

Including Empty Groups

```
SELECT x.manufacturer, count(y.pid)
FROM Product x LEFT OUTER JOIN Purchase y
ON x.pname = y.product
GROUP BY x.manufacturer
```

Product

pname	manufacturer	…
Gizmo	GizmoWorks	
Camera	Canon	
OneClick	Hitachi	

Purchase

product	price	...
Camera	150	
Camera	300	
OneClick	180	

Left Outer Join(Product, Purchase)

pname	manufacturer	…	product	price	…
Camera	Canon		Camera	150	
Camera	Canon		Camera	300	
OneClick	Hitachi		OneClick	180	
Gizmo	GizmoWorks	…	NULL	NULL	NULL

Why 0 for GizmoWorks?

Final results

manufacturer	Count(y.pid)
Canon	2
Hitachi	1
GizmoWorks	0

GizmoWorks is paired with NULLs

Including Empty Groups

```
SELECT x.manufacturer, count(*)
FROM Product x LEFT OUTER JOIN Purchase y
ON x.pname = y.product
GROUP BY x.manufacturer
```

Product

pname	manufacturer	…
Gizmo	GizmoWorks	
Camera	Canon	
OneClick	Hitachi	

Purchase

product	price	...
Camera	150	
Camera	300	
OneClick	180	

Left Outer Join(Product, Purchase)

pname	manufacturer	…	product	price	…
Camera	Canon		Camera	150	
Camera	Canon		Camera	300	
OneClick	Hitachi		OneClick	180	
Gizmo	GizmoWorks	…	NULL	NULL	NULL

Final results

manufacturer	Count(*)
Canon	2
Hitachi	1
GizmoWorks	1

Probably not what we want!

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

1. Subqueries in SELECT

Recall: count of an empty table is 0

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
```

Side note: This is not a correlated subquery. (why?)

Try unnest this query !

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

2. Subqueries in FROM

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

=

A subquery whose result we called myTable

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

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

3. Subqueries in WHERE

Find all companies that make some products with 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

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

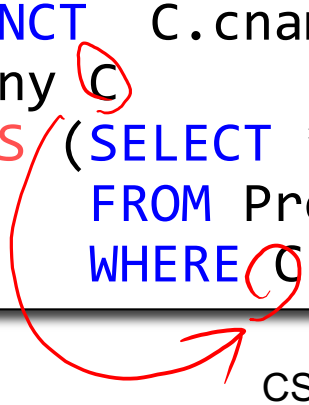
3. Subqueries in WHERE

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

Using ANY:

```
SELECT DISTINCT  C.cname
FROM  Company C
WHERE 200 > 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 < 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
```

```
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

```
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

```
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 that make some product ≥ 200

```
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 s.t. all their products have price < 200

1. Find *the other* companies that make 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

NOT ANY

Using ALL:

```
SELECT DISTINCT  C.cname
FROM  Company C
WHERE 200 >= ALL (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 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)
```

Not supported in sqlite

Question for Database Theory Fans and their Friends

- Can we unnest the *universal quantifier* query?

- We need to first discuss the concept of *monotonicity*

```
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)
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

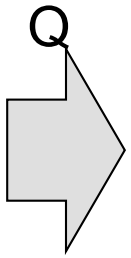
---

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



pname	city
Gizmo	Lyon
Camera	Lodtz

```
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

pname	city
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

pname	city
Gizmo	Lyon
Camera	Lodtz
iPad	Lyon

So far it looks monotone...

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

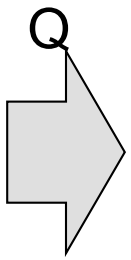
---

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



pname	city
Gizmo	Lyon
Camera	Lodtz

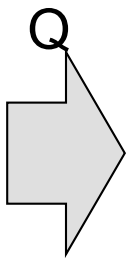
Product

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
c004	Crafter	Lodtz

Q is not monotone!



pname	city
Gizmo	Lodtz
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.

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

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

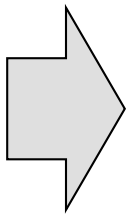
Monotone Queries

- 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

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

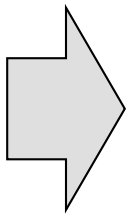
Monotone Queries

- The query:

Find all companies s.t. all their products have price < 200
is not monotone

pname	price	cid
Gizmo	19.99	c001

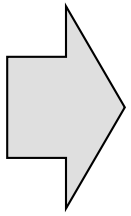
cid	cname	city
c001	Sunworks	Bonn



cname
Sunworks

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c001

cid	cname	city
c001	Sunworks	Bonn



cname

- Consequence: If a query is not monotonic, then 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

Queries that must be nested

- Queries with universal quantifiers or with negation

- Queries that use aggregates in certain ways
  - `sum(..)` and `count(*)` are NOT monotone, because they do not satisfy set containment
  - `select count(*) from R` is not monotone!

SQL Idioms

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

Finding Witnesses

For each city, find the most expensive product made in that city

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

Finding Witnesses

For each city, find the most expensive product made in that city

Finding the maximum price is easy…

```
SELECT x.city, max(y.price)
FROM   Company x, Product y
WHERE  x.cid = y.cid
GROUP BY x.city;
```

But we need the *witnesses*, i.e., the products with max price

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

Finding Witnesses

To find the witnesses, compute the maximum price in a subquery (in FROM or in WITH)

```
WITH CityMax AS
  (SELECT x.city, max(y.price) as maxprice
   FROM Company x, Product y
   WHERE x.cid = y.cid
   GROUP BY x.city)
SELECT DISTINCT u.city, v.pname, v.price
FROM Company u, Product v, CityMax w
WHERE u.cid = v.cid
      and u.city = w.city
      and v.price = w.maxprice;
```

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

Finding Witnesses

To find the witnesses, compute the maximum price in a subquery (in FROM or in WITH)

```
SELECT DISTINCT u.city, v.pname, v.price
FROM Company u, Product v,
     (SELECT x.city, max(y.price) as maxprice
      FROM Company x, Product y
      WHERE x.cid = y.cid
      GROUP BY x.city) w
WHERE u.cid = v.cid
      and u.city = w.city
      and v.price = w.maxprice;
```

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

Finding Witnesses

Or we can use a subquery in where clause

```
SELECT u.city, v.pname, v.price
FROM Company u, Product v
WHERE u.cid = v.cid
  and v.price >= ALL (SELECT y.price
                      FROM Company x, Product y
                      WHERE u.city=x.city
                      and x.cid=y.cid);
```

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

Finding Witnesses

There is a more concise solution here:

```
SELECT u.city, v.pname, v.price
FROM Company u, Product v, Company x, Product y
WHERE u.cid = v.cid and u.city = x.city
and x.cid = y.cid
GROUP BY u.city, v.pname, v.price
HAVING v.price = max(y.price)
```

SQL: Our first language for the relational model

- Projections
- Selections
- Joins (inner and outer)
- Inserts, updates, and deletes
- Aggregates
- Grouping
- Ordering
- Nested queries