CSE 344

JUNE 27TH

SUBQUERIES

ADMINISTRIVIA

- **HW1 Due Tonight (11pm)**
  - Make sure the grading script passes
  - Run the turn-in script to submit & tag
  - Check on gitlab after submitting
- **WQ2 Due Friday**
- **HW2 Out Tomorrow**
  - Due next Wednesday (April 11)

SEMANTICS OF SQL WITH GROUP-BY

```
SELECT    S
FROM      R1,…,Rn
WHERE     C1
GROUP BY  a1,…,ak
HAVING    C2
```

FWGHOS

Evaluation steps:

1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group rows with same values in the attributes $a_1,\ldots,a_k$
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

REVIEW

- **SQL**
  - inner & outer joins (FROM & WHERE clauses)
  - group by
    - subsequent processing applies to groups not rows
    - can only use group-by columns and aggregation
      - count, sum, average, min, max
  - having filter on groups (vs where on rows)
  - order by
  - select is processed **last**
  - (almost done with SQL now…)

```
Purchase(pid, product, price, quantity, month)
```

MYSTERY QUERY

What do they compute?

```
SELECT      month, sum(quantity), max(price)
FROM        Purchase
GROUP BY    month
```

```
SELECT      month, sum(quantity)
FROM        Purchase
GROUP BY    month
```

```
SELECT      month
FROM        Purchase
GROUP BY    month
```

```
Purchase(pid, product, price, quantity, month)
```

MYSTERY QUERY

What do they compute?

```
SELECT     month, sum(quantity), max(price)
FROM       Purchase
GROUP BY   month
```

```
SELECT     month, sum(quantity)
FROM       Purchase
GROUP BY   month
```

```
SELECT     month
FROM       Purchase
GROUP BY   month
```

Lesson:
DISTINCT is
a special case
of GROUP BY

```
Product(product_id,pname,manufacturer)
Purchase(pid,product_id,price,month)
```

AGGREGATE + JOIN

For each manufacturer, compute how many products with price > $100 they sold

```
Product(product_id,pname,manufacturer)
Purchase(pid,product_id,price,month)
```

AGGREGATE + JOIN

For each manufacturer, compute how many products with price > $100 they sold

Problem: manufacturer is in Purchase, price is in Product...

```
Product(product_id,pname,manufacturer)
Purchase(pid,product_id,price,month)
```

AGGREGATE + JOIN

For each manufacturer, compute how many products with price > $100 they sold

Problem: manufacturer is in Purchase, price is in Product...

```
-- step 1: think about their join
SELECT ...
FROM Product x, Purchase y
WHERE x.product_id = y.product_id
  and y.price > 100
```

manu facturer	...	price	...
Hitachi		150	
Canon		300	
Hitachi		180	

```
Product(product_id,pname,manufacturer)
Purchase(pid,product_id,price,month)
```

AGGREGATE + JOIN

For each manufacturer, compute how many products with price > $100 they sold

Problem: manufacturer is in Purchase, price is in Product...

```
-- step 1: think about their join
SELECT ...
FROM Product x, Purchase y
WHERE x.product_id = y.product_id
  and y.price > 100
```

manufacturer	...	price	...
Hitachi		150	
Canon		300	
Hitachi		180	

```
-- step 2: do the group-by on the join
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.product_id = y.product_id
  and y.price > 100
GROUP BY  x.manufacturer
```

manufacturer	count(*)
Hitachi	2
Canon	1
...	

```
Product(product_id,pname,manufacturer)
Purchase(pid,product_id,price,month)
```

AGGREGATE + JOIN

Variant:
For each manufacturer, compute how many products with price > $100 they sold in each month

```
SELECT x.manufacturer, y.month, count(*)
FROM Product x, Purchase y
WHERE x.product_id = y.product_id
  and y.price > 100
GROUP BY  x.manufacturer, y.month
```

manufacturer	month	count(*)
Hitachi	Jan	2
Hitachi	Feb	1
Canon	Jan	3
...		

INCLUDING EMPTY GROUPS

FWGHOS

In the result of a group by query, there is one row per group in the result

Count(*) is never 0

```sql
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.product_id = y.product_id
GROUP BY x.manufacturer
```

INCLUDING EMPTY GROUPS

Count(pid) is 0
when all pids in
the group are NULL

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

SUBQUERIES

A subquery is a SQL query nested inside another query

- inner query is also called a “nested query”

A subquery may occur in:

- `SELECT` clause
- `FROM` clause
- `WHERE` clause

Rule of thumb: avoid nested queries when possible

- But sometimes it’s impossible to avoid, as we will see
- (And those in the FROM clause are not usually a problem)

SUBQUERIES…

- **Can return a single value to be included in a `SELECT` clause**
  - query must return relation with 1 row & 1 column
- **Can return a relation to be included in the `FROM` clause, aliased using a tuple variable**
- **Can return a single value to be compared with another value in a `WHERE` clause**
- **Can return a relation to be used in the `WHERE or` `HAVING` clause under an existential quantifier**

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

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
```

```
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 with 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
```

```
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
```

```
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
```

Try to unnest this query !

```
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 to unnest this query !

2. SUBQUERIES IN FROM

Sometimes we need to compute an intermediate table only to use it later in a SELECT-FROM-WHERE

Option 1: use a subquery in the FROM clause

Option 2: use the WITH clause

```
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

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:

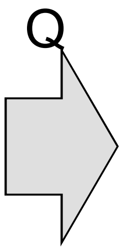
- Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples

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

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
c004	Crafter	Lodtz

Q

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 $a_1$, $a_2$, …, $a_k$
FROM   $R_1$ AS $x_1$, $R_2$ AS $x_2$, …, $R_n$ AS $x_n$
WHERE  Conditions
```

```
for $x_1$ in $R_1$ do
  for $x_2$ in $R_2$ do
    …
    for $x_n$ in $R_n$ do
      if Conditions
        output ($a_1$,…,$a_k$)
```

```
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

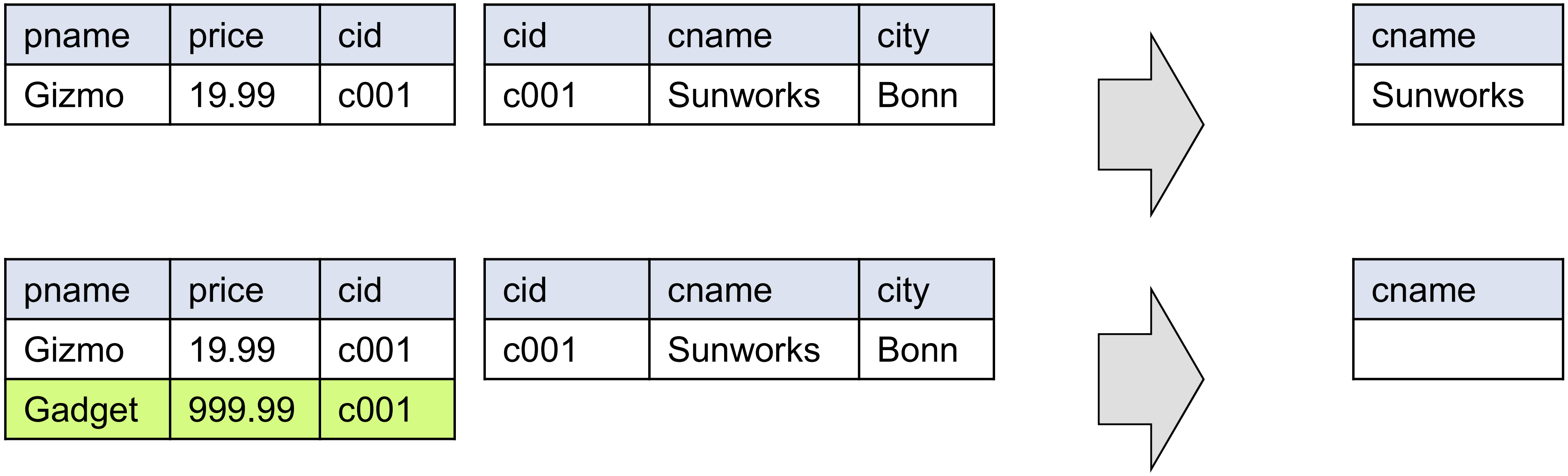
```
Product (pname,  price, cid)
Company (cid, cname, city)
```

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 grouping or nested subqueries

Purchase(pid, product, quantity, price)

GROUP BY V.S. NESTED QUERIES

```
SELECT   product, Sum(quantity) AS TotalSales
FROM     Purchase
WHERE    price > 1
GROUP BY product
```

```
SELECT DISTINCT x.product, (SELECT Sum(y.quantity)
                            FROM     Purchase y
                            WHERE x.product = y.product
                              AND y.price > 1)
                            AS TotalSales
FROM  Purchase x
WHERE x.price > 1
```

Why twice ?

```
Author(login,name)
Wrote(login,url)
```

MORE UNNESTING

Find authors who wrote ≥ 10 documents:

```
Author(login,name)
Wrote(login,url)
```

MORE UNNESTING

Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

This is SQL by a novice

```
SELECT DISTINCT Author.name
FROM         Author
WHERE         (SELECT count(Wrote.url)
               FROM Wrote
               WHERE Author.login=Wrote.login)
                  >= 10
```

```
Author(login,name)
Wrote(login,url)
```

MORE UNNESTING

Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

Attempt 2: using GROUP BY and HAVING

```
SELECT      Author.name
FROM        Author, Wrote
WHERE       Author.login=Wrote.login
GROUP BY Author.name
HAVING      count(wrote.url) >= 10
```

This is
SQL by
an expert

```
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)
```