Introduction to Database Systems
CSE 414

Lecture 6: SQL Subqueries

Announcements

- Web Quiz 2 due Friday night
- HW 2 due Tuesday at midnight
- Section this week important for HW 3, must attend

Announcements

- Many students did not turn in hw1 correctly – need to make sure your files are here:

https://gitlab.cs.washington.edu/cse414-2018au/cse414-[username]/tree/master/hw/hw[homework#]/submission

E.g. https://gitlab.cs.washington.edu/cse414-2018au/cse414-maas/tree/master/hw/hw1/submission

AND you have the hw1 tag here:

https://gitlab.cs.washington.edu/cse414-2018au/cse414-[username]/tags

- Commit, then use ./turnInHw.sh hw2 script.
- MUST have this correct for HW2

Semantics of SQL With Group-By

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

Evaluation steps:

1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by 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

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

Aggregate + Join

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

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

Aggregate + Join

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

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

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

Aggregate + Join

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

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

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

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

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

Aggregate + Join

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

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

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

manu facturer	...	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.pid = y.product_id
  and y.price > 100
GROUP BY  x.manufacturer
```

manu facturer	count(*)
Hitachi	2
Canon	1
...	9

```
Product(pid,pname,manufacturer)
Purchase(id,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.pid = y.product_id
  and y.price > 100
GROUP BY  x.manufacturer, y.month
```

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

FWGHOS

Including Empty Groups

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

INNER JOINS will not allow us to manipulate these empty groups as no corresponding entry in table x can be found for y need an outer join

Count(*) is not 0 because there are no tuples to count!

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

Including Empty Groups

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

FWGHOS

Product

pname	manufacturer	…
Gizmo	GizmoWorks	
Camera	Canon	
OneClick	Hitachi	

Purchase

product	price	...
Camera	150	
Camera	300	
OneClick	180	

Join(Product, Purchase)

pname	manu facturer	…	manu facturer	price	…
Camera	Canon		Canon	150	
Camera	Canon		Canon	300	
OneClick	Hitachi		Hitachi	180	

No GizmoWorks!

Final results

manufacturer	Count(*)
Canon	2
Hitachi	1

FWGHOS

Including Empty Groups

count(*) would NOT work here

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

group still exists when we get to select statement

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

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

NULLS are NOT counted

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!

count (*) will count the the rows that have NULL entries because there are some NON-null entries in the row (i.e. the row exists)

What we have in our SQL toolbox

- Projections (`SELECT * / SELECT c1, c2, …`)
- Selections (aka filtering) (`WHERE cond, HAVING`)
- Joins (inner and outer)
- Aggregates
- Group by
- Inserts, updates, and deletes

Make sure you read the textbook!

Subqueries

- In the relational model, the output of a query is also a relation

- Can use output of one query as input to another

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

can be really slow

- Rule of thumb: avoid nested queries when possible
  - But sometimes it's impossible, as we will see

FWGHOS

Subqueries…

- Can return a single value to be included in a `SELECT` clause
- 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

Subqueries…

Subqueries are often:

- Intuitive to write
- Slow

Be careful!

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
```

can ONLY return a single value

"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
```

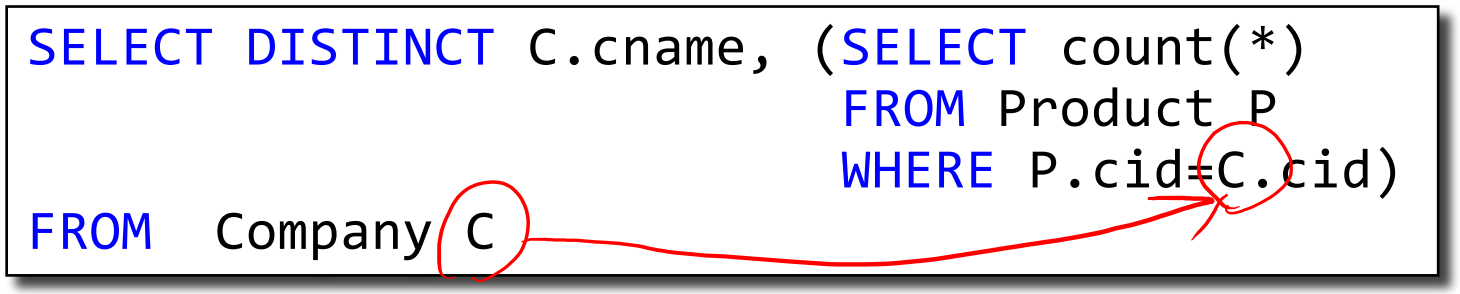
We have "unnested" the query

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 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
```

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
```

when doing the cross product, since there is no C.cid = P.pid pair for company with no products, it is not part of returned response

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

treats subquery as a separate table that we are extracting elements 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
```

have to give the subquery a name

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
```

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

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?)

add a second WHERE condition (X.price > 20) AND (X.price < 500)

Try to unnest this query !

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

3. Subqueries in WHERE

Find all companies that make some products with price < 200

```
SELECT C.cname
FROM Company
WHERE 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 that make some products with price < 200

Existential quantifiers

SELECT …… WHERE EXISTS (subquery);

SELECT …. WHERE NOT EXISTS (subquery);

SELECT….. attribute IN (subquery)
SELECT….. attribute NOT IN (subquery)

SELECT….. attribute > ANY(subquery)

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

3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

For each tuple in Company C, checks whether the subquery returns at least 1 tuple

Using EXISTS:

select company name that makes some product that costs < $200

```
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

checks whether the given attribute value can be found in one of the tuples returned by subquery; subquery must ONLY return tuples with a SINGLE attribute

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

not SELECT *

```
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:

returns only if ANY value returned by subquery satisfies condition
subquery must only return a SINGLE attribute

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

there is also an ALL qualifier; only returns tuple if all tuples in subquery satisfy the given condition (try changing the ANY query on last slide with ALL instead)

```
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)
```

negate it!

```
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

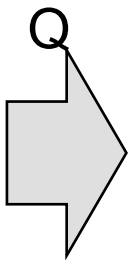
---

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

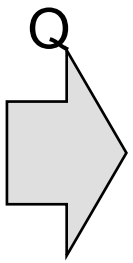
---

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



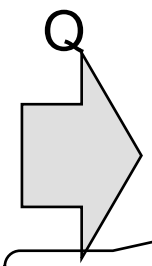
pname	city
Gizmo	Lyon
Camera	Lodtz

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



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

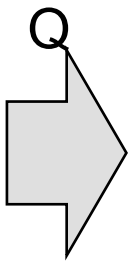
---

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



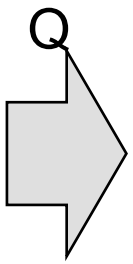
pname	city
Gizmo	Lyon
Camera	Lodtz

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)
```

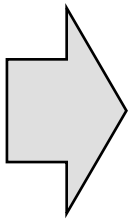
Monotone Queries

- 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)
```

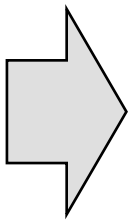
Monotone Queries

- The query:

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

pname	price	cid
Gizmo	19.99	c001

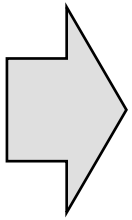
cid	cname	city
c001	Sunworks	Bonn



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

i.e. for loop semantics fails because some tuples are being added to output prematurely (before whole table processed) and thus might not belong their depending on future tuples

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!

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)

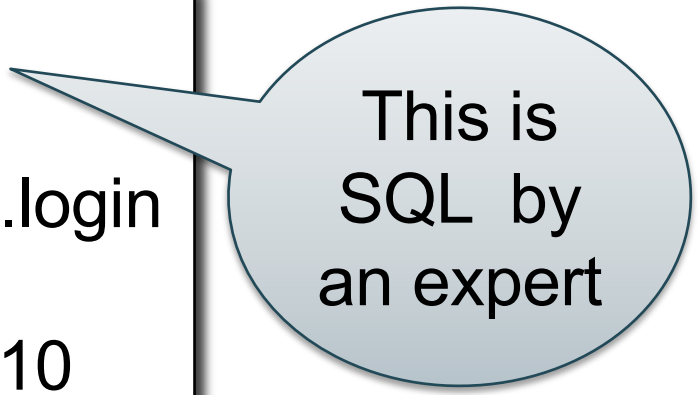
More Unnesting

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
```



```
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)

```
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;
```

Joining three tables

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)
```