Database Systems
CSE 344

Lecture 7: SQL Wrap-up
Monday July 3

Announcements

- WQ3 is out, due next Monday 11pm
- HW2 is due Wednesday (July 5) 11pm
  - H3 will be posted later this week
  - You will be using Microsoft Azure
  - Will post instructions for setting up account on Thursday.

Recap from last lecture

- Subqueries can occur in many clauses:
  - SELECT
  - FROM
  - WHERE
- Monotone queries: SELECT-FROM-WHERE
  - Existential quantifier
- Non-monotone queries
  - Universal quantifier
  - Aggregation

Examples of Complex Queries

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

1. People that frequent some restaurant that serves some food they like.
2. People that frequent some restaurant that serves only food they don't like.
3. People that frequent only restaurants that serves some food they like.

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 1

People that frequent some restaurant that serves some food they like.

$$Q(p) = \exists r, \exists f, (F(p,r) \wedge S(r, f) \wedge L(p, f))$$

Existential quantifiers are easy

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 1

People that frequent some restaurant that serves some food they like.

```
SELECT DISTINCT F.person
FROM Frequents F, Serves S, Likes L
WHERE F.restaurant = S.restaurant AND
    S.food = L.food AND
    F.person = L.person
```

inner join

What happens if we didn't write DISTINCT?

Could also us GROUP BY

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 1

People that frequent some restaurant that serves some food they like.

```
SELECT DISTINCT F.person
FROM Frequents F, Serves S, Likes L
WHERE F.restaurant = S.restaurant AND
    S.food = L.food AND
    F.person = L.person
```

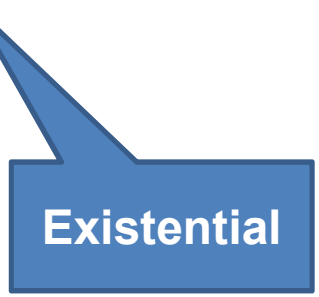
person + restaurant they frequent + food served that they like
=> person is an answer

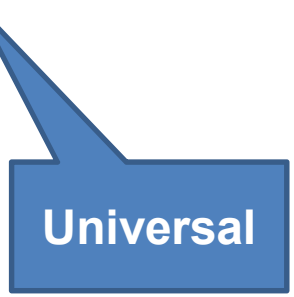
(even though we only want the person,
we need the rest to know it's an answer.)

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 2

People that frequent some restaurant that serves only food they don't like





$$Q(p) = \exists r\,(F(p,r) \wedge \forall f\,(S(r,f) \rightarrow \neg L(p,r)))$$

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 2

People that frequent some restaurant that serves only food they don't like

$$Q(p) = \exists r\,(F(p,r) \wedge \forall f\,(S(r,f) \rightarrow \neg L(p,r)))$$

Restaurant serves only food that X does not like

Equivalent To

Restaurant that does NOT serve some food that X does like

$$Q(p) = \exists r\,F(p,r) \wedge \neg \exists f\,(S(r,f) \wedge L(p,f))$$

Let's find the others (drop the NOT): → Example 1

People that frequent some restaurant that serves some food they like.

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 2

People that frequent some restaurant that serves only food they don't like

Let's find the others (drop the NOT):
People that frequent some restaurant that serves some food they like.

That's the previous query…

```
SELECT DISTINCT F.person
FROM Frequents F, Serves S, Likes L
WHERE F.restaurant = S.restaurant AND
    S.food = L.food AND
    F.person = L.person
```

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 2

People that frequent some restaurant that serves only food they don't like

Let's find the others (drop the NOT):
People that frequent some restaurant that serves some food they like.

That's the previous query… Let's write it with a subquery:

```
SELECT DISTINCT F.person
FROM Frequents F WHERE EXISTS (
        SELECT *
        FROM Serves S, Likes L
        WHERE F.restaurant =S.restuarnt
            AND  F.person = L.person
            AND  S.food = L.food
)
```

Example 1

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 2

People that frequent some restaurant that serves only food they don't like

Let's find the others (drop the NOT):
People that frequent some restaurant that serves some food they like.

That's the previous query… Let's write it with a subquery:

Now negate!

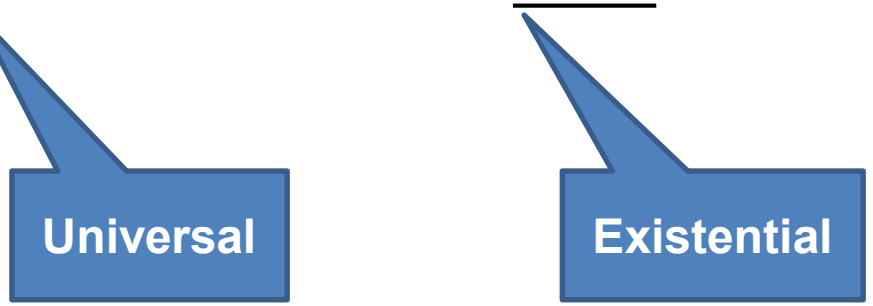
```
SELECT DISTINCT F.person
FROM Frequents F WHERE NOT EXISTS (
        SELECT *
        FROM Serves S, Likes L
        WHERE F.restaurant =S.restuarnt
           AND  F.person = L.person
           AND  S.food = L.food
)
```

Example 1

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 3

People that frequent only restaurants that serves some food they like.



$$Q(p) = Person(p) \wedge \forall r\,(F(p, r) \rightarrow \exists f\ (\ S(r, f) \wedge L(p, f))\ )$$

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 3

People that frequent only restaurants that serves some food they like.

$$Q(p) = Person(p) \wedge \forall r\,(F(p, r) \rightarrow \exists f\ (\ S(r, f) \wedge L(p, f))\ )$$

X frequents only restaurants that serve some food X likes

=

X does NOT frequent some restaurant that serves only food X doesn't like

$$Q(p) = Person(p) \wedge \neg\ \exists r\,(F(p, r) \wedge \neg\ \exists f\ (S(r, f) \wedge L(p, f))$$

Let's find the others (drop the NOT):

Person that frequent some restaurant that serves only food they don't like.

↳ Example 2

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 3

People that frequent only restaurants that serves some food they like.

Let's find the others (drop the NOT):

Person that frequent some restaurant that serves only food they don't like.

That's the previous query!

Example 2

```
SELECT DISTINCT F.person
FROM Frequents F WHERE NOT EXISTS (
        SELECT *
        FROM Serves S, Likes L
        WHERE F.restaurant =S.restuarnt
           AND  F.person = L.person
           AND  S.food = L.food
)
```

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 3

People that frequent only restaurants that serves some food they like.

Let's find the others (drop the NOT):

Person that frequent some restaurant that serves only food they don't like.

That's the previous query! But write it as a nested query:

```
SELECT DISTINCT U.person
FROM Frequents U
WHERE U.person IN
    (SELECT DISTINCT F.person
     FROM Frequents F WHERE NOT EXISTS (
          SELECT *
               FROM Serves S, Likes L
               WHERE F.restaurant =S.restuarnt
               AND  F.person = L.person
               AND  S.food = L.food
          )
    )
```

Likes(person, food)
Frequents(person, restaurant)
Serves(restaurant, food)

Example 3

People that frequent only restaurants that serves some food they like.

Let's find the others (drop the NOT):

Person that frequent some restaurant that serves only food they don't like.

That's the previous query! But write it as a nested query:

Now negate!

Now need three nested queries

```
SELECT DISTINCT U.person
FROM Frequents U
WHERE U.person NOT IN
    (SELECT DISTINCT F.person
     FROM Frequents F WHERE NOT EXISTS (
          SELECT *
               FROM Serves S, Likes L
               WHERE F.restaurant =S.restuarnt
               AND  F.person = L.person
               AND  S.food = L.food
          )
    )
```

Example 2

Example 1

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

Unnesting Aggregates

Find the number of companies in each city

```
SELECT DISTINCT X.city, (SELECT count(*)
                         FROM Company Y
                         WHERE X.city = Y.city)
FROM  Company X
```

```
SELECT city,  count(*)
FROM   Company
GROUP BY city
```

Note: no need for DISTINCT
(DISTINCT *is the same* as GROUP BY)

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

Unnesting Aggregates

Find the number of companies in each city

```
SELECT DISTINCT X.city, (SELECT count(*)
                         FROM Company Y
                         WHERE X.city = Y.city)
FROM  Company X
```

```
SELECT city,  count(*)
FROM   Company
GROUP BY city
```

Equivalent queries

Should be LEFT OUTER JOIN for to be equivalent

Purchase(pid, product, quantity, price)

Grouping vs 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
```

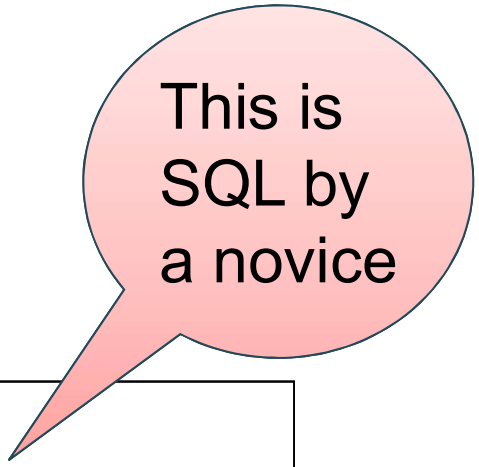
Why twice ?

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

More Unnesting

Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries




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

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

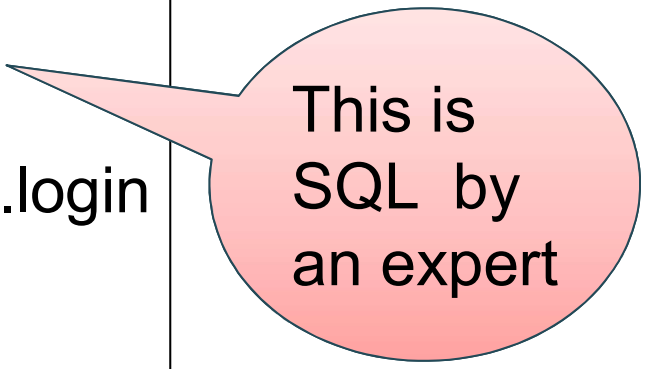
More Unnesting

Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

Attempt 2: using GROUP BY and HAVING

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



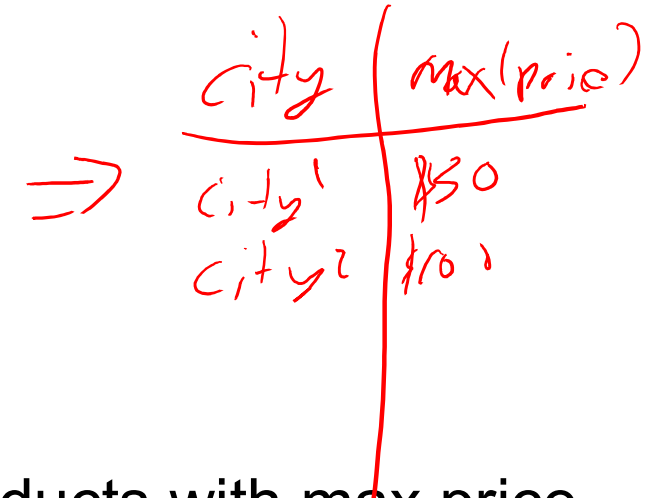
Product (pname, price, cid)
Company(cid, cname, city)

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

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);
```

Correlated subquery! Not good.

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

Finding Witnesses

Or can use a subquery in the FROM clause.
(Join on new table with max price per city)

```
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;
```

only x, y used

Subquery is in FROM can probably rewrite

Not a bad solution…

Not a correlated subquery. Why?

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

Finding Witnesses

There is a more concise solution here:

Idea: Product JOIN Product ON "made in the same city"
Then group by first product.
Then check that first product is more expensive than all of the second products in the group.

```
SELECT C1.city, P1.pname, P1.price
FROM Company C1, Product P1, Company C2, Product P2
WHERE C1.cid = P1.cid
      and C1.city = C2.city
      and C2.cid = P2.cid
GROUP BY C1.city, P1.pname, P1.price
HAVING P1.price = max(P2.price);
```

Why use SQLite?

All quarter we have talked about the limitations of SQLite

- No strict type definitions
- Allows attributes not in group by or aggregate function

So who uses SQLite? One use imbedded database.

Chrome History

- C:\Users\<username>\AppData\Local\Google\Chrome\<Profile>
- /Users/<username>/Library/Application Support/Google/Chrome/<Profile>
- /home/<username>/.config/google-chrome/<profile>

Firefox, Safari and Edge store data in similar locations

SQLite Browser

There are multiple tools for working with larger SQLite databases