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
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
(and SQLite simply ignores the extra values…)

“correlated subquery”
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
```

```
SELECT C.cname, count(pname) 
FROM Company C LEFT OUTER JOIN Product P 
ON C.cid=P.cid 
GROUP BY C.cname
```

No! Different results if a company has no products.
Simple Aggregations

Five basic aggregate operations in SQL

```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
```

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
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
<th>Final results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pname</strong></td>
<td><strong>product</strong></td>
<td><strong>manufacturer</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Camera</td>
<td>Camera</td>
<td>Canon</td>
</tr>
<tr>
<td>OneClick</td>
<td>OneClick</td>
<td>Hitachi</td>
</tr>
<tr>
<td>Gizmo</td>
<td></td>
<td>NULL</td>
</tr>
</tbody>
</table>

Why 0 for GizmoWorks? 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
```

<table>
<thead>
<tr>
<th>pname</th>
<th>manufacturer</th>
<th>...</th>
<th>product</th>
<th>price</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>GizmoWorks</td>
<td></td>
<td>Camera</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Canon</td>
<td></td>
<td>Camera</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>OneClick</td>
<td>Hitachi</td>
<td></td>
<td>OneClick</td>
<td>180</td>
<td></td>
</tr>
</tbody>
</table>

**Left Outer Join(Product, Purchase)**

<table>
<thead>
<tr>
<th>pname</th>
<th>manufacturer</th>
<th>...</th>
<th>product</th>
<th>price</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera</td>
<td>Canon</td>
<td></td>
<td>Camera</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Canon</td>
<td></td>
<td>Camera</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>OneClick</td>
<td>Hitachi</td>
<td></td>
<td>OneClick</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>Gizmo</td>
<td>GizmoWorks</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

**Final results**

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>Count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canon</td>
<td>2</td>
</tr>
<tr>
<td>Hitachi</td>
<td>1</td>
</tr>
<tr>
<td>GizmoWorks</td>
<td>1</td>
</tr>
</tbody>
</table>

Probably not what we want!
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

SELECT C.cname, count(pname)
FROM Company C LEFT OUTER JOIN Product P
ON C.cid=P.cid
GROUP BY C.cname

Recall: count of an empty table is 0

No! Different results if a company has no products
Find all products whose prices is > 20 and < 500

```sql
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!
2. Subqueries in FROM

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

A subquery whose result we called myTable
3. Subqueries in WHERE

Find all companies that make some products with price < 200
Find all companies that make some products with price < 200
Find all companies that make some products with price < 200

Using **EXISTS**:  

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE EXISTS (SELECT * 
               FROM Product P 
               WHERE C.cid = P.cid and P.price < 200)
```
Find all companies that make some products with price < 200

Using IN

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

Find all companies that make some products with price < 200

Using ANY:

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

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

3. Subqueries in WHERE

Find all companies that make some products with price < 200

Using ANY:

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

Existential quantifiers

Not supported in sqlite
Find all companies that make some products with price < 200
3. Subqueries in WHERE

Find all companies that make some products with price < 200

```
SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid = P.cid and P.price < 200
```

Existential quantifiers are easy!
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
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
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 are hard! 😞
3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

1. Find the other companies that make some product $\geq 200$

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

```sql
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

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid NOT IN (SELECT P.cid
FROM Product P
WHERE P.price >= 200)
```
3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

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)
```
3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Using **ALL**:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 >= ALL (SELECT price
                    FROM Product P
                    WHERE P.cid = C.cid)
```
3. Subqueries in WHERE

Find all companies s.t. **all** their products have price < 200

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*
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.
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.

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>price</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
</tr>
<tr>
<td>Gadget</td>
<td>999.99</td>
</tr>
<tr>
<td>Camera</td>
<td>149.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Lyon</td>
</tr>
<tr>
<td>Camera</td>
<td>Lodtz</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{pname}$</td>
<td>$\text{price}$</td>
</tr>
<tr>
<td>$\text{Gizmo}$</td>
<td>19.99</td>
</tr>
<tr>
<td>$\text{Gadget}$</td>
<td>999.99</td>
</tr>
<tr>
<td>$\text{Camera}$</td>
<td>149.99</td>
</tr>
<tr>
<td>$\text{iPad}$</td>
<td>499.99</td>
</tr>
<tr>
<td>$\text{cid}$</td>
<td>$\text{cname}$</td>
</tr>
<tr>
<td>c002</td>
<td>Sunworks</td>
</tr>
<tr>
<td>c001</td>
<td>DB Inc.</td>
</tr>
<tr>
<td>c003</td>
<td>Builder</td>
</tr>
<tr>
<td>$\text{pname}$</td>
<td>$\text{city}$</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Lyon</td>
</tr>
<tr>
<td>Camera</td>
<td>Lodtz</td>
</tr>
</tbody>
</table>

So far it looks monotone...
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
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.

```sql
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
```

```python
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)
```
Monotone Queries

• The query:

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

- The query:

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

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>c001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>c001</td>
<td>Sunworks</td>
<td>Bonn</td>
</tr>
</tbody>
</table>
Monotone Queries

• The query:

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

- 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
Finding Witnesses

For each city, find the most expensive product made in that 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.
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;
```
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;
```
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);
```
There is a more concise solution here:

```sql
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