ADMINISTRIVIA

• HW1 Due Tonight (11:30)
  • Don’t forget to git add and tag your assignment
  • Check on gitlab after submitting
  • Use AS for aliasing
• HW2 Out Tonight
  • Due next Wednesday (April 11)
  • Contains AWS instructions
• OQ1 Due Friday (11:00)
• OQ2 Out Due April 11
QUERY COMPLEXITY

• As the information we want gets more complex, we need to utilize more elements of the RDBMS
  • Multi-table queries -> join
  • Data statistics -> grouping

• Whatever you can do in SQL, you should
  • Optimization
  • Basic analysis tools
    • Sum, min, average, max, count
SEMANTICS OF SQL WITH GROUP-BY

Evaluation steps:

1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by the attributes $a_1, \ldots, a_k$
3. Apply condition $C_2$ to each group (may have aggregates)
4. Compute aggregates in $S$ and return the result

```
SELECT S
FROM $R_1, \ldots, R_n$
WHERE $C_1$
GROUP BY $a_1, \ldots, a_k$
HAVING $C_2$
```
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 to avoid, as we will see
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
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

Whenever possible, don’t use a nested queries:

\[
\begin{align*}
\text{SELECT} & \quad \text{X.pname, (SELECT Y.city FROM Company Y WHERE Y.cid=X.cid) as City} \\
\text{FROM} & \quad \text{Product X}
\end{align*}
\]

We have "unnested" the query
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
```
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
```
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
```
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
```
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
```
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 unnest this query!
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
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
```
3. SUBQUERIES IN WHERE

Find all companies that make some products with price < 200
3. SUBQUERIES IN WHERE

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

Using \textbf{EXISTS}:

\begin{verbatim}
SELECT DISTINCT C.cname
FROM Company C
WHERE EXISTS (SELECT *
              FROM Product P
              WHERE C.cid = P.cid and P.price < 200)
\end{verbatim}
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
FROM Product P
WHERE P.price < 200)

Using **IN**

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

**3. SUBQUERIES IN WHERE**

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

3. SUBQUERIES IN WHERE

Find all companies that make some products with price < 200

Using ANY:

\[
\begin{align*}
\text{SELECT} & \quad \text{DISTINCT} & \quad C.cname \\
\text{FROM} & \quad \text{Company} & \quad C \\
\text{WHERE} & \quad 200 > \text{ANY} & \quad (\text{SELECT} \quad \text{price} \\
& \quad \text{FROM} & \quad \text{Product} \quad P \\
& \quad \text{WHERE} & \quad P.cid = C.cid)
\end{align*}
\]
3. SUBQUERIES IN WHERE

Find all companies that make some products with price < 200

Now let’s unnest it:

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

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 are hard!
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)
```
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**:  

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

Not supported in sqlite
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></th>
<th></th>
<th></th>
<th>Company</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>price</td>
<td>cid</td>
<td></td>
<td>cid</td>
<td>cname</td>
<td>city</td>
<td></td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>c001</td>
<td></td>
<td>c002</td>
<td>Sunworks</td>
<td>Bonn</td>
<td></td>
</tr>
<tr>
<td>Gadget</td>
<td>999.99</td>
<td>c004</td>
<td></td>
<td>c001</td>
<td>DB Inc.</td>
<td>Lyon</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>149.99</td>
<td>c003</td>
<td></td>
<td>c003</td>
<td>Builder</td>
<td>Lodtz</td>
<td></td>
</tr>
</tbody>
</table>

\[ Q \]
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

<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>
<tr>
<td>Gadget</td>
<td>999.99</td>
<td>c004</td>
</tr>
<tr>
<td>Camera</td>
<td>149.99</td>
<td>c003</td>
</tr>
</tbody>
</table>

### Company

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>c002</td>
<td>Sunworks</td>
<td>Bonn</td>
</tr>
<tr>
<td>c001</td>
<td>DB Inc.</td>
<td>Lyon</td>
</tr>
<tr>
<td>c003</td>
<td>Builder</td>
<td>Lodtz</td>
</tr>
</tbody>
</table>

Q is not monotone!
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)
```
MONOTONE QUERIES

The query:

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

is not monotone
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

Consequence: If a query is not monotonic, then we cannot write it as a SELECT-FROM-WHERE query without nested subqueries
Purchase(pid, product, quantity, price)

**GROUP BY V.S.**

**NESTED QUERIES**

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

```sql
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:
MORE UNNESTING

Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

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

This is SQL by a novice
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 \((p\text{name}, \text{ price}, \text{ cid})\)  
Company \((\text{cid}, \text{ cname}, \text{ city})\)

**FINDING WITNESSES**

For each city, find the most expensive product made in that city.
Finding the maximum price is easy...

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

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