CSE 344

JANUARY 19\textsuperscript{TH} – SUBQUERIES 2 AND RELATIONAL ALGEBRA
ASSORTED MINUTIAE

• Winter storm Inga

• Online quiz out after class
  • Still due Wednesday, will be shorter but through today’s lecture

• For SQLite submissions, please use AS when aliasing
TODAY’S LECTURE

• Review of ordering
• Subqueries in FROM and WHERE
• Intro to relational algebra (maybe)
SEMANTICS OF SQL WITH GROUP-BY

Evaluation steps:

1. Evaluate FROM-WHERE using Nested Loop Semantic
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
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
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:

```
SELECT X.pname, (SELECT Y.city
    FROM Company Y
    WHERE Y.cid=X.cid) as City
FROM Product X
```

We have “unnested” the query

```
SELECT X.pname, Y.city
FROM Product X, Company Y
WHERE X.cid=Y.cid
```
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
```
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?

```sql
SELECT DISTINCT C.cname, (SELECT count(*)
   FROM Product P
   WHERE P.cid=C.cid)
FROM Company C
```

```sql
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?

\[
\text{SELECT DISTINCT C.cname, (SELECT count(*) FROM Product P WHERE P.cid=C.cid)}\]

FROM Company C

\[
\text{SELECT C.cname, count(*) FROM Company C, Product P WHERE C.cid=P.cid GROUP BY C.cname}
\]

No! Different results if a company has no products

\[
\text{SELECT C.cname, count(pname) FROM Company C LEFT OUTER JOIN Product P ON C.cid=P.cid GROUP BY C.cname}
\]
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
```
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 unnest this query!
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!

Side note: This is not a correlated subquery. (why?)
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
2. SUBQUERIES IN FROM

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

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
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 `EXISTS`:

```
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

```sql
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:

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

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

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!
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
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 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 ≥ 200

```sql
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 $\geq$ 200

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
FROM Product P
WHERE P.price $\geq$ 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 $\geq$ 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

**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.

\[
\begin{align*}
\text{SELECT} & \quad a_1, a_2, \ldots, a_k \\
\text{FROM} & \quad R_1 \text{ AS } x_1, R_2 \text{ AS } x_2, \ldots, R_n \text{ AS } x_n \\
\text{WHERE} & \quad \text{Conditions}
\end{align*}
\]

for \( x_1 \) in \( R_1 \) do
  for \( x_2 \) in \( R_2 \) do
    ... 
    for \( x_n \) in \( R_n \) do
      if Conditions
        output \( (a_1, \ldots, a_k) \)
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

<table>
<thead>
<tr>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunworks</td>
<td>Bonn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product (pname, price, cid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company (cid, cname, city)</td>
</tr>
</tbody>
</table>

<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></td>
</tr>
</tbody>
</table>
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

- \text{sum(..)} and \text{count(*)} are NOT monotone, because they do not satisfy set containment
- \text{select count(*) from R} is not monotone!