Lecture 5: Nested Queries in SQL
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 writing nested queries when possible; keep in mind that sometimes it’s impossible
Subqueries…

• Can return a single constant and this constant can be compared with another value in a WHERE clause
• Can return relations that can be used in various ways in WHERE clauses
• Can appear in FROM clauses, followed by a tuple variable that represents the tuples in the result of the subquery
• Can appear as computed values in a SELECT clause
1. Subqueries in SELECT

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

For each product return the city where it is manufactured
1. Subqueries in SELECT

Product (\texttt{pname, price, cid})
Company(\texttt{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
```
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
(SQLite simply ignores the extra values)
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
(SQLite simply ignores the extra values)
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
(SQLite simply ignores the extra values)
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
```
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
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 by 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
```

```
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
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
```

Unnest this query!
```
SELECT pname FROM Product
WHERE price > 20 and price < 500
```
2. Subqueries in FROM

At the end of the lecture we will see that sometimes we really need a subquery and one option will be to put it in the FROM clause.
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

Product (pname, price, cid)
Company(cid, cname, city)
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}
3. Subqueries in WHERE

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)
```
Product \((\text{pname, price, cid})\)
Company\((\text{cid, cname, city})\)

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)
```
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)

Not supported in sqlite
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Now let's unnest it:

```sql
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

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 😊
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 \((pname, \text{ price, cid})\)

Company\((\text{cid, cname, city})\)

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
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: i.e. s.t. 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: i.e. s.t. some product ≥ 200

\[
\text{SELECT DISTINCT C.cname}
\text{FROM Company C}
\text{WHERE C.cid IN (SELECT P.cid}
\text{\quad FROM Product P}
\text{\quad WHERE P.price >= 200)}
\]

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

\[
\text{SELECT DISTINCT C.cname}
\text{FROM Company C}
\text{WHERE C.cid NOT IN (SELECT P.cid}
\text{\quad FROM Product P}
\text{\quad WHERE P.price >= 200)}
\]

Product (pname, price, cid)
Company(cid, cname, city)
3. Subqueries in WHERE

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

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**: 

```sql
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)
```

**Universal quantifiers**

Not supported in sqlite
Question for Database Fans and their Friends

- Can we unnest the *universal quantifier* query?
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.
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
```

```plaintext
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>

<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>

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadget</td>
<td>999.99</td>
<td>c001</td>
</tr>
</tbody>
</table>

Consequence: we cannot write it as a SELECT-FROM-WHERE query without nested subqueries
Queries that must be nested

- Queries with universal quantifiers
Where We Are

• Motivation for using a DBMS for managing data

• SQL, SQL, SQL
  – Declaring the schema for our data (CREATE TABLE)
  – Inserting data one row at a time or in bulk (INSERT/.import)
  – Modifying the schema and updating the data (ALTER/UPDATE)
  – Querying the data (SELECT)
  – Practice on SQLite

• Next step: More knowledge of how DBMSs work
  – Relational algebra, query execution, and physical tuning