Announcements

- WQ2 is due on Sunday 11pm
  - no late days
- HW2 is due on Tuesday 11pm

Lecture Goals

- Today we will learn how to write (even) more powerful SQL queries
- Reading: Ch. 6.3

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; keep in mind that sometimes it’s impossible
  - (though use in FROM is often not as bad)

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

\[
\text{SELECT X.pname, (SELECT Y.city FROM Company Y WHERE Y.cid = X.cid) as City FROM Product X}
\]

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

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
```

No! Different results if a company has no products

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

2. 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
```

But are these really equivalent?

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

No! Different results if a company has no products

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

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
```

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
```

- We will see that sometimes we really need a subquery
- We will see most compelling examples next lecture
- In that case, we can put it in the FROM clause
3. Subqueries in WHERE

Find all companies that make some products with price < 100

Using EXISTS:

\[
\begin{align*}
\text{SELECT DISTINCT } & \text{C.cname} \\
\text{FROM } & \text{Company C} \\
\text{WHERE } & \exists \left( \text{SELECT * FROM Product P WHERE C.cid = P.cid and P.price < 100} \right)
\end{align*}
\]

Existential quantifiers

Using IN:

\[
\begin{align*}
\text{SELECT DISTINCT } & \text{C.cname} \\
\text{FROM } & \text{Company C} \\
\text{WHERE } & \text{C.cid IN (SELECT P.cid FROM Product P WHERE P.price < 100)}
\end{align*}
\]

Existential quantifiers

Now let’s unnest it:

\[
\begin{align*}
\text{SELECT DISTINCT } & \text{C.cname} \\
\text{FROM } & \text{Company C, Product P} \\
\text{WHERE } & \text{C.cid = P.cid and P.price < 100}
\end{align*}
\]

Existential quantifiers are easy!

Universal quantifiers are hard!

Universal quantifiers are not supported in sqlite.
3. Subqueries in WHERE
Find all companies where all their products have price < 100

Using EXISTS:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS (SELECT *
    FROM Product P
    WHERE P.cid = C.cid and P.price > 100)
```

Using ALL:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 100 >= ALL (SELECT price
    FROM Product P
    WHERE P.cid = C.cid)
```

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

- 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

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

(That is, cannot be CFV queries)

• Queries with universal quantifiers or negation

• Queries that use aggregates in certain ways
  – Note: sum(…) etc. are NOT monotone
  – select count(*) from R is not monotone!