CSE 344

JUNE 27TH

SUBQUERIES
ADMINISTRIVIA

- HW1 Due Tonight (11pm)
  - Make sure the grading script passes
  - Run the turn-in script to submit & tag
  - Check on gitlab after submitting
- WQ2 Due Friday
- HW2 Out Tomorrow
  - Due next Wednesday (April 11)
SEMANTICS OF SQL WITH GROUP-BY

```
SELECT S
FROM R_1,...,R_n
WHERE C1
GROUP BY a_1,...,a_k
HAVING C2
```

Evaluation steps:
1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group rows with same values in the attributes a_1,...,a_k
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result
REVIEW

- **SQL**
  - inner & outer joins (FROM & WHERE clauses)
  - group by
    - subsequent processing applies to groups not rows
    - can only use group-by columns and aggregation
      - count, sum, average, min, max
  - having filter on groups (vs where on rows)
  - order by
  - select is processed **last**
  - (almost done with SQL now…)
What do they compute?

**MYSTERY QUERY**

```
SELECT  month, sum(quantity), max(price)
FROM    Purchase
GROUP BY month
```

```
SELECT  month, sum(quantity)
FROM    Purchase
GROUP BY month
```

```
SELECT  month
FROM    Purchase
GROUP BY month
```
Purchase(pid, product, price, quantity, month)

**MYSTERY QUERY**

What do they compute?

```sql
SELECT month, sum(quantity), max(price)
FROM Purchase
GROUP BY month
```

```sql
SELECT month, sum(quantity)
FROM Purchase
GROUP BY month
```

```sql
SELECT month
FROM Purchase
GROUP BY month
```

Lesson: DISTINCT is a special case of GROUP BY
Product(product_id,pname,manufacturer)
Purchase(pid,product_id,price,month)

**AGGREGATE + JOIN**

For each manufacturer, compute how many products with price > $100 they sold
AGGREGATE + JOIN

For each manufacturer, compute how many products with price > $100 they sold

Problem: manufacturer is in Purchase, price is in Product...
Product(product_id, pname, manufacturer)
Purchase(pid, product_id, price, month)

AGGREGATE + JOIN

For each manufacturer, compute how many products with price > $100 they sold

Problem: manufacturer is in Purchase, price is in Product...

-- step 1: think about their join

```
SELECT ...
FROM Product x, Purchase y
WHERE x.product_id = y.product_id
  and y.price > 100
```

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hitachi</td>
<td>150</td>
</tr>
<tr>
<td>Canon</td>
<td>300</td>
</tr>
<tr>
<td>Hitachi</td>
<td>180</td>
</tr>
</tbody>
</table>
Product(`product_id`, `pname`, `manufacturer`)
Purchase(`pid`, `product_id`, `price`, `month`)

**AGGREGATE + JOIN**

For each manufacturer, compute how many products with price > $100 they sold

Problem: manufacturer is in Purchase, price is in Product...

---

**Step 1: Think about their join**

```
SELECT ...
FROM Product x, Purchase y
WHERE x.product_id = y.product_id and y.price > 100
```

---

**Step 2: Do the group-by on the join**

```
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.product_id = y.product_id and y.price > 100
GROUP BY x.manufacturer
```
AGGREGATE + JOIN

Variant:
For each manufacturer, compute how many products with price > $100 they sold in each month

```
SELECT x.manufacturer, y.month, count(*)
FROM Product x, Purchase y
WHERE x.product_id = y.product_id
    and y.price > 100
GROUP BY x.manufacturer, y.month
```
INCLUDING EMPTY GROUPS

In the result of a group by query, there is one row per group in the result

```
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.product_id = y.product_id
GROUP BY x.manufacturer
```

Count(*) is never 0
INCLUDING EMPTY GROUPS

SELECT x.manufacturer, count(y.pid)
FROM Product x LEFT OUTER JOIN Purchase y
ON x.product_id = y.product_id
GROUP BY x.manufacturer

Count(pid) is 0 when all pids in the group are NULL
A subquery is a SQL query nested inside another query
- inner query is also called a “nested query”

A subquery may occur in:
- SELECT clause
- FROM clause
- WHERE clause

Rule of thumb: avoid nested queries when possible
- But sometimes it’s impossible to avoid, as we will see
- (And those in the FROM clause are not usually a problem)
SUBQUERIES...

- Can return a single value to be included in a SELECT clause
  - query must return relation with 1 row & 1 column
- 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
```
1. SUBQUERY 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
```

Better: we can unnest with 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?

```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
```

No! Different results if a company has no products

```sql
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 to unnest this query!
Product (pname, price, cid)
Company (cid, cname, city)

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
2. SUBQUERIES IN FROM

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

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
3. SUBQUERIES IN WHERE

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

3. SUBQUERIES IN WHERE

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 \textbf{ANY}:

\begin{verbatim}
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 > ANY (SELECT price
                    FROM Product P
                    WHERE P.cid = C.cid)
\end{verbatim}
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

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

Not supported in sqlite
Product (pname, price, cid)
Company (cid, cname, city)

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

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

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

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

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

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

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

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.

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

**Product Company**

<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>

**Q is not monotone!**

<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>
<tr>
<td>c004</td>
<td>Crafter</td>
<td>Lodtz</td>
</tr>
</tbody>
</table>

**Q**

<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>
<tr>
<td>iPad</td>
<td>Lyon</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.
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.
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>
Product \((pname, \ price, \ cid)\)  
Company \((cid, \ cname, \ city)\)

**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>
<tr>
<td>Gadget</td>
<td>999.99</td>
<td>c001</td>
</tr>
</tbody>
</table>

**Consequence:** If a query is not monotonic, then we cannot write it as a SELECT-FROM-WHERE query without grouping or nested subqueries
GROUP BY V.S.
NESTED QUERIES

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

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:
Author(login,name)  
Wrote(login,url)

MORE UNNESTING

Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

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

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
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;
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)
```