Announcements

• Web Quiz 2 due Friday night

• HW 2 due Tuesday at midnight

• Section this week important for HW 3, must attend
Announcements

• Many students did not turn in hw1 correctly – need to make sure your files are here:

https://gitlab.cs.washington.edu/cse414-2018au/cse414-
[username]/tree/master/hw/hw[homework#]/submission

E.g. https://gitlab.cs.washington.edu/cse414-2018au/cse414-
maas/tree/master/hw/hw1/submission

AND you have the hw1 tag here:


• Commit, then use ./turnInHw.sh hw2 script.
• MUST have this correct for HW2
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 C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result
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 Product, price is in Purchase...
Aggregate + Join

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

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

\[
\text{SELECT} \quad ... \\
\text{FROM} \quad \text{Product} \ x, \ \text{Purchase} \ y \\
\text{WHERE} \quad x.\text{pid} = y.\text{product_id} \\
\quad \text{and} \quad y.\text{price} > 100
\]
Product(pid,pname,manufacturer)
Purchase(id,product_id,price,month)

Aggregate + Join

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

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

-- step 1: think about their join
SELECT ...
FROM Product x, Purchase y
WHERE x.pid = 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.pid = y.product_id
    and y.price > 100
GROUP BY x.manufacturer

<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>

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hitachi</td>
<td>2</td>
</tr>
<tr>
<td>Canon</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>9</td>
</tr>
</tbody>
</table>
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.pid = y.product_id
    and y.price > 100
GROUP BY x.manufacturer, y.month
```

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>month</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hitachi</td>
<td>Jan</td>
<td>2</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Feb</td>
<td>1</td>
</tr>
<tr>
<td>Canon</td>
<td>Jan</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>
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.pname = y.product
GROUP BY x.manufacturer
```

Count(*) is not 0 because there are no tuples to count!
Including Empty Groups

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

**Final results**

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>Count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canon</td>
<td>2</td>
</tr>
<tr>
<td>Hitachi</td>
<td>1</td>
</tr>
</tbody>
</table>

No GizmoWorks!
Including Empty Groups

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

Count(pid) is 0 when all pid’s in the group are NULL
Including Empty Groups

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

Final results

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>Count(y.pid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canon</td>
<td>2</td>
</tr>
<tr>
<td>Hitachi</td>
<td>1</td>
</tr>
<tr>
<td>GizmoWorks</td>
<td>0</td>
</tr>
</tbody>
</table>

Why 0 for GizmoWorks?

GizmoWorks is paired with NULLs
Including Empty Groups

```
SELECT x.manufacturer, count(*)
FROM Product x LEFT OUTER JOIN Purchase y
ON x.pname = y.product
GROUP BY x.manufacturer
```

Final results

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>Count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canon</td>
<td>2</td>
</tr>
<tr>
<td>Hitachi</td>
<td>1</td>
</tr>
<tr>
<td>GizmoWorks</td>
<td>1</td>
</tr>
</tbody>
</table>

Probably not what we want!
What we have in our SQL toolbox

• Projections (SELECT * / SELECT c1, c2, …)
• Selections (aka filtering) (WHERE cond, HAVING)
• Joins (inner and outer)
• Aggregates
• Group by
• Inserts, updates, and deletes

Make sure you read the textbook!
Subqueries

• In the relational model, the output of a query is also a relation

• Can use output of one query as input to another
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
  - 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
Subqueries…

Subqueries are often:
• Intuitive to write
• Slow

Be careful!
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:

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

We have “unnested” the query:

```sql
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
```

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

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

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
```
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 to 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!
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)

Existential quantifiers
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Using `EXISTS`:

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE EXISTS (SELECT *
               FROM 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

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 **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)
```

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

3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers are easy!

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, 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

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 \( \geq 200 \)

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
                 FROM Product P
                 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

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)
```

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

```
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 \textit{EXISTS}:

\begin{verbatim}
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS (SELECT *
    FROM Product P
    WHERE P.cid = C.cid and P.price >= 200)
\end{verbatim}
Product (pname, price, cid)
Company (cid, cname, city)

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**:

```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*
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

- **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>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pname</strong></td>
<td><strong>cid</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>c001</td>
</tr>
<tr>
<td>Gadget</td>
<td>c004</td>
</tr>
<tr>
<td>Camera</td>
<td>c003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pname</strong></td>
<td><strong>city</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>Lyon</td>
</tr>
<tr>
<td>Camera</td>
<td>Lodtz</td>
</tr>
</tbody>
</table>
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.

---

**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>

**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>

**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>
<tr>
<td>iPad</td>
<td>499.99</td>
<td>c001</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>

**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.
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>

CSE 414 - Autumn 2018
Monotone Queries

• The query:

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

• 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 $\text{count(\ast)}$ are NOT monotone, because they do not satisfy set containment
  - $\text{select count(\ast) from R}$ is not monotone!
More Unnesting

Find authors who wrote $\geq 10$ 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
```
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)

```sql
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:

```sql
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)
```