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/hw1/submission
  E.g. https://gitlab.cs.washington.edu/cse414-2018au/cse414-
  [username]/tree/master/hw/hw1/submission
  AND you have the hw1 tag here: https://gitlab.cs.washington.edu/cse414-2018au/cse414-
  [username]/tags
  - Commit, then use ./turninHw.sh hw2 script.
  - MUST have this correct for HW2

Evaluation steps:
1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by the attributes $a_1$,...,$a_k$
3. Apply condition $C_2$ 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

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...

---

step 1: think about their join

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

```sql
<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>
```

---

step 2: do the group-by on the join

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

```sql
<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>
</tbody>
</table>
```

Including Empty Groups

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

```sql
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

```sql
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

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
```

```
<table>
<thead>
<tr>
<th>product</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera</td>
<td>150</td>
</tr>
<tr>
<td>Camera</td>
<td>300</td>
</tr>
<tr>
<td>OneClick</td>
<td>180</td>
</tr>
</tbody>
</table>
```

Why 0 for GizmoWorks?

Final results

```
<table>
<thead>
<tr>
<th>manufacturer</th>
<th>Count(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>
```

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>
```

Subqueries

- In the relational model, the output of a query is also a relation
- Can use output of one query as input to another

```
<table>
<thead>
<tr>
<th>manufacture</th>
<th>product</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canon</td>
<td>Camera</td>
<td>150</td>
</tr>
<tr>
<td>Canon</td>
<td>Camera</td>
<td>300</td>
</tr>
<tr>
<td>Hitachi</td>
<td>OneClick</td>
<td>180</td>
</tr>
</tbody>
</table>
```

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

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...
Subqueries are often:
• Intuitive to write
• Slow
Be careful!

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

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
GROUP BY C.cname
```

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
```

But are these really equivalent?

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

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

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
```

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

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
```

Try to unnest this query!

Side note: This is not a correlated subquery. (why?)

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

3. Subqueries in WHERE

Find all companies that make some products with price < 200

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

Existential quantifiers

Product (pname, price, cid)
Company (cid, cname, city)
Find all companies that make some products with price < 200

Using \textit{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}

Using \textit{IN}:

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

Using \textit{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}

Using \textit{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}

Now let’s unnest it:

\begin{verbatim}
SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid = P.cid and P.price < 200
\end{verbatim}
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! 😞

Using EXISTS:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS (SELECT * FROM Product P
WHERE P.price >= 200)
```

Universal quantifiers
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**

**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

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<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>

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<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>

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<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>

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>iPad</td>
<td>499.99</td>
</tr>
</tbody>
</table>

**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.

### 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
```

For $x_i$ in $R_i$ do
- For $x_j$ in $R_j$ do
  - For conditions output $(a_1, ..., a_k)$

Monotone Queries

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

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

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

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

```
SELECT x.city, max(y.price) as maxprice
FROM Company x, Product y
WHERE x.cid = y.cid
GROUP BY x.city;
```

Joining three tables

Finding Witnesses

To find the witnesses, compute the maximum price in a subquery (in FROM)

```
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;
```

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 v.price = max(y.price)
```

Joining three tables