Introduction to Data Management
CSE 344

Lecture 7: SQL Wrap-up
Relational Algebra
Announcements

• Webquiz 3 is open, due next Tuesday

• Homework 3 has been posted, due on Wednesday, 2/1
  – Azure can take significant time to set up
  – Don’t wait until the last minute to start
  – We support Windows
Using Electronics in Class

- Opened laptops create disturbances to your neighbors.
- Please sit in the back if you use your laptop to take notes.
- OK if you use surfaces.

- And please don’t check your email / sms / youtube / fb / etc during class.
  - If people are doing this we will have to ban all laptops 😞.
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 $C_2$ to each group (may have aggregates)
4. Compute aggregates in $S$ and return the result

```
SELECT S
FROM $R_1, \ldots, R_n$
WHERE $C_1$
GROUP BY $a_1, \ldots, a_k$
HAVING $C_2$
```
What We Learned Yesterday

• Subqueries can occur in every clause:
  – SELECT
  – FROM
  – WHERE
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

Universal quantifiers are hard! 😞
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)
```
Product (pname, price, cid)
Company (cid, cname, city)

3. Subqueries in WHERE

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

Using \textbf{EXISTS}:

\begin{verbatim}
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS (SELECT *
FROM Product P
WHERE P.cid = C.cid \text{ and } P.price \geq 200)
\end{verbatim}
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 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></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>price</td>
<td>cid</td>
</tr>
<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>

<table>
<thead>
<tr>
<th>Company</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cid</td>
<td>cname</td>
<td>city</td>
</tr>
<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>

$Q$
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 $a_1, a_2, \ldots, a_k$
FROM $R_1$ AS $x_1$, $R_2$ AS $x_2$, \ldots, $R_n$ AS $x_n$
WHERE Conditions
```

```sql
for $x_1$ in $R_1$ do
  for $x_2$ in $R_2$ do
    \ldots
    for $x_n$ in $R_n$ do
      if Conditions
        output $(a_1, \ldots, a_k)$
  
```
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>
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>

<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>

- Consequence: 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
  – \texttt{sum(..)} and \texttt{count(*)} are NOT monotone, because they do not satisfy set containment
  – \texttt{select count(*) from R} is not monotone!
Author(login, name)
Wrote(login, url)

More Unnesting

Find authors who wrote ≥ 10 documents:
More Unnesting

Find authors who wrote $\geq 10$ 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
```
Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

Attempt 2: using GROUP BY and HAVING

\[
\begin{align*}
\text{SELECT} & \quad \text{Author.name} \\
\text{FROM} & \quad \text{Author, Wrote} \\
\text{WHERE} & \quad \text{Author.login} = \text{Wrote.login} \\
\text{GROUP BY} & \quad \text{Author.name} \\
\text{HAVING} & \quad \text{count(wrote.url)} \geq 10
\end{align*}
\]
In-class Exercise

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.

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

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)
```
Where We Are

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

• Next step: More knowledge of how DBMSs work
  – Relational algebra, query execution, and physical tuning
  – Client-server architecture
The Relational Model

• **Instance**
  – Organized as “table” or “relation”

• **Schema**
  – tables and columns / relations and attributes

• **Query languages**
  – SQL
    – Relational algebra (RA)

• **We will learn RA by studying the internals of DBMS**
Query Evaluation Steps

1. **SQL query**
   - **Parse & Check Query**
     - Translate query string into internal representation
     - Check syntax, access control, table names, etc.
   - **Decide how best to answer query: query optimization**
     - Logical plan → physical plan
     - Relational Algebra
   - **Query Execution**
   - **Return Results**

CSE 344 - Winter 2017
The WHAT and the HOW

• SQL = WHAT we want to get from the data

• Relational Algebra = HOW to get the data we want

• The passage from WHAT to HOW is called query optimization
  – SQL $\rightarrow$ Relational Algebra $\rightarrow$ Physical Plan
  – Relational Algebra = Logical Plan
Overview: SQL = WHAT

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid
and x.price > 100 and z.city = 'Seattle'
```

It’s clear WHAT we want, unclear HOW to get it
Relation Algebra

- Relations and attributes
- Functions that are applied to relations
  - Return relations
  - Can be composed together
  - Often displayed using a tree rather than linearly
  - Uses Greek symbols: $\sigma$, $\pi$, $\delta$, etc
- Language for describing query plans
Overview: Relational Algebra = HOW

```sql
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid
and y.cid = z.cid
and x.price > 100
and z.city = 'Seattle'
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid
and y.cid = z.cid
and x.price > 100
and z.city = 'Seattle'
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid
and y.cid = z.cid
and x.price > 100
and z.city = ‘Seattle’
Overview: Relational Algebra = HOW

SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid
and y.cid = z.cid
and x.price > 100
and z.city = 'Seattle'
Overview: Relational Algebra = HOW

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid
    and y.cid = z.cid
    and x.price > 100
    and z.city = 'Seattle'
```
Overview: Relational Algebra = HOW

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid
    and y.cid = z.cid
    and x.price > 100
    and z.city = 'Seattle'
```

Execution order is now clearly specified

Final answer

T4(name,name)

T3(...)
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid
and y.cid = z.cid
and x.price > 100
and z.city = 'Seattle'

Final answer

Logical plan
Many physical details are still left open!

Execution order is now clearly specified
Overview: Relational Algebra = HOW

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid
and y.cid = z.cid
and x.price > 100
and z.city = 'Seattle'
```

**Logical plan**
Many physical details are still left open!

**Physical plan**
Concrete algorithm for each operator

Execution order is now clearly specified