Introduction to Data Management
CSE 344

Lectures 10: Relational Algebra
Announcements

• Webquiz 4 is out

• HW3 is out, due next Monday

• HW4 will be out tomorrow

• Midterm in 2 weeks (Monday 2/9, in class)
Where We Are

• Motivation for using a DBMS for managing data
• SQL, SQL, SQL
  – Declaring the schema for our data (CREATE TABLE)
  – Inserting data one row at a time or in bulk (INSERT/.import)
  – Modifying the schema and updating the data (ALTER/UPDATE)
  – Querying the data (SELECT)
  – Tuning queries (CREATE INDEX)

• Next step: More knowledge of how DBMSs work
  – Client-server architecture
  – Relational algebra and query execution
Query Evaluation Steps

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

CSE 344 - Winter 2015
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
Sets v.s. Bags

• Sets: \{a,b,c\}, \{a,d,e,f\}, \{\}\, \ldots
• Bags: \{a, a, b, c\}, \{b, b, b, b, b\}, \ldots

Relational Algebra has two semantics:
• Set semantics = standard Relational Algebra
• Bag semantics = extended Relational Algebra

DB systems implement bag semantics (Why?)
Relational Algebra Operators

- Union $\cup$, intersection $\cap$, difference $-$
- Selection $\sigma$
- Projection $\Pi$
- Cartesian product $\times$, join $\Join$
- Rename $\rho$
- Duplicate elimination $\delta$
- Grouping and aggregation $\gamma$
- Sorting $\tau$
What about Intersection?

- Derived operator using minus
  \[ R_1 \cap R_2 = R_1 - (R_1 - R_2) \]
- Derived using join (will explain later)
  \[ R_1 \cap R_2 = R_1 \bowtie R_2 \]
Natural Join

\[
R_1 \bowtie R_2
\]

• Meaning: \( R_1 \bowtie R_2 = \Pi_A(\sigma(R_1 \times R_2)) \)

• Where:
  – Selection \( \sigma \) checks equality of all common attributes
  – Projection eliminates duplicate all common attributes
Theta Join

- A join that involves a predicate

\[ R_1 \bowtie_\theta R_2 = \sigma_\theta (R_1 \times R_2) \]

- Here \( \theta \) can be any condition
- For our voters/patients example:

\[ P \bowtie P.zip = V.zip \text{ and } P.age > V.age \]

SELECT *
FROM P, V
WHERE P.zip = V.zip AND
    P.age > V.age
Equijoin

- A theta join where $\theta$ is an equality

$$R1 \bowtie_{A=B} R2 = \pi_A(\sigma_{A=B} (R1 \times R2))$$

- This is by far the most used variant of join in practice
### Equijoin Example

**AnonPatient** P

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
</tbody>
</table>

**Voters** V

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

\[
P \Join_{P\.age=V\.age} V
\]

<table>
<thead>
<tr>
<th>age</th>
<th>P.zip</th>
<th>disease</th>
<th>name</th>
<th>V.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>p1</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>p2</td>
<td>98120</td>
</tr>
</tbody>
</table>
Join Summary

• **Theta-join**: \( R \bowtie_{\theta} S = \sigma_{\theta}(R \times S) \)
  - Join of \( R \) and \( S \) with a join condition \( \theta \)
  - Cross-product followed by selection \( \theta \)

• **Equijoin**: \( R \bowtie_{\theta} S = \pi_A (\sigma_{\theta}(R \times S)) \)
  - Join condition \( \theta \) consists only of equalities
  - Projection \( \pi_A \) drops all redundant attributes

• **Natural join**: \( R \bowtie S = \pi_A (\sigma_{\theta}(R \times S)) \)
  - Equijoin
  - Equality on **all** fields with same name in \( R \) and in \( S \)
  - Projection \( \pi_A \) drops all redundant attributes
So Which Join Is It?

- When we write $R \bowtie S$, we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context.
More Joins

• **Outer join**
  – Include tuples with no matches in the output
  – Use NULL values for missing attributes
  – Does not eliminate duplicate columns

• **Variants**
  – Left outer join
  – Right outer join
  – Full outer join
## Outer Join Example

### AnonPatient P

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
</tr>
</tbody>
</table>

### AnonJob J

<table>
<thead>
<tr>
<th>job</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>lawyer</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>cashier</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>disease</th>
<th>job</th>
<th>J.age</th>
<th>J.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>lawyer</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
<td>20</td>
<td>98120</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
<td>null</td>
<td>33</td>
<td>98120</td>
</tr>
</tbody>
</table>
Some Examples

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,qty,price)

Name of supplier of parts with size greater than 10
\( \pi_{sname} \left( \text{Supplier} \Join \text{Supply} \Join \left( \sigma_{psize>10} \ (\text{Part}) \right) \right) \)

Name of supplier of red parts or parts with size greater than 10
\( \pi_{sname} \left( \text{Supplier} \Join \text{Supply} \Join \left( \sigma_{psize>10} \ (\text{Part}) \cup \sigma_{pcolor='red'} \ (\text{Part}) \right) \right) \)
From SQL to RA
Query Evaluation Steps

1. **Parse & Check Query**
   - SQL query
   - Check syntax, access control, table names, etc.
   - Translate query string into internal representation
   - Logical plan → physical plan

2. **Decide how best to answer query: query optimization**

3. **Query Execution**

4. **Return Results**

---

CSE 344 - Winter 2015
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
    x.price > 100 and z.city = 'Seattle'
From SQL to RA

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

\[\text{SELECT DISTINCT } x.\text{name}, z.\text{name} \]
\[\text{FROM Product } x, \text{ Purchase } y, \text{ Customer } z\]
\[\text{WHERE } x.\text{pid} = y.\text{pid} \text{ and } y.\text{cid} = z.\text{cid} \text{ and}\]
\[x.\text{price} > 100 \text{ and}\]
\[z.\text{city} = 'Seattle'\]
An Equivalent Expression

Query optimization = finding cheaper, equivalent expressions

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’
Extended RA: Operators on Bags

- Duplicate elimination $\delta$
- Grouping $\gamma$
- Sorting $\tau$
Logical Query Plan

SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100

T1, T2, T3 = temporary tables

T1(city, p, c)
T2(city, p, c)
T3(city, c)

sales(product, city, price)
Typical Plan for Block (1/2)

\[
\begin{align*}
\text{SELECT-PROJECT-JOIN} \\
\text{Query}
\end{align*}
\]
Typical Plan For Block (2/2)

\[ \pi \text{fields, sum/count/min/max(fields)} \]
\[ \gamma \text{having condition} \]
\[ \sigma \text{selection condition} \]
\[ \text{join condition} \]

\[ \cdots \]
How about Subqueries?

```
SELECT  Q.sno
FROM    Supplier Q
WHERE   Q.sstate = 'WA'
        and not exists
        (SELECT *
         FROM Supply P
         WHERE P.sno = Q.sno
         and P.price > 100)
```
How about Subqueries?

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
    and not exists
    (SELECT *
     FROM Supply P
     WHERE P.sno = Q.sno
     and P.price > 100)
```
How about Subqueries?

**De-Correlation**

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
  and not exists
    (SELECT *
     FROM Supply P
     WHERE P.sno = Q.sno
     and P.price > 100)
```

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
  and Q.sno not in
    (SELECT P.sno
     FROM Supply P
     WHERE P.price > 100)
```
How about Subqueries?

```sql
(SELECT Q.sno
 FROM Supplier Q
 WHERE Q.sstate = 'WA')
 EXCEPT
(SELECT P.sno
 FROM Supply P
 WHERE P.price > 100)
 EXCEPT = set difference

SELECT Q.sno
 FROM Supplier Q
 WHERE Q.sstate = 'WA'
 and Q.sno not in
(SELECT P.sno
 FROM Supply P
 WHERE P.price > 100)
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
How about Subqueries?

(\text{SELECT} \ Q.\text{sno} \\
\text{FROM} \ \text{Supplier} \ Q \\
\text{WHERE} \ Q.\text{sstate} = \text{‘WA’}) \\
\text{EXCEPT} \\
(\text{SELECT} \ P.\text{sno} \\
\text{FROM} \ \text{Supply} \ P \\
\text{WHERE} \ P.\text{price} > 100)