Database Systems
CSE 414

Lectures 14: Relational Algebra (part 2) and Query Evaluation (Ch. 5.2 & 16.3 (skim 16.3.2))
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**: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - Join condition $\theta$ consists only of equalities

• **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>

### Result of Join P $\bowtie$ J

<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>null</td>
<td>null</td>
</tr>
</tbody>
</table>
More Examples

\[
\text{Supplier}(sno, \text{sname, scity, sstate})
\]
\[
\text{Part}(pno, \text{pname, psize, pcolor})
\]
\[
\text{Supply}(sno, pno, qty, price)
\]

Name of supplier of parts with size greater than 10
\[
\pi_{\text{sname}}(\text{Supplier } \bowtie \text{Supply } \bowtie(\sigma_{\text{psize}>10} (\text{Part}))
\]

Name of supplier of red parts or parts with size greater than 10
\[
\pi_{\text{sname}}(\text{Supplier } \bowtie \text{Supply } \bowtie(\sigma_{\text{psize}>10} (\text{Part}) \cup \sigma_{\text{pcolor}='red'} (\text{Part}) ))
\]
Query Evaluation Steps

1. Parse & Check Query
   - Translate query string into internal representation
   - Check syntax, access control, table names, etc.

2. Decide how best to answer query: query optimization
   - Logical plan → physical plan

3. Query Execution
   - Return Results

SQL query

CSE 414 - Fall 2017
From SQL to RA

```
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’

Can you think of another plan?

Push selections down the query plan!

Query optimization: find an equivalent optimal plan

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)
Extended RA: Operators on Bags

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

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

$\text{T1(city, p, c)}$

$\text{T2(city, p, c)}$

$\text{T3(city, c)}$

$\gamma_{\text{city, sum(price)} \rightarrow p, \text{count(*)} \rightarrow c}$

T1, T2, T3 = temporary tables

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

SELECT fields
FROM R, S, ...
WHERE condition

SELECT-PROJECT-JOIN
Query
Typical Plan for Block (2/2)

\[ \pi_{\text{fields}} \]
\[ \sigma_{\text{having condition}} \]
\[ \gamma_{\text{fields, sum/count/min/max(fields)}} \]
\[ \sigma_{\text{where condition}} \]
join condition

\[ \text{SELECT fields} \]
FROM R, S, ...
WHERE condition
GROUP BY fields
HAVING condition
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)
```

```
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)
```

De-Correlation

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} \\
(\(\text{SELECT} \ P.\text{sno} \\
\text{FROM} \ \text{Supply} \ P \\
\text{WHERE} \ P.\text{price} > 100\)) \\
\text{EXCEPT} = \text{set difference}

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

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

\[
\begin{align*}
\text{(SELECT } & Q.\text{sno} \\
\text{FROM } & \text{Supplier Q} \\
\text{WHERE } & Q.\text{sstate} = 'WA') \\
\text{EXCEPT} & \\
\text{(SELECT } & P.\text{sno} \\
\text{FROM } & \text{Supply P} \\
\text{WHERE } & P.\text{price} > 100)\end{align*}
\]

Finally…

\[
\begin{align*}
\pi_{\text{sno}} & - \\
\sigma_{\text{sstate}='WA'} & \\
\pi_{\text{sno}} & \\
\sigma_{\text{Price > 100}} & \\
\text{Supplier} & \text{Supply}
\end{align*}
\]
From Logical Plans to Physical Plans
Physical Operators

Each of the logical operators may have one or more implementations = physical operators

Will discuss several basic physical operators, with a focus on join
Main Memory Algorithms

Logical operator:

\[ \text{Product}(\text{pid, name, price}) \Join_{\text{pid} = \text{pid}} \text{Purchase}(\text{pid, cid, store}) \]

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join \( O(??) \)
2. Merge join \( O(??) \)
3. Hash join \( O(??) \)

(note that pid is a key)
Main Memory Algorithms

Logical operator:

\[
\text{Product}(\text{pid}, \text{name}, \text{price}) \bowtie_{\text{pid} = \text{pid}} \text{Purchase}(\text{pid}, \text{cid}, \text{store})
\]

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join \(O(n^2)\)
2. Merge join \(O(??)\)
3. Hash join \(O(??)\)
Main Memory Algorithms

Logical operator:

\[ \text{Product}(\text{pid}, \text{name}, \text{price}) \Join_{\text{pid}=\text{pid}} \text{Purchase}(\text{pid}, \text{cid}, \text{store}) \]

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join \( O(n^2) \)
2. Merge join \( O(n \log n) \)
3. Hash join \( O(???) \)

\text{sort both} – \( O(n \log n) \)
\text{merge} – \( O(n) \)
Main Memory Algorithms

Logical operator:

Product(pid, name, price) \Join_{\text{pid}=\text{pid}} \text{Purchase(pid, cid, store)}

Propose three physical operators for the join, assuming the tables are in main memory:

1. Nested Loop Join \(O(n^2)\)
2. Merge join \(O(n \log n)\)
3. Hash join \(O(n) \ldots O(n^2)\)

add n to hash \(-O(n)?\)
lookup n in hash \(-O(n)\)?
BRIEF Review of Hash Tables

Separate chaining:

A (naïve) hash function:
\[ h(x) = x \mod 10 \]

Operations:
- find(103) = ??
- insert(488) = ??
BRIEF Review of Hash Tables

• $\text{insert}(k, v) = \text{inserts a key } k \text{ with value } v$

• Many values for one key
  – Hence, duplicate $k$’s are OK

• $\text{find}(k) = \text{return the list of all values } v$ associated to the key $k$
Query Evaluation Steps Review

1. Parse & Rewrite Query
2. Select Logical Plan
3. Select Physical Plan
4. Query Execution

Query optimization

SQL query

Disk

Logical plan

Physical plan

CSE 414 - Fall 2017
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Relational Algebra

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

Give a relational algebra expression for this query
Relational Algebra

\[
\pi_{\text{sname}} \left( \sigma_{\text{scity} = 'Seattle' \land \text{sstate} = 'WA' \land \text{pno} = 2} \left( \text{Supplier} \bowtie_{\text{sid} = \text{sid}} \text{Supply} \right) \right)
\]
Relational Algebra

SELECT sname 
FROM Supplier x, Supply y 
WHERE x.sid = y.sid 
    and y.pno = 2 
    and x.scity = 'Seattle' 
    and x.sstate = 'WA'

Relational algebra expression is also called the “logical query plan”
A physical query plan is a logical query plan annotated with physical implementation details.

```sql
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    AND y.pno = 2
    AND x.scity = 'Seattle'
    AND x.sstate = 'WA'
```
Physical Query Plan 2

Same logical query plan
Different physical plan

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'

(Supplier(sid, sname, scity, sstate))
(Supply(sid, pno, quantity))

(On the fly) (File scan)
(On the fly) (File scan)
(Hash join)
Physical Query Plan 3

(On the fly)

π \text{sname}

(Sort-merge join)

\sigma \text{scity=‘Seattle’ } \land \text{sstate=‘WA’}

(Scan & write to T1)

(Scan & write to T2)

Supplier (File scan)

Supply (File scan)

Different but equivalent logical query plan; different physical plan

SELECT \text{sname}
FROM \text{Supplier x, Supply y}
WHERE \text{x.sid = y.sid}
and \text{y.pno = 2}
and \text{x.scity = ‘Seattle’}
and \text{x.sstate = ‘WA’}
Query Optimization Problem

• For each SQL query… many logical plans

• For each logical plan… many physical plans

• How to find a fast physical plan?
  – Will discuss in a few lectures