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**: \( R \bowtie \pi_{\theta}(R \times S) \)
  - Join condition \( \theta \) consists only of equalities
- **Natural join**: \( R \bowtie S = \pi_{\theta} \pi_{\alpha}(R \times S) \)
  - Equijoin
  - Equality on all fields with same name in \( R \) and in \( S \)
  - Projection \( \pi_{\theta} \) 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

<table>
<thead>
<tr>
<th>AnonPatient</th>
<th>AnonJob</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>zip</td>
</tr>
<tr>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
</tr>
</tbody>
</table>

Announcements

- Should have used SQL / Azure now
  - let us know if you had any setup problems
- WQ3 is due on Sunday
- HW3 is due one week from Tuesday
- HW1 grades should be posted tonight
More 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
\[ \sigma_{\text{psize} > 10} \left( \text{Supplier} \right) \times \left( \text{Part} \right) \]

Name of supplier of parts or parts with size greater than 10
\[ \sigma_{\text{psize} > 10} \left( \text{Supplier} \right) \times \left( \text{Part} \right) \}

Name of supplier of red parts or parts with size greater than 10
\[ \sigma_{\text{psize} > 10} \left( \text{Supplier} \right) \times \left( \text{Part} \right) \}
\text{where} \quad \text{pcolor} = \text{red} \]

Query Evaluation Steps

1. Parse & Check Query
2. Decide how best to answer query: query optimization
3. Query Execution
4. Return Results

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'
```

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
```

T1, T2, T3 = temporary tables
sales(product, city, price)

---

Typical Plan for Block (1/2)

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

R
S

Typical Plan for Block (2/2)

```
SELECT fields FROM R, S, ...
WHERE condition
GROUP BY fields
HAVING condition
```

R
S

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)
```

De-Correlation

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

Un-nesting

```
(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)
EXCEPT = set difference
```
How about Subqueries?

Finally...

<table>
<thead>
<tr>
<th>Supplier(sno, name, sstate)</th>
<th>Part(pno, pname, pcolor, pcolor)</th>
<th>Supply(sno, pno, price)</th>
</tr>
</thead>
</table>

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

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:
Product(pid, name, price) ⫷ pid=pid Purchase(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:
Product(pid, name, price) ⫷ pid=pid 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)  two nested loops
2. Merge join  O(??)
3. Hash join  O(??)

sort both – O(n log n)
merge – O(n)
Main Memory Algorithms

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

Propose three physical operators for the join, assuming the tables are in main memory:
1. Nested Loop Join \( O(n^2) \)
2. Merge join \( O(n \log n) \)
3. Hash join \( O(n) \)

```
Product(pid, name, price)
Purchase(pid, cid, store)
```

BRIEF Review of Hash Tables

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

Operations:
\[ \text{find}(103) = ?? \]
\[ \text{insert}(488) = ?? \]

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>503</td>
<td>503</td>
<td>503</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Separate chaining:

```
add n to hash – O(1)?
lookup n in hash – O(n)?
```

BRIEF Review of Hash Tables

- \( \text{insert}(k, v) = \) inserts a key \( k \) with value \( v \)
- Many values for one key
  - Hence, duplicate \( k \)'s are OK
- \( \text{find}(k) = \) returns the list of all values \( v \) associated to the key \( k \)

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

Query Evaluation Steps Review

SQL query

```
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

```
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)
```

```
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

**Relational Algebra** is also called the “logical query plan.”

- **Supplier**
  - `sid`, `sname`, `scity`, `sstate`

- **Supply**
  - `sid`, `pno`, `quantity`

**Relational Algebra Expression:**

\[
\text{SELECT } sname \\
\text{FROM Supplier x, Supply y} \\
\text{WHERE } x.sid = y.sid \land x.scity = 'Seattle' \land x.sstate = 'WA' \land y.pno = 2
\]

**Physical Query Plan 1**

- **On the fly**
  - \(\pi_{sname}\)
  - \(\sigma_{scity='Seattle'} \land sstate='WA' \land pno=2\)

**Physical Query Plan 2**

- **(Hash join)**
  - \(\pi_{sname}\)
  - \(\sigma_{scity='Seattle'} \land sstate='WA' \land pno=2\)

**Physical Query Plan 3**

- **(Scan & write to T1)**
  - \(\sigma_{scity='Seattle'} \land sstate='WA' \land pno=2\)

**Query Optimization Problem**

- For each SQL query... many logical plans
- For each logical plan... many physical plans
- How do find a fast physical plan?
  - Will discuss in a few lectures