Join Summary

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

- **Equijoin**: $R \Theta \theta S = \sigma_\theta (R \times S)$
  - Join condition $\theta$ consists only of equalities

- **Natural join**: $R \Theta \theta S = \pi_A (\sigma_\theta (R \times S))$
  - 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

<table>
<thead>
<tr>
<th>AnonPatient P</th>
<th>AnonJob J</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>
| P ⋈ J

<table>
<thead>
<tr>
<th>P</th>
<th>J</th>
<th>Page</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>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
<td>20</td>
<td>98120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

More Examples

Name of supplier of parts with size greater than 10

$\pi_{sname} (\sigma_{psize>10} (\text{Supply}))$

Name of supplier of red parts or parts with size greater than 10

$\pi_{sname} (\sigma_{pcolor='red'} (\text{Part})) \cup \sigma_{psize>10} (\text{Part})$
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

3. Logical plan → physical plan

4. Query Execution

5. 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'

Can you think of another plan?

Extended RA: Operators on Bags

- Duplicate elimination \( \delta \)
- Grouping & aggregation \( \gamma \)
- Sorting \( \tau \)

Logical Query Plan

\[
\begin{align*}
\text{T1} & (\text{city}, p, c) \\
\text{T2} & (\text{city}, p, c) \\
\text{T3} & (\text{city}, c) \\
\end{align*}
\]
How about Subqueries?

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

EXCEPT

EXCEPT = set difference

How about Subqueries?

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

Finally…

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

Supplier(sno, sname, scity, sstate)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, price)
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) \text{\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 \(O(??)\)
2. Merge join \(O(??)\)
3. Hash join \(O(??)\)

(note that pid is a key)
**BRIEF Review of Hash Tables**

Separate chaining:

A (naïve) hash function:

- \( h(x) = x \mod 10 \)

Operations:

- find(103) = ??
- insert(488) = ??

Duplicates OK, WHY ??

Value can be different tuple ids.

- find(k) = returns the list of all values v associated to the key k

- insert(k, v) = inserts a key k with value v

- Many values for one key
  - Hence, duplicate k’s are OK

**Query Evaluation Steps Review**

SQL query

- Parse & Rewrite Query
- Select Logical Plan
- Select Physical Plan
- Physical plan
- Query Execution
- Disk

**Relational Algebra**

- Select Logical Plan
- Select Physical Plan
- Logical plan

Give a relational algebra expression for this query

Relational algebra is also called the "logical query plan"

**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'

\[ \pi_{\text{sname}}(\sigma_{\text{scity} = 'Seattle' \land \text{sstate} = 'WA' \land \text{pno} = 2}(\text{Supplier} \bowtie \text{Supply})) \]
A physical query plan is a logical query plan annotated with physical implementation details.

**Physical Query Plan 1**

```
π sname 
```

```
σ scity='Seattle' ∧ sstate='WA' ∧ pno=2
```

```
(On the fly)
```

```
(Nested loop)
```

```
(On the fly)
```

```
(File scan)
```

```
(File scan)
```

**Physical Query Plan 2**

```
π sname 
```

```
σ scity='Seattle' ∧ sstate='WA' ∧ pno=2
```

```
(On the fly)
```

```
(Hash join)
```

```
(On the fly)
```

```
(File scan)
```

```
(File scan)
```

**Physical Query Plan 3**

```
π sname 
```

```
σ scity='Seattle' ∧ sstate='WA'
```

```
(Scan & write to T1)
```

```
(Scan & write to T2)
```

```
(Scan & write to T1)
```

```
(Scan & write to T2)
```

```
(File scan)
```

```
(File scan)
```

**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