Database Applications (15-415)

DBMS Internals- Part VII
Lecture 18, March 29, 2015

Mohammad Hammoud
Today...

- **Last Session:**
  - DBMS Internals- Part VI
    - Algorithms for Relational Operations

- **Today’s Session:**
  - DBMS Internals- Part VII
    - Algorithms for Relational Operations (*Cont’d*)

- **Announcements:**
  - Project 3 is due on Thursday, April 2\textsuperscript{nd} by midnight
  - Quiz II will be held on Thursday, April 9\textsuperscript{th} (all concepts covered after the midterm are included)
Relational Operations

- We will consider how to implement:
  - *Selection* \((\sigma)\)
  - *Projection* \((\Pi)\)
  - *Join* \(\bowtie\)
  - *Set-difference* \((\setminus)\)
  - *Union* \(\cup\)
  - *Aggregation* (SUM, MIN, etc.) and GROUP BY

- Since each operation returns a relation, ops can be *composed*!

- After we cover how to implement operations, we will discuss how to *optimize* queries (formed by composing operators)
The Join Operation

- Consider the following query, Q, which implies a join:

```sql
SELECT *
FROM Reserves R, Sailors S
WHERE R.sid = S.sid
```

- How can we evaluate Q?
  - Compute $R \times S$
  - Select (and project) as required

- But, the result of a cross-product is typically much larger than the result of a join

- Hence, it is very important to implement joins *without* materializing the underlying cross-product
The Join Operation

- We will study *five* join algorithms, *two* which enumerate the cross-product and *three* which do not

- Join algorithms which enumerate the cross-product:
  - Simple Nested Loops Join
  - Block Nested Loops Join

- Join algorithms which *do not* enumerate the cross-product:
  - Index Nested Loops Join
  - Sort-Merge Join
  - Hash Join
Assumptions

- We assume the following two relations:
  - Sailors \((sid: \text{integer}, \ sname: \text{string}, \ rating: \text{integer}, \ age: \text{real})\)
  - Reserves \((sid: \text{integer}, \ bid: \text{integer}, \ day: \text{dates}, \ rname: \text{string})\)

- For Reserves, we assume:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages

- For Sailors, we assume:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages

- Our cost metric is the number of I/Os
Assumptions (Cont’d)

- We assume *equality* joins with:
  - \( R \) representing Reserves and \( S \) representing Sailors
  - \( M \) pages in \( R \), \( p_R \) tuples per page, \( m \) tuples total
  - \( N \) pages in \( S \), \( p_S \) tuples per page, \( n \) tuples total

- We ignore the output and computational costs
The Join Operation

- We will study five join algorithms, two which enumerate the cross-product and three which do not.

- Join algorithms which enumerate the cross-product:
  - Simple Nested Loops Join
  - Block Nested Loops Join

- Join algorithms which do not enumerate the cross-product:
  - Index Nested Loops Join
  - Sort-Merge Join
  - Hash Join
Simple Nested Loops Join

- Algorithm #0: (*naive*) nested loop (*SLOW!*)

\[
\begin{align*}
R(A, \ldots) & \quad m \\
S(A, \ldots) & \quad n
\end{align*}
\]
Simple Nested Loops Join

• Algorithm #0: *(naive)* nested loop *(SLOW!)*

```plaintext
for each tuple r of R
  for each tuple s of S
    print, if they match
```

```
R(A,..)
```

```
S(A, ......)
```

```
m
```

```
n
```
Simple Nested Loops Join

- Algorithm #0: *(naive)* nested loop *(SLOW!)*

\[
\begin{align*}
\text{for each tuple } r \text{ of } R & \quad \text{for each tuple } s \text{ of } S \\
\text{print, if they match} & \\
\end{align*}
\]
Simple Nested Loops Join

- Algorithm #0: *(naive)* nested loop *(SLOW!)*

How many disk accesses (‘M’ and ‘N’ are the numbers of pages for ‘R’ and ‘S’)?
Simple Nested Loops Join

- Algorithm #0: (*naive*) nested loop (*SLOW!*)

How many disk accesses (‘M’ and ‘N’ are the numbers of pages for ‘R’ and ‘S’)?

\[ \text{I/O Cost} = M + m \cdot N \]
Simple Nested Loops Join

- Algorithm #0: *(naive)* nested loop *(SLOW!)*

- Cost = $M + (p_R \times M) \times N = 1000 + 100 \times 1000 \times 500$ I/Os
- At 10ms/IO, total = ~6 days (!)

I/O Cost = $M + m \times N$

Can we do better?
Nested Loops Join: A Simple Refinement

- Algorithm:
  - Read in a page of R
  - Read in a page of S
  - Print matching tuples

\[
\text{COST} = \ ?
\]
Nested Loops Join: A Simple Refinement

- Algorithm:

  Read in a *page* of R
  Read in a *page* of S
  Print matching tuples

  \[ \text{COST} = M + M \times N \]

\[ \begin{array}{c}
\text{R}(A, \ldots) \\
M \text{ pages, } m \text{ tuples}
\end{array} \quad \begin{array}{c}
\text{S}(A, \ldots) \\
N \text{ pages, } n \text{ tuples}
\end{array} \]
Nested Loops Join

- Which relation should be the *outer*?

\[ \text{COST} = M + M \times N \]
Nested Loops Join

- Which relation should be the *outer*?
- A: The smaller (page-wise)

\[
\text{COST} = M + M \times N
\]
Nested Loops Join

- M=1000, N=500 - *if larger is the outer*:
- Cost = 1000 + 1000*500 = 501,000
  = 5010 sec (~ 1.4h)

\[
\text{COST} = M + M \times N
\]
Nested Loops Join

- M=1000, N=500 - if smaller is the outer:
- Cost = 500 + 1000*500 = 500,500
  = 5005 sec (~ 1.4h)

\[
\text{COST} = N + M \times N
\]
Summary: Simple Nested Loops Join

- What if we do not apply the page-oriented refinement?
  - Cost = M + (pR * M) * N = 1000 + 100*1000*500 I/Os
  - At 10ms/IO, total = ~6 days (!)

- What if we apply the page-oriented refinement?
  - Cost = M * N + M = 1000*500+1000 I/Os
  - At 10ms/IO, total = 1.4 hours (!)

- What if the smaller relation is the outer?
  - Slightly better
The Join Operation

- We will study *five* join algorithms, *two* which enumerate the cross-product and *three* which do not

Join algorithms which enumerate the cross-product:
- Simple Nested Loops Join
- Block Nested Loops Join

Join algorithms which *do not* enumerate the cross-product:
- Index Nested Loops Join
- Sort-Merge Join
- Hash Join
Block Nested Loops

- What if we have $B$ buffer pages available?
Block Nested Loops

- What if we have $B$ buffer pages available?
- A: Give $B - 2$ buffer pages to outer, 1 to inner, 1 for output
Block Nested Loops

- Algorithm:
  - Read in $B-2$ pages of $R$
  - Read in a page of $S$
  - Print matching tuples

\[
\begin{align*}
\text{COST} &= \ ? \\
\end{align*}
\]
Block Nested Loops

- Algorithm:
  
  Read in $B-2$ pages of $R$
  Read in a page of $S$
  Print matching tuples

$$\text{COST} = M + \frac{M}{B-2} \times N$$
Block Nested Loops

- And, actually:
- \( \text{Cost} = M + \lceil M/(B-2) \rceil \times N \)

\[
\text{COST} = M + \frac{M}{B-2} \times N
\]
Block Nested Loops

- If the smallest (outer) relation fits in memory?
- That is, $B = N+2$
- Cost =?
Block Nested Loops

- If the smallest (outer) relation fits in memory?
- That is, \[ B = N + 2 \]
- Cost = \[ N + M \] (minimum!)
Nested Loops - Guidelines

- Pick as outer the smallest table (= fewest pages)
- Fit as much of it in memory as possible
- Loop over the inner
The Join Operation

- We will study five join algorithms, two which enumerate the cross-product and three which do not.

- Join algorithms which enumerate the cross-product:
  - Simple Nested Loops Join
  - Block Nested Loops Join

- Join algorithms which do not enumerate the cross-product:
  - Index Nested Loops Join
  - Sort-Merge Join
  - Hash Join
What if there is an index on one of the relations on the join attribute(s)?

A: Leverage the index by making the indexed relation *inner*
Index Nested Loops Join

- Assuming an index on S:

  for each tuple \( r \) of \( R \)
  for each tuple \( s \) of \( S \) where \( r_i = s_j \)
  Add \( (r, s) \) to result
Index Nested Loops Join

- What will be the cost?
- Cost: $M + m \times c$ (c: look-up cost)

‘c’ depends on the type of index, the adopted alternative and whether the index is clustered or un-clustered!
The Join Operation

- We will study five join algorithms, two which enumerate the cross-product and three which do not

Join algorithms which enumerate the cross-product:
- Simple Nested Loops Join
- Block Nested Loops Join

Join algorithms which do not enumerate the cross-product:
- Index Nested Loops Join
- Sort-Merge Join
- Hash Join
Sort-Merge Join

- Sort both relations on join attribute(s)
- Scan each relation and merge
- This works only for equality join conditions!
### Sort-Merge Join: An Example

#### Table 1: Ratings

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>28</td>
<td>uppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

#### Table 2: Bid Days

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>103</td>
<td>12/4/96</td>
<td>guppy</td>
</tr>
<tr>
<td>28</td>
<td>103</td>
<td>11/3/96</td>
<td>yuppy</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>10/12/96</td>
<td>lubber</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/11/96</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>dustin</td>
</tr>
</tbody>
</table>
**Sort-Merge Join: An Example**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>103</td>
<td>12/4/96</td>
<td>guppy</td>
</tr>
<tr>
<td>28</td>
<td>103</td>
<td>11/3/96</td>
<td>yuppy</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>10/12/96</td>
<td>lubber</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/11/96</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>dustin</td>
</tr>
</tbody>
</table>
## Sort-Merge Join: An Example

### Table 1: User Information
<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

### Table 2: Bid Information
<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>103</td>
<td>12/4/96</td>
<td>guppy</td>
</tr>
<tr>
<td>28</td>
<td>103</td>
<td>11/3/96</td>
<td>yuppy</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>10/12/96</td>
<td>lubber</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/11/96</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>dustin</td>
</tr>
</tbody>
</table>
## Sort-Merge Join: An Example

### Example 1

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>28</td>
<td>uppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

### Example 2

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>103</td>
<td>12/4/96</td>
<td>guppy</td>
</tr>
<tr>
<td>28</td>
<td>103</td>
<td>11/3/96</td>
<td>uppy</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>10/12/96</td>
<td>lubber</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/11/96</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>dustin</td>
</tr>
</tbody>
</table>

Output the two tuples.
Sort-Merge Join: An Example

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>103</td>
<td>12/4/96</td>
<td>guppy</td>
</tr>
<tr>
<td>28</td>
<td>103</td>
<td>11/3/96</td>
<td>yuppy</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>10/12/96</td>
<td>lubber</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/11/96</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>dustin</td>
</tr>
</tbody>
</table>
Sort-Merge Join: An Example

### Table 1: Sid, Sname, Rating, Age

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

### Table 2: Sid, Bid, Day, Rname

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>103</td>
<td>12/4/96</td>
<td>guppy</td>
</tr>
<tr>
<td>28</td>
<td>103</td>
<td>11/3/96</td>
<td>yuppy</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>10/12/96</td>
<td>lubber</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/11/96</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>dustin</td>
</tr>
</tbody>
</table>

![Diagram showing the join process]

The join process is indicated by the red and blue boxes, with the resulting data shown in the rightmost column.
Sort-Merge Join: An Example

Output the two tuples
## Sort-Merge Join: An Example

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>103</td>
<td>12/4/96</td>
<td>guppy</td>
</tr>
<tr>
<td>28</td>
<td>103</td>
<td>11/3/96</td>
<td>yuppy</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>10/12/96</td>
<td>lubber</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/11/96</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>dustin</td>
</tr>
</tbody>
</table>
Sort-Merge Join: An Example

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>28</td>
<td>uppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>103</td>
<td>12/4/96</td>
<td>guppy</td>
</tr>
<tr>
<td>28</td>
<td>103</td>
<td>11/3/96</td>
<td>upto</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>10/12/96</td>
<td>lubber</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/11/96</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>dustin</td>
</tr>
</tbody>
</table>

The join condition is not satisfied because the sname of the first relation (dusty) does not match the sname of the second relation (yuppy) at their corresponding positions.
Sort-Merge Join: An Example

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>103</td>
<td>12/4/96</td>
<td>guppy</td>
</tr>
<tr>
<td>28</td>
<td>103</td>
<td>11/3/96</td>
<td>yuppy</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>10/12/96</td>
<td>lubber</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/11/96</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>dustin</td>
</tr>
</tbody>
</table>
### Sort-Merge Join: An Example

#### Table 1: $sid$, $sname$, $rating$, $age$

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>28</td>
<td>uppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

#### Table 2: $sid$, $bid$, $day$, $rname$

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>103</td>
<td>12/4/96</td>
<td>guppy</td>
</tr>
<tr>
<td>28</td>
<td>103</td>
<td>11/3/96</td>
<td>yuppy</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/10/96</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>10/12/96</td>
<td>lubber</td>
</tr>
<tr>
<td>31</td>
<td>101</td>
<td>10/11/96</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
<td>dustin</td>
</tr>
</tbody>
</table>

- **Output the two tuples**

- **Continue the same way!**
Sort-Merge Join

- What is the cost?
- \( \sim 2^\cdot M^\cdot \log M / \log B + 2^\cdot N^\cdot \log N / \log B + M + N \)
Sort-Merge Join

- Assuming 100 buffer pages, Reserves and Sailors can be sorted in 2 passes
- Total cost = 7500 I/Os
- Cost of Block Nested Loops Join = 7500 I/Os
Sort-Merge Join

- Assuming 35 buffer pages, Reserves and Sailors can be sorted in 2 passes
- Total cost = 7500 I/Os
- Cost of Block Nested Loops Join = 15000 I/Os

\[
\begin{align*}
\text{M pages,} & \quad \text{R(A,..)} \quad \text{S(A, ......)} \quad \text{N pages,} \\
m \text{tuples} & \quad \text{m tuples} & \quad \text{n tuples}
\end{align*}
\]
Sort-Merge Join

- Assuming 300 buffer pages, Reserves and Sailors can be sorted in 2 passes
- Total cost = 7500 I/Os
- Cost of Block Nested Loops Join = 2500 I/Os

The Block Nested Loops Join is more sensitive to the buffer size!
The Join Operation

- We will study *five* join algorithms, *two* which enumerate the cross-product and *three* which do not.

- Join algorithms which enumerate the cross-product:
  - Simple Nested Loops Join
  - Block Nested Loops Join

- Join algorithms which do *not* enumerate the cross-product:
  - Index Nested Loops Join
  - Sort-Merge Join
  - Hash Join
Hash Join

- The join algorithm based on hashing has two phases:
  - Partitioning (also called Building) Phase
  - Probing (also called Matching) Phase

- **Idea**: Hash both relations on the join attribute into $k$ partitions, using the same hash function $h$

- **Premise**: R tuples in partition $i$ can join only with S tuples in the same partition $i$
Hash Join: Partitioning Phase

- Partition both relations using hash function $h$

Two tuples that belong to different partitions are guaranteed not to match.
Hash Join: Probing Phase

- Read in a partition of R, hash it using $h2 (<> h)$

- Scan the corresponding partition of S and search for matches
Hash Join: Cost

- **What is the cost of the partitioning phase?**
  - We need to scan R and S, and write them out once
  - Hence, cost is $2(M+N)$ I/Os

- **What is the cost of the probing phase?**
  - We need to scan each partition once *(assuming no partition overflows)* of R and S
  - Hence, cost is $M + N$ I/Os

- Total Cost = $3(M + N)$
Hash Join: Cost (Cont’d)

- Total Cost = 3 (M + N)

- Joining Reserves and Sailors would cost 3 (500 + 1000) = 4500 I/Os

- Assuming 10ms per I/O, hash join takes less than 1 minute!

- This underscores the importance of using a good join algorithm (e.g., Simple NL Join takes ~140 hours!)

But, so far we have been assuming that partitions fit in memory!
Memory Requirements and Overflow Handling

- How can we increase the chances for a given partition in the probing phase to fit in memory?
  - Maximize the number of partitions in the building phase

- If we partition R (or S) into $k$ partitions, what would be the size of each partition (in terms of $B$)?
  - At least $k$ output buffer pages and 1 input buffer page
  - Given $B$ buffer pages, $k = B - 1$
  - Hence, the size of an R (or S) partition = $M/B-1$

- What is the number of pages in the (in-memory) hash table built during the probing phase per a partition?
  - $f.M/B-1$, where $f$ is a fudge factor
Memory Requirements and Overflow Handling

- What do we need else in the probing phase?
  - A buffer page for scanning the S partition
  - An output buffer page

- What is a good value of B as such?
  - $B > \frac{f.M}{B} - 1 + 2$
  - Therefore, we need $B > \sqrt{f \cdot M}$

- What if a partition overflows?
  - Apply the hash join technique *recursively* (as is the case with the projection operation)
Hash Join vs. Sort-Merge Join

- If $B > \sqrt{M}$ (M is the # of pages in the smaller relation) and we assume uniform partitioning, the cost of hash join is $3(M+N) \text{ I/Os}$

- If $B > \sqrt{N}$ (N is the # of pages in the larger relation), the cost of sort-merge join is $3(M+N) \text{ I/Os}$

Which algorithm to use, hash join or sort-merge join?
Hash Join vs. Sort-Merge Join

- If the available number of buffer pages falls between \( \sqrt{M} \) and \( \sqrt{N} \), hash join is preferred (why?)

- Hash Join shown to be highly parallelizable (*beyond the scope of the class*)

- Hash join is sensitive to data skew while sort-merge join is not

- Results are sorted after applying sort-merge join (may help “upstream” operators)

- Sort-merge join goes fast if one of the input relations is already sorted
The Join Operation

- We will study *five* join algorithms, *two* which enumerate the cross-product and *three* which do not.

Join algorithms which enumerate the cross-product:
- Simple Nested Loops Join
- Block Nested Loops Join

Join algorithms which do not enumerate the cross-product:
- Index Nested Loops Join
- Sort-Merge Join
- Hash Join
Next Class

Query Optimization and Execution

Relational Operators

Files and Access Methods

Buffer Management

Disk Space Management

Queries

Transaction Manager

Lock Manager

Recovery Manager

DB

Continue...