Where we are / and where we go

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Additional Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feb 7</td>
<td>Transactions: Concurrency Control</td>
<td>Midterm</td>
</tr>
<tr>
<td></td>
<td>lecture 14-15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Midterm review on the board</td>
<td></td>
</tr>
<tr>
<td>Feb 14</td>
<td>Database Tuning</td>
<td>Query Processing Overview</td>
</tr>
<tr>
<td></td>
<td>lecture 17</td>
<td>lecture 19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Project 3 due</td>
</tr>
<tr>
<td>Feb 21</td>
<td>No class (Presidents Day)</td>
<td>Operator Algorithms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Query Optimization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Homework 3 due</td>
</tr>
<tr>
<td>Feb 28</td>
<td>Query Optimization</td>
<td>Query Optimization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parallel and Distributed DBMSs</td>
</tr>
<tr>
<td>Mar 7</td>
<td>Pig Latin</td>
<td>TBA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Wrap-up</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Project 4 due</td>
</tr>
<tr>
<td>Mar 14</td>
<td>Final Exam</td>
<td>Thursday, March 17, 8:30am-10:20am, in class</td>
</tr>
</tbody>
</table>
Why Learn About Operator Algorithms?

- Implemented in commercial DBMSs
  - DBMSs implement different subsets of known algorithms

- Good algorithms can greatly improve performance

- Need to know about physical operators to understand query optimization
Cost Parameters

- In database systems the data is on disk
- **Cost = total number of I/Os**

Parameters:
- \( B(R) \) = \# of blocks (i.e., pages) for relation R
- \( T(R) \) = \# of tuples in relation R
- \( V(R, a) \) = \# of distinct values of attribute a
  - When a is a key, \( V(R, a) = T(R) \)
  - When a is not a key, \( V(R, a) \) can be anything \(< T(R) \)
- \( M \) = \# of max. pages in main memory
Cost

- Cost of an operation = number of disk I/Os to
  - Read the operands
  - Compute the result

- Cost of writing the result to disk is *not included*
  - Need to count it separately when applicable
Cost of Scanning a Table

- Result may be unsorted: $B(R)$
- Result needs to be sorted: $3 \times B(R)$
  - We will discuss sorting later

http://www.cs.washington.edu/education/courses/cse444/11wi/
Outline for Today

- Join operator algorithms
  - One-pass algorithms (Sec. 15.2 and 15.3)
  - Index-based algorithms (Sec 15.6)
  - Two-pass algorithms (Sec 15.4 and 15.5)

Note about readings:
- In class, we will discuss only join operator algorithms (because other operators are easier)
- Read the book to get more details about these algos and about algots for other operators
Basic Join Algorithms

- Logical operator:
  - Product(pname, cname) ⋈ Company(cname, city)

- Propose three physical operators for the join, assuming the tables are in main memory:
  - Hash join
  - Nested loop join
  - Sort-merge join
1. Hash Join

Hash join: $R \bowtie S$

- Scan $R$, build buckets in main memory
- Then scan $S$ and join
- Cost: $B(R) + B(S)$

- One-pass algorithm when $B(R) \leq M$
  - By “one pass”, we mean that the operator reads its operands only once. It does not write intermediate results back to disk.

http://www.cs.washington.edu/education/courses/cse444/11wi/
1. Hash Join Example

Patient(pid, name, address)
Insurance(pid, provider, policy_nb)

Two tuples per page
1. Hash Join Example

Patient \( \Join \) Insurance

Showing only pid; note a page contains 2 tuples

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Memory M = 21 pages

http://www.cs.washington.edu/education/courses/cse444/11wi/
1. Hash Join Example

Step 1: Scan Patient and create hash table in memory

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Memory M = 21 pages

Hash h: pid % 5

| 5 | 1 | 6 | 2 | 3 8 | 4 9 |

Input buffer

http://www.cs.washington.edu/education/courses/cse444/11wi/
# 1. Hash Join Example

**Patient ⋈ Insurance**

**Step 1:** Scan Patient and create hash table in memory

**Step 2:** Scan Insurance and probe into hash table

**Memory M = 21 pages**

**Hash h: pid % 5**

<table>
<thead>
<tr>
<th>Disk</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

**Input buffer**

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

**Output buffer**

| 2 2 |

write to disk

http://www.cs.washington.edu/education/courses/cse444/11wi/
1. Hash Join Example

**Patient ⊖ Insurance**

Step 1: Scan Patient and create hash table in memory

Step 2: Scan Insurance and probe into hash table

Memory M = 21 pages

Hash h: pid % 5

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Input buffer

Output buffer

http://www.cs.washington.edu/education/courses/cse444/11wi/
1. Hash Join Example

Step 1: Scan Patient and create hash table in memory
Step 2: Scan Insurance and probe into hash table

Memory M = 21 pages

Hash h: pid % 5

<table>
<thead>
<tr>
<th>5</th>
<th></th>
<th>1</th>
<th>6</th>
<th>2</th>
<th>3</th>
<th>8</th>
<th>4</th>
<th>9</th>
</tr>
</thead>
</table>

Keep going until read all of Insurance

Cost: $B(R) + B(S)$

http://www.cs.washington.edu/education/courses/cse444/11wi/
1. Hash Join Details

```java
Open( ) {
    H = newHashTable( );
    R.Open( );
    x = R.GetNext( );
    while (x != null) {
        H.insert(x);
        x = R.GetNext( );
    }
    R.Close( );
    S.Open( );
    buffer = [ ];
}
```

http://www.cs.washington.edu/education/courses/cse444/11wi/
1. Hash Join Details

```c
GetNext( ) {
    while (buffer == [ ]) {
        x = S.GetNext( );
        if (x == Null) return NULL;
        buffer = H.find(x);
    }
    z = buffer.first( );
    buffer = buffer.rest( );
    return z;
}
```
1. Hash Join Details

```java
Close() {
    release memory (H, buffer, etc.);
    S.Close( )
}
```
2. Nested Loop Joins

Tuple-based nested loop $R \bowtie S$

- $R$ is the outer relation, $S$ is the inner relation

```java
for each tuple r in R do
    for each tuple s in S do
        if r and s join then output (r,s)
```

- Cost: $B(R) + T(R) B(S)$
- One-pass only over outer relation
  - But $S$ is read many times
2. Page-at-a-time Refinement

- **Cost:** $B(R) + B(R)B(S)$

```plaintext
for each page of tuples r in R do
  for each page of tuples s in S do
    for all pairs of tuples
      if r and s join then output (r, s)
```

http://www.cs.washington.edu/education/courses/cse444/11wi/
2. Nested Loop Example

Patient ▪ Insurance

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Input buffer for Patient

| 1 2 |

Input buffer for Insurance

| 2 4 |

Output buffer

| 2 2 |
2. Nested Loop Example

Patient $\times$ Insurance

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Input buffer for Patient

Input buffer for Insurance

Output buffer
2. Nested Loop Example

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Input buffer for Patient

Input buffer for Insurance

Keep going until read all of Insurance

Then repeat for next page of Patient... until end of Patient

Output buffer

Cost: $B(R) + B(R) \cdot B(S)$
2b. Nested-block join (Nested-loop join)

M = 101 pages

Cost R: 500
Cost S: 5000 = 5\cdot1000
SUM: 5500

B(R) + B(R)/M \cdot B(S)
500 + (500/100) \cdot 1000 = 5500

Cost S: 1000
Cost R: 5000 = 10\cdot500
SUM: 6000

B(S) + B(R)/M \cdot B(S)
1000 + (1000/100) \cdot 500 = 6000
3. Sort-Merge Join

Sort-merge join: \( R \bowtie S \)

- Scan \( R \) and sort in main memory
- Scan \( S \) and sort in main memory
- Merge \( R \) and \( S \)

Cost: \( B(R) + B(S) \)

- One pass algorithm when \( B(S) + B(R) \leq M \)
- Typically, this is NOT a one pass algorithm
3. Sort-Merge Join Example

Step 1: Scan Patient and sort in memory

Memory $M = 21$ pages
3. Sort-Merge Join Example

**Step 1:** Scan Patient and sort in memory

**Step 2:** Scan Insurance and sort in memory

Memory $M = 21$ pages

---

http://www.cs.washington.edu/education/courses/cse444/11wi/
3. Sort-Merge Join Example

Step 1: Scan Patient and sort in memory
Step 2: Scan Insurance and sort in memory
Step 3: Merge Patient and Insurance

Memory $M = 21$ pages

Output buffer
3. Sort-Merge Join Example

Step 1: Scan Patient and sort in memory
Step 2: Scan Insurance and sort in memory
Step 3: Merge Patient and Insurance

Memory M = 21 pages

Disk

Patient

Insurance

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

| 2 | 4 |
| 6 | 6 |
| 4 | 3 |
| 1 | 3 |
| 2 | 8 |
| 8 | 9 |

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

| 2 | 2 |

Output buffer

Keep going until end of first relation

http://www.cs.washington.edu/education/courses/cse444/11wi/
Outline for Today

- Join operator algorithms
  - One-pass algorithms (Sec. 15.2 and 15.3)
  - Index-based algorithms (Sec 15.6)
  - Two-pass algorithms (Sec 15.4 and 15.5)
Review: Access Methods

- **Heap file**
  - Scan tuples one at the time

- **Hash-based index**
  - Efficient selection on equality predicates
  - Can also scan data entries in index

- **Tree-based index**
  - Efficient selection on equality or range predicates
  - Can also scan data entries in index
Index Based Selection

- Selection on equality: $\sigma_{a=v}(R)$

- $V(R,a) = \#$ of distinct values of attribute $a$

- Cost Clustered index on $a$: $B(R)/V(R,a)$
- Cost Unclustered index on $a$: $T(R)/V(R,a)$

- Note: we ignored I/O cost for index pages
Index Based Selection

Example:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T(R)</td>
<td>100,000</td>
</tr>
<tr>
<td>B(R)</td>
<td>2,000</td>
</tr>
<tr>
<td>V(R, a)</td>
<td>20</td>
</tr>
</tbody>
</table>

Cost of \( s_{a=v}(R) = ? \)

Table scan: \( B(R) = 2,000 \) I/Os

Index based selection

- If index is clustered: \( B(R)/V(R,a) = 100 \) I/Os
- If index is unclustered: \( T(R)/V(R,a) = 5,000 \) I/Os

Lesson

- Don’t build unclustered indexes when \( V(R,a) \) is small, i.e. many tuples with same attribute values a (here 5,000)!

http://www.cs.washington.edu/education/courses/cse444/11wi/
4. Index Nested Loop Join

Index-nested loop join \( R \bowtie S \)

- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple (s) from S

Cost

- If index on S is clustered:
  \[ B(R) + T(R) \frac{B(S)}{V(S, a)} \]
- If index on S is unclustered:
  \[ B(R) + T(R) \frac{T(S)}{V(S, a)} \]

Expected number of tuples from S that join with a tuple from R

http://www.cs.washington.edu/education/courses/cse444/11wi/
Outline for Today

- Join operator algorithms
  - One-pass algorithms (Sec. 15.2 and 15.3)
  - Index-based algorithms (Sec 15.6)
  - Two-pass algorithms (Sec 15.4 and 15.5)
Two-Pass Algorithms

- What if data does not fit in memory?
- Need to process it in multiple passes

- Two key techniques
  - 1. Hashing
  - 2. Sorting
5. Two-Pass Join Alg. based on Hashing

- Idea: partition a relation R into buckets, on disk
- Each bucket has size $\approx \frac{B(R)}{M}$ pages

Does each bucket fit in main memory?
Yes if $\frac{B(R)}{M} \leq M$, i.e. $B(R) \leq M^2$
5. Partitioned (Grace) Hash Join

Hash Join \( R \bowtie S \)

- **Step 1:**
  - Hash \( S \) into \( M-1 \) buckets
  - Send all buckets to disk

- **Step 2**
  - Hash \( R \) into \( M-1 \) buckets
  - Send all buckets to disk

- **Step 3**
  - Join every pair of buckets
5. Partitioned Hash Join

- Partition both relations using hash function $h$
- $R$ tuples in partition $i$ will only match $S$ tuples in partition $i$. 

http://www.cs.washington.edu/education/courses/cse444/11wi/
5. Partitioned Hash Join

- Read in partition of R, hash it using $h_2 \neq h$
  - Build phase
- Scan matching partition of S, search for matches
  - Probe phase

![Diagram of Partitioned Hash Join](http://www.cs.washington.edu/education/courses/cse444/11wi/)
5. Partitioned Hash Join

- **Cost:** $3B(R) + 3B(S)$
- **Assumption:** $\min(B(R), B(S)) \leq M^2$

What is max. size of smaller table?
- **1 Gb** main memory = $2^{30}$ b
- **64 Kb** block size = $2^{16}$ b
Then $M$ (# blocks) = $2^{14} = 16$ K
Then $B \leq M^2 = 2^{28}$
Then total size = $2^{44}$ b = 16 Tb

Calculate cost with nested block join for two 16 Tb tables: Cost = $B + B^2/M$
- $B(R) = 16$ Tb / 64 Kb = $2^{28}$
- Cost = $2^{28} + 2^{42} \approx 2^{42}$
- Cost $\approx 2^{14} B(R) = 16$ K $B(R)$
  vs. 6 $B(R)$ for Part. Hash Join

Example 15.5 in the book says 4 Tb, b/c "$2^{14} = 64$ K" ??? 😞
External Sorting

- Problem: Sort a file of size $B$ with memory $M$

- Where we need this:
  - ORDER BY in SQL queries
  - Several physical operators
  - Bulk loading of B+-tree indexes.

- Sorting is two-pass when $B < M^2$
External Merge-Sort: Step 1

Step 1: Load $M$ pages in memory, sort

Each run is of length $M$ pages (maximal $M-1$ runs)
**External Merge-Sort: Step 2**

**Step 1:** Load $M$ pages in memory, sort

**Step 2:** Merge $M - 1$ runs into a new run

Result: runs of length $M$ ($M - 1$) $\approx M^2$

Cost: read + write + read: $3B(R)$

Assumption: $B(R) \leq M^2$

http://www.cs.washington.edu/education/courses/cse444/11wi/
6. Two-Pass Join Alg. based on Sorting

Sort-based Join \( R \bowtie S \)
- Step 1: sort both \( R \) and \( S \) on the join attribute:
  - Cost: \( 4B(R)+4B(S) \) (because need to write to disk)
- Step 2: Read both relations in sorted order, match tuples
  - Cost: \( B(R)+B(S) \)
- Total cost: \( 5B(R)+5B(S) \)
- Assumption: \( B(R) \leq M^2, B(S) \leq M^2 \)

http://www.cs.washington.edu/education/courses/cse444/11wi/
6. Two-Pass Join Alg. based on Sorting

Sort Merge Join $R \bowtie S$

- If $B(R) + B(S) \leq M^2$
- If the number of tuples in $R$ matching those in $S$ is small (or vice versa)
- We can compute the join during the merge phase

- Total cost: $3B(R) + 3B(S)$
Summary of Join Algorithms

- **Nested Loop Join**: \( B(R) + \frac{B(R)B(S)}{M} \)
  - Assuming block-at-a-time refinement
  - With page-at-a-time, the formula would be: \( B(R) + B(R)B(S) \)

- **Hash Join**: \( 3B(R) + 3B(S) \)
  - Assuming: \( \min(B(R), B(S)) \leq M^2 \)

- **Sort-Merge Join**: \( 3B(R) + 3B(S) \)
  - Assuming \( B(R)+B(S) \leq M^2 \)

- **Index Nested Loop Join**: \( B(R) + \frac{T(R)B(S)}{V(S,a)} \)
  - Assuming \( S \) has clustered index on attribute \( a \)
Summary of Query Execution

- For each logical query plan
  - There exist many physical query plans
  - Each plan has a different cost
  - Cost depends on the data
- Additionally, for each query
  - There exist several logical plans
- Next 3 lectures: query optimization
  - How to compute the cost of a complete plan?
  - How to pick a good query plan for a query?