# CSE 444: Database Internals 

## Lecture 7

Query Execution and
Operator Algorithms (part 1)

## What We Have Learned So Far

- Overview of the architecture of a DBMS
- Access methods
- Heap files, sequential files, Indexes (hash or B+ trees)
- Role of buffer manager
- Practiced the concepts in hw1 and lab1


## DBMS Architecture

| Admission Control | Parser |
| :---: | :---: |
| Connection Mgr | Query Rewrite |
|  | Optimizer |
| Process Manager | Query Processor |
| Executor |  |
| Access Methods | Buffer Manager |
| Lock Manager | Log Manager |
|  |  |
| Storage Manager |  |

## Next Lectures

- How to answer queries efficiently!
- Physical query plans and operator algorithms
- How to automatically find good query plans
- How to compute the cost of a complete plan
- How to pick a good query plan for a query
- i.e., Query optimization


## Query Execution Bottom Line

- SQL query transformed into physical plan
- Access path selection for each relation
- Implementation choice for each operator
- Scheduling decisions for operators
- Execution of the physical plan is pull-based
- Operators given a limited amount of memory


## Pipelined Query Execution

(On the fly)
(On the fly)

$$
\sigma_{\text {sscity }}=\text { 'Seattle' } \wedge \text { sstate }=‘ W A^{\prime} \wedge \mathrm{pno}=2
$$

open()
(Nested loop)

## Pipelined Query Execution

(On the fly)
(On the fly)
 nex()
$\pi_{\text {sname }}$
next()
$\sigma_{\text {sscity }}=$ 'Seattle' $\wedge$ sstate='WA' $\wedge$ pno=2
next()
(Nested loop)


CSE 444 - Winter 2017

## Intermediate Tuple Materialization

(On the fly)
(Sort-merge join)

$\boldsymbol{\pi}_{\text {sname }}$

(Scan: write to T1)

$$
\sigma_{\text {sscity }}=' \text { Seattle' } \wedge \text { sstate }=' W A^{\prime}
$$

Suppliers
(File scan)

$$
\sigma_{\mathrm{pno}}=2
$$

(Scan: write to T2)

(File scan)

## Memory Management

Each operator:

- Pre-allocates heap space for tuples
- Pointers to base data in buffer pool
- Or new tuples on the heap
- Allocates memory for its internal state
- Either on heap or buffer pool (depends on system)

DMBS may limit how much memory each operator, or each query can use

## Operator Algorithms

## Operator Algorithms

## Design criteria

- Cost: IO, CPU, Network
- Memory utilization
- Load balance (for parallel operators)


## Cost Parameters

- Cost = total number of I/Os
- This is a simplification that ignores CPU, network
- Parameters:
$-B(R)=\#$ of blocks (i.e., pages) for relation $R$
$-T(R)=\#$ of tuples in relation $R$
$-\mathbf{V}(\mathbf{R}, \mathrm{a})=$ \# of distinct values of attribute a
- When a is a key, $V(R, a)=T(R)$
- When $\mathbf{a}$ is not a key, $\mathbf{V}(\mathbf{R}, \mathrm{a})$ can be anything $<\mathbf{T}(\mathbf{R})$


## Convention

- Cost $=$ the cost of reading operands from disk
- Cost of writing the result to disk is not included; need to count it separately when applicable


## Outline

- 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 discuss only algorithms for joins
- Other operators are easier: read the book


## Join Algorithms

- Hash join
- Nested loop join
- Sort-merge join


## 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 $B(R) \leq M$


## Hash Join Example

Patient(pid, name, address)
Insurance(pid, provider, policy_nb)
Patient $\bowtie$ Insurance

Two tuples per page

Patient

| 1 | 'Bob' | 'Seattle' |
| :--- | :--- | :--- |
| 2 | 'Ela' | 'Everett' |


| 3 | 'Jill' | 'Kent' |
| :--- | :---: | :---: |
| 4 | 'Joe' | 'Seattle' |

Insurance

| 2 | 'Blue' | 123 |
| :---: | :---: | :---: |
| 4 | 'Prem' | 432 |


| 4 | 'Prem' | 343 |
| :---: | :---: | :---: |
| 3 | 'GrpH' | 554 |

## Hash Join Example

Patient $\bowtie$ Insurance
Some largeenough nb

Memory M = 21 pages

Disk
Patient Insurance


This is one page with two tuples

## Hash Join Example

Step 1: Scan Patient and build hash table in memory

Can be done in method open()

Disk
Patient Insurance

| 12 | 2 | 4 | 6 | 6 |
| :---: | :---: | :---: | :---: | :---: |
| 34 | 4 | 3 | 1 | 3 |
| 96 | 2 | 8 |  |  |
| 85 | 8 | 9 |  |  |

Memory M = 21 pages
Hash h: pid \% 5


Input buffer

## Hash Join Example

Step 2: Scan Insurance and probe into hash table

Done during calls to next()

Memory M = 21 pages
Hash h: pid \% 5


Disk
Patient Insurance

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


| 2 | 4 |
| :--- | :--- |

Input buffer

## 22

Output buffer

Write to disk or pass to next operator

## Hash Join Example

Step 2: Scan Insurance and probe into hash table
Done during calls to next()

Memory M = 21 pages
Hash h: pid \% 5


| 2 | 4 |
| :--- | :--- |

Input buffer

| 4 | 4 |
| :--- | :--- |

Output buffer

## Hash Join Example

Step 2: Scan Insurance and probe into hash table

Done during calls to next()

Memory M = 21 pages
Hash h: pid \% 5


Disk
Patient Insurance

| 12 | 2 | 4 | 6 | 6 |
| :---: | :---: | :---: | :---: | :---: |
| 3 4 | 4 | 3 | 1 | 3 |
| 9 6 | 2 | 8 |  |  |
| 8 | 8 | 9 |  |  |


| 4 | 3 |
| :--- | :--- |

Input buffer
Keep going until read all of Insurance

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

## Nested Loop Joins

- Tuple-based nested loop $R \bowtie S$
- $R$ is the outer relation, $S$ is the inner relation

$$
\begin{aligned}
& \text { for each tuple } t_{1} \text { in } R \text { do } \\
& \text { for each tuple } t_{2} \text { in } \mathrm{S} \text { do } \\
& \quad \text { if } t_{1} \text { and } t_{2} \text { join then output }\left(t_{1}, t_{2}\right)
\end{aligned}
$$

What is the Cost?

## Nested Loop Joins

- Tuple-based nested loop $R \bowtie S$
- $R$ is the outer relation, $S$ is the inner relation

$$
\begin{aligned}
& \text { for each tuple } t_{1} \text { in } R \text { do } \\
& \text { for each tuple } t_{2} \text { in } \mathrm{S} \text { do } \\
& \quad \text { if } t_{1} \text { and } t_{2} \text { join then output }\left(t_{1}, t_{2}\right)
\end{aligned}
$$

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

What is the Cost?

- Multiple-pass since $S$ is read many times


## Page-at-a-time Refinement

for each page of tuples $r$ in $R$ do for each page of tuples $s$ in $S$ do<br>for all pairs of tuples $t_{1}$ in $r, t_{2}$ in $s$<br>if $t_{1}$ and $t_{2}$ join then output $\left(t_{1}, t_{2}\right)$

What is the Cost?

## Page-at-a-time Refinement

for each page of tuples $r$ in $R$ do for each page of tuples $s$ in S do
for all pairs of tuples $t_{1}$ in $r, t_{2}$ in $s$ if $t_{1}$ and $t_{2}$ join then output $\left(t_{1}, t_{2}\right)$

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


## Page-at-a-time Refinement



## Page-at-a-time Refinement



## Page-at-a-time Refinement

Disk

## Patient Insurance

| 12 | 2 | 4 | 6 | 6 |
| :---: | :---: | :---: | :---: | :---: |
|  | 4 | 3 | 1 | 3 |
| 9 6 | 2 | 8 |  |  |
| 8 5 | 8 | 9 |  |  |


| 1 | 2 | Input buffer for Patient |
| :--- | :--- | :--- |


| 2 | 8 | Input buffer for Insurance |
| :--- | :--- | :--- |

Keep going until read all of Insurance
Then repeat for next

## 22

Output buffer page of Patient... until end of Patient

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

## Block-Nested-Loop Refinement

for each group of $\mathrm{M}-1$ pages r in R do for each page of tuples $s$ in $S$ do<br>for all pairs of tuples $t_{1}$ in $r, t_{2}$ in $s$<br>if $t_{1}$ and $t_{2}$ join then output $\left(t_{1}, t_{2}\right)$

What is the Cost?

## Block-Nested-Loop Refinement

for each group of $\mathrm{M}-1$ pages $r$ in $R$ do for each page of tuples $s$ in $S$ do
for all pairs of tuples $t_{1}$ in $r, t_{2}$ in $s$
if $t_{1}$ and $t_{2}$ join then output $\left(t_{1}, t_{2}\right)$

- Cost: $B(R)+B(R) B(S) /(M-1)$


## 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 $B(S)+B(R)<=M$
- Typically, this is NOT a one pass algorithm


## Sort-Merge Join Example

Step 1: Scan Patient and sort in memory
Memory M = 21 pages

Disk
Patient Insurance

| 1 | 2 | 2 | 4 |  | 6 |
| :--- | :--- | :--- | :--- | :--- | :--- |
| 3 | 4 | 4 3 1 3 <br> 9 6  2 <br>  8   <br> 8 5  8 <br>  8   |  |  |  |


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

## Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory
Memory M = 21 pages
$\square$

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

## Sort-Merge Join Example

Step 3: Merge Patient and Insurance
Memory M = 21 pages

| 1 2 3 4 5 6 8 9 <br> 1 2 2 3 3 4 4 6 <br> 6 8 8 9    1 1 |
| :--- |
| \begin{tabular}{\|l|l|l|l|l|}
\hline
\end{tabular} |

## Sort-Merge Join Example

Step 3: Merge Patient and Insurance
Memory M = 21 pages

| 1 | 2 | 3 | 4 | 5 |  | 8 |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 2 | 2 | 3 | 3 | 4 | 4 | 6 |
| 6 | 8 | 8 | 9 | 22 |  |  |  |
| Output buffer |  |  |  |  |  |  |  |
| Keep going until end of first relation |  |  |  |  |  |  |  |

