Recommended Readings

• Join processing in database systems with large main memories. Leonard Shapiro. ACM Transactions on Database Systems 11(3), 1986. Also in Red Book (3rd and 4th ed)


• Database management systems. Ramakrishnan and Gehrke. Third Ed. Chapters 12, 13 and 14.
Outline

• Steps involved in processing a query
  – Logical query plan
  – Physical query plan
  – Query execution overview

• Operator implementations
  – One pass algorithms
  – Two-pass algorithms
  – Index-based algorithms
Example Database Schema

Supplier(sno, sname, scity, sstate)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, price)

View: Suppliers in Seattle
CREATE VIEW NearbySupp AS
SELECT sno, sname
FROM Supplier
WHERE scity='Seattle' AND sstate='WA'
Example Query

• Find the names of all suppliers in Seattle who supply part number 2

SELECT sname FROM NearbySupp
WHERE sno IN ( SELECT sno
  FROM Supplies
  WHERE pno = 2 )
Lifecycle of a Query (1)

• **Step 0: admission control**
  – User connects to the db with username, password
  – User sends query in text format

• **Step 1: Query parsing**
  – Parses query into an internal format
  – Performs various checks using catalog:
    Correctness, authorization, integrity constraints

• **Step 2: Query rewrite**
  – View rewriting, flattening, decorrelation, etc.
Original query:
SELECT sname
FROM NearbySupp
WHERE sno IN ( SELECT sno
                FROM Supplies
                WHERE pno = 2 )

Rewritten query:
SELECT S.sname
FROM Supplier S, Supplies U
WHERE S.scity='Seattle' AND S.sstate='WA'
AND S.sno = U.sno
AND U.pno = 2;

View rewriting
= view inlining
= view expansion

Flattening
= unnesting
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)
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)
Decorrelation

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)

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)
Decorrelation

Un-nesting

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

EXCEPT = set difference

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)

Supervisor(sno, sname, scity, sstate)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, price)
(SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA')
EXCEPT
(SELECT P.sno
FROM Supply P
WHERE P.price > 100)
Lifecycle of a Query (2)

• **Step 3: Query optimization**
  – Find an efficient query plan for executing the query
  – We will spend next lecture on this topic

• **A query plan is**
  – **Logical query plan**: an extended relational algebra tree
  – **Physical query plan**: with additional annotations at each node
Extended Algebra Operators

- Union $\cup$, intersection $\cap$, difference -
- Selection $\sigma$
- Projection $\pi$
- Join $\Join$
- Duplicate elimination $\delta$
- Grouping and aggregation $\gamma$
- Sorting $\tau$
- Rename $\rho$

Bag semantics!
Logical Query Plan

\[ \pi_{\text{sname}} \]

\[ \sigma_{\text{sscity} = 'Seattle' \land \text{sstate} = 'WA' \land \text{pno} = 2} \]

\[ \sigma_{\text{sno} = \text{sno}} \]

Suppliers \quad Supplies
Query Block

• Most optimizers operate on individual query blocks

• A query block is an SQL query with no nesting
  – Exactly one
    • SELECT clause
    • FROM clause
  – At most one
    • WHERE clause
    • GROUP BY clause
    • HAVING clause
Typical Plan For Block

\[ \sigma_{\text{having-condition}} \]

\[ \gamma \text{ fields, sum/count/min/max(fields)} \]

\[ \pi \text{ fields} \]

\[ \sigma_{\text{where-condition}} \]

\[ \text{join condition} \]

\[ \cdots \]

\[ \text{SELECT-PROJECT-JOIN Query} \]
Physical Query Plan

\[ \pi_{\text{sname}} \]

\[ \sigma_{\text{sscity}='Seattle' \land \text{sstate}='WA' \land pno=2} \]

Physical plan = Logical plan + choice of algorithms + choice of access path
Final Step in Query Processing

• **Step 4: Query execution**
  – How to synchronize operators?
  – How to pass data between operators?

• Standard approach:
  – Iterator interface and
  – Pipelined execution or
  – Intermediate result materialization
Iterator Interface

• Each **operator implements this interface**
• Interface has only three methods
• **open()**
  – Initializes operator state
  – Sets parameters such as selection condition
• **get_next()**
  – Operator invokes get_next() recursively on its inputs
  – Performs processing and produces an output tuple
• **close()**: clean-up state
Pipelined Execution

• Applies parent operator to tuples directly as they are produced by child operators

• Benefits
  – No operator synchronization issues
  – Saves cost of writing intermediate data to disk
  – Saves cost of reading intermediate data from disk
  – Good resource utilizations on single processor

• This approach is used whenever possible
Pipelined Execution

(On the fly) \( \pi_{sname} \)

(On the fly) \( \sigma_{sscity='Seattle'} \land sstate='WA' \land pno=2 \)

(Nested loop) \( \text{sno} = \text{sno} \)

Suppliers (File scan)

Supplies (Index lookup)
Intermediate Tuple Materialization

• Writes the results of an operator to an intermediate table on disk

• Necessary for some operator implementations
• When operator needs to examine the same tuples multiple times
Intermediate Tuple Materialization

(On the fly)

(Sort-merge join)

(Scan: write to T1)

(Scan: write to T2)

\[ \sigma_{\text{sscity}='Seattle' \ \land \text{sstate}='WA'} \]

\[ \pi_{\text{sname}} \]

\[ \sigma_{\text{pno}=2} \]

Suppliers (File scan)

Supplies (File scan)

Scan: write to T1

Scan: write to T2

Intermediate Tuple Materialization
Lifecycle of a Query

SQL query

Parse & Rewrite Query

Select Logical Plan

Select Physical Plan

Query Execution

Disk

Query optimization

Logical plan

Physical plan
Outline

• **Steps involved in processing a query**
  - Logical query plan
  - Physical query plan
  - Query execution overview

• **Operator implementations**
  - One pass algorithms
  - Two-pass algorithms
  - Index-based algorithms
Cost Parameters

• In database systems the data is on disk

• 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 $a$
  – $M = \#$ pages available in main memory

• Cost = total number of I/Os

• Convention: writing the final result to disk is *not included*
One-pass Algorithms

Selection $\sigma(R)$, projection $\Pi(R)$
- Both are *tuple-at-a-time* algorithms
- Cost: $B(R)$, the cost of scanning the relation
Main Memory Join Algorithms

Three standard main memory algorithms:

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

Review in class
One Pass Hash Join

Hash join:  \( R \bowtie S \)

- Scan \( R \), build buckets in main memory
- Then scan \( S \), probe hash table to join

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

- One pass algorithm when \( B(R) \leq M \)
Nested Loop Joins

• Tuple-based nested loop \( R \bowtie S \)
• \( R \) is the outer relation, \( S \) is the inner relation

\[
\text{for each tuple } r \text{ in } R \text{ do} \\
\quad \text{for each tuple } s \text{ in } S \text{ do} \\
\quad \quad \text{if } r \text{ and } s \text{ join then output } (r,s)
\]

• Cost: \( B(R) + T(R) \cdot B(S) \)
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
            if r and s join then output (r,s)

• Cost: B(R) + B(R)B(S)
Nested Loop Joins

• We can be much more clever

• How would you compute the join in the following cases? What is the cost?
  – $B(R) = 1000, B(S) = 2, M = 4$
  – $B(R) = 1000, B(S) = 3, M = 4$
  – $B(R) = 1000, B(S) = 6, M = 4$
Nested Loop Joins

- Block Nested Loop Join
- Group of (M-2) pages of S is called a “block”

```plaintext
for each (M-2) pages ps of S do
    for each page pr of R do
        for each tuple s in ps
            for each tuple r in pr do
                if r and s join then output(r,s)
```

Main memory hash-join

\[ ps \bowtie pr \]
Nested Loop Joins

Hash table for block of S (M-2 pages)

Input buffer for R

Output buffer

Join Result
Cost of block-based nested loop join

- Read S once: $B(S)$

- Outer loop runs $B(S)/(M-2)$ times, each iteration reads the entire R: $B(S)B(R)/(M-2)$

- Total cost: $B(S) + B(S)B(R)/(M-2)$

Notice: it is better to iterate over the smaller relation first
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
Example

Grouping:

Product(name, department, quantity)

γ_{department, sum(quantity)} (Product) → Answer(department, sum)

In class: describe a one-pass algorithms. Cost=?
Outline

• **Steps involved in processing a query**
  – Logical query plan
  – Physical query plan
  – Query execution overview

• **Operator implementations**
  – One pass algorithms
  – Two-pass algorithms
  – Index-based algorithms
Two-Pass Algorithms

• When data is larger than main memory, need two or more passes

• Two key techniques
  – Hashing
  – Sorting
Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
- Each bucket has size approx. B(R)/M

Does each bucket fit in main memory?
- Yes if B(R)/M <= M, i.e. B(R) <= M^2
Hash Based Algorithms for $\gamma$

- Recall: $\gamma(R) =$ grouping and aggregation

- Step 1. Partition $R$ into buckets
- Step 2. Apply $\gamma$ to each bucket

- Cost: $3B(R)$
- Assumption: $B(R) \leq M^2$
Simple Hash Join

\[ R \bowtie S \]

- **Step 1:**
  - \( P = \min( M-3, B(S) ) \)
  - Choose hash function \( h \) and set of hash values s.t. \( P \) blocks of \( S \) tuples will hash into that set
  - Hash \( S \) and either insert tuple into hash table or write to disk

- **Step 2**
  - Hash \( R \) and either probe the hash table for \( S \) or write to disk

- **Step 3**
  - Repeat steps 1 and 2 until all tuples are processed

Will skip in class
See [Shapiro]
Simple Hash Join

- Build a hash-table for M-3 pages of S
- Write remaining pages of S back to disk

Original relation S

Hash table for P blocks of S (M-3 pages)

Passed over tuples of S

Input buffer for S

Output buffer

Will skip in class
See [Shapiro]
Simple Hash Join

- Hash R using the same hash function
- Probe hash table for S or write tuples of R back to disk

- Repeat these two steps until all tuples are processed
- Requires many passes
Partitioned (Grace) 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
Partitioned Hash Join

- Partition both relations using hash fn $h$
- $R$ tuples in partition $i$ will only match $S$ tuples in partition $i$. 
Partitioned Hash Join

- Read in partition of R, hash it using h2 (≠ h)
  - Build phase
- Scan matching partition of S, search for matches
  - Probe phase

- Hash function
- Disk
- Input buffer for Ri
- Output buffer
- Hash table for partition Si (≤ M-1 pages)
- Partitions of R & S
- B main memory buffers
- Join Result
Partitioned Hash Join

- Cost: $3B(R) + 3B(S)$
- Assumption: $\min(B(R), B(S)) \leq M^2$
Hybrid Hash Join Algorithm

• Assume we have extra memory available

• Partition S into k buckets
  t buckets $S_1, \ldots, S_t$ stay in memory
  k-t buckets $S_{t+1}, \ldots, S_k$ to disk

• Partition R into k buckets
  – First t buckets join immediately with S
  – Rest k-t buckets go to disk

• Finally, join k-t pairs of buckets:
  $(R_{t+1}, S_{t+1}), (R_{t+2}, S_{t+2}), \ldots, (R_k, S_k)$
Hybrid Hash Join Algorithm

• How to choose $k$ and $t$?
  – The first $t$ buckets must fit in $M$: $t/k \cdot B(S) \leq M$
  – Need room for $k-t$ additional pages: $k-t \leq M$
  – Thus: $t/k \cdot B(S) + k-t \leq M$

• Assuming $t/k \cdot B(S) \gg k-t$: $t/k = M/B(S)$
Hybrid Hash Join Algorithm

• How many I/Os?

• Cost of partitioned hash join: $3B(R) + 3B(S)$

• Hybrid join saves 2 I/Os for a $t/k$ fraction of buckets

• Hybrid join saves $2t/k(B(R) + B(S))$ I/Os

• Cost: $(3-2t/k)(B(R) + B(S)) = (3-2M/B(S))(B(R) + B(S))$
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.

• Will discuss only 2-pass sorting, for when B < M^2
External Merge-Sort: Step 1

• Phase one: load M pages in memory, sort

![Diagram showing phase one of external merge-sort]

- Size M pages
- Main memory
- Disk
- Disk
- Runs of length M
  #Runs = B(R)/M
External Merge-Sort: Step 2

- Merge $M - 1$ runs into a new run
- Result: runs of length $M$ ($M - 1) \approx M^2$

If $B \leq M^2$ then we are done
External Merge-Sort

• Cost:
  – Read+write+read = 3B(R)
  – Assumption: B(R) <= M^2

• Other considerations
  – In general, a lot of optimizations are possible
Two-Pass Algorithms Based on Sorting

Grouping: \( \gamma_a, \sum(b) \) (R)

Sort, then compute the sum(b) for each group of a’s

- Step 1: sort chunks of size M, write
  - cost \( 2B(R) \)

- Step 2: merge M-1 runs, combining groups by addition
  - cost \( B(R) \)

- Total cost: \( 3B(R) \), Assumption: \( B(R) \leq M^2 \)
Join R \times S

- Start by creating initial runs of length M, for R and S:
  - Cost: 2B(R)+2B(S)
- Merge (and join) $M_1$ runs from R, $M_2$ runs from S:
  - Cost: $B(R)+B(S)$
- Total cost: $3B(R)+3B(S)$
- Assumption:
  - R has $M_1=B(R)/M$ runs, S has $M_2=B(S)/M$ runs
  - $M_1 + M_2 \leq M$
  - Hence: $B(R)+B(S)\leq M^2$
Outline

• **Steps involved in processing a query**
  – Logical query plan
  – Physical query plan
  – Query execution overview

• **Operator implementations**
  – One pass algorithms
  – Two-pass algorithms
  – Index-based algorithms
Review: Index Classification

- **Clustered/unclustered**
  - Clustered = records close in index are close in data
    - Option 1: Data inside data file is sorted on disk
    - Option 2: Store data directly inside the index (no separate files)
  - Unclustered = records close in index may be far in data

- **Primary/secondary**
  - Meaning 1:
    - Primary = is over attributes that include the primary key
    - Secondary = otherwise
  - Meaning 2: means the same as clustered/unclustered

- **Organization** B+ tree or Hash table
Clustered vs Unclustered

Every table can have **only one** clustered and **many** unclustered indexes.
Index Based Selection

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

• $V(R, a) = \# \text{ of distinct values of attribute } a$

• **Clustered index on } a: \text{ cost } B(R)/V(R,a)$

• **Unclustered index on } a: \text{ cost } T(R)/V(R,a)$

• Note: we ignored the I/O cost for the index pages (why?)
Index Based Selection

- Example:
  - Table scan (assuming R is clustered)
    - $B(R) = 2,000$ 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!
Index Based Selection

• Example:
  
  \[ B(R) = 2000 \]
  \[ T(R) = 100,000 \]
  \[ V(R, a) = 20 \]

  \[ \text{cost of } s_{a=v}(R) = ? \]

• Table scan (assuming R is clustered)
  
  – \( B(R) = 2,000 \) 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!

The 2% rule!

Note: the “2” in 2% decreases yearly (why?)
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:
  – Assuming R is clustered
  – If index on S is clustered: \( B(R) + T(R)B(S)/V(S,a) \)
  – If index on S is unclustered: \( B(R) + T(R)T(S)/V(S,a) \)
• Block Nested Loop Join: $B(R) + B(R)\cdot B(S)/M$

• Hybrid Hash Join: $(3-2M/B(S))(B(R) + B(S))$
  Assuming $t/k \cdot B(S) >> k-t$

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

• Index Nested Loop Join: $B(R) + T(R)B(S)/V(S,a)$
  Assuming $R$ is clustered and $S$ has clustered index on $a$