Lecture 6 – Lifecycle of a Query Plan
Announcements

• HW1 is due Thursday

• Projects proposals are due on Wednesday

• Office hour canceled today
  – CSE affiliates research talks!

<table>
<thead>
<tr>
<th>Session II</th>
<th>Big Data Management and Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:30 - 2:35pm</td>
<td>CSE 305</td>
</tr>
</tbody>
</table>
References


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, etc.
Rewritten Version of Our Query

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;
Lifecycle of a Query (2)

• **Step 3: Query optimization**
  – Find an efficient query plan for executing the query
  – We will spend a whole 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
    • Access method to use for each relation
    • Implementation to use for each relational operator
Extended Algebra Operators

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

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

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

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 (1/2)

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

\[ \sigma \text{ selection condition} \]

\[ \pi \text{ fields} \]

\[ \sigma \text{ selection condition} \]

\[ \text{join condition} \]

\[ \text{join condition} \]

\[ \ldots \]

\[ R \]

\[ S \]
Typical Plan For Block (2/2)

\[ \pi \text{ fields} \]
\[ \gamma \text{ fields, sum/count/min/max(fields)} \]
\[ \sigma \text{ selection condition} \]
\[ \text{join condition} \]
\[ \ldots \ldots \ldots \]
How about Subqueries?

```
SELECT   Q.name
FROM     Person Q
WHERE    Q.age > 25
         and not exists
         SELECT *
         FROM Purchase P
         WHERE P.buyer = Q.name
         and P.price > 100
```
How about Subqueries?

```
SELECT Q.name
FROM Person Q
WHERE Q.age > 25
and not exists
    SELECT *
    FROM Purchase P
    WHERE P.buyer = Q.name
    and P.price > 100
```
Physical Query Plan

• Logical query plan with extra annotations

• **Access path selection** for each relation
  – Use a file scan or use an index

• **Implementation choice** for each operator

• **Scheduling decisions** for operators
Physical Query Plan

(On the fly)

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

(Nested loop)

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

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

Suppliers (File scan)

Supplies (File scan)
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)

\( \pi_{\text{sname}} \)

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

(Nested loop)

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

Suppliers (File scan)  Suppliers (File scan)

CSE 544 - Fall 2015
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)

Suppliers
(File scan)

Supplies
(File scan)

(Scan: write to T2)

$\sigma_{\text{sscity}=\text{Seattle} \land \text{sstate}=\text{WA}}$

$\pi_{\text{sname}}$

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

$sno = sno$
Life cycle of a Query

1. SQL query
2. Parse & Rewrite Query
3. Select Logical Plan
4. Select Physical Plan
5. Query Execution
6. 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
Why Learn About Op Algos?

- Implemented in commercial DBMSs
- Different DBMSs implement different subsets of these 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:
  – $B(R) = \# \text{ of blocks (i.e., pages) for relation } R$
  – $T(R) = \# \text{ of tuples in relation } R$
  – $V(R, a) = \# \text{ of distinct values of attribute } a$
  – $M = \# \text{ pages available in main memory}$
Cost

• Cost of an operation = number of disk I/Os to
  – read the operands
  – compute the result

• Cost of writing the final result to disk is *not included*
  – Need to count it separately when applicable
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
Join Algorithms

• Logical operator:
  – Product(pname, cname) \(\bowtie\) Company(cname, city)

• Some well-known physical operators for the join, assuming the tables are in main memory:
  – Hash join
  – Nested loop join
  – Sort-merge join
Hash Join

Hash join:  \( R \bowtie S \)
- Scan \( R \), build buckets in main memory
- Then scan \( S \), 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 $R \Join S$
- $R$ is the outer relation, $S$ is the inner relation

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

- Cost: $B(R) + T(R) B(S)$
Page-at-a-time Refinement

\[
\text{for each page of tuples } r \text{ in } R \text{ do } \\
\quad \text{for each page of tuples } s \text{ in } S \text{ do } \\
\quad \quad \text{for all pairs of tuples } \\
\quad \quad \quad \text{if } r \text{ and } s \text{ join then output } (r, s)
\]

- 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)
```
Nested Loop Joins

R & S

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

Input buffer for R

Output buffer

Join Result
Nested Loop Joins

- Cost of block-based nested loop join
  - Read S once: cost $B(S)$
  - Outer loop runs $B(S)/(M-2)$ times, and each time need to read R: costs $B(S)B(R)/(M-2)$
  - 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
More One-pass Algorithms

Duplicate elimination $\delta(R)$

- Need to keep tuples in memory
- When new tuple arrives, need to compare it with previously seen tuples
- Balanced search tree or hash table
- Cost: $B(R)$
- Assumption: $B(\delta(R)) \leq M$
Even More One-pass Algorithms

Grouping:

\[ \text{Product(name, department, quantity)} \]

\[ \gamma_{\text{department, sum(quantity)}} (\text{Product}) \rightarrow \text{Answer(department, sum)} \]

How can we compute this in main memory?
Even More One-pass Algorithms

• Grouping: \( \gamma \text{department, sum(quantity)} \) (R)

• Need to store all departments in memory
• Also store the sum(quantity) for each department
• Balanced search tree or hash table
• Cost: B(R)
• Assumption: number of depts fits in memory
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

• What if data does not fit in memory?
• Need to process it in multiple 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 $\delta$

- Recall: $\delta(R) = \text{duplicate elimination}$

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

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

- Build a hash-table for M-3 pages of S
- Write remaining pages of S back to disk
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 \land 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 function $h$
- $R$ tuples in partition $i$ will only match $S$ tuples in partition $i$. 

CSE 544 - Fall 2015
Partitioned Hash Join

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

\begin{itemize}
  \item Read in partition of R, hash it using h2 \(\neq h\)
    - Build phase
  \item Scan matching partition of S, search for matches
    - Probe phase
\end{itemize}
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?
  – Choose k large but s.t. \( k \leq M \)
  – Choose \( t/k \) large but s.t. \( t/k \cdot B(S) \leq M \)
  – Moreover: \( t/k \cdot B(S) + k-t \leq M \)

• 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
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

Duplicate elimination $\delta(R)$

- Trivial idea: sort first, then eliminate duplicates
- Step 1: sort chunks of size $M$, write
  - cost $2B(R)$
- Step 2: merge $M-1$ runs, but include each tuple only once
  - cost $B(R)$
- Total cost: $3B(R)$, Assumption: $B(R) \leq M^2$
Two-Pass Algorithms Based on Sorting

Grouping: \( \gamma_a, \text{sum}(b) \) (R)

- Same as before: sort, then compute the sum(b) for each group of a’s
- Total cost: 3B(R)
- Assumption: B(R) <= M^2
Two-Pass Algorithms Based on Sorting

Join \( R \bowtie S \)

- Start by sorting both \( R \) and \( S \) on the join attribute:
  - Cost: \( 4B(R)+4B(S) \) (because need to write to disk)
- Read both relations in sorted order, match tuples
  - Cost: \( B(R)+B(S) \)
- Total cost: \( 5B(R)+5B(S) \)
- Assumption: \( B(R) \leq M^2, B(S) \leq M^2 \)
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: 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: $\sigma_{a=v}(R)$

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

• Clustered index on $a$: cost $B(R)/V(R,a)$

• Unclustered index on $a$: cost $T(R)/V(R,a)$

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

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

  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!
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: \( B(R) + T(R)T(S)/V(S,a) \)
Summary of External Join Algorithms

• Block Nested Loop Join: \( B(R) + \frac{B(R) \times B(S)}{M} \)

• Hybrid Hash Join: \( (3 - \frac{2M}{B(S)}) (B(R) + B(S)) \)
  Assuming \( \frac{t}{k} \times B(S) \gg k-t \)

• 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 \( R \) is clustered and \( S \) has clustered index on \( a \)
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 lecture: query optimization
  – How to compute the cost of a complete plan?
  – How to pick a good query plan for a query?