Architecture of a Database Engine

Logical Algebra Operators
- Union, intersection, difference
- Selection $\sigma$
- Projection $\Pi$
- Join $\bowtie$
- Duplicate elimination $\delta$
- Grouping $\gamma$
- Sorting $\tau$

Physical Operators
Will learn today and the following lectures:
- Join:
  - Main-memory hash based join
  - Block-based nested-loop join
  - Partitioned hash-based join
  - Merge-join
  - Index-join
- Group-by / Duplicate-elimination:
  - 

Question in Class
Logical operator:

Propose three physical operators for the join, assuming the tables are in main memory:
1.
2.
3.
Question in Class

Product(pname, cname) △◁ Company(cname, city)

- 1000000 products
- 1000 companies

How much time do the following physical operators take if the data is in main memory?

- Nested loop join: \( \text{time} = \) (cost parameters)
- Sort and merge = merge-join: \( \text{time} = \) (cost parameters)
- Hash join: \( \text{time} = \) (cost parameters)

Cost Parameters

The cost of an operation = total number of I/Os
- result assumed to be delivered in main memory

Cost parameters:

- \( B(R) \) = number of blocks for relation \( R \)
- \( T(R) \) = number of tuples in relation \( R \)
- \( V(R, a) \) = number of distinct values of attribute \( a \)
- \( M \) = size of main memory buffer pool, in blocks

Cost Parameters

- Clustered table \( R \):
  - Blocks consists only of records from this table
  - \( B(R) \ll T(R) \)
- Unclustered table \( R \):
  - Its records are placed on blocks with other tables
  - \( B(R) = T(R) \)

- When \( a \) is a key, \( V(R, a) = T(R) \)
- When \( a \) is not a key, \( V(R, a) \)

Selection and Projection

Selection \( \sigma(R) \), projection \( \Pi(R) \)
- Both are tuple-at-a-time algorithms
- Cost: \( B(R) \)

Selection and Projection

Hash Tables

- Key data structure used in many operators
- May also be used for indexes, as alternative to B+trees
- Recall basics:
  - There are \( n \) buckets
  - A hash function \( f(k) \) maps a key \( k \) to \( \{0, 1, \ldots, n-1\} \)
  - Store in bucket \( f(k) \) a pointer to record with key \( k \)
- Secondary storage: bucket = block, use overflow blocks when needed

Hash Table Example

- Assume 1 bucket (block) stores 2 keys + pointers
- \( h(e) = 0 \)
- \( h(b) = h(f) = 1 \)
- \( h(g) = 2 \)
- \( h(a) = h(c) = 3 \)

Here: \( h(x) = x \mod 4 \)
Searching in a Hash Table

- Search for a:
- Compute $h(a) = 3$
- Read bucket 3
- 1 disk access

| 0 | e ------------- |
| 1 | b ------------- |
| 2 | g ------------- |
| 3 | a ------------- |

Insertion in Hash Table

- Place in right bucket, if space
- E.g. $h(d) = 2$

| 0 | e ------------- |
| 1 | b ------------- |
| 2 | f ------------- |
| 3 | d ------------- |

Insertion in Hash Table

- Create overflow block, if no space
- E.g. $h(k) = 1$

| 0 | e ------------- |
| 1 | b ------------- |
| 2 | g ------------- |
| 3 | a ------------- |

- More overflow blocks may be needed

Hash Table Performance

- Excellent, if no overflow blocks
- Degrades considerably when number of keys exceeds the number of buckets (i.e. many overflow blocks).

Main Memory Hash Join

Hash join: $R \bowtie S$
- Scan $S$, build buckets in main memory
- Then scan $R$ and join

- Cost: $B(R) + B(S)$
- Assumption: $B(S) \leq M$

Duplicate Elimination

Duplicate elimination $\delta(R)$
- Hash table in main memory

- Cost: $B(R)$
- Assumption: $B(\delta(R)) \leq M$
Grouping

Grouping:
Product(name, department, quantity)
\( \gamma_{\text{department}, \text{sum(quantity)}} \) (Product) \( \rightarrow \)
Answer(department, sum)

Main memory hash table
Question: How?

Nested Loop Joins

• Tuple-based nested loop \( R \bowtie S \)

\begin{verbatim}
for each tuple r in R do
  for each tuple s in S do
    if r and s join then output (r, s)
\end{verbatim}

• Cost: \( T(R) B(S) \) when \( S \) is clustered
• Cost: \( T(R) T(S) \) when \( S \) is unclustered

Nested Loop Joins

• We can be much more clever

• Question: how would you compute the join in the following cases? What is the cost?
  - \( B(R) = 1000, B(S) = 2, M = 4 \)
  - \( B(R) = 1000, B(S) = 3, M = 4 \)
  - \( B(R) = 1000, B(S) = 6, M = 4 \)

Block-Based Nested-loop Join

\begin{verbatim}
for each (M-2) blocks bs of S do
  for each block br of R do
    for each tuple s in bs
      for each tuple r in br do
        if “r and s join” then output(r, s)
\end{verbatim}

Block-Based Nested-loop Join

• Cost:
  - 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
• \( R \bowtie S: \ R=\text{outer relation}, S=\text{inner relation} \)
Index Based Join

• $R \bowtie S$
• Assume $S$ has an index on the join attribute

\[
\text{for each tuple } r \text{ in } R \text{ do}
\]

lookup the tuple(s) $s$ in $S$ using the index
output $(r,s)$

Cost (Assuming $R$ is clustered):

• If index is clustered: $B(R) + T(R)B(S)/V(S,a)$
• If index is unclustered: $B(R) + T(R)T(S)/V(S,a)$

Index Based Selection

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

• Clustered index on $a$: cost $B(R)/V(R,a)$
• Unclustered index on $a$: cost $T(R)/V(R,a)$
  – We have seen that this is like a join

Example:

\[
\begin{align*}
B(R) &= 2000 \\
T(R) &= 100,000 \\
V(R, a) &= 20
\end{align*}
\]

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

• 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!

Operations on Very Large Tables

• Partitioned hash algorithms
• Merge-sort algorithms

Partitioned Hash Algorithms

• 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 $B(R)/M \leq M$, i.e. $B(R) \leq M^2$
Duplicate Elimination

- Recall: $\delta(R) =$ duplicate elimination
- Step 1. Partition R into buckets
- Step 2. Apply $\delta$ to each bucket (may read in main memory)
  - Cost: $3B(R)$
  - Assumption: $B(R) \leq M^2$

Grouping

- Recall: $\gamma(R) =$ grouping and aggregation
- Step 1. Partition R into buckets
- Step 2. Apply $\gamma$ to each bucket (may read in main memory)
  - Cost: $3B(R)$
  - Assumption: $B(R) \leq M^2$

Partitioned Hash Join

$R >\bowtie S$

- Step 1:
  - Hash S into M buckets
  - Send all buckets to disk
- Step 2
  - Hash R into M buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets
  - Cost: $3B(R) + 3B(S)$
  - Assumption: min($B(R)$, $B(S)$) $\leq M^2$

Hash-Join

- Partition both relations using hash fn $h$: R tuples in partition i will only match S tuples in partition i.
- Read in a partition of R, hash it using $h_2 (< h_1)$. Scan matching partition of S, search for matches.

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 bytes in memory, sort

External Merge-Sort: Step 2

• Merge M – 1 runs into a new run
• Result: runs of length M (M – 1) ≈ M²

Cost of External Merge Sort

• Read+write+read = 3B(R)
• Assumption: B(R) <= M²

Duplicate Elimination

Duplicate elimination δ(R)
• Idea: do a two step merge sort, but change one of the steps
• Question in class: which step needs to be changed and how?
• Cost = 3B(R)
• Assumption: B(δ(R)) <= M²

Grouping

Grouping: γₐ, 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) <= M²

Merge-Join

Join R ⋈ S
• Step 1a: initial runs for R
• Step 1b: initial runs for S
• Step 2: merge and join
Two-Pass Algorithms Based on Sorting

Join R \bowtie S

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

Summary of External Join Algorithms

- Block Nested Loop: $B(S) + B(R) \cdot B(S)/M$
- Index Join: $B(R) + T(R)B(S)/V(S,a)$
- Partitioned Hash: $3B(R) + 3B(S)$; $\min(B(R),B(S)) \leq M^2$
- Merge Join: $3B(R) + 3B(S)$; $B(R) + B(S) \leq M^2$

$M_1 = B(R)/M$ runs for R
$M_2 = B(S)/M$ runs for S
If $B \leq M^2$ then we are done