Query Execution
Query Compiler (Chapter 16) -> Query Plan

Metadata

Query Execution (Chapter 15) -> Data

Data
Parse Query

Select "logical" query plan

Select "physical" query plan

Execute Plan
Plan

• Mostly we consider various algorithms to implement the JOIN operator.

• Then, we look at other operators.

COST: # of disk blocks read (or written) to execute (we ignore CPU costs).
Example  \( R_1 \times R_2 \) over common attribute \( C \)

Number of tuples: 10000 (R1), 5000 (R2)
Size of each tuple = 1/10 block
Memory available = 101 blocks
Assume tuples packed into blocks.

→ Metric: # of IOs
    (ignoring writing of result)
Options

• Transformations: $R_1 \Join R_2$, $R_2 \Join R_1$

• Join algorithms:
  – Nested loops (Iteration)
  – Merge join
  – Join with index
  – Hash join
Nested Loop Join

for each \( r \in R1 \) do
  for each \( s \in R2 \) do
    if \( r.C = s.C \) then output \( r, s \) pair
Nested Join (Block wise)

for each block X of R1 do
for each block Y of R2 do
   Read X, Y.
   Take join of tuples in X and Y

Cost = 1000 + 1000 (500) = 501k
Better: Read R1 in Chunks

(1) Read 100 blocks of R1
(2) Read all of R2 (using 1 block) + join
(3) Repeat until done

Cost = 1000 + 10 (500) = 6000.

If whole of R1 can fit in main memory, then known as One-Pass Join Algorithm
Even Better: Reverse Join Order

Reverse join order: $R2 \bowtie R1$

Total $= 500 + 5 \times 1000 = 5500$ IOs
Options

• Transformations: $R_1 \bowtie R_2, \quad R_2 \bowtie R_1$

• Join algorithms:
  – Nested loops (Iteration)
  – Merge join
  – Join with index
  – Hash join
Merge Join

SORT R1 and R2 (if not already sorted)

i ← 1; j ← 1;
While (i ≤ T(R1)) ∧ (j ≤ T(R2)) do
    if R1{ i }.C = R2{ j }.C
        outputTuples (ignoring details here)
    elseif R1{ i }.C > R2{ j }.C
        j ← j+1
    elseif R1{ i }.C < R2{ j }.C
        i ← i+1
Cost of Merge Join

- Both R1, R2 ordered by C

Cost: $1000 + 500 = 1500$ IOs
How to Sort a Table – 1

(i) For each 100 blk chunk of R:
- Read chunk
- Sort in memory
- Write to disk
How to Sort a Table – 2.

(ii) Read all chunks + merge + write out
Cost for Sort

Each tuple is read, written, read, written

so...

Sort cost R1: 4 x 1,000 = 4,000
Sort cost R2: 4 x 500 = 2,000
Merge Join Cost (with Sorting)

Total cost = sort cost + join cost
  = 6,000 + 1,500 = 7,500 IOs

Cost is linear in sizes.
Thus, will beat nested-loop for large sizes.
Memory Requirements for Merge Sort

E.g: Say I have 10 memory blocks

\[ 10 \quad 100 \text{ chunks} \Rightarrow \text{to merge, need 100 blocks!} \]
In general:

Say $M$ blocks in memory

$x$ blocks for relation sort

# chunks = $(x/M)$  size of chunk = $M$

# chunks $\leq$ buffers available for merge

so... $(x/M) \leq M$

or $M^2 \geq x$  or  $M \geq \sqrt{x}$
In our example

R1 is 1000 blocks, $M \geq 31.62$
R2 is 500 blocks, $M \geq 22.36$

Need at least 32 buffers
Improving Merge Join

Do we really need the fully sorted files?

R1 → \{ \text{sorted runs} \} → \text{Join?} → \{ \text{sorted runs} \} → R2
Cost of improved merge join:

\[ C = \text{Read R1} + \text{write R1 into runs} + \text{read R2} + \text{write R2 into runs} + \text{join} \]
\[ = 2000 + 1000 + 1500 = 4500 \]

--> Memory requirement?
Options

- Transformations: R1 ⋈ R2,  R2 ⋈ R1

- Join algorithms:
  - Nested loops (Iteration)
  - Merge join
  - Join with index
  - Hash join
Index Join

For each $s \in R2$ do

\[
X \leftarrow \text{index}(R1, C, s.C)
\]

for each $r \in X$ do

output $r,s$ pair

Assume $R1.C$ index

Note: $X \leftarrow \text{index}(rel, attr, value)$ is equivalent to:
Set $X = \text{set of rel tuples with attr} = \text{value}$
Index-Join cost

B(R2) + T(R2)(probe-cost + retrieving matches)

- Index is small enough in memory.
  Probe-cost = 0.
- Index is big.
  E.g., 201 blocks of B-Tree, 2-levels.
  Store root and half of the leaves in MM.
  Avg. probe-cost = 1/2.
Expected # of matching tuples

• If R1.C is key, then expected # of matches = 1
• If number of distinct C values = 5000 and T(R1) = 10,000, then expected # of matches = 2 (assuming uniformity).
<table>
<thead>
<tr>
<th></th>
<th>Iterate R2 ( \times ) R1</th>
<th>55,000 (best)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not contiguous</td>
<td>Merge Join</td>
<td>_____________</td>
</tr>
<tr>
<td></td>
<td>Sort+ Merge Join</td>
<td>_____________</td>
</tr>
<tr>
<td></td>
<td>R1.C Index</td>
<td>_____________</td>
</tr>
<tr>
<td></td>
<td>R2.C Index</td>
<td>_____________</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Iterate R2 ( \times ) R1</th>
<th>5500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contiguous</td>
<td>Merge join</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td>Sort+Merge Join</td>
<td>7500 ( \rightarrow ) 4500</td>
</tr>
<tr>
<td></td>
<td>R1.C Index</td>
<td>5500</td>
</tr>
<tr>
<td></td>
<td>R2.C Index</td>
<td>_____________</td>
</tr>
</tbody>
</table>
Options

• Transformations: R1 \Join R2,  R2 \Join R1

• Join algorithms:
  – Nested loops (Iteration)
  – Merge join
  – Join with index
  – Hash join
Hash Join

- Hash function $h$, range $0 \rightarrow k$
- Buckets for $R1$: $G0, G1, \ldots, Gk$
- Buckets for $R2$: $H0, H1, \ldots, Hk$

Algorithm
(1) Hash $R1$ tuples into $G$ buckets
(2) Hash $R2$ tuples into $H$ buckets
(3) For $i = 0$ to $k$ do
    match tuples in $Gi, Hi$ buckets
Simple example

hash: even/odd

Buckets

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>14</td>
</tr>
</tbody>
</table>

Even:

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
</tr>
</tbody>
</table>

Odd:

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Even:</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Odd:</td>
<td>3</td>
<td>5</td>
<td>9</td>
</tr>
</tbody>
</table>
Example 1(f) Hash Join

R1, R2 contiguous (un-ordered)
→ Use 100 buckets
→ Read R1, hash, + write buckets

![Diagram showing hash join process]

10 blocks
-> Same for R2
-> Read one R1 bucket; build memory hash table
-> Read corresponding R2 bucket + hash probe

Then repeat for all buckets
Hash Join Cost

“Bucketize:”

Read R1 + write

Read R2 + write

Join:

Read R1, R2

Total cost = 3 \times [1000+500] = 4500

Note: this is an approximation since buckets will vary in size and we have to round up to blocks
Minimum memory requirements

Let, \( M \) = number of memory buffers
\( B \) = number of R1 blocks

For efficient hashing: Number of buckets < \( M \)
For efficient 2\(^{nd}\) stage: Size of a bucket < \( M \)

Size of R1 bucket = \((B/\text{number of buckets}) > B/M\)
\( B/M < M \quad \Rightarrow \quad M > \sqrt{B} \)

Need \( B+1 \) total memory buffers
Trick #1: keep some buckets in MM

E.g., 33 buckets; R1 Bucket = 31 blocks

Keep 2 R1 buckets in memory

Memory use:
- G0: 31 buffers
- G1: 31 buffers
- Output: 33-2 buffers
- R1 input: 1
- Total: 94 buffers

6 buffers to spare!!

called hybrid hash-join
Next: Bucketize R2

- R2 buckets = $\frac{500}{33} = 16$ blocks
- Two of the R2 buckets joined immediately with G0, G1
Finally: Join remaining buckets
– for each bucket pair:
  • read one of the buckets into memory
  • join with second bucket

memory

ans

out

one full R2 bucket

Gi

one R1 buffer

R2 buckets

16

33-2=31

R1 buckets

31

33-2=31
Cost

- Bucketize R1 = 1000 + 31 \times 31 = 1961
- To bucketize R2, only write 31 buckets:
  so, cost = 500 + 31 \times 16 = 996
- To compare join (2 buckets already done)
  read 31 \times 31 + 31 \times 16 = 1457

Total cost = 1961 + 996 + 1457 = 4414
How many buckets in memory?

See textbook for answer...
Trick #2: Store only <val, ptr> pairs in buckets

- Only write into buckets <val, ptr> pairs
- When we get a match in join phase, must fetch tuples
• To illustrate cost computation, assume:
  – 100 <val(ptr)> pairs/block
  – expected number of result tuples is 100

• Build hash table for R2 in memory
  5000 tuples $\rightarrow$ 5000/100 = 50 blocks

• Read R1 and match

• Read $\sim$ 100 R2 tuples

\[
\text{Total cost} = \begin{align*}
\text{Read R2:} & \quad 500 \\
\text{Read R1:} & \quad 1000 \\
\text{Get tuples:} & \quad 100 \\
& \quad \frac{1600}{1600}
\end{align*}
\]
So far:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterate</td>
<td>5500</td>
</tr>
<tr>
<td>Merge join</td>
<td>1500</td>
</tr>
<tr>
<td>Sort+merge joint</td>
<td>7500</td>
</tr>
<tr>
<td>R1.C index</td>
<td>5500</td>
</tr>
<tr>
<td>R2.C index</td>
<td>______</td>
</tr>
<tr>
<td>Build R.C index</td>
<td>______</td>
</tr>
<tr>
<td>Build S.C index</td>
<td>______</td>
</tr>
<tr>
<td>Hash join</td>
<td>4500+</td>
</tr>
<tr>
<td>with trick, R1 first</td>
<td>4414</td>
</tr>
<tr>
<td>with trick, R2 first</td>
<td>______</td>
</tr>
<tr>
<td>Hash join, pointers</td>
<td>1600</td>
</tr>
</tbody>
</table>
Summary

- Iteration ok for “small” relations (relative to memory size)
- If sorted, use merge-join.
- If index available, consider index-join.
- For equi-join, use hash join.
- Else, use sort + merge-join.

[Above, just guidelines]
Other Operations

- Duplicate Elimination, Aggregation
  - Keep one group/copy in memory
  - Use efficient main-memory data structure.

- Union, Intersection, Minus:
  - Either “one-pass”, or if MM is too small, use sorting, hashing, or index-based strategies.