QUERY PROCESSING (CHAPTER 12)

Software Architecture of a DBMS

TERM DEFINITION

- P(R): Number of pages that constitute R
- t(R): Number of tuples that constitute R
- ν(A,R): the number of unique values for attribute A of R
- min(A,R): the minimum value for attribute A of R
- max(A,R): the maximum value for attribute A of R
- P(I_R,A): the number of pages that constitute the B+-tree index on attribute A of R
- d(I_R,A): the depth of a B+-tree index on attribute A of R
- lp(I_R,A): the number of leaf pages for a B+-tree index on attribute A of R
- B(I_R,A): the number of buckets for a hash index on attribute A of R

HEAP FILE ORGANIZATION

- Assume a student table:  Student(name, age, gpa, major)
  - t(Student) = 16
  - P(Student) = 4
  - Bob, 21, 3.7, CS
  - Mary, 24, 3, ECE
  - Tom, 20, 3.2, EE
  - Kathy, 18, 3.8, LS
  - Kane, 19, 3.8, ME
  - Lam, 22, 2.8, ME
  - Chang, 18, 2.5, CS
  - Vera, 17, 3.9, EE
  - Louis, 32, 4, LS
  - Martha, 29, 3.8, CS
  - James, 24, 3.1, ME
  - Pat, 19, 2.8, EE
  - Shideh, 16, 4, CS

Non-Clustered Hash Index

- A non-clustered hash index on the age attribute with 4 buckets,
- h(age) = age % B

Clustered Hash Index

- A clustered hash index on the age attribute with 4 buckets,
- h(age) = age % B
Non-Clustered Secondary B+-Tree

- A non-clustered secondary B+-tree on the gpa attribute

Bob, 21, 3.7, CS
Mary, 24, 3.0, ECE
Tom, 20, 3.2, EE
Kathy, 18, 3.8, LS
Kane, 19, 3.8, ME
Lam, 22, 2.8, ME
Chang, 18, 2.5, CS
Vera, 17, 3.9, EE
Louis, 32, 4.0, LS
Martha, 29, 3.8, CS
James, 24, 3.1, ME
Pat, 19, 2.8, EE
Shideh, 16, 4.0, CS

Non-Clustered Primary B+-Tree

- A non-clustered primary B+-tree on the gpa attribute

Bob, 21, 3.7, CS
Mary, 24, 3.0, ECE
Tom, 20, 3.2, EE
Kathy, 18, 3.8, LS
Kane, 19, 3.8, ME
Lam, 22, 2.8, ME
Chang, 18, 2.5, CS
Vera, 17, 3.9, EE
Louis, 32, 4.0, LS
Martha, 29, 3.8, CS
James, 24, 3.1, ME
Pat, 19, 2.8, EE
Shideh, 16, 4.0, CS

Clustered B+-Tree

- A clustered B+-tree on the gpa attribute

Chad, 28, 2.3, LS
Mary, 24, 3.0, ECE
Tom, 20, 3.2, EE
Pat, 19, 2.8, EE

Clustered B+-Tree

- It is impossible to have a clustered secondary B+-tree on an attribute.

Cost of Performing Select(θ(R)) Operator

Exact match queries (θ = attribute A equals constant C):

- Heap: Scan the relation one page after another, until end of relation: \( P(R) \)
- Primary B+-tree: Use the constant to traverse the depth of the tree to the leftmost data record, begin to search forward: \( d(I) + \frac{P(R)}{\nu(A,R)} \)
- Secondary B+-tree: Locate the left most record using the constant, initiate the search using the records in the leaf nodes. For each index record that matches, perform a random I/O to the data file: \( d(I) + \frac{lp(I)}{\nu(A,R)} + \frac{t(R)}{\nu(A,R)} \)
- Hash index assuming a uniform distribution of tuples across pages: probe the hash index with the constant: \( \frac{P(R)}{B(I)} \)

Cost of Performing Select(θ(R)) Operator (Cont...)

Range queries (θ = attribute A greater than constant C):

Number of tuples that satisfy the selection predicate: \( \frac{t(\theta = A > C)}{\text{max}(A,R) - \text{min}(A,R)} \times t(R) \)
- Heap: Scan the relation one page after another, until end of relation: \( P(R) \)
- Primary B+-tree: Use the constant to traverse the depth of the tree to the leftmost data record, begin to search forward: \( d(I) + \frac{P(R)}{\nu(A,R)} \)
- Secondary B+-tree: Locate the left most record using the constant, initiate the search using the records in the leaf nodes. For each index record that matches, perform a random I/O to the data file: \( d(I) + \frac{lp(I)}{\nu(A,R)} + \frac{t(R)}{\nu(A,R)} \)
- Hash index assuming a uniform distribution of tuples across pages: probe the hash index with the constant: \( \frac{P(R)}{B(I)} \)

Estimating Number of Resulting Records

- For exact match selection predicates assume a uniform distribution of records across the number of unique values. E.g., the selection predicate is gpa = 3.3
- For range selection predicates assume a uniform distribution of records across the range of available values defined by min and max. In this case, one must think about the interval. E.g., gpa > 3.5

© 2013 Shahram Ghandeharizadeh
**ESTIMATING NUMBER OF RESULTING RECORDS (Cont...)**

- **Actual GPA Values**

**PROJECTION**

- The previous lecture described the selection operator and techniques to estimate both its costs and produced number of tuples. These techniques assume a uniform distribution of values within the minimum and maximum values of an attribute, uniform distribution of tuples across the buckets of a hash index.

- Projection (πᵣᵢ(R)) Logically, the system scans R and produces as output the unique values for attribute A (eliminates duplicates). The number of tuples produced by this operator is (πᵣᵢ(R)) with duplicate elimination and (R) without duplicate elimination. Without duplicate elimination, the cost of this operator is one scan of relation R; cost equals P(R). With duplicate elimination, its cost equals: sort(R) + P(R).

**EXTERNAL SORTING**

- Sort a relation that is larger than the available main memory.
- Consider a relation R with P(R) pages and w buffer pages (w ≥ 3).
- Assume that P(R)/w = 2ᵏ⁻¹.

**First approach, 2-way merge sort**

1. Read in, sort in main memory, and write out chunks of w pages. This step produces 2ᵏ⁻¹ sorted chunks of w pages.
2. Read in, merge in main memory, and write out pairs of sorted chunks of w pages. This produces 2ᵏ⁻² sorted chunks of w × 2 pages.
3. Read in, merge in main memory, and write out pairs of sorted chunks of w pages. This produces 2ᵏ⁻³ sorted chunks of w × 2² pages.
... k: Read in, merge in main memory, and write out pairs of sorted chunks of w × 2ᵏ⁻² pages. This produces 2.sorted chunks of w × 2ᵏ⁻¹ pages.

**Second approach, (w-1)-way merge sort**

1. Read in, sort in main memory, and write out chunks of w pages. This step produces (w-1) sorted chunks of w pages.
2. Read in, merge in main memory, and write out sets of (w-1) sorted chunks of w pages. This produces (w-1) × (w-1) sorted chunks of (w-1) w pages.
3. Read in, merge in main memory, and write out sets of (w-1) sorted chunks of (w-1) w pages. This produces (w-1)² sorted chunks of (w-1)² w pages.
... l: Read in, merge in main memory, and write out sets of (w-1) sorted chunks of (w-1)²⁻¹ w pages. This produces (w-1).sorted chunks of (w-1)²⁻¹ w pages.

**EXTERNAL SORTING (Cont...)**

- Example: Sort a 20 page relation assuming a five page buffer pool.

- Each pass requires reading and writing of the entire file. The algorithm’s I/O cost is:
  \[ \text{cost}(\text{2-way sort}) = 2 \times P(R) \times \log_2(P(R)/w) + 1 \]

- If P(R)/w is not a power of two then we take the ceiling of the logarithm.

**EXTERNAL SORTING (Cont...)**

- Second approach, (w-1)-way merge sort:
  1. Read in, sort in main memory, and write out chunks of w pages. This step produces (w-1) sorted chunks of w pages.
  2. Read in, merge in main memory, and write out sets of (w-1) sorted chunks of w pages. This produces (w-1) × (w-1) sorted chunks of (w-1) w pages.
  3. Read in, merge in main memory, and write out sets of (w-1) sorted chunks of (w-1) w pages. This produces (w-1)² sorted chunks of (w-1)² w pages.
  ... l: Read in, merge in main memory, and write out sets of (w-1) sorted chunks of (w-1)²⁻¹ w pages. This produces (w-1) sorted chunks of (w-1)²⁻¹ w pages.
EXTERNAL SORTING (Cont…)

- Merging \((w-1)\) temporary files is done by reading one page from each stream and simultaneously comparing the values of \((w-1)\) tuples, one from each stream.

- Example: Sort a 20 page relation assuming a five page buffer pool.

\[
\begin{align*}
&\text{Page 1:} & &\text{Page 2:} & &\text{Page 3:} & &\text{Page 4:} & \\
&\text{Page 5:} & &\text{Page 6:} & &\text{Page 7:} & &\text{Page 8:} & \\
&\text{Page 9:} & &\text{Page 10:} & &\text{Page 11:} & &\text{Page 12:} & \\
&\text{Page 13:} & &\text{Page 14:} & &\text{Page 15:} & &\text{Page 16:} & \\
&\text{Page 17:} & &\text{Page 18:} & &\text{Page 19:} & &\text{Page 20:} & \\
\end{align*}
\]

EXTERNAL SORTING (Cont…)

- Each pass requires reading and writing of the entire file. The algorithm's I/O cost is:

\[
\text{cost((w-1)way sort)} = 2 \times P(R) \times \left(\frac{\log_{w-1}(P(R)/w)}{w-1}\right) + 1
\]

- Example: How does the I/O cost of 2-way merge sort compares with \((w-1)\)-way merge sort for the 20 page relation?

- 2-way merge sort is more expensive in I/O, but it is simpler to implement. Overall it loses because the time attributed to performing I/O operations is more significant as compared to CPU execution time of a simple routine.

EQUALITY JOIN

- Estimated number of output tuples is:

\[
t(R) \times t(S) / \nu(A,R)
\]

- Two algorithms for performing the join operator: nested loops and merge-scan.

- Nested-loops:

  - Tuple nested loops:

    \[
    \begin{align*}
    &\text{for each tuple } r \text{ in } R \text{ do} & &\text{for each tuple } s \text{ in } S \text{ do} \\
    &\quad \text{if } r.A = s.A \text{ then output } r, s \text{ in the result relation} & &\quad \text{end-for} & &\text{end-for} \\
    &\quad \text{end-for} & &\text{end-for}
    \end{align*}
    \]

  - Estimated cost of tuple nested loops:

    \[
    P(R) + (t(R) \times P(S))
    \]

- Page nested loops:

  \[
  \begin{align*}
  &\text{for each page } rp \text{ in } R \text{ do} & &\text{fix } rp \text{ in the buffer pool} & &\text{for each page } sp \text{ in } S \text{ do} \\
  &\quad \text{for each tuple } r \text{ in } rp \text{ do} & &\text{for each tuple } s \text{ in } sp \text{ do} & &\text{if } r.A = s.A \text{ then output } r, s \text{ in the result} & &\text{end-for} & &\text{end-for} & &\text{end-for} & &\text{unfix } rp
  \end{align*}
  \]

- Estimated cost of page nested loops:

  \[
  P(R) + (P(R) \times P(S))
  \]

EQUALITY JOIN (Cont…)

- Merge-scan:

  1. Sort \(R\) on attribute \(A\)
  2. Sort \(S\) on attribute \(A\)
  3. Scan \(R\) and \(S\) in parallel, merging tuples with matching \(A\) values

  Estimated cost of merge-scan: \(\text{sort}(R) + \text{sort}(S) + P(R) + P(S)\)

- Merge-scan with alternative index structures:

  - Heap: \(\text{sort}(R) + \text{sort}(S)\)
  - Clustered hash index: \(\text{sort}(R) + \text{sort}(S) + \nu(A,R)\)
  - Non-clustered hash index: \(\text{sort}(R) + \text{sort}(S) + \nu(A,R)\)
  - Clustered \(B^+\)-tree: \(\text{sort}(R) + \nu(A,R) + \nu(A,S)\)
  - Non-clustered \(B^+\)-tree: \(\text{sort}(R) + \nu(A,R) + \nu(A,S)\)

EQUALITY JOIN (Cont…)

- Merge scan with alternative index structures. It is not important whether a relation is hashed or not. Assume \(w\) buffer pages. We sort a relation using either a 2-way or \((w-1)\)-way merge sort:

- Example: Assume the following statistics on the Employee and Department relations: \(t(\text{Dept})=1000\) tuples, \(P(\text{Dept})=100\) disk pages, \(\nu(\text{Dept},dno)=1000\), \(\nu(\text{Dept},dname)=500\). \(t(\text{Employee})=100,000\) tuples and \(P(\text{Employee})=10,000\) pages. Note that 10 tuples of each relation fit on a disk page. Assume that a concatenation of one Employee and one Dept record is wide enough to enable a disk page to hold five such records.

  Let's compare the cost of two alternative algebraic expressions for processing a query that retrieves those employees that work for the toy department:

  - \(\text{Emp} \bowtie \text{Dept} = \{\text{Emp} | \text{dname=Toy}\}\)
  - \(\text{Emp} \bowtie \text{Dept} = \{\text{Dept} | \text{dname=Toy}\}\)
EQUALITY JOIN (Cont…)

- $\sigma_{\text{dname}=\text{Toy}}(\text{Emp} \bowtie \text{Dept})$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Estimate</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{t(Tmp1)}$</td>
<td>$100,000$</td>
<td></td>
</tr>
<tr>
<td>$\text{p(Tmp1)}$</td>
<td>$100,000 / 5 = 20,000$</td>
<td></td>
</tr>
<tr>
<td>$\text{C(Tmp1)}$</td>
<td>$20,000$</td>
<td></td>
</tr>
<tr>
<td>$\text{C(D)}$</td>
<td>$20,000$</td>
<td></td>
</tr>
<tr>
<td>$\text{t(Tmp2)}$</td>
<td>$100,000 / 500 = 200$</td>
<td></td>
</tr>
<tr>
<td>$\text{p(Tmp2)}$</td>
<td>$200 / 5 = 40$</td>
<td></td>
</tr>
<tr>
<td>$\text{C(Tmp2)}$</td>
<td>$40$</td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td>$1,000,100 + 20,000 + 20,000 + 40$</td>
<td></td>
</tr>
<tr>
<td>$\text{C_{w}(\bowtie)}$</td>
<td>$20,000$</td>
<td></td>
</tr>
<tr>
<td>$\text{C_{w}(\text{Tmp1})}$</td>
<td>$1$</td>
<td></td>
</tr>
<tr>
<td>$\text{C(\text{Tmp1})}$</td>
<td>$1 + 10,000 = 10,001$</td>
<td></td>
</tr>
<tr>
<td>$\text{C_{w}(\text{Tmp2})}$</td>
<td>$40$</td>
<td></td>
</tr>
<tr>
<td>$\text{Cost} = \text{C(\bowtie)} + \text{C_{w}(\text{Tmp1})} + \text{C(\text{Tmp1})} + \text{C_{w}(\text{Tmp2})}$</td>
<td>$10,142$</td>
<td>I/O</td>
</tr>
</tbody>
</table>

EQUALITY JOIN (Cont…)

- $\sigma_{\text{dname}=\text{Toy}}(\text{Dept})$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Estimate</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{t(Tmp1)}$</td>
<td>$100,000$</td>
<td></td>
</tr>
<tr>
<td>$\text{p(Tmp1)}$</td>
<td>$100,000 / 5 = 20,000$</td>
<td></td>
</tr>
<tr>
<td>$\text{C(Tmp1)}$</td>
<td>$20,000$</td>
<td></td>
</tr>
<tr>
<td>$\text{C(D)}$</td>
<td>$20,000$</td>
<td></td>
</tr>
<tr>
<td>$\text{t(Tmp2)}$</td>
<td>$100,000 / 500 = 200$</td>
<td></td>
</tr>
<tr>
<td>$\text{p(Tmp2)}$</td>
<td>$200 / 5 = 40$</td>
<td></td>
</tr>
<tr>
<td>$\text{C(Tmp2)}$</td>
<td>$40$</td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td>$1,000,100 + 20,000 + 20,000 + 40$</td>
<td></td>
</tr>
<tr>
<td>$\text{C_{w}(\bowtie)}$</td>
<td>$100$</td>
<td></td>
</tr>
<tr>
<td>$\text{C_{w}(\text{Tmp1})}$</td>
<td>$1$</td>
<td></td>
</tr>
<tr>
<td>$\text{C(\text{Tmp1})}$</td>
<td>$1 + 10,000 = 10,001$</td>
<td></td>
</tr>
<tr>
<td>$\text{C_{w}(\text{Tmp2})}$</td>
<td>$40$</td>
<td></td>
</tr>
<tr>
<td>$\text{Cost} = \text{C(\bowtie)} + \text{C_{w}(\text{Tmp1})} + \text{C(\text{Tmp1})} + \text{C_{w}(\text{Tmp2})}$</td>
<td>$10,142$</td>
<td>I/O</td>
</tr>
</tbody>
</table>

The role of a query optimizer is to re-arrange operators of a query-tree to minimize the cost of executing a query. It employs heuristic search to compute a low-cost execution-plan:

1. Given a query plan, push the selection and projection operators down the query-tree. The system must be careful with the projection operator.
2. Combine two operators whenever possible, e.g., selection and join can be combined, projection and selection can be combined, projection and join can be combined.
3. Join is associative, consider different orderings of the join operators:
   - $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
   - Never compute the Cartesian product of two relations. For example if the join clause is $(R.A = S.A \& B = T.B)$ then it would be a mistake to use the following clause $(S \bowtie T \bowtie R)$ and $R.A = S.A \& B = T.B$
   - Always join an intermediate result with an original relation:
     - $(R \bowtie S) \bowtie T$ is Okay
     - $(R \bowtie S) \bowtie (R \bowtie T)$ is not Okay

© 2013 Shahram Ghandeharizadeh