Query Processing Strategies and Optimization

CPS352: Database Systems

Simon Miner
Gordon College
Last Revised: 10/25/12
Agenda

• Check-in
• Design Project Presentations
• Query Processing
• Programming Project
• Exam 1 (time permitting)
• Homework 4
Check-in
Design Project Presentations
Query Processing and Optimization
Different Ways to Execute Queries

- Database creates a plan to get the results for a query
  - Not just one way to do this.

- Example: Find the titles of all books written by Korth.
  - $\pi_{\text{title}} \sigma_{\text{author} = \text{'Korth'}} \text{Book} \mid X \mid \text{BookAuthor}$

- $\pi_{\text{title}} \text{Book} \mid X \mid \sigma_{\text{author} = \text{'Korth'}} \text{BookAuthor}$

- Good DBMS will transform queries to make them as efficient as possible
  - Minimize disk accesses
Selection Strategies

- Linear search – full table scan
  - Cost of potentially accessing each disk block containing the desired data

- Binary search (with B+ tree index)
  - Exact matches
  - Multiple matches
  - Range queries
  - Complex queries

- Index often requires disk accesses the index structure as well as for actual data
  - Typically far fewer for data than linear search
  - Index root and (perhaps) first level kept in buffer pool
# Query Type vs. Index Type

<table>
<thead>
<tr>
<th>Condition</th>
<th>Example</th>
<th>Clustering / Primary Index</th>
<th>Secondary Index</th>
<th>Hashed Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact match on candidate key id = 12345</td>
<td>Great!</td>
<td>Great!</td>
<td>Great!</td>
<td>Great!</td>
</tr>
<tr>
<td>Exact match on non-key status = ‘Active’</td>
<td>N/A</td>
<td>Find first match (+ potential scan)</td>
<td>Find first match (+ potential scan)</td>
<td></td>
</tr>
<tr>
<td>Range query age between 21 and 65</td>
<td>Find first match + sequential scan</td>
<td>Less helpful</td>
<td>Not useful</td>
<td></td>
</tr>
<tr>
<td>Complex query color = ‘blue’ or status = ‘Inactive’</td>
<td>Not useful</td>
<td>Not useful (multiple indexes help)</td>
<td>Not useful</td>
<td></td>
</tr>
</tbody>
</table>
Join Strategies

- Joins are most expensive part of query processing
  - Number of tuples examined can approach the product of the number of records in tables being joined

- Example
  - \( \sigma \text{Borrower.lastName} = \text{BookAuthor.authorName} \)  **Borrower X BookAuthor**
    - Where BookAuthor has 10K tuples and Borrower has 2K tuples
    - Cartesian join yields 20 million tuples to process
for (int i = 0; i < 2000; i++)
{
    retrieve Borrower[i];
    for (int j = 0; j < 10000; j++)
    {
        retrieve BookAuthor[j];
        if (Borrower[i].lastName ==
            BookAuthor[j].authorName)
            construct tuple from Borrower[i] &
            BookAuthor[j];
    }
}
for (int i = 0; i < 2000; i += 20) {
    retrieve block containing Borrower[i]..Borrower[i+19];
    for (int j = 0; j < 10000; j += 20) {
        retrieve block containing BookAuthor[j] ..
        BookAuthor[j+19];
        for (int k = 0; k < 19; k ++)
            for (int l = 0; l < 20; l ++)
                if (Borrower[i+k].lastName ==
                    BookAuthor[j+l].authorName)
                    construct tuple from Borrower[i+k] &
                    BookAuthor[j+l];
    }
}
Buffering an Entire Relation

```java
for (int i = 0; i < 2000; i += 20)
    retrieve and buffer block containing
    Borrower[i]..Borrower[i+19];

for (int j = 0; j < 10000; j += 20)
{
    retrieve block containing BookAuthor[j] ..
    BookAuthor[j+19];

    for (int k = 0; k < 2000; k ++)
        for (int l = 0; l < 20; l ++)
            if (Borrower[k].lastName ==
                BookAuthor[j+l].authorName)
                construct tuple from Borrower[k] &
                BookAuthor[j+l];

}
Using Indexes to Speed Up Joins

- **Example: Borrower \( \times \) CheckedOut**
  - Assume
    - 2K Borrower tuples, 1K CheckedOut tuples
    - 20 records per block (so 100 and 50 blocks for each table, respectively)
    - We cannot buffer either table entirely
  - Without indexes – nested block join takes 5050 or 5100 disk accesses, depending on which table is in the outer loop
  - With index on Borrower.borrowerID – exactly one match (PK)
    - Scan all 1000 CheckedOut records (50 blocks) – each matches exactly one Borrower record, which can be looked up in the index
      - Requires processing only 2000 tuples
    - Not quite as good as it seems
      - Each borrower may require a separate disk access (50 + 1000 = 1050 accesses)
      - Traversing index might take multiple disk accesses (especially B+ Tree indexes)
Temporary Indexes

- Indexes created and buffered for the purpose of a single query and then discarded
- Example: neither Borrower nor CheckedOut is indexed
  - Borrower |X| CheckedOut might cause a temporary index to be built on Borrower.borrowerID
  - If each (dense) index entry takes ~10 bytes, entire index will be ~20K
  - Index construction requires reading all 2K borrowers = 100 disk accesses
  - Join itself costs up to 1050 disk accesses (see previous slide)
  - Total of 1150 disk accesses
get first tuple from Borrower;
get first tuple from CheckedOut
while (we still have valid tuples from both relations)
{
    if (Borrower.borrowerID == CheckedOut.borrowerID)
    {
        output one tuple to the result;
        get next tuple from CheckedOut
        // We might have more checkouts for this borrower,
        // so keep current borrower tuple
    }
    else if (Borrower.borrowerID < CheckedOut.borrowerID)
        get next tuple from Borrower;
    else
        get next tuple from CheckedOut;
}
Order of Joins

• For multiple joins, performance can be greatly impacted by the order in which the joins are done

• Example
  • \( \pi \) last, first, authorName Borrower |X| BookAuthor |X| CheckedOut
  • Assume 2K borrowers, 1K CheckedOut records, and 10K authors
    • Each book has an average of 2 authors
  • 3 ways to do the (binary commutative) join operations
    • (Borrower |X| BookAuthor) |X| CheckedOut
    • (BookAuthor |X| CheckedOut) |X| Borrower
    • (Borrower |X| CheckedOut) \X| BookAuthor
  • Final number of tuples is the same, but intermediate joins create temporary tables which are then joined with the remaining table
    • Which way is most efficient in light of this?
Rules of Equivalence

- Two formulations of a query are equivalent if the produce the same set of results
  - Not necessarily in the same order

- Example: Find the titles of all books written by Korth.
  - `select title
    from Book natural join BookAuthor
    where authorName = 'Korth';`
  - Equivalent relational algebra queries
    - $\pi_{\text{title}} \sigma_{\text{author} = 'Korth'} \text{Book} \times \text{BookAuthor}$
    - $\pi_{\text{title}} \text{Book} \times \sigma_{\text{author} = 'Korth'} \text{BookAuthor}$
    - Equivalent, but the same in terms of performance
Equivalence Rules

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections.

\[ (E)_{1,2} = (E)_1 \cap (E)_2 \]

2. Selection operations are commutative.

\[ (E)_{1,2} = (E)_{2,1} \]

3. Only the last in a sequence of projection operations is needed, the others can be omitted.

\[ \Pi_{L_1} (\Pi_{L_2} (\ldots (\Pi_{L_n} (E)) \ldots)) = \Pi_{L_1} (E) \]

4. Selections can be combined with Cartesian products and theta joins.
   a. \( \sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2 \)
   b. \( \sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \land \theta_2} E_2 \)
Equivalence Rules (Cont.)

5. Theta-join operations (and natural joins) are commutative.
\[ E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1 \]

6. (a) Natural join operations are associative:
\[(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)\]

(b) Theta joins are associative in the following manner:
\[(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \land \theta_3} E_3 = E_1 \bowtie_{\theta_1 \land \theta_3} (E_2 \bowtie_{\theta_2} E_3)\]

where \(\theta_2\) involves attributes from only \(E_2\) and \(E_3\).
Equivalence Rules (Cont.)

7. The selection operation distributes over the theta join operation under the following two conditions:
   (a) When all the attributes in $\theta_0$ involve only the attributes of one of the expressions ($E_1$) being joined.

   $$\sigma_{\theta_0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_{\theta} E_2$$

   (b) When $\theta_1$ involves only the attributes of $E_1$ and $\theta_2$ involves only the attributes of $E_2$.

   $$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} (\sigma_{\theta_2}(E_2))$$
8. The projection operation distributes over the theta join operation as follows:

(a) if $\theta$ involves only attributes from $L_1 \cup L_2$:

$$
\Pi_{L_1 \cup L_2} (E_1 \bowtie_{\theta} E_2) = (\Pi_{L_1} (E_1)) \bowtie_{\theta} (\Pi_{L_2} (E_2))
$$

(b) Consider a join $E_1 \bowtie_{\theta} E_2$.

- Let $L_1$ and $L_2$ be sets of attributes from $E_1$ and $E_2$, respectively.
- Let $L_3$ be attributes of $E_1$ that are involved in join condition $\theta$, but are not in $L_1 \cup L_2$, and
- let $L_4$ be attributes of $E_2$ that are involved in join condition $\theta$, but are not in $L_1 \cup L_2$.

$$
\Pi_{L_1 \cup L_2} (E_1 \bowtie_{\theta} E_2) = \Pi_{L_1 \cup L_2} ((\Pi_{L_1 \cup L_3} (E_1)) \bowtie_{\theta} (\Pi_{L_2 \cup L_4} (E_2)))
$$
Equivalence Rules (Cont.)

9. The set operations union and intersection are commutative
   \[ E_1 \cup E_2 = E_2 \cup E_1 \]
   \[ E_1 \cap E_2 = E_2 \cap E_1 \]
   (set difference is not commutative).

10. Set union and intersection are associative.
    \[(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)\]
    \[(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)\]

11. The selection operation distributes over \(\cup\), \(\cap\) and \(-\).
    \[ \sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - \sigma_\theta(E_2) \]
    and similarly for \(\cup\) and \(\cap\) in place of \(-\).
    Also: \[ \sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - E_2 \]
    and similarly for \(\cap\) in place of \(-\), but not for \(\cup\).

12. The projection operation distributes over union
    \[ \Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2)) \]
Push Selections Inward

- Do selections as early as possible
  - Reduces ("flattens") the number of records in the relation(s) being joined

- Example:
  - \( \pi_{\text{title}} \sigma_{\text{author} = \text{‘Korth’}} \) \( \text{Book} \mid X \mid \text{BookAuthor} \)
  - \( \pi_{\text{title}} \text{Book} \mid X \mid \sigma_{\text{author} = \text{‘Korth’}} \text{BookAuthor} \)

- Sometimes this is not feasible
  - \( \sigma_{\text{Borrower.lastName} = \text{BookAuthor.authorName}} \) \( \text{Borrower} \times \text{BookAuthor} \)
  - i.e. when there are no shared attributes

- Alter the structure of the selection itself
  - Find late checked out books that cost more than $20.00.
    - \( \sigma_{\text{purchasePrice} > 20 \land \text{dateDue} < \text{today}} \) \( \text{Book} \mid X \mid \text{CheckedOut} \)
    - \( \sigma_{\text{purchasePrice} > 20} \text{Book} \mid X \mid \sigma_{\text{dateDue} < \text{today}} \text{CheckedOut} \)
Push Projections Inward

• Do projections as early as possible
  • Reduces ("narrows") the number of columns in the relation(s) being joined

• Example:
  • \( \pi \) lastName, firstName, title, dateDue \hspace{1cm} \text{Borrower} \mid X \mid \text{CheckedOut} \mid X \mid \text{Book} 
  • \( \pi \) lastName, firstName, title, dateDue \hspace{1cm} \text{Borrower} \mid X \mid 
    \hspace{1cm} (\pi \text{borrowerID, title, dateDue} \hspace{1cm} \text{CheckedOut} \mid X \mid \text{Book}) 
  • Reduces the number of columns in the temporary table from the intermediate join
Statistics and Query Optimization

- Using statistics about database objects can help speed up queries
- Maintaining statistics as the data in the database changes is a manageable process
- Types of statistics
  - Table statistics
  - Column statistics
Table Statistics

- On a relation $r$
- $n_r$ = number of tuples in the relation
- $b_r$ = number of blocks used by the relation
- $l_r$ = size (in bytes) of a tuple in the relation
- $f_r$ = blocking factor, number of tuples per block
  - Note that $f_r = \text{floor}(\text{block size} / l_r)$ if tuples do not span blocks
  - Note that $b_r = \text{ceiling}(n_r / f_r)$ if tuples in $r$ reside in a single file and are not clustered with other relations
Column Statistics

- on a column A

- \( V( A, r ) = \) number of distinct values in the column
  - If A is a superkey, then \( V( A, r ) = n_r \)
  - If A is not a superkey, the number of times each column value occurs can be estimated by \( n_r / V( A, r ) \)
  - If column A is indexed, \( V( A, r ) \) is relatively easy to maintain
    - Keep track of the count of entries in the index

- Histogram of the relative frequency of column values in different ranges
Estimating the Size of a Join

- **Cartesian product** – $r \times s$
  - Number of tuples in join = $n_r \times n_s = n_r \times n_s$
  - Size of each tuple in join = $l_r \times l_s = l_r + l_s$

- **Natural join** – $r \mid X \mid s$, where $r$ and $s$ have $A$ in common
  - The size of the join can be estimated in two ways
    - The $n_s$ tuples of $s$ will join with $n_r / V(A, r)$ tuples of $r$
      for $n_s \times n_r / V(A, r)$ total tuples
    - The $n_r$ tuples of $r$ will join with $n_s / V(A, s)$ tuples of $s$
      for $n_r \times n_s / V(A, s)$ total tuples
  - We want to use the smaller of these estimates
    - $\min(n_r \times n_s / V(A, s), n_s \times n_r / V(A, r)) = n_s \times n_r / \max(V(A, r), V(A, s))$  
    - Also note that $V(A, r \mid X \mid s) = \min(V(A, r), V(A, s))$
      - Some tuples in the relation with the larger number of column values do not join with any tuples in the other relation
Example Join Estimation

- \( \pi \) last, first, authorName \( \text{Borrower} \ | X | \text{BookAuthor} \ | X | \text{CheckedOut} \)

- 3 ways to do the join operations – Which is most efficient?
  - ( \( \text{Book} \ | X | \text{BookAuthor} \) ) \( X \) \( \text{CheckedOut} \)
  - ( \( \text{BookAuthor} \ | X | \text{CheckedOut} \) ) \( X \) \( \text{Borrower} \)
  - ( \( \text{Borrower} \ | X | \text{CheckedOut} \ | X | \text{BookAuthor} \)

- Statistics

<table>
<thead>
<tr>
<th>( n_r )</th>
<th>( V(A, r) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_{\text{Borrower}} = 2000 )</td>
<td>( V(\text{borrowerID, Borrower} ) = 2000 )</td>
</tr>
<tr>
<td>( n_{\text{CheckedOut}} = 1000 )</td>
<td>( V(\text{borrower, CheckedOut} ) = 100 )</td>
</tr>
<tr>
<td>( n_{\text{BookAuthor}} = 10,000 )</td>
<td>( V(\text{callNo, CheckedOut} ) = 500 )</td>
</tr>
<tr>
<td></td>
<td>( V(\text{callNo, BookAuthor} ) = 5000 )</td>
</tr>
</tbody>
</table>
Programming Project

Part I
Exam 1