INFSCI 2711: Advanced Topics in Database Management

Spring 2024

Homework #2: Distributed Query Optimization

1. Consider the following relations.

Tables	*Table is stored in the city:*
author (ID, Name, YearOfBirth, Gender, Country)	*Boston*
book (ID, Name, Type, YearPublished, PublisherID)	*NYC*
writes (BookID, AuthorID)	*LA*
publisher (ID, Name, Country)	*SF*

Specify SQL expressions, and describe **at least two strategies** for processing of each of the following query:
a) [10pt] Find the names of books that are published after 1995 in the USA.
b) [10pt] Find the names of books that are published in the author's country.
c) [5pt] for each query, explain advantages and disadvantages of each of described strategies. Explain how that benefit depends on **number of tuples for each table.**

Solution:

a) To Find the names of books that are published after 1995 in the USA.
SQL Query:

```
SELECT book.Name
FROM book
JOIN publisher ON book.PublisherID = publisher.ID
WHERE book.YearPublished > 1995 AND publisher.Country = 'USA';
```

Strategies:

i. **Local Filtering before Join**: Apply a filter on the `book` table to retrieve only records with `YearPublished > 1995`, and on the `publisher` table to retrieve only records where `Country = 'USA'` before performing the join.

ii. **Join then Filter:** Join the `book` and `publisher` tables first and then apply the filters.

b) Find the names of books that are published in the author's country.

SQL Query:

```
SELECT book.Name
FROM book
JOIN publisher ON book.PublisherID = publisher.ID
JOIN writes ON writes.BookID = book.ID
JOIN author ON writes.AuthorID = author.ID
WHERE author.Country = publisher.Country;
```

Strategies:

i. **Parallel Join:** Perform the join between `writes` and `book` and between `author` and `publisher` in parallel, then join these two results.

ii. **Sequential Join:** First join `writes` with `book`, then join this result with `author`, and finally with `publisher`.

c) For each query, explain advantages and disadvantages of each described strategy. Explain how that benefit depends on the number of tuples for each table.

- **Local Filtering before Join:** The benefits increase as the number of tuples in the unfiltered table grows because the cost of transferring and processing large amounts of data is reduced. However, if the filtering does not significantly reduce the size of the table, this strategy may not provide a benefit.

- **Join then Filter**: This approach can be beneficial when the tables are already indexed on the join keys, which can make the join operation faster. However, if the number of tuples is large, this can result in a large intermediate result set, which can negatively impact performance.

- **Parallel Join:** The advantages of this strategy become more pronounced as the size of the tables increases. Parallel processing can significantly reduce query time by distributing the load. However, this depends on the system's ability to effectively run operations in parallel.

- **Sequential Join:** This strategy can be more advantageous when dealing with smaller numbers of tuples because each join operation can be individually optimized. However, as the number of tuples grows, the intermediate result sets may become too large, which can make this strategy less efficient.

For the best performance, the query optimizer should consider the size of the tables, the selectivity of the filters, the distribution of data across different locations, and the availability of indexes on the columns involved in the join and where conditions.

2. Explain the difference between the query optimization objectives in a centralized and distributed database system.

Solution:

In centralized database systems, query optimization is primarily concerned with reducing the number of disk accesses to speed up data retrieval and processing. This involves strategies like efficient indexing and minimizing disk I/O through careful planning of joins and execution paths.
In distributed databases, however, the optimization must account for additional complexities. The cost of transferring data across the network often outweighs the cost of local disk accesses, necessitating strategies that minimize data movement. Additionally, distributed systems can leverage parallel processing, allowing multiple sites to work on different parts of a query simultaneously, thus enhancing performance. However, this must be balanced against the current load and capacities of each site to prevent bottlenecks. Data localization strategies also play a significant role, aiming to execute queries as close to the data's storage location as possible to reduce network load. Finally, the optimizer must also consider the overhead of distributed transaction management to maintain the integrity and consistency of the database.

3. Explain the difference between logical and physical query optimization.

Solution:

Logical and physical query optimization are two stages in the process of improving the efficiency of a query in a database system, each focusing on different aspects of the execution plan:

- Logical Query Optimization
  - It involves reorganizing the operations in a query to produce the same logical result more efficiently, without considering how the data is physically stored or the specific algorithms used to execute operations.
  - Includes the use of algebraic transformations such as predicate pushdown, join reordering, and join elimination. It is concerned with re-writing the query into a logically equivalent one that can be processed more efficiently.
  - The main goal is to reduce the complexity of the query by minimizing the size of intermediate results and the number of operations performed.
  - Logical optimization is generally independent of the underlying database system's physical architecture and can be applied across different systems.

- Physical Query Optimization
  - This stage considers the actual layout of the data on the storage media, the presence of indexes, the statistics of the data distribution, and the specific algorithms used for operations like joins and selections.
  - Involves decisions like choosing which index to use, whether to perform a table scan or use an index scan, selecting the type of join (nested loop, merge join, hash join), and determining the order of join operations.
  - To find the most efficient execution plan considering the database's physical characteristics and current state, which often involves estimating the cost of different execution plans in terms of resources like CPU, I/O, and memory usage.
  - Physical optimization is highly dependent on the specific database system, its architecture, and its performance characteristics.

In summary, Logical query optimization restructures a query at the abstract level to reduce complexity, using algebraic transformations like reordering joins and pushing down predicates to create a more efficient equivalent query. It is hardware-agnostic and focuses on the theoretical efficiency of the query structure. Physical query optimization, on the other hand, tailors the query plan to the specific physical characteristics of the database system, such as data distribution, indexing, and the cost of various operations like disk I/O. It focuses on choosing the most resource-efficient way to execute the query in the given database environment.

4. Consider joining two tables ***r*** and ***s*** with 200 and 600 tuples correspondingly. Assume that tables ***r*** and ***s*** take 50 and 150 blocks on the disk correspondingly. Estimate number of disk blocks transfers required to perform nested loop join for the best and worst cases (fill in the following table) and explain your answer.

Solution:

Scenario	Number of blocks, best case	Number of blocks, worst case
r is outer	**200**	**30050**
s is outer	**200**	**30150**

- **When r is the outer table,**

  Best Case

- Number of blocks for r: 50
- Number of blocks for s: 150
- Total block transfers: 50 + 150 = 200

  Worst Case

- Number of blocks for r: 50
- Number of blocks for s: 150
- Total block transfers: 200 * 150 + 50 = 30050

- **When s is the outer table,**

  Best Case

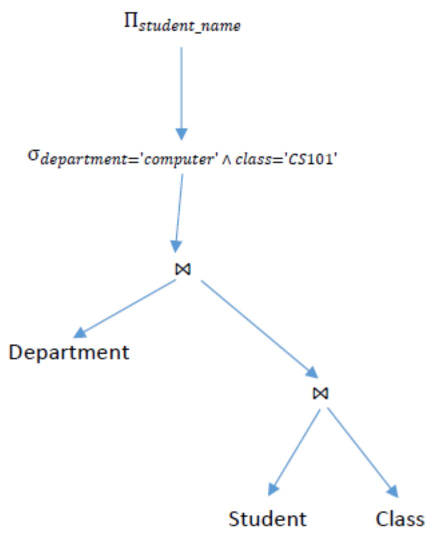
- Number of blocks for r: 50
- Number of blocks for s: 150
- Total block transfers: 50 + 150 = 200

  Worst Case

- Number of blocks for s: 50
- Number of blocks for r: 150
- Total block transfers: 600 * 50 + 150 = 30150

The best case assumes that the entire inner table can be kept in memory, thus it only needs to be read from disk once. The worst case assumes that the inner table cannot be kept in memory at all, necessitating a read from disk for every single block access of the outer table. The slight difference in the worst-case scenario between r as outer and s as outer is due to the differing block sizes of the two tables.

5. **[10 pts]** Find the Query Optimization of the following expression with the multiple transformation (include all steps).



__Solution:__

We have `Department`, `Student`, and `Class`. The operation is followed by a selection condition (σ) that filters results where `Department = 'computer'` and `Class = 'CS101'`, and then a projection (Π) that selects the `student_name` attribute.

step-by-step transformation that could optimize this query,

- Selection Pushdown: Move the selection operations as close to the leaf nodes as possible. This reduces the number of tuples early on, which makes the subsequent join operations cheaper. In this case, apply the filters on `Department` and `Class` before the joins.

- Join Reordering: If statistics or indexes indicate that one of the tables is much smaller after the selection pushdown or there is a more efficient way to join the tables, the order of the join operations might be adjusted to start with the smallest or most restrictive set to minimize the intermediate result size.

- Projection Pushdown: Just like selections, projections (removing unnecessary columns) should be pushed down as far as possible to reduce the width of the tuples being passed around.

Step-by-step query optimization:

- Apply the selection conditions directly to the `Department` and `Class` tables.
- Perform the joins, potentially starting with the table that has the fewest remaining tuples after the selection.
- Project the `student_name` from the result.
- Assuming `Department` and `Class` have been filtered:
  - The size of the `Department` table is reduced by selecting only 'computer'.
  - The size of the `Class` table is reduced by selecting only 'CS101'.

The join would then proceed with the smaller of the two tables. After joining these two tables, the result would then be joined with `Student`, and finally, the `student_name` would be projected from the result set.
The specific order of joins and the decision to push selections or projections first depend on the size of the tables and the selectivity of the filters. If indexes exist on the `Department` or `Class` attributes, they should be used to speed up the selection. If foreign key relationships are present and indexed, they could guide the order of the joins for further optimization.