Project

CIS611
SS Chung

Building a Simple Query Optimizer with
Performance Evaluation Experiment on Query Rewrite Optimization

This project is to simulate a simple Query Optimizer that:

1. **Evaluates query processing cost for a given SQL query and its optimized rewritten query.**
2. **Choose the best execution plan among them.**

You will design a simple optimizer that experiments performance evaluation for a given query and its optimized rewritten query by evaluating query processing cost for every possible query execution plan for the query and the rewritten query to choose the best plan at the end.

1. First, optimize the following query Q1 by rewriting it to RQ1.
Q1:

```
SELECT    T1.x1, SUM(T2.x2)
FROM      T1,T2
WHERE     T2.x2  = ( SELECT      SUM(T3.x3)
                     FROM        T3
                     WHERE       T1.x1 = T3.x3)
GROUP BY  T1.x1;
```

=>
RQ1

2. Find the optimal plan to process the query by evaluating query processing costs for all possible execution plans for each Q1 and RQ1 and comparing the costs to choose the minimum cost plan at the end.

Write a module Opt_Choose_Best_Plan() in Query Optimizer that chooses the best query processing plan for a given query based on Disk I/O calculation for each possible query execution plan by iterating all the different join orders and join methods.

3. Given Input of System Parameters and Table Info are as below:

- Assume that all the columns in tables are unsorted, with no index, no clustered, and uniformly distributed. No skew in data in each column.

- Page Size: 4096 Byte
- Block Size: 100 pages
- Table Size:
  T1: Each tuple is 20 bytes long, how many tuples per page, 1000 pages.
  T2: Each tuple is 40 bytes long, how many tuples per page, 500 pages.
  T3: **Each tuple is 100 bytes long, how many tuples per page, 2000 pages.**

- Predicate Selectivity for : T2.x2 = T3.x3 : 10%
  T1.x1 = T3.x3 : 20 %
  T1.Rowid = temp.rowid : 15 %
  temp join t3 : 15%
  t1 join t2 : Use 15% selectivity .

- Projection rate: All of project rates to 70%

We define the followings:

Query Processing Cost
= Disk I/O Cost + CPU Computation Cost (we ignore this cost here)

Disk I/O Cost
= Disk Access Time + Data Transfer Time

Disk Acess Time
= Seek Time + Latency

Assume Data Transfer rate = 45 MB/sec
Average Seek time = 8 ms
Average Latency = 4 ms
When we ignore CPU Computation Cost and Data Transfer Time here to make a process simpler,

Query Processing Cost = Disk I/O Cost
= # of Disk I/O * Disk Access Time
= # of Disk I/O * (8 ms + 4 ms)
= Total # of Disk Block access needed * 12ms

For each cost calculated at the end each loop, convert this cost to time in Hour/Min/Sec for print out.

2. Factors to consider to compute estimated Disk I/O for each plan and compare them in the loop to find the optimal one in the end.

2-1 Original Query Q1 vs Rewritten Query RQ1 :

- Assume Query Rewrite Module already rewrote Q1 to RQ1 and named RQ1 before entering Optimizer.
- Assume Parser already parsed each query and processed and stored it as sequenced operator steps in Query Execution tree.
- For this assignment, you only need to identify operators in query processing steps for each query Q1, RQ1 to calculate the processing cost

2-2 Join Methods

1) Tuple Nested Loop Join: TNL
2) Page Nested Loop Join: PNL
3) Block Nested Loop Join: with Buffer memory B =50: BNJM
4) Sort Merge Join with buffer memory B=50: SMJM
   – Assume to be optimized with Sort phase of outer and inner tables with merging phase for matching for join
5) Hash Join with Buffer memory requirement B=50 pages for hash table: HJM
6) Hash Join with less Buffer memory B= 30 pages : HJL
7) Block Nested Loop Join: with less Buffer memory B =30: BNJL
8) Sort Merge Join with less buffer memory B=30: SMJL
   – Assume to be optimized with Sort phase of outer and inner tables with merging phase for matching for join

2-3 Every possible Join Orders between t1, t2, t3, (or Temp)
(For 3 tables, there are 12 ways of Join Order, each costs differently)
Make an enumerating process, we will only consider ways to choose first 2 out of 3 then switching Left Table and Right Table for each order:
For Example:

t1 join (t2 join t3)
(t1 join t2) join t3
(t1 join t3) join t2
For t1 join (t2 join t3)
t1 join (t3 join t2)
(t2 join t3) join t1
(t3 join t2) join t1
For (t1 join t2) join t3
t3 join (t1 join t2)
…

3. Operators to Compute to Process each query Q1 and RQ1

- Join: Scan, Sort if necessary

- Projection : Scan, Sort + Duplicate Elimination
- Group By : Scan, Sort

Guideline of Procedure:

Input: Parameters given above, Query Execution Steps of Q1, Query Execution Steps of RQ1

Output: Each Query Plan(join order, join method) and Cost Computed
Final Query Plan and Best Cost

1. Data Structure for Query_Plan (Best_Plan, Curent_Plan) that has values indicating
1)Query: Q or RQ
2)Join Method
3)Join Order : Sequence of binary join of 3 tables
4)Query Cost computed.

2. Data Structure for Query Execution Step for Q1 and RQ1: Assume that this is coming in a Query Execution Tree form as input. To make the process simpler, instead of tree structure, you can make this as a Flat file with one operator with operands and other input per line in order of execution.

For Example,
A possible Query Execution Steps of Q1:

1 Join t1 t2 : any join methods for (temp1 <= t1 join t2)
2 Join temp1 t3 :
For each tuple of temp1: only for TNJ (because of correlation) with
two selectivities 10 % * 20 % for the result
temp2 ← temp1 join t3
on the predicates T2.x2 = T3.x3 : 10% and
T1.x1 = T3.x3 : 20 %

3 Project temp2 : temp3 ← temp2
4.Aggregate without Group By

4 GroupBy temp3 with Aggregation

For RQ:

RQ1:

- SELECT T1.x1, SUM(T2.x2)
  FROM T1,T2,
  (Select T1.Rowid, SUM(T3.x3)
  From T1, T3
  Where T1.x1 = T3.x3
  Group By T1.Rowid
) as Temp1 (rowid, x3)
  WHERE T2.x2 = Temp.x3 and T1.Rowid = Temp.rowid
  GROUP BY T1.x1;

This means:

1. Process the subquery (push T1 into subquery to join and group by first to save the result to Temp1), ➔ To rewrite (to apply Kim's decorrelation) the correlated subquery to noncorrelated Subq

2. Then join with the other outer T2, T1 (We need to join with Outer T1 again to project T1.x1 at the end)

Temp1 (rowid, x3) ← Select T1.Rowid, SUM(T3.x3)
From T1, T3
Where T1.x1 = T3.x3
Group By T1.Rowid ;

SELECT T1.x1, SUM(T2.x2)
FROM T1,T2,
WHERE T2.x2 = Temp.x3 and T1.Rowid = Temp.rowid
GROUP BY T1.x1;

A possible Query Execution Steps of RQ1:

A possible Execution Steps for that:
1 Join t1 t3 : any join methods for (temp1 ← t1 join t3) with
selectivity 20 % on T1.x1 = T3.x3 for the result temp0 ← t1 join t3

2.Group By temp0 on T1.Rowid with Aggregate ➔Result table called Temp1

3 Join t1 Temp1 : any join methods for (temp2 ← t1 join Temp1) with

selectivity 15 % on T1.Rowid = temp1.rowid for the result temp2

4. Join t2 temp2 : any join methods for (temp3 <= t2 join temp2) with selectivity 10 % on T2.x2 = temp2.x3 for the result temp3

5 Project temp3 : temp3 <= temp2
6. GROUP BY temp3. X1 with Aggregate

For example of calculation to estimate a temp table size resulting from a previous intermediate join result with Selectivity:

For Q1,
Because of correlation, this would be so called correlated tuple join, which is,
calculate scanning full t3 per the number of tuples in temp (<=t1 join t2),
For the size estimation of the result of this correlation join, you need a selectivity on temp join t3. I see I only gave 15% selectivity for t1 join temp
Use the same selectivity 15% for temp join t3 as well !
Use 15% selectivity for t1 join t2 as well.

For RQ,
1) temp 1 <= t1 Join t3
2) temp2 <= temp 1 join t1 (for this 15% selectivity is given)

For each operator given above (join, project, group by), write a Cal_Operator_Function that calculates cost of the operator. For example, for Join operator a function like Cal_Join_Func that takes sizes of left, right tables, join method, etc as input then computes join processing cost using left and right tables, the given join method so that it can be called inside loop for each possible combination of Join Order with Join Method.

For example, a function for join
Cal_Binary_Join_Func (LeftTableSize, RightTableSize, JoinMethod)
{ 1. You can use Case statement for Join Methods for example SMJ
Calculate Disk I/O cost to Join LeftTable and RightTable using Join Method = SMJ
2. Store the cost
}

Opt_Choose_Best_Plan()

{

1. Preprocessing before this module Opt_Choose_Best_Plan() :
   1-1 Rewrite the query Q1 to an optimized one RQ1 if there is.
   1-2 Assume that Query Rewriting Module transformed Q1 to RQ1. (Show rewritten query from Q1 to RQ1 manually here.)
   1-3 For each query Q1, RQ1, identify query processing steps to identify all the operators and operands needed to execute the query in terms of the operators given above. Ignore the rest other than those 4 operators above.

1-4 Assume all the execution steps with operators and operands are processed and stored in Query Execution Tree as data structure under the name Q1 and RQ1 each.
Show query execution steps needed for Q1 and RQ1 (Write down as Comments before Program)

2. Initialization of Best_Cost, Current_Cost, Best_Plan, Current_Plan… all the others.

3. For each Query (Original Q1 or Rewritten RQ1)
{
Initialization
For each possible Join methods and Order of each binary Join for all tables in the query
{
Initialization for Current_Plan and Current_Cost
1. Compute Query Processing Cost to Current_Cost
1-1 Estimate Expected size of Temp tables or tables of each binary join
1-2 Calculate Query Processing Cost in Disk I/O by calling calculating routines

2. Print Out Query Name (Q1 or RQ1), current Join Method, Join Order used to compute this current cost, Current cost in hh/mm/sec

3. if (Best_Cost > Current_Cost just computed)
{
Make Current_Cost as Best_Cost
Make Current_Plan as Best_Plan
}
}

}

4. Print Out the Best_ Plan(Query Name, Join Method, Join Order) and Best_Cost as the final query processing plan

}