Optimization of Large Join Query using Heuristic Greedy Algorithm

Vishal P. Patel, Hardik R. Kadiya
1. PG Student of Computer Department, Merchant Engineering College, Basna, Gujarat, India
2. Faculty of Computer Department, Merchant Engineering College, Basna, Gujarat, India

Abstract - SQL statements are used to retrieve data from the database. One can get the same results by writing different SQL queries. But use of the best query is important when performance is to be considered. So you need to use SQL query tuning based on the requirement. As queries are stated in non-procedural manner, the need of optimizer arises that transform the straight forward translation of a query into a cost-effective evaluation plan. Due to their high evaluation costs, joins are a primary target of query optimizers. Use of Heuristic to cut-down alternatives and Greedy Algorithm for finding suboptimal solution in less time rather than using dynamic programming and branch and bound to find optimal solution in more time than greedy algorithm.

Keywords - Query Optimization, Relational Databases Cost-Based optimization, Heuristic greedy algorithm, Hybrid query optimization.

1. Introduction

First the parsed query must pass the query transformer inside the optimizer the query transformer rewrite the query using heuristic like[2]

(1) Perform selection and projection as early as possible
(2) Predicate pushdown
(3) Subquery unnesting

Example: Select A.id, u from A where A.id in (select Aid from B where V=0); Can be transformed in to query using join method Select A.id, u from A inner join B on (A.id=B.id) where v=0;

Next the transformed query passes the estimator. This section estimates the costs (e.g. number of rows) of different operation which might be relevant to execute the query. Therefore a dictionary (which contains statistical information about the data) can be used to be able to estimate the number of rows which match a where clause. Finally, the plan generator is determining the expected optimal query plan to execute the query.

2. Database Statistics

Success of estimation depends on statistical information DBMS holds. Keeping statistics current can be problematic. If statistics updated every time tuple is changed, this would impact performance, so DBMS provide statistical information on base data. DBMS could update statistics on a periodic basis, for example nightly, or whenever the system is idle. This could lead to inaccurate estimates. Query optimization tries to find best possible plan within a minimum amount of time using mostly semi accurate statistical information.

3. Join Scheduling for Query Optimization

Algorithm applied to explore search space and determine the best query execution plan (QEP) based on join selectivity

Join selectivity = ratio of the number of tuples in the result/number of tuples in the Cartesian product.

Classes of strategies solve problem of join scheduling[4]

Deterministic Strategy: Starting from base relation, joining one or more relations at each step till complete plans are obtained.
Randomized Strategies: These strategies do not guarantee optimal plan but they avoid high cost optimization in terms of memory and time consumption. Example: Iterative improvement, Simulated Annealing.

4. Explain Statement in SQL

An explain statement in SQL is a statement which consists of the keyword “explain” followed by a select statement. The select statement will then be parsed by the parser of the RDBMS. Afterwards, the optimizer will decide which one is the expected optimal query plan to execute the query.

Example: Explain select A.id, B.id, u, v from A inner join B on (A.id=B.Aid)[5]

QUERY PLAN

Hash Join (cost=1.07 ... 2.18 rows=3 width=6)
Hash Cond : (b.aid=a.id)
→ Seq scan on b (cost = 0.00...1.07 width=8)
   Filter: (p=0)
→ Hash (cost=1.03...1.03 rows=3 width=6)
→ Seq scan on a (cost=0.00...1.03 rows=3 width=6)

5. Shapes of join Query Tree

![Shapes of join Query Tree](image)

Number of possible join tree for a given n relation is determined by using \(C_n, (C_{(n-1)}*n)\) where \(C_n\) is the Catalan number that is satisfied using following formula

\[C_n = \frac{(2n)!}{((n+1)!*n!)}\]

Require to proceed further by using Left-deep tree is preferable because by using Left-deep tree we required to consider only \(n!\) possible tree where \(n\) is the number of relation in a query. By using Left-deep tree no require to store intermediate result and pipeline can be possible. In left-deep tree all leaf node as a relation and all intermediate node as a join operator.

<table>
<thead>
<tr>
<th>Number of Relations</th>
<th>Catalan Number</th>
<th>Join Trees (C_n)</th>
<th>Left Deep Trees</th>
<th>Table 1. Number of Possible Trees [8]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>12</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>1680</td>
<td>24</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>42</td>
<td>30240</td>
<td>720</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>137</td>
<td>166590</td>
<td>5040</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>425</td>
<td>17297280</td>
<td>40320</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>4952</td>
<td>1784325600</td>
<td>3638800</td>
<td>12</td>
</tr>
</tbody>
</table>

5.1 Transformation rules for Join operation[9]:

- Join Method choice: \(A \bowtie_{method} B\) \(\Rightarrow A \bowtie_{method} B\)
- Join Commutative: \(A \bowtie B \Rightarrow B \bowtie A\)
- Join Associativity: \((A \bowtie B) \bowtie C \Rightarrow A \bowtie (B \bowtie C)\)
- Left Join Exchange: \((A \bowtie B) \bowtie C \Rightarrow (A \bowtie C) \bowtie B\)
- Right Join Exchange: \((A \bowtie B) \bowtie C \Rightarrow A \bowtie (B \bowtie C)\)
- Swap: exchange the positions of two arbitrary positions in the sequence \(R_1 R_2 R_3 R_4 R_5 \Rightarrow R_1 R_4 R_3 R_2 R_5\)
- 3-cycle: cyclic rotation of three arbitrary position in the sequence \(R_1 R_2 R_3 R_4 R_5 \Rightarrow R_5 R_2 R_1 R_4 R_3\)

5.2 Shapes of Query[10]

1) Star Query: In star query there is a subset of attributes common to all relations in the query

2) Chain Query: In chain query all except two queries have common attribute with exactly two other relations but the first and last relations have attributes in common with only one other query. Each attribute is common to at most two relations.

3) Circular Query: In a circular query, each query has common attribute with exactly two other relations and each attribute is common to at most two relations.

4) Clique query: In a clique Query, every pair of relation has a unique subset of common attributes. Proceed further by using chain query is preferable because most database operation of type chain query

Fig. 2: Shapes of join Query Tree[6]
6. Experiments & Results

6.1 Example (Cartesian Join):

```
SELECT classroom.capacity, course.dept_name, department.dept_name
FROM classroom, course, department
WHERE classroom.capacity = 10 AND department.dept_name = 'english';
```

![Diagram of Initial Query Tree & Using Algorithm](image)

Table 2. Database Statistics (Cartesian Join Example)

<table>
<thead>
<tr>
<th>Table Name</th>
<th>No of Rows</th>
<th>Avg Row Length</th>
<th>Table Size</th>
<th>Affected Rows</th>
<th>Result Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Department (T0)</td>
<td>20</td>
<td>30</td>
<td>600</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>Classroom (T1)</td>
<td>30</td>
<td>20</td>
<td>600</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>Course (T2)</td>
<td>200</td>
<td>37</td>
<td>7400</td>
<td>200</td>
<td>7400</td>
</tr>
</tbody>
</table>

Cost of Initial Query Tree [Space usage] =

\[
A \cdot \text{Cost of crossjoin T1 and T2} + B \cdot \text{Cost of crossjoin A and T0} + C \cdot \text{Number of rows after apply selection on B}
\]

\[
= A \cdot ([30 \cdot 200] \cdot [20+30]) + B \cdot ([6000 \cdot 20] \cdot [57+30]) + C \cdot [1000 \cdot 87]
\]

\[
= A \cdot [342000] + B \cdot [10440000] + C \cdot [87000]
\]

Cost of Query Tree by Algorithm [Space usage] =

\[
A \cdot \text{Cost of applying selection on T0} + B \cdot \text{Cost of applying selection on T1} + C \cdot \text{Cost of apply cross join on A and B} + D \cdot \text{cost of apply cross join on C and T2}
\]

\[
= A \cdot [1 \cdot 30] + B \cdot [5 \cdot 20] + C \cdot [1 \cdot 5] \cdot [30+20] + D \cdot [5 \cdot 20] \cdot [50+37]
\]

\[
= A \cdot [30] + B \cdot [100] + C \cdot [250] + D \cdot [87000]
\]

6.2 Example (Equi Join):

```
SELECT T1.dept_name, T2.dept_name, T1.capacity, T1.building, T1.room_number, T2.dept_name, T2.course_id
FROM classroom, course, department
WHERE classroom.capacity = 10 AND department.dept_name = 'english' AND T1.building = T2.building AND T1.dept_name = T2.dept_name
```

![Diagram of Initial Query Tree Using Algorithm](image)

Table 3. Database Statistics (Equi Join Example)

<table>
<thead>
<tr>
<th>Table Name</th>
<th>No of Rows</th>
<th>Avg Row Length</th>
<th>Table Size</th>
<th>Affected Rows</th>
<th>Result Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student (T0)</td>
<td>2000</td>
<td>29</td>
<td>58000</td>
<td>12</td>
<td>348</td>
</tr>
<tr>
<td>Classroom (T1)</td>
<td>30</td>
<td>20</td>
<td>600</td>
<td>30</td>
<td>600</td>
</tr>
<tr>
<td>Department (T2)</td>
<td>20</td>
<td>30</td>
<td>600</td>
<td>20</td>
<td>600</td>
</tr>
</tbody>
</table>

Cost of Initial Query Tree [Space usage] =

\[
A \cdot \text{Cost of crossjoin T1 and T2} + B \cdot \text{Cost of crossjoin A and T0} + C \cdot \text{Number of rows after apply selection on B}
\]

\[
= A \cdot ([30 \cdot 20] \cdot [20+30]) + B \cdot ([50 \cdot 2000] \cdot [50+29]) + C \cdot [1 \cdot 0+79]
\]

\[
= A \cdot [30000] + B \cdot [7900000] + C \cdot [790]
\]

Cost of Query Tree by Algorithm [Space usage] =

\[
A \cdot \text{Cost of applying selection on T0} + B \cdot \text{Cost of applying join on T2 and A} + C \cdot \text{Cost of applying join on B and T1}
\]

\[
= A \cdot [12 \cdot 29] + B \cdot [12 \cdot 29+30] + C \cdot [10 \cdot 29+20+30]
\]

\[
= A \cdot [348] + B \cdot [708] + C \cdot [790]
\]

\[\text{SELECT room_number, budget, name FROM classroom, department, student WHERE classroom.building = department.building AND department.dept_name = student.dept_name AND tot_cred = 4}\]
SELECT ROOM_NUMBER, BUDGET_NAME from ( Select NAME,ID from STUDENT where (tx_cost = 4) in T1 ) ( Select ROOM_NUMBER,BUILDING from CLASSROOM ) as T1, ( Select BUDGET,DEPT_NAME from DEPARTMENT ) as T2 where (T1.BUILDING = T2.BUILDING) and (T1.DEPT_NAME = T2.DEPT_NAME)

Fig.4: Initial Query Tree & Using Algorithm

7. Conclusions and Future Work

Reduce intermediate result size ultimately reduces the execution time. Experiment result shows Combine Heuristic and Greedy approach provide better performance for optimization of Large join query. Optimization only on select-project-join queries also need to handle complex queries (e.g. Top-k query, Group by, aggregations, materialized view, recursive query, dynamic query). Optimize query on centralized system can be extended to implement on parallel database and Distributed Database.

References


[14] CJ.Date, “An Introduction to database systems”Addison wisely


Mr. Vishal P. Patel, He graduated from L.C institute of technology Bhandu (Affiliated to H.N.G.U Patan), presently pursuing M.Tech. in computer engineering at MEC.Basna affiliated to Gujarat technological university, Gujarat, India. His interested area in object-oriented systems, database and computer algorithms

Mr. Hardik R. Kadiya, He is Asst.Professor in Merchant engineering college,Basna,Gujarat,India. He received bachelor degree in information technology from government engineering college,Modasa,Gujarat,India. And M.E from merchant engineering college,affiliated to Gujarat technological universityand his area of interest are image processing,database and computer network.