Efficient Query Evaluation Using Joins in Relational Database

Dr. K V S N Jawahar Babu¹, V. Harshavardhan², J.S. Ananda Kumar³
Department of MCA, K M M Institute of Post Graduate Studies, Tirupati (A.P.), India.

Abstract—This paper will introduce the reader to the basic concepts of query processing and query optimization in the relational database domain. How a database processes a query as well as some of the algorithms and rule-sets utilized to produce more efficient queries will also be presented. This is responsible for translating a user submitted query usually written in a non-procedural language into an efficient query evaluation program that can be executed against database.

Keywords—Database, Equivalence, Join, Optimization, Query Processing, Relational Algebra

I. INTRODUCTION

In a relational database all information can be found in a series of tables. A query therefore consists of operations on tables for retrieving the data from tables. The most common queries are Select-Project-Join queries

Here a list of the most commonly performed operations:

• Select (σ): Returns tuples that satisfy a given predicate
• Project (π): Returns attributes listed
• Join (⋈): Returns a filtered cross product of its arguments
• Set operations: Union, Intersect, and Difference

Consider an Example 1.1

To illustrate how a query is performed look at the following basic database:

Student: { s_id, sname, sem }
Lecture: { lect_id, ltitle, lecturer }
Professor: { prof_id, name }
Enrolment: { s_id, lect_id }

The question we want to ask is:

Which semester are the students in, which are enrolled in a course of professor Newton?

If we translate this into an SQL query, in a first step, we might get:

Select distinct s.sem
From student s, professor p, enrolment e, lecture l
Where p.name = ‘Newton’ and
l.lecturer = p.prof_id and
l.lect_id = e.lect_id and
e.s_id = s.s_id

II. QUERY PROCESSING

Most of the relational database application programs are written in high-level languages integrating a relational language. Query Processing is the process of transforming a high level query in to a plan that executes and retrieves data from the database. It involves four phases which are Query decomposition, Query optimization, Code generation and Run time query Execution.

(a) Query decomposition

In this phase, a query is checked as to whether or not it conforms to the syntax of the language used (mostly SQL) and in case it does not, the error message is generated. If the query conforms to the syntax, it is broken into small pieces and represented in an equivalent relational algebra expression (parsing). A system catalog is used to cross check the consistency of the query with the schema.

(b) Query optimization

In this phase, the best execution plan is generated. This is done by putting into account the resources required to execute the query as well as the resources required to get the plan. Database statistics are used to make appropriate decisions.

(c) Code generation

After the optimizer has got the best execution plan, the code generator creates the code equivalent to the plan. This is sent to the internal ANSI spark architecture level of the database for execution.

(d) Query execution

In this phase, the code interacts with the database and retrieves the data for consumption of the process or individual who sent the query.
Two relational algebra expressions are said to be equivalent if on every legal database instance the two expressions generate the same set of tuples.

- Two expressions in the multiset version of the relational algebra are said to be equivalent if on every legal database instance the two expressions generate the same multiset of tuples.
- Order of tuples does not matter.

3.1.1 Equivalence rules

An equivalence rule says that expressions of two forms are equivalent (for expressions $E$, $E_1$, $E_2$, conditions $F_i$).

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections; cascade of $\sigma$.
\[ \sigma_{\theta_{1} \land \theta_{2}}(E) = \sigma_{\theta_{1}}(\sigma_{\theta_{2}}(E)) \]

2. Selection operations are commutative:
\[ \sigma_{\theta_{1}}(\sigma_{\theta_{2}}(E)) = \sigma_{\theta_{2}}(\sigma_{\theta_{1}}(E)) \]

3. Only the final operations in a sequence of projection operations is needed, the others can be omitted; cascade of $\Pi$.
\[ \Pi_{L_{n}, L_{n-1}, \ldots, L_{1}}(E) = \Pi_{L_{1}, \ldots, L_{n}}(E) \]

4. Selections can be combined with Cartesian products and theta joins:
\[ \sigma_{\theta_{0}}(E_1 \times E_2) = E_1 \bowtie_{\theta_{0}} E_2 \]
\[ \sigma_{\theta_{0}}(E_1 \bowtie_{\theta_{0}} E_2) = E_1 \bowtie_{\theta_{0}} \sigma_{\theta_{0}}(E_2) \]

5. Theta join operations are commutative:
\[ E_1 \bowtie_{\theta_{0}} E_2 = E_2 \bowtie_{\theta_{0}} E_1 \]

6. Natural-join operations are associative:
\[ (E_1 \bowtie_{\theta_{0}} E_2) \bowtie_{\theta_{0}} E_3 = E_1 \bowtie_{\theta_{0}} (E_2 \bowtie_{\theta_{0}} E_3) \]

7. The selection operation distributes over the theta join operation under the following two conditions:
   (a) It distributes when all the attributes in the selection condition $\theta_{0}$ involve only the attributes of one of the expressions ($E_1$) being joined.
\[ \sigma_{\theta_{0}}(E_1 \bowtie_{\theta_{0}} E_2) = (\sigma_{\theta_{0}}(E_1)) \bowtie_{\theta_{0}} E_2 \]

   (b) It distributes when the selection condition $\theta_{0}$ involves only the attributes of $E_1$ and $\theta_{1}$ involves only the attributes of $E_2$.
\[ \sigma_{\theta_{1} \land \theta_{0}}(E_1 \bowtie_{\theta_{0}} E_2) = (\sigma_{\theta_{1}}(E_1)) \bowtie_{\theta_{0}} \sigma_{\theta_{0}}(E_2) \]
8. The projection operation distributes over the theta join.
   (a) Let $L_1$ and $L_2$ be attributes of $E_1$ and $E_2$ respectively. Suppose that the join condition $\theta$ involves only attributes in $L_1 \cup L_2$. Then
   \[
   \Pi_{L_1 \cup L_2} (E_1 \bowtie \theta E_2) = (\Pi_{L_1} (E_1)) \bowtie \theta (\Pi_{L_2} (E_2))
   \]
   (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 the 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 the join condition $\theta$, but are not in $L_1 \cup L_2$. Then
   \[
   \Pi_{L_1 \cup L_2} (E_1 \bowtie \theta E_2) = \Pi_{L_1 \cup L_2}
   ((\Pi_{L_1 \cup L_3} (E_1)) \bowtie \theta (\Pi_{L_2 \cup L_4} (E_2)))
   \]

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 the union, intersection, and set-difference operations.
    \[
    \sigma_p (E_1 - E_2) = \sigma_p (E_1) - \sigma_p (E_2)
    \]

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

3.1.2 Cost-based query Optimization:

From the it above example 1.1 Query gives the following access plan:

We have three cross products, which means that we create a table whose number of rows is $|s|*|e|*|l|*|p|$. For large tables s, e, l and p this result is not acceptable. A possible way of improving this is to perform the selections earlier on in the search as shown in Figure 3.1 which is a Non-left-deep join tree.

By doing the selections earlier on, we make restrictions and our final table is of smaller size than the former one. We can improve our result even more by substituting the left-deep join operators as shown in Figure 3.2.

We have improved our original query model. But now we are at a point where it gets harder to tell how to further improve our model: Knowing that the join operator is commutative as well as associative, we don’t know which order of these joins would suit us best to minimize the resulting table at each step. Even though the result of the joins will be the same, it makes a difference, which join ordering we use.
We consider the three relations P, Q and R as shown in Table 1, 2, 3. Each one of the tables consists of a number of columns and rows. By illustrating the join procedure for the cases \((P \bowtie Q) \bowtie R\) or \(P \bowtie (Q \bowtie R)\), we will see, why the join ordering does indeed matter. The result of these operations will be the same, as expected. But we will see that the tables produced at the intermediate step will vary considerably with regards to their size:

IV. CONCLUSION

Query Optimization provide solutions to various problems and discusses issues like how to reduce the cost associated with Queries, increase the performance of the CPU, reduces communication cost and easy to communicate with the relations. Joins is one of the techniques which addresses these issues.

REFERENCES

[1] An Introduction to Database Systems by C. J. Date, 8th Edition