The Efficient Query Evaluation Plan in the Spatial Database Engine

*Zhao-Hong Liu[1], Sung-Hee Kim[1], Jae-Dong Lee[2], Hae-Young Bae[1]
[1]Dept. of Computer Sci. & Eng., Inha Univ., Inchon, 402-751, Korea
[2]Dept. of Computer Science, Dankook University, Korea
*macromliu@hotmail.com

Abstract

A new GIS software Spatial Database Engine (SDE) has been developed to integrated with spatial database that combines conventional and spatially related data. As we known well in the traditional relation database system, the query evaluation techniques are a well-researched subject, many useful and efficient algorithms have been proposed; but in the spatial database system, it is a litter difference with the traditional ones. Based on the Query Graph Model (QGM), we implemented our own query evaluation plan in the SDE, which can deal with the full functionality query statement SELECT-FROM-WHERE-GROUPBY-HAVING, and treat the spatial data and non-spatial data seamlessly. We proposed a novel multi way join algorithm base on nest loop that may be attractive.

1. Introduction

Data intensive geographic applications such as cartography, urban planning, and natural resource management are build by GISs. GISs are database systems that allow the manipulation, storage, and retrieval of geographic data and the display of data in the form of maps. The spatial DBMS provides the underlying database technology for GISs and other applications. Spatial DBMS (i) is a database system; (ii) offers spatial data types (SDT) in its data model and query language; (iii) supports spatial data types in its implementation, providing at least spatial indexing and efficient algorithms for spatial join. [7]

We adopted a conventional storage manger system to store the data. It supports the traditional atomic data type and BLOB data type. Base on the storage manager, we implemented our own Spatial Database Engine (SDE), which use the SQL language as the query language. In the SQL, based on the Open-GIS and SQL-99, we support the standard SQL as more as possible, thus we need deal with the complex query such as SELECT-FROM-WHERE-GROUBY-HAVING, we add the complex query support function to the system, and at last constructed our own query evaluate party. This paper will address the issues of query evaluation plan for queries. Retrieval for data manipulation (UPDATE, DELETE) is treated similarly.

Section 2 gives an introduction about the Spatial Database Engine and the Query Graph Model (QGM). Section 3 presents the query evaluation plan model, and the query plan in our system, a novel multi way join algorithm is also been proposed. Section 4 dates some conclusions.

2. Spatial Database Engine and Query Graph Model

The Spatial Database Engine is the kernel of the spatial database system.

In the Figure 1 architecture, the Client part provides the GUI (Graphics User Interface). It is composed of Communication Module, Cache management, Local Query Processor, Drawing Module and User Interface Module.

The Spatial Database Engine is the system's server program, it responses for storing and managing the spatial and non-spatial data in a DBMS, provides computing function, and serves multi-client at the same time. The communication module translates the packet format or SQL query received from user into database query. The QPE (Query Processing Engine) parses the queries received from user, executes it using the interface supported by MiDAS-III. In the SDE, it uses the spatial query

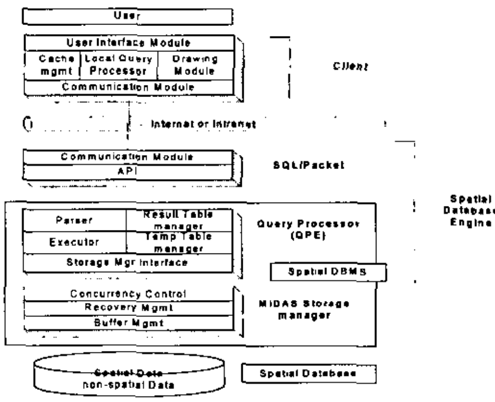

Figure 1: Architecture

language supporting spatial operation, which is extended from basic SQL and compatible with the Open GIS, and supports the well-known spatial logical operation, such as intersect, disjoint, contain, equal, touch, cross and overlap etc, spatial arithmetic operation, such as xcenter, ycenter, length, perimeter, distance, union, difference, intersection, xor, buffer, convexhull, as also the spatial aggregate function groupunion.

The Query Graph Model was systemically introduced in the IBM DB2's research paper[2]. Query are parsed and internally represented in a Query Graph Model (QGM), which is stored in an object-relationship in-memory c++ structure. QGM supports arbitrary table operations whose inputs/outputs are tables. Examples of operations are SELECT, GROUP BY, UNION, INTERSECT and EXCEPT (set difference). Note that the operation SELECT handles restriction, projection and join in SQL. We present the QGM through a simple SQL query:

```
select distinct q1.partno, q1.descry, q2.suppno
from inventory q1, quotations q2
where q1.partno = q2.parno and q1.descry='engine';
```

This query returns information about suppliers and 'engine' parts. Figure 1 shows the QGM for this query. The graph contains three boxes (or equivalently operations). Boxes 1 and 2 are associated with base tables *inventory* and *quotations*, and box 3 is a SELECT box associated with the main party of the query. Each box has a head and a body. The head describes the output table produced by the box, and the body specifies the operation required to compute the output table. Base tables can be considered to have empty or non-existent bodies.

This research is supported by Ministry of Information and Communication under work of University S/W research center.

Let's consider Box 3. The head specifies output columns specified in the select list of the query. The head has a Boolean attribute called *distinct* which indicates whether the associated table contains only distinct tuples (i.e. head.distinct is true), or whether it may contain duplicates (i.e. head.distinct is false).

The body of a box contains a graph. The vertices of this graph (the dark circles in the diagrams) represent quantified tuples variables, called quantifiers. In Box 3, we have quantifiers *q1* and *q2* ---they range over the base tables inventory and quotations respectively, and correspond to the table references in the FROM clause of the SQL query. These quantifiers have a quantifier type attribute, F (*ForEach*), and some of their columns are used in the output; and a distinct attribute which has the value of *enforce*, *preserve*, *permit*. The edge between *q1* and *q2* specified the join predicate. The (loop) edge attached to *q1* is the local predicate on *q1*.

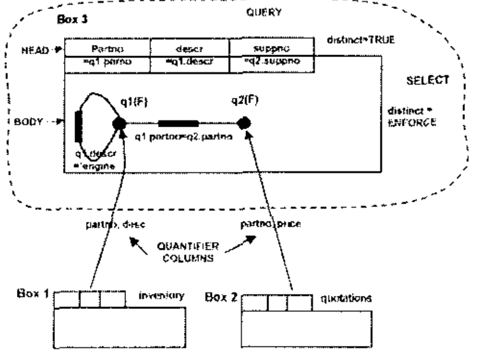

Figure2. Example QGM graph

The body of every box has an attribute called *distinct* with values **enforce**, **preserve** or **permit**. Enforce means that the operation must eliminate duplicates in order to enforce the output put to be duplicate free (i.e., head.distinct is true). Preserver means that the operation must preserve the duplicates. This could be because head.distinct is false, or because head.distinct is true and no duplicate could exist in the output of the operation even without duplicate elimination. Permit means that the operation is permitted to eliminate duplicates arbitrarily. For example, subquery boxes can have the value permit because duplicates returned by subquery do not affect the answer set.

3. Efficient Query Evaluation Plan

In this section, we firstly give a brief introduction of the query evaluation plan models; secondly, the phase in the query evaluation model; thirdly, the query plan in the SDE, available way presented to solve the problem, and the advantage and shortcomings; lastly, our own spatial query plan methods also be introduced.

3.1 The Query Evaluation Plan Model

The typical query evaluation models is composed of parse, validate, rewrite (optimization), plan (optimization) and evaluation phases.

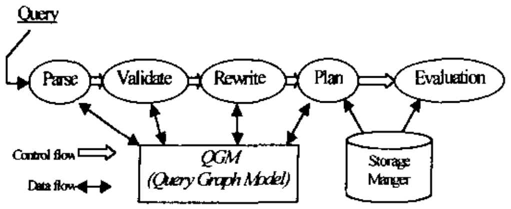

Figure 3. Phase of Query Processing

3.2 Parse, Validate and Rewrite

In the parser phase, the SQL text commands are translated into internally presentation, we translate the DDL part into parser tree, the DML part into QGM directly. The having clause has been translated into a where clause in a SELECT box, the group by clause has been translated into a GROUPBY box with the child box is the original query SELECT-FROM-WHERE part.

After the parser tree/QGM is constructed by the parser part, we apply the validate check, it will check the syntax and semantic error. The validation accumulates the names of tables and columns referenced in the query, looks them up in the system's catalog to verify their existence and to retrieve information about them. After obtained the table catalog information, the validation rescans the SELECT-List and Where-tree to check for semantic error in both expressions and predicate comparisons.

Here, we normalize the where clause, having clause (in the QGM,) with the conjunctive normal form.

We consider the entire spatial predicate into the *where* clause, including the *spatial* predicate, so we can deal with it as same as the alphanumeric predicate, and in the Normalization phrase, we adopted the Conjunction Normalization algorithm.

Definition:
CNF Tree: predicates linked by the OR, i.e.: $a \vee b \vee c \vee d$
CNF Forest: CNF trees linked by the AND,
i.e. : $(a \vee b \vee c \vee d) \wedge (a \vee b \vee e \vee f) \wedge (a \vee e \vee g)$

Algorithm 3.1:
```
Input: Binary Tree composed by predicate
Output: Conjunctive Normal Form Forest
Method:
i) For the terminal node (predicate)
        Return a AND node pointing to an OR node, and
        attach the terminal node to the OR node;
ii) For the intermediate AND node
        Normalize the left sub tree;
        Normalize the right sub tree;
        Return CNF_AND_MERGE(left, right); (plus)
iii) For the intermediate OR node
        Normalize the left sub tree;
        Normalize the right sub tree;
        Return CNF_OR_MERGER (left, right); (multiply)
```

In the rewrite phase, the QGM are translated equally from one state to another state. Such as

After the rewrite phase, the QGM has been translated to a state, it does not have correlated query and the local predicate has been putted on the basic table block properly, the each Conjunctive Normal Form trees (composed by the OR) also has been putted on the related box. For example, the query in Figure 2, after the rewrite phase, the local predicate/filter q1.descry = 'engine' will firstly be presented in the Conjunctive Normal Form, then moved from Box 3 to the Box 1. Of course, one CNF tree mixed with non-spatial predicate and spatial predicate, but only referenced to one table, is also moved to the related box.

3.3 Plan, Evaluation

The typical plan procedure is divided into logical plan and physical plan.

In the logical plan phase, the queries that presented in query graph (QGM) are translated to logical algebra (such as Selection, Projection, Join, Sort, Group By, Remove Duplicate, Aggregate).

Secondly, the logical algebra used in the logical plan is translated to physical algebra.

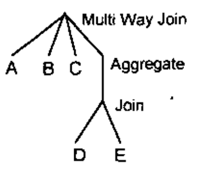

Figure 4. a bushy-tree like multi way join plan

Different logical algebra are mapped into related physical algebra, the Logical Selection is been mapped into Selection (intermediate node), Sequence Scan, Indexed Scan (base table), the Projection is been mapped to Projection, the Join is been mapped Nest Loop Join, Indexed Nest Loop Join, Equal Join, Merge Join, Product, Spatial Join (none index, one index, two index) based on the filter condition attached with this execute node.

In one Box, if there are more than 2 quantifiers, the traditional plan need consider the order in which the blocks should be chosen to join. There are three types: left-deep, busy, and right-deep. We prefer the bushy tree and multi way join, i.e., in Figure 4 it is a bushy tree like multi way join. In the following it is the novel multi-way join algorithm.

Algorithm 3.2: multi-way join algorithm

```
0. Every execute node, there is a member data m_pFilter with
   the type CNF forest; a member data m_pFilterArry[] with the
   type CNF forest. For simplicity, we suppose the quantifier list
   as Q0, Q1, Q2, Q3, Q4 by orderly; a buffer array to save
   every child block's one record; a m_nState array with the
   value FIRST, NEXT, EMPTY;

1. Preparations:
   For every CNF tree in the m_pFilter, check the referenced
   table list. Initial the m_pFilterArray[QuantifierCount-1],
   collect all the CNF tree which's reference table is (Q4, Q3),
   construct them into a CNF forest, assign to
   m_pFilterArray[0], the CNF forest with the CNF trees with
   the referenced table in (Q4, Q3, Q2) assign to
   m_pFilterArray[1] .......

2. TotalInitial()
   Initial the buffer array with QuantifierCount Element;
   Initial the m_nState[QuantifierCount] with the initial value
   FIRST;
   For (int i = 0; i < QuantifierCount; i++) {
     Call $i^{th}$ quantifier's related subblock open();
     Read First record to $i^{th}$ record buffer;
     If the return is null, return false;
   }
   Return true;

3. PartInitial (int pos)
   FOR (int i = 0; i < pos; i++) {
       Call $i^{th}$ quantifier's related subblock Open();
       Read the record to $i^{th}$ buffer;
       m_nState[I] = FIRST;
   }
   FOR (int j = i ; j <= QuantifierCount - 1; j++) {
     Call $j^{th}$ quantifier's related subblock next(),
     Read the record to $j^{th}$ buffer;
     IF return value is NULL {
       Set m_pState[j] to FIRST;
       $j^{th}$ Block Open();
       read first record to $j^{th}$ buffer;
     }ELSE {
       set m_nState[j] to NEXT;
       Return false; //mean continue
     }
   }
   IF ((j == QuantifierCount) && (m_nState[QuantifierCount]
   ==FIRST)) return true;

4. JoinAlgorithm()
   IF TotalInital() is FALSE, end;
   While(1){
     int i = 0;
     DO{
       revV = Valiate the m_pFilterArray[i];
       IF revV is false {
         IF PartInitial( QuantifierCount -2 - i) Break;
         revV = TRUE; i = 0;
         } ELSE i ++;
       }WHILE (i < QuantifierCount -1) && (revV is TRUE)
     Construct a record, append it to the result.
     //Read $0^{th}$ Quantifier Block's next record to buffer;
     //If return value is NULL
     IF PartInitial(0) break;
 };
```

With the iterator technique in Volcano query model specified in [4], we implemented the methods open, next, close at every execute node. Every node's filter is a Conjunctive normal form, which mixed with the spatial predicate and non-spatial predicate, in the terminate execute node, we gather the conjunctive normal form like filter which the storage manager can deal with, this part is sent to the storage manager, if there is spatial predicate, we gather the MBR information from the predicate, and translate the it into the arithmetic type, with the logical operation windows query or disjoint query (called in the Spatial Query), we can say this is the filter step in the multi-step technique in spatial query, and the remained filter will be evaluated by the expression evaluate procedure.

For, we have adopted the bushy tree like plan, so every node is belong to non-stop node or stop node, non-stop means it need not store the intermediate result at this node, the non-stop node means we need store the intermediate result at this node, this may be resulted of the aggregate, group by, duplicate remove. But in another situation, the count(field) has been pushed just after the field's original related execute box the last execute node, this strategy can reduce the time spend on large aggregate operation, as a result, every node will have a factor, indicate that the output result repeat times, or this will result in a bug, and produce the uncorrected result.

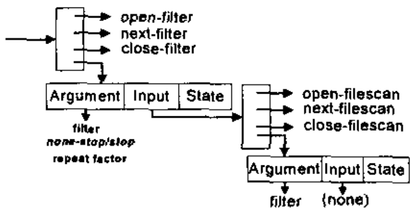

Figure 5. Two operators in a Volcano query plan

4. Conclusions and Future Research

In this paper, we've discussed the implemented query evaluation plan used in the SDE. We proposed the models, steps, methods and principles for query evaluation plan based on the query graph (QGM) and Conjunctive normal form; at last, we implemented our own evaluation plan in the SDE, which has the full query capabilities of SQL on the non-spatial part of the database while being tightly integrated with the spatial part, the proposed multi way join algorithm may be attractive.

About the future research, we'll support subquery, and apply more works on query rewrite optimization and rule based plan optimization.

REFERENCES

[1] M. Astrahan et al., System R: A Relational Approach to Data Base Management, ACM TODS, pp.97-137, 1976

[2] T.Y.Cliff Leung etc., Query Rewrite Optimization rules in IBM DB2 Universal Database, Readings In Database System 3rd Edition, pp153

[3] Open GIS, OpenGIS Simple Features Specification for SQL R1.0, http://www.opengis.org/techno/specs.htm

[4] Geetz Graefe, Query Evaluation Techniques for Large Databases. ACM Computing Surveys, 25(2), pp.73, 1993

[5] P. Griffiths Selinger etc., Access Path Selection in a Relational Database Management System, ACM SIGMOD 79, pp.23, 1979

[6] Peter Gulutzan and Trudy Pelzer, SQL-99 Complete, Really, R&D Books, Lawrence, Kansas 66046

[7] Ralf Hartmut Guting, An Introduction to Spatial Database Systems, VLDB Vol.3 No.4, pp.357-399, 1994.

[8] Inderpal S. Mumick etc., Implementation of Magic-sets in a Relational Database System, ACM SIGMOD, pp.103, 1994