

# QUERY OPTIMIZATION IN EXTENDED DBMS

*Dissertation submitted to the Jawaharlal Nehru University  
in partial fulfilment of the requirements  
for the award of the Degree of  
MASTER OF TECHNOLOGY*

in  
COMPUTER SCIENCE & TECHNOLOGY

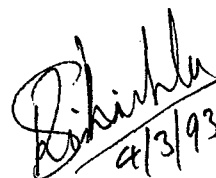
by  
**RISHI SHUKLA**

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES  
JAWAHARLAL NEHRU UNIVERSITY  
NEW DELHI - 110 067  
INDIA  
JANUARY 1993

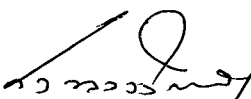
CERTIFICATE

This is to certify that the dissertation entitled **QUERY OPTIMIZATION IN EXTENDED DBMS**, being submitted by me to Jawaharlal Nehru University in the partial fulfilment of the requirements for the award of the degree of **Master of Technology**, is a record of original work done by me under the supervision of **Dr P.C.Saxena**, Associate Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University during the year 1992, Monsoon Semester.


The examples taken and results reported in this dissertation have not been submitted in part or full to any other university or Institute for the award of any degree or diploma, etc.



( RISHI SHUKLA )



**PROF.R.G. Gupta**  
Dean, 4/11/93  
School of Computer and  
Systems Sciences,  
J .N .U .,  
New Delhi.



**Dr. P. C. Saxena**  
Associate Professor,  
School of Computer and  
Systems Sciences,  
J. N .U.,  
New Delhi

### ACKNOWLEDGEMENT

I express my humble gratitude towards Dr. P.C. Saxena, Associate Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi who was always there to guide me to the safe exit in case of any crisis and was all the way with me to boost my morale and to provide his valuable guidance.

I am also highly thankful to Prof. R.G.Gupta, Dean, School of Computer and Systems Sciences, who took great pains to provide me with the environment and all the facilities required for the successful completion of my project.

I also take this opportunity to thank all faculty and staff members as well as my friends who have been directly or indirectly helpful in eliminating a variety of problems encountered by me in the course of completing this dissertation.

(RISHI SHUKLA)

## CONTENTS

	Preface	
1.	Introduction	1-6
	1.1 Extending Techniques	3
	1.2 Extended Optimization	3
	1.3 Data Architecture	6
2.	Logical Transformation and Decomposition	7-12
	2.1 Logical Transformation	7
	2.2 Decomposition	9
3.	Subquery Sequences and Plan Formulation	13-23
	3.1 Result Formulation	13
	3.2 Sequential Query Plan (SQP) Formulation	19
4.	The Hybrid Database Architecture for Optimization	24-35
	4.1 Hybrid Architecture	24
	4.2 Implication of Hybrid Architecture	27
	4.3 Extended Operators and Spatial Relations	31
	4.4 Special Features of a Hybrid System	34
5.	Query Processing and Optimization	36-43
	5.1 Query Processing	36
	5.2 Analysis of an Example	37
	5.3 Description of Plan 1	38
	5.4 Other Possibilities	40
6.	Conclusion	44-46
7.	References	47-51

## PREFACE

The purpose of this dissertation is to introduce the techniques for extending the existing DBMS for the applications involving hybrid (spatial and aspatial) queries. Special systems which are application specific have been developed for GIS, CAD/CAM and IMAGE PROCESSING. Here a scheme has been proposed which, in general, is applicable to any DBMS with little variation in different systems.

**CHAPTER 1** is the introductory chapter and introduces in short, the overall strategy and approach for extending a system.

**CHAPTER 2** discusses the first two stages involved in the extending technique namely Logical Transformation and Decomposition, examples of which have been shown to give more insight of the system.

**CHAPTER 3** deals with the process of plan formulation for query processing and their execution. Special treatment has been given to spatial and aspatial parts of the query. Different algorithms for query processing have been dealt with.

**CHAPTER 4** discusses the data architecture of a hybrid system, different arrangements for storing spatial and aspatial attributes of an object and how the link between the two attributes is established for the quick query processing. Operations such as the sp\_extract and db-extract have been discussed.

CHAPTER 5 discusses an example and the application of different techniques at different stages, in particular at the plan formulation stage.

The concluding chapter discusses the limitations and performance of such systems.

Some of the systems with the similar techniques have been found to be functioning successfully.

## CHAPTER 1

### INTRODUCTION

The query optimizer serves as the integral part of any RDBMS(Relational Data Base Management System), its main task is to select an efficient query evaluation plan for a given query. The conventional optimizers in existing system are not suitable for supporting CAD/CAM ,GIS, and IMAGE DATABASE. The current focus is on design of extensible DBMS so that newly constructed generalised DBMS can support unconventional applications easily. Another approach which is widely advocated is to extend an existing DBMS for new applications. In the latter case the existing DBMS is supplemented by new indexing structure and evaluation subsystem in order to process spatial queries. The approach takes the advantage of existing mature techniques for aspatial query evaluation while providing, at low cost, efficient implementation of spatial indexing techniques for general queries. The purpose of the dissertation is to describe an optimization strategy based on this approach.

To extend a DBMS for hybrid applications involving spatial queries it is necessary to augment the external language and to provide a special processing subsystem to evaluate predicates that cannot be efficiently processed by conventional DBMS. For executing the hybrid queries efficiently it is necessary to construct an extended optimizer which also supports new indexing/file structures. Thus a hybrid system must provide a

storage and information processing architecture that integrates both spatial and aspatial components of database. In such an integrated system user is provided with a single query language capable of expressing selection criteria including both aspatial and spatial qualifications.

Let us take the example of finding all parks within ten kilometers of the university named "JNU" in Delhi.

```
SELECT L.name L.usage
      FROM Land-use L, Land-use U
      WHERE U.usage = "university" AND
            L.usage = "park"
            AND Within(U.location, L.location, 10)
            AND In_window(L.location, W)
```

The response to such queries depends on the ability of the optimizer to find an efficient query evaluation plan. Without considering how selection and projection may be performed there are several global strategies to execute the above query (shown in implementation verification). A spatial index can be used to reduce the number of parks entities that must be processed or else each park must be examined in turn. If the university relation is very large and no index exists then a different method must be used. The aim is that an optimizer must be able to make a choice among these schemes and select an efficient low level operation.



The aim of this dissertation is to extend an existing DBMS such that the spatial queries can be optimized as a whole. But the three issues must be investigated before extending a system. First requirement is that of application domain, second is data architecture needed to fulfil these requirements, third the design should be prototype and should be capable of being built on top of an existing extensible system. Extended operations have been introduced to integrate homogenously both spatial and non-spatial operations.

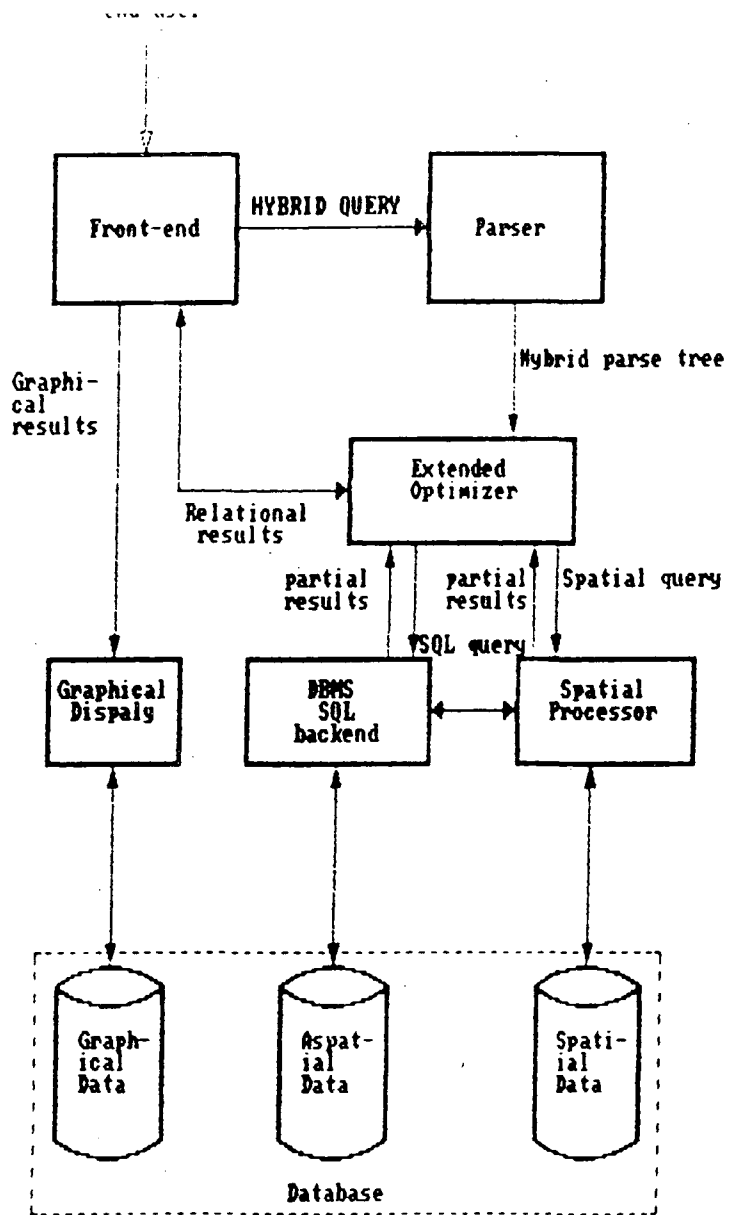
#### **1.1 EXTENDING TECHNIQUES :-**

##### **Overview of extended system :-**

New applications involving hybrid queries require extra auxiliary indexing structure to facilitate the query retrieval based on the new relationships. These new indexing structures cannot be supported in a conventional DBMS as they require special subsystems which interfaces with the SQL backend through which it accesses the database, and performs necessary spatial operations. On the top of these two processors, the SQL backend and special processor, is an extended optimizer responsible for determining an efficient query evaluation plan. The architecture is shown in figure .

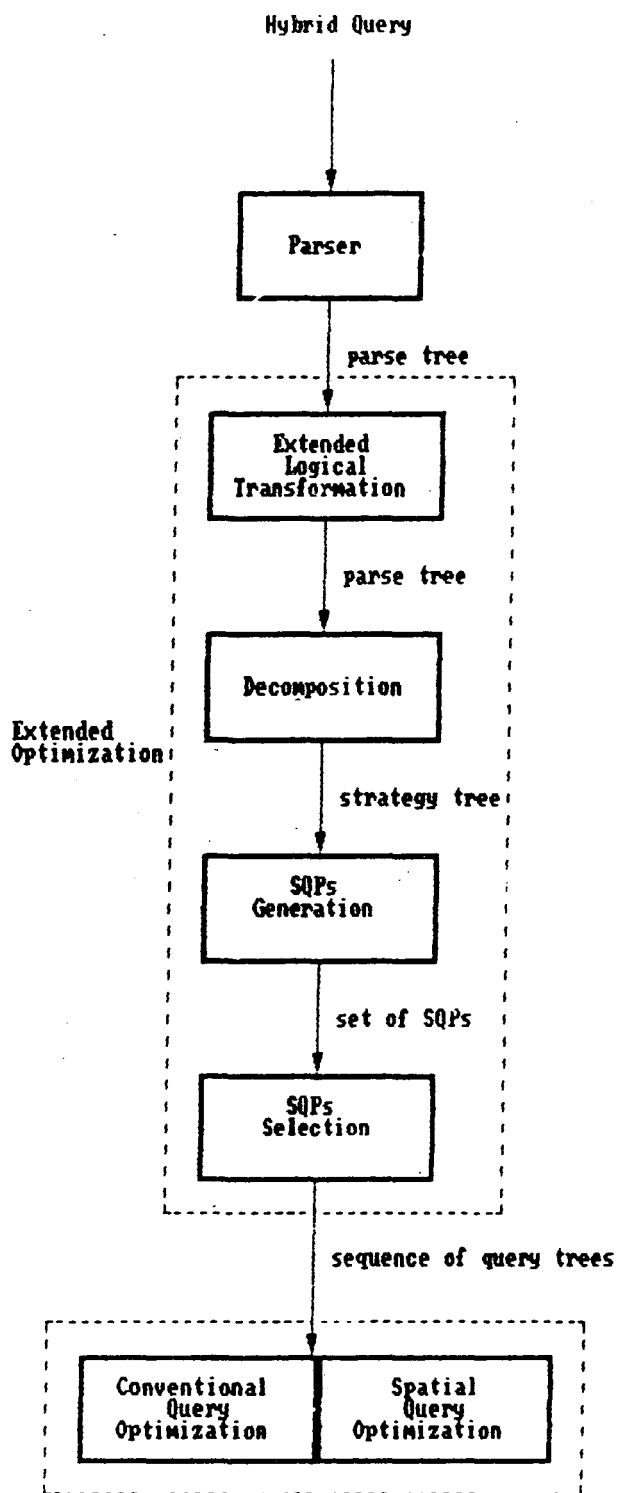
##### **1.2 Extended optimization :-**

Our main aim is to make use of existing SQL backend. To this end a query is broken into subqueries that are either totally spatial or aspatial so that totally aspatial subqueries



A System Architecture for A Hybrid (GIS) System

Fig. 1



An Extended Optimization Strategy

Fig. 2

can be executed by an existing SQL backend, and the spatial queries can be executed by the newly installed spatial processors that minimizes the overall query cost. The strategy consists of following four major steps.

**(1) LOGICAL TRANSFORMATION :-**

As part of the optimization, the query tree produced by parsing the initial query is rearranged such that the new representation is more amenable to efficient evaluation. Also to include conventional logical transformation a hybrid query tree must be restructured so that the spatial indexing can be used whenever possible.

**(2) DECOMPOSITION :-**

The parsed tree produced by the stage one is partitioned into subtrees which are either totally spatial or aspatial. Each subtree represents a subquery which must be executed by the query processor. The aspatial subquery will be executed by the existing SQL backend and spatial subquery which cannot be processed by the SQL backend will be executed by the spatial processor.

**(3) PLAN FORMULATION AND SELECTION :-**

From the set of subqueries obtained in the previous step, different orders (subquery plans or SQP) are formed. The best or the cheapest is selected among all subquery sequences.

#### (4) PLAN EXECUTION :-

From the chosen subquery sequences, SQL subqueries are passed to SQL backend and spatial subqueries are passed to spatial processor.

The extended SQL includes predicates of the form *geo\_term, geo\_op, geo\_term*, where *geo\_op* is one of the spatial operators, example Intersect, Adjacent, Join, End at, Contains, Situated at, Within, Closest, and Farthest, the *geo\_op* "Within" has been used later.

*Geo\_term* is one of the following :

(1) A geographic entity class (i.e a *geo\_obj* as defined by table of variables name).

(2) A virtual geographic entity of the form :

(a) Line joining *geo\_obj* or

(b) *geo\_obj* bounded by *geo\_obj* and *geo\_obj*

(3) A window definition  $(x_1, x_2, y_1, y_2)$  where  $(x_1, y_1)$  and  $(x_2, y_2)$  are top right and bottom left corner of the window, or a WINDOW keyword that assumes the current window in which a skeleton map is displayed as the co-ordinates.

Since SQL supports nested formulation of queries, algorithms which transform a nested query to an equivalent unnested query are an important component of SQL query processor, as hybrid spatial predicates do not allow subqueries as operands, the hybrid language introduces no new nesting complexity as compared to SQL, and so the existing SQL unnesting algorithm can be applied with only minor modifications to transform the

hybrid nested query into a set of partially ordered unnested queries. The result of this unnesting is sequence of unnested hybrid queries. Unnesting is considered as a preprocessing stage, and the optimization strategies that follow consider each unnested query in isolation. Consequently the only scope for optimization between the nested and unnested component occurs in the unnesting phase. An existing SQL parser can be extended for hybrid queries. For subsequent query transformation, it is highly desirable that the structure used to represent a parsed hybrid query allows a great degree of freedom for restructuring. The internal nodes may have more than two children.

### **1.3 DATA ARCHITECTURE**

Data architecture of a hybrid system is its very important part. In such architectures special provision for storing spatial and aspatial data are needed. Their both spatial and aspatial attributes are stored in different regions and a proper link between the two is maintained for faster search. The concept of forward and backward links which help in locating the objects is also given, the two extract operators `sp_extract` and `db_extract` make possible the two links. The `sp_extract` finds the spatial part from the given database tuples, and `db_extract` does the reverse.

## CHAPTER 2

### LOGICAL TRANSFORMATION AND DECOMPOSITION

#### 2.1 Logical Transformation

As in SQL, several hybrid queries with different syntactic forms but with the same semantics may be formed. These alternative syntactic forms may exhibit different potential for execution optimization, and greatly increase the complexity of the optimization procedures. One of the preprocesses involved in query analysis is to convert all queries into some canonical form, with the objective of reducing the syntactic variants amongst semantically equivalent queries and yielding syntactic forms that are most amenable to the optimization heuristics. Assuming that a syntactically correct query is translated into a parse tree, the predicate subtrees corresponding to the WHERE clause are initially simplified.

It is important to make use of spatial indexes whenever possible. Therefore, the representation of predicates in the predicate tree must not cause an early cartesian product to be formed when this product may inhibit the use of spatial indexes.

Consider the following predicate

```
road is adjacent to park AND  
(park.usage = "recreational" OR  
road.name = "Africa Avenue")
```

If the OR predicate is executed first, the cross product formed precludes the use of spatial indexes from the spatial predicate. However the expression in the following equivalent form would not inhibit the use of spatial indexes on both spatial predicates.

(road is adjacent to park AND  
road.name = "Africa Avenue") OR  
(road is adjacent to park AND  
park.usage = "recreational").

In this example the restrictions may be used to reduce the cost of evaluating the spatial expressions and no cross product is involved.

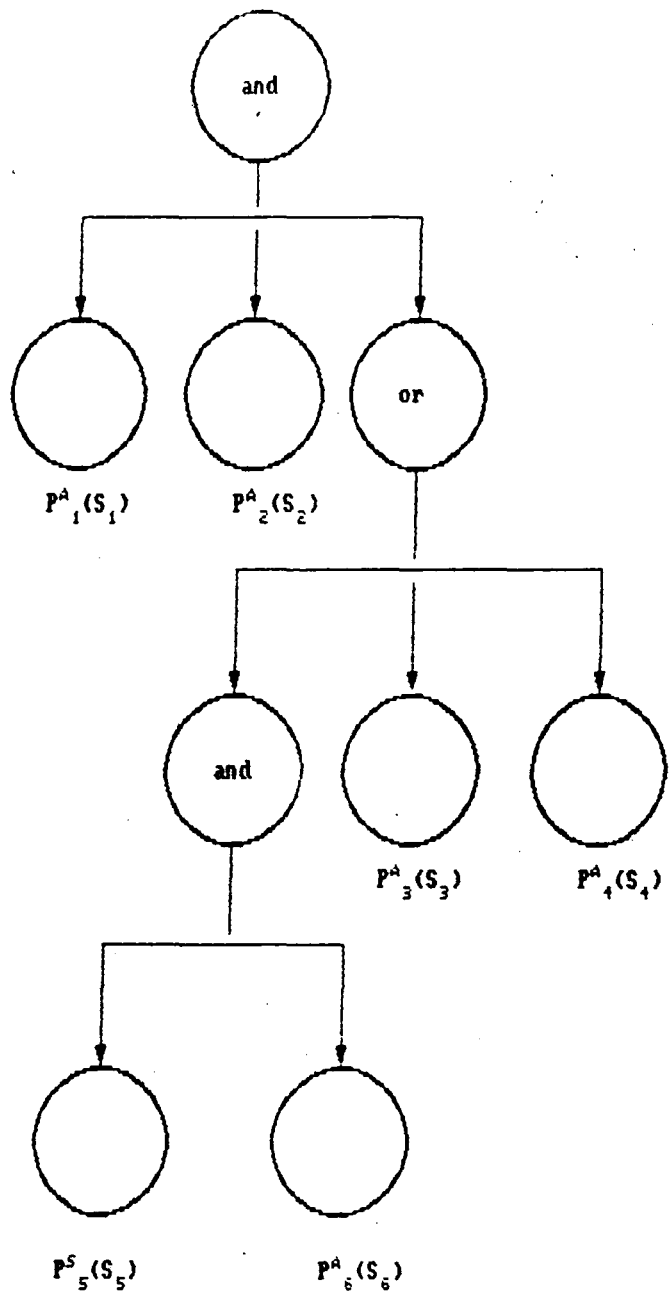
Another objective of logical transformation is to reduce the number of terms in an expression by identifying identical terms. This objective can conflict with the logical transformation. Allowing the use of spatial indexes is given higher priority than reducing the number of terms. Thus the transformation required on top of the existing conventional logical transformations can be stated as follows:

$$\text{AND}(P_1^s(S_1), \text{OR}(P_2(S_2), P_3(S_3))) \implies$$

$$\text{OR}(\text{AND}(P_1^s(S_1), P_2(S_2)), \text{AND}(P_1^s(S_1), P_3(S_3)))$$

$$\text{if } [(S_1 \cap S_2) \text{ OR } (S_1 \cap S_3)] = \text{true}$$





Grouping of Predicates.

Fig. 3

The above transformation is necessary, irrespective of the predicate types of  $P_2$  and  $P_3$ . The above transformation has no effect on the following expression :

*park.usage = "recreational" AND  
road.name = "Africa Avenue" AND  
(park is adjacent to road OR  
park is adjacent to railway)*

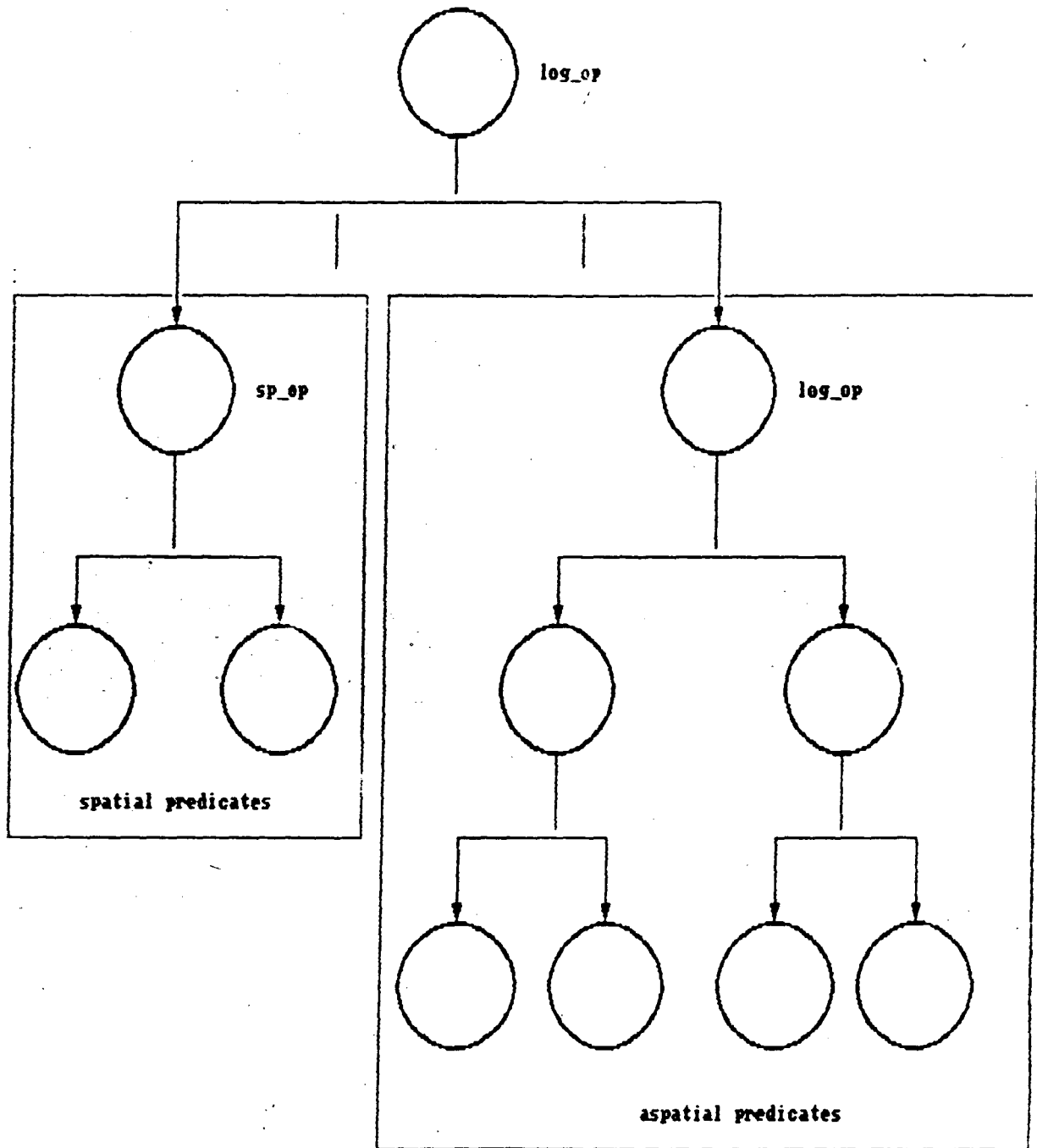
For this expression, the spatial predicates are not impeded from using spatial indexes and can be more efficiently evaluated by using the partial results of the restrictions. No transformation is necessary.

All aspatial predicates are grouped according to the relations they reference.

## **2.2 Decomposition**

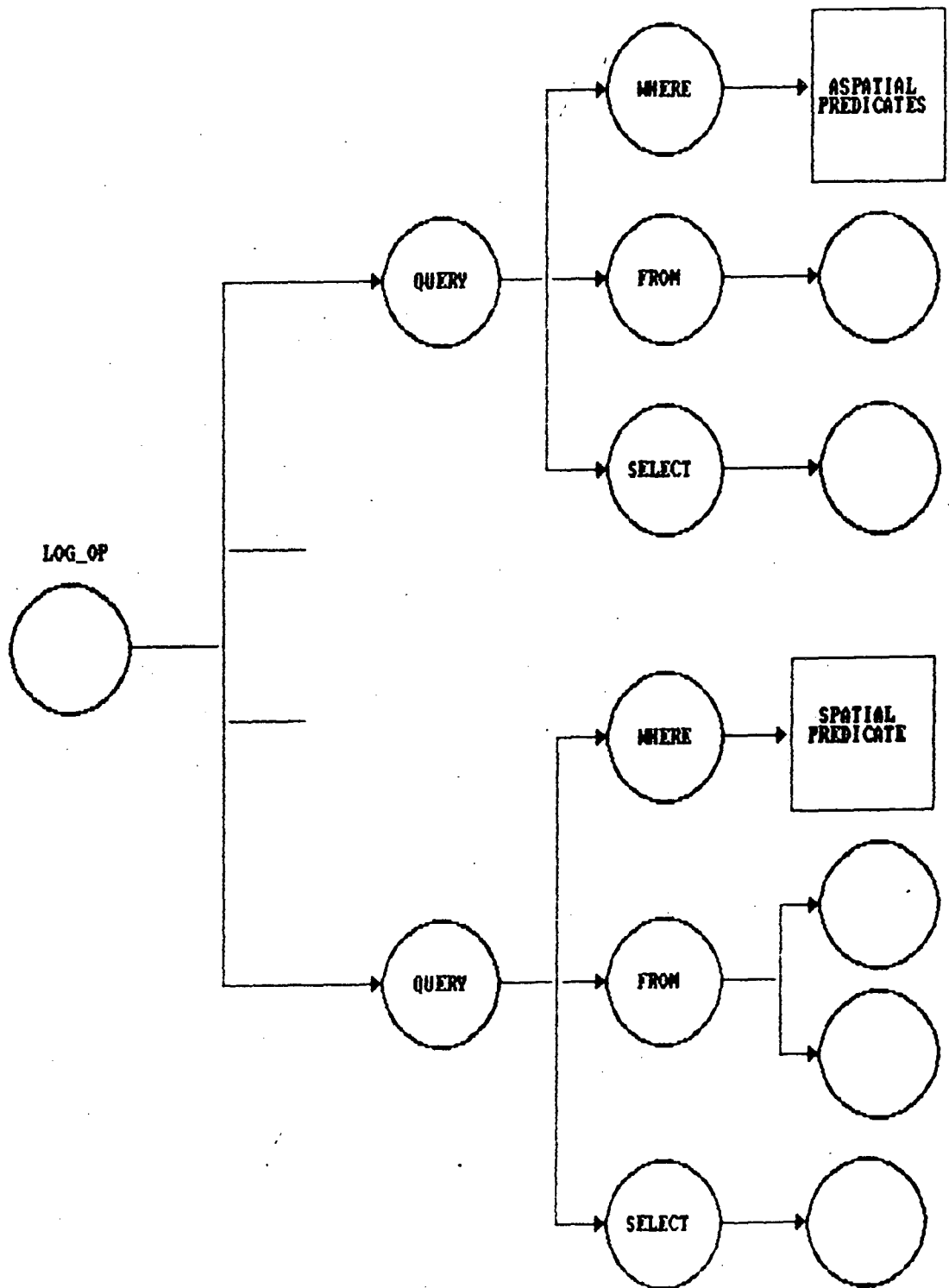
The technique is used in query optimization to decompose a query into simpler components to reduce the complexity of optimization. In general, the simpler subqueries are designed to reference fewer relations than the original query. In our context, an unnested hybrid query is decomposed into simpler subqueries that are either purely spatial or purely aspatial to enable global ordering of subqueries such that the overall evaluation cost is minimum.

The temporary relation created by  $P_2^a(S_2)$  may affect the strategy employed in evaluating  $P_5^s(S_5)$  if both predicates reference some common relations. On the other hand,



A Predicate Tree

Fig. 4



A Corresponding (Fig. 4) Strategy Tree  
 Fig. 5

the partial result of an earlier execution of  $P_3^A$  or  $P_4^A$  will not affect the execution of  $P_5^S$  irrespective of the relations in  $S_3$  and  $S_4$ . Hence  $P_5^S$ ,  $P_6^A$  and  $OR(P_3, P_4)$  should be detached as subqueries.  $P_1^A$  and  $P_2^A$  should form another two subqueries if  $S_1$  or  $S_2$  intersected  $S_5$ ; otherwise the conjunction of the two predicates would form the subquery. A more complicated decomposition policy would form  $P_1^A$  and  $P_2^A$  as two subqueries if  $S_1$  or  $S_2$  intersected any of  $S_3$ ,  $S_4$ ,  $S_5$  and  $S_6$ . However, this more complicated strategy would lead to severe increase in the cost of determining the optimal subquery sequence. Our strategy is to form subqueries

1. for each spatial predicate.
2. for each SQL predicate (which may be compound) that is related to a spatial predicate by a conjunction and references the relation (or relations) involved in spatial predicate.
3. for the remaining SQL predicates in a logical expression, which do not satisfy condition 2.

The decomposition scheme maps a hybrid expression  $log\_op(P_1, \dots, P_k)$  into a query expression  $log\_op(E_1, E_2, \dots, E_n)$  where  $E_i$  is either a query expression or a query  $Q_k(S_k)$ . More precisely, it is a mapping of a predicate tree into a tree whose internal nodes are logical operators, and external nodes are queries (pointers to a corresponding query trees). We call the data structure formed by the decomposition algorithms the "strategy tree". Fig. 4.5 illustrates such a mapping. With such

representation, the dependency of the subqueries is captured without any ordering.

A join  $R_1.A = R_2.A$  where one of the relations does not participate in any other predicate and the target list (answer) is implicitly a restriction. Hence we regard such joins as a restriction in our decomposition algorithm.

As an example of the decomposition strategy, the following contrived query (disregard its semantics) is provided to illustrate the power of the decomposition strategy.

```
SELECT road.name, park.name
FROM road, region, park
WHERE road.name ≠ region.name AND
      road is adjacent to region AND
      park.name ≠ "APPU PARK" AND
      ( region.name = " JNU " OR
        region.name = "SOUTH DELHI")
```

The subqueries formed are as below

```
-----
Q1      SELECT  *
        FROM    region, road
        WHERE   road is adjacent to region
-----
Q2      SELECT  *
        FROM    region
        WHERE   region.name = "JNU" OR
               region.name = "South Delhi"
-----
```

```
-----  
Q3      SELECT  *  
        FROM    road, region  
        WHERE   road.name ≠ region.name  
-----
```

```
Q4      SELECT  *  
        FROM    park  
        WHERE   park.name ≠ "APPU PARK"  
-----
```

The strategy tree for the above query is an internal node with an AND logical operator pointing to four offspring subqueries. Note that Q1, Q2 and Q3, and Q4 are formed under conditions 1, 2 and 3 respectively.

It should be noted that the decomposition algorithm only decomposes a query into multiple subqueries, it does not specify any order for the subqueries, and the strategy tree allows us to arrange the subqueries in any order we prefer so long as the final answer is correct. Hence the SELECT clause of subqueries cannot be defined until the full ordering is known.

## CHAPTER 3

### SUBQUERY SEQUENCES AND PLAN FORMULATION

#### 3.1 Result Formulation

Since a strategy tree is semantically equivalent to the predicate tree from which it originates, the subqueries in the strategy tree and their corresponding predicates in the original query tree exhibit the same dependencies. While ordering the subqueries, the dependencies captured by the strategy tree structure can be expressed by introduction of extra SQL queries AND/OR terms rewriting. The extra queries are used to merge multiple partial results (Temporary relations), while the technique of terms rewriting propagates the partial results of a query to another query. Three result rewriting rules are introduced in the chapter to ensure the correctness of the final result. These conditions for which three rules are applied are mutually exclusive.

If the subqueries that are connected by a conjunction, reference some common relations and executed independently, then the partial results must be merged by a natural join. However if a subquery( subquery Q) uses the partial result of other subquery (subquery Q ) then no natural join is required since the result that is produced by Q also satisfies the selection in Q .



### Result Rewriting rule 1

Given an expression and  $(Q_1(S_1))$ ,  $Q_2(S_2)$  and the precondition  $S_1 \cap S_2 \neq \emptyset$ . Let  $I = S_1 \cap S_2$  and let  $C \subseteq I$ . Suppose  $Q_1$  is evaluated before  $Q_2$ , resulting in the partial result  $T_1$ , then for each  $R \in C$  if  $C \neq \emptyset$  replace  $R$  by  $T_1$  in  $Q_2$  and

1. if  $C = I$ , no further rewriting is necessary.
2. if  $C \subset I$ , ( $C$  may be  $\emptyset$ ) and suppose  $T_2$  is the temporary relation produced by  $Q_2$  and  $A_1, \dots, A_n$  are the common unqualified attributes in  $I$ , then the following SQL query (an equijoin) is introduced to merge the partial results :

```
SELECT *
FROM T1, T2
WHERE T1.A1 = T2.A1 AND
      T1.A2 = T2.A2 AND
      T1.An = T2.An.
```

Using a restricted instance of a base relation or a base relation with indexes is determined by the optimizer to minimize the processing cost.

---

```
Q1      SELECT *
        FROM   Ri, Rj
        WHERE  P1(Ri, Rj) AND P2(Ri)
```

---

```
Q2      SELECT *
        FROM   P(T2, Rj)
        WHERE  P(T2, Rj)
```

```

T2 = SELECT *
      FROM Ri
      WHERE P (Ri)

```

---

```

Q3  SELECT *
      FROM T3a, T3b
      WHERE T3a.A1 = T3b.A1 AND T3a.A2 = T3b.A2

```

```

T3a = SELECT *
      FROM Ri
      WHERE P2 (Ri)

```

```

T3b = SELECT *
      FROM Ri, Rj
      WHERE P1 (Ri, Rj)

```

---

$Q_1$  can be thought of as first fetching a tuple of  $R_i$  if  $P_1$  is true, fetch all the tuples of  $R_j$  such that  $P_1$  is true. Then the result projected is the final answer. This is operationally the same as  $Q_2$ , except that  $Q_2$  will have the partial result of  $R_j$  for which  $P_1$  is true and stored as the temporary relation  $T_3$ . Therefore, each tuple of  $T_2$  and for all tuples of  $R_j$ , the result is formed if  $P_1$  is true. Another way which is similar to  $Q_2$  is firstly evaluating  $P_1$  and then, based on the partial result,  $P_2$  is evaluated. Alternatively  $P_1$  and  $P_2$  are evaluated independently, and a join of two temporary results on the common attributes is performed. This is expressed as  $Q_3$ . Hence the three queries produce the same result. The evaluation of query can be visualized as

firstly forming the cartesian product of all the relations that are reflected, and evaluating the predicates.

When two conjuncted subqueries reference different relations, the order of their evaluation is not important to cost saving. However, the partial results must be propagated to become part of the final answer. Subquery  $Q_4$  in fig. is a case in point. The partial result of  $Q_4$  must participate in the final result.

#### Result Rewriting Rule 2

Suppose we are given an expression and  $(Q_1(S_1), Q_2(S_2))$ , a condition  $S_1 \wedge S_2$ . Suppose that  $Q_1$  is evaluated before  $Q_2$ , then put the temporary relation  $T_1$  produced by  $Q_1$  in the FROM clause of  $Q_2$ . Consider the following examples.

```
-----  
Q 4 :      SELECT  *  
           FROM    Ri, Rj, Rk.  
           WHERE   P1 (Ri, Rj) AND P2(Rk)  
-----  
Q 5 :      SELECT  *  
           FROM    Ri, Rj, T5  
           WHERE   P1 (Ri, Rj)  
T 5 =      SELECT  *  
           FROM    Rk  
           WHERE   P2 (Rk)  
-----
```

Both predicates  $P_1$  and  $P_2$  can be evaluated independently, and the answer is the cross product of  $P_1$  and  $P_2$ . This is the same as forming the cross product of  $R_1, R_j$  and  $R_k$  and then retaining the tuples for which the  $P_1$  and  $P_2$  evaluate to true at the same time. For  $Q_5, P_2$  is evaluated first and the result is stored in  $T_5$ . Then for each tuple of  $R_j$  and for each tuple of  $R_k$ , the two tuples are merged if  $P_1$  is satisfied and the resultant relation is then crossed with  $T_5$ . As  $P_1$  and  $P_2$  can be evaluated independently, the sequence of evaluation is immaterial.

Two disjuncted subqueries that reference different set of relations must be made union capable and hence, each partial result must be "crossed" with the relations that are not referenced by the subquery.

### Result Rewriting Rule 3

Given an expression OR  $(Q_1(S_1), \dots, Q_k(S_k))$ , we let  $S = S_1 \cup S_2 \cup S_3 \cup S_4 \dots \cup S_k$ . The FROM clause of  $Q_i$  ( $i=1 \dots k$ ) is rewritten with the set  $S$  as the referenced relations. Suppose  $T$  is the temporary relation produced by  $Q$ , then the result is formed by:

Consider the following example

```

-----
Q7:      SELECT      *
          FROM        Ri, Rj, Rk.
          WHERE       P1(Ri, Rj) OR
                   P2(Rj, Rk)
-----

```

```

-----
Q8:      Tθa U Tθb

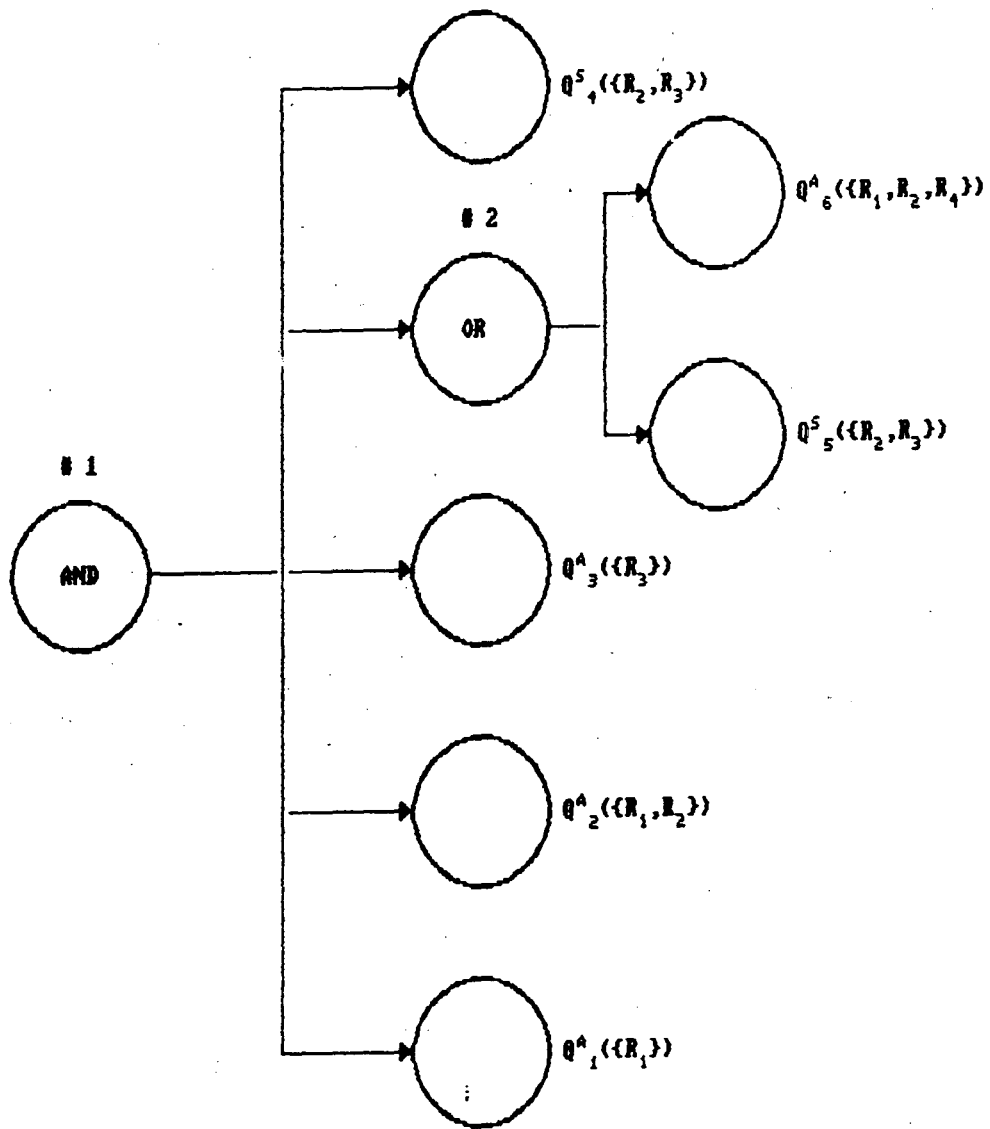
Tθa=    SELECT      *
          FROM        Ri , Rj , Rk .
          WHERE       P1 ( Ri , Rj )

Tθb=    SELECT      *
          FROM        Ri , Rj , Rk .
          WHERE       P2 ( Rj , Rk ) .
-----

```

Q7 can be thought of as firstly forming the cartesian product of the relations and duplicating them as two temporary relations. Then evaluate  $P_1$  on one of the temporary relations and  $P_2$  on the other. The results obtained by both evaluations are then unioned to form the final answer. This is equivalent to  $Q_8$ .

Notice that the first rewriting rule does not handle a general case like  $\log\_op(E_1, \dots, E_n)$ , where  $E_i$  may be a subquery or an expression. Consider the example  $AND(Q_1(R_1))$ ,  $OR(Q_2(R_1, R_2), Q_3(R_1))$  and suppose  $Q_1$  is the first to be evaluated. Suppose we chose to use the partial result of  $Q_1$  in evaluating  $Q_2$  but not  $Q_3$  for some reason (like using indexes). Then a natural join between the results of the OR expression and the result of  $Q_1$  is required. However, if both  $Q_2$  and  $Q_3$  use the partial result of  $Q_1$ , then the partial result can be considered as being propagated to the OR expression and hence, no extra natural join is required.



Use of Partial Results

Fig. 6

The partial result relation of one subquery may be used as the substitute for a base relation of other subquery if the first common ancestor of both subquery is on the same node .In figure ,any of the partial result of  $Q_1^a, Q_2^a, Q_3^a$  and  $Q_4^a$  may be used by  $Q_5^s$  and  $Q_6^a$  .A child of an AND node may use its brother's results. Suppose instead of  $Q_6$  ,we have an AND node with two children subqueries. These two subqueries may then use partial results of each other and the partial result of  $Q_i (1 \leq i \leq 4)$  but not of  $Q_5^s$  .

### 3.2 Sequential Query Plan ( SQP ) Formulation

For n subqueries n! distinct sequences of subquery plans can be formulated by the brute force method .Without pruning techniques, the optimizer would be unacceptably slow. Therefore, a rule based on heuristic strategy is required to reduce the number of SQP considered .Following the conventional DBMS ,a rule based optimizer should employ certain "rules" or "musts", to avoid the generation of sequences that are obviously ineffective, and heuristics to further prune the search space.

The set of subqueries formed by the decomposition strategy is small. However, it is important not to consider any alternatives that are obviously ineffective. Heuristic techniques are used to generate SQP that are efficient. A relaxed heuristic uses less rules, hence produces more alternatives than those produced by the stricter heuristic which has more rules. A disadvantage of a stricter heuristics is that the likelihood of a

good alternative being rejected is higher.

The following rules are proposed to reduce the subquery sequences being generated and thereby reducing the search space for finding the best sequence.

**Heuristic rule 1 :**

For an expression AND  $(Q_1(S_1), \dots, Q_n(S_n))$ , if  $S_i \cap (S_1 \cup S_2 \cup \dots \cup S_n)$  then  $Q_i(S_i)$  is executed just before the result is required.

The execution of  $Q_i$  does not offset any intermediate results of  $Q_j$  for  $1 \leq j \leq n (i \neq j)$ , therefore, the sequence of its execution is not important. To avoid rerecording of the partial result of  $Q_i$  which is not required till the formation of the final answer, the merging of the partial result can be postponed until  $n-2$  siblings subqueries have been evaluated. One of the  $n-1$  queries,  $Q_j (j \neq i)$ , is not executed before the final answer through that query (as in result rewriting rule 2).

The above rule is applicable for a general case in which there is more than one independent subquery. However, for each end node of the query tree, the decomposition strategy will produce at most one query for which the reference relations do not exist in the other conjoined queries.



**Heuristic Rule 2 :**

For the expression OR ( $Q_1(S_1), \dots, Q_n(S_n)$ ), the order of execution of subqueries is immaterial to cost saving, hence, the queries may be executed in any specific order.

Since the subqueries are independent, that is, one does not depend upon the evaluation of the other, it is immaterial which query is executed first. The above rule states that sum of the cost in executing subqueries is invariant to the subqueries. However, if the OR expression is the part of the conjuncted expression, say AND ( $Q_{p1}, Q_{pM} \dots \text{OR}(Q_1, Q_2 \dots)$ ), then the queries in the disjunction may use the partial result of the conjuncted ancestor queries ( $Q_{p1} \dots Q_{pM}$ ). The order of the execution of the OR query expression is important with respect to queries  $Q_{p1}, \dots, \text{and } Q_{pM}$ , so the whole disjunction is treated as a single entity. The following illustrate some of the subquery sequences of the example shown in fig.

[  $Q_1^a, Q_2^a, Q_3^a, Q_4^s (Q_5^s, Q_6^a)$  ]

[  $Q_1^a, Q_2^a, Q_3^a, (Q_5^s, Q_6^a), Q_4^s$  ]

[  $Q_1^a, Q_2^a, (Q_5^s, Q_6^a), Q_3^a, Q_4^s$  ]

[  $Q_1^a, (Q_5^s, Q_6^a), Q_2^a, Q_3^a, Q_4^s$  ]

[  $(Q_5^s, Q_6^a), Q_2^a, Q_3^a, Q_4^s, Q_1^a$  ].

While subqueries in [...] are conjuncted, subqueries in (..) are disjuncted. For each OR node of the strategy tree, the decomposition strategy produces atleast one child node which



TH-4528

is an aspatial query although there may be a number of child nodes representing spatial queries.

**Heuristic Rule 3 :**

For an expression , AND ( $Q_1(S_1)$  , . . . . ( $Q_i^a(S_i)$  ,  $Q_j^a(S_j)$  ,  $Q_k^s(S_k)$  , . . .  $Q_n(S_n)$  ) , suppose  $|S_i|$  , and  $|S_j| = 1$  , and  $S_i, S_j, S_k$  , then atleast one of the  $Q_i^a$  or  $Q_j^a$  must be executed before  $Q_k^s$ .

The argument follows from the fact that the outer relation should be as small as possible to reduce the search.

**Generating SQP :**

During the generation of different SQPs, it is necessary to minimize the number of SQPs, for which cost estimation is performed. A method for generating SQPs is proposed here. This method generates different plans lexicographically in such a way that all arrangements with common initial subsequences are generated consecutively. In this way ,the rejection of next few sequences with the same initial subsequence can be made without having to examine them.

Following the heuristic rule 2 ,the maximum number of sequences that may be generated is{ }.

To specify the order of execution ,the strategy tree may be rearranged such that the depth first left to right traversal produces the desired order. Alternatively, an array that describes the ordering can be used. For example one of the sequences can be represented as follows:

$$SQP [ Q_1^a, Q_2^a, Q_3^a, ( Q_5^s, Q_6^a ), Q_4^s ] .$$

AND#1	Q1 <sup>A</sup>	Q2 <sup>A</sup>	Q3 <sup>A</sup>	OR#2	Q4 <sup>S</sup>	Q5 <sup>S</sup>	Q6 <sup>A</sup>
0	1	1	1	1	1	4	4

SOP(Q1<sup>A</sup>, Q2<sup>A</sup>, Q3<sup>A</sup>, (Q5<sup>S</sup>, Q6<sup>A</sup>), Q4<sup>S</sup>)

An entry consisting of a query is a logical operator and the parent index (inner box). The first entry is always a logical operator and is followed by the children. The SQP can be obtained by recursively scanning the array from left to right. Start from the first entry, its child entries are searched from left to right. When a child entry is a logical operator, the process is repeated from that point. For array in the fig. on reaching entry 4 the search for entries 6 and 7 starts. Searching for the remaining children of the first entry resumes when the searching of the children of the fourth entry is completed.

While constructing an SQP, the attributes in the SELECT clause of the subqueries are required for further evaluation of the answer.

## CHAPTER 4

### THE HYBRID DATABASE ARCHITECTURE FOR OPTIMIZATION

#### 4.1 HYBRID ARCHITECTURE

For efficient working of a hybrid system which has been extended, the data structure architecture is important. In such systems proper link between the spatial and aspatial attributes of any object must be done for the fast operation and access of the query. Therefore, a discussion of database architecture for such hybrid system is done. Also, different operations which link aspatial and spatial attributes are given.

#### **Main Features Of Hybrid architecture**

In the typical spatial database environment, spatial objects are described by spatial as well as aspatial attributes. As an example, a road is described in a database by the coordinates of their end points (i.e spatial attributes). We make the following assumption about the way in which aspatial and spatial attributes describing an entity are linked.

(1) Each data entry is represented by the most efficient form that suits the operational purpose, that is why spatial attributes are stored in the spatial data structures and not flattened in database record.

(2) Each set, say  $O$ , of homogeneous objects that are spatially related to each other is logically clustered in the database.  $O$  is also described by the schema that has two sets of attributes:  $R$  and  $S$ .  $R$  is the set of aspatial attributes and  $S$  is the set of spatial attributes.

All data instances of a spatial attribute over the set of homogeneous objects are stored in one spatial data structure suitable for handling attribute's spatial type (e.g. region, line, or point). We refer to this as a map. A map is associated with each spatial attribute in the schema. A map is used as an index for spatial objects as well as a medium for performing aspatial operations (e.g. rotation and scaling for images, window, polygon intersection, area of a region, connected component retrieval, proximity query etc.).

The instances of a spatial attribute are merely some spatial indexes that point to the spatial description of the objects stored inside the spatial data structures. On the other hand aspatial are stored in database relations. It is important to mention that the database relation is used in hybrid system only as a repository for clustering data instances of aspatial attributes. In other words the hybrid spatial database architecture does not depend heavily on the fact that relational or arelational system is being used. It is how spatial and aspatial data are linked between the spatial and aspatial repositories that is important. In the hybrid system the spatial

and aspatial description of an object are linked to each other by what we mean "spatial relations". A spatial relation consists of two main components: a spatial and an aspatial repository. The spatial repository may consist of one or more spatial data structure while the aspatial repository is simply a relation.

In particular, in a spatial relation, spatial and aspatial attribute values of an object are linked by two types of links: **forward** and **backward** links. A forward link maps the aspatial description of an object that is stored inside a tuple into the object's spatial description that is stored inside a spatial data structure. A backward link serves as a mapping in the opposite direction. i.e from spatial description of an object into its aspatial description. The description of a spatial object can point back, via backward links, into more than one tuple in the relation.

A forward link is stored inside the tuple as the instance value of the attribute of the relation. Its purpose is to index and uniquely identify a spatial object from among a set of related spatial objects in the same spatial data structure. Regardless of the type of the spatial attributes being indexed, the forward link index value is of fixed length and does not depend upon the size of spatial object being referenced. Backward links are stored inside the spatial data structure along with each spatial object description, or in a separate look-up table that maps the spatial description of an

object back to its aspatial description (i.e tuple) inside a database relation. A backward link can be implemented by storing the tuple identifier of the object's corresponding tuple.

#### **4.2 Implications Of Hybrid Architecture:-**

The hybrid architecture structurally separates the spatial from the aspatial data. This has several implications on data manipulations and query processing. Here we demonstrate the implications that are related to insertion and deletion of objects, as well as relational and spatial based selection and join. Later their implementation is shown.

We use the following notations. A set of homogeneous objects  $O$  is represented by a relation  $R$  and data structures. We assume that  $O$ 's schema has only one spatial attribute in it. Extending the description to more than one spatial attribute is easy.

#### **Insertion And Deletion:-**

Performing insertion and deletion in hybrid system for spatial objects is simple. Basically, if any object  $o$  is inserted into (or deleted from) the homogeneous set of objects  $O$ , then  $o$  has to be inserted (deleted) as well. Finally forward and backward links are established between both components of  $o$ .

#### **Relational Based Selection:**

If some tuples are selected from relation  $R$  via a relational based selection to form a new relation  $R'$ , then a new data structure  $S'$  is also formed to store the spatial aspect of



the selected object in R'. This is achieved by the "sp\_extract" operator, as shown below. sp\_extract extracts the spatial data from S via the use of forward link from R to S. The extracted spatial data is inserted into S'. Backward link of the selected objects in S' have to be adjusted to point to the corresponding tuples in R' rather than R.

```
sp_extract (R1,S)
    /* For all the tuples in R1, retrieve their
description from S and stores it in a new
spatial data structure U.* /
    begin
        initialize U;
        traverse R1
        for each tuple t in R1 do
    begin
        sid:= get t's spatial- id;
        retrieve sid's spatial object o from S;
        insert o into U;
    end;
    return U;
end;
```

### **Spatial -based selection**

If some tuples are selected from data structure S via spatial based selection to form a new data structure S', then a new relation R' is also formed to store the corresponding tuples ( the non-spatial aspects) of the selected objects in S'. This is achieved by **db\_extract** operator( db\_ denotes database correlation ) of **db\_extract** tuples from R via the use of backward link from S to R and insert them into R'. Forward links of the selected object R' have to be adjusted to point to their corresponding spatial entity in S' rather than S.

```
db_extract (S1 ,R)
```

```
/* For all the spatial description in S1  
retrieve their tuples from R.  
store it in R' * /
```

```
begin
```

```
    initialise R';
```

```
    traverse S1;
```

```
    for each description in S1 do
```

```
begin
```

```
    db := get tuples;
```

```
    retrieve db relation from R:
```

```
    insert into R';
```

```
end;
```

```
adjust R' for proper link to return;
```

```
end .
```

## Relational and Spatial Joint :

The standard relational joint operation behaves slightly different when performed on top of the hybrid data\_architecture. When we join the relation where one or both of them have spatial attributes, we have to build new spatial data structure to store the spatial information of the tuples participating in the join operations. This involves the execution of the operation sp\_extract twice; one for each spatial attributes, to store the object region that participated in the relational join.

On the other hand, a spatial join on two spatial relations, each having spatial attributes is performed as follows; the spatial operation corresponding to the spatial join is executed first. It identifies the matching pairs of spatial object that will participate in the resulting spatial relation. Two spatial data structures are created here, each of them stores the qualifying spatial objects that originate from one of the two input spatial relations. Two invocations of the operator db\_extract are needed in order to build the resulting join relation. We need to retrieve the tuples corresponding to the qualifying application and object and each of the two input spatial relations have to be retrieved. Then tuples of matching spatial objects are merged and stored in the join relation. All the operations involved in performing spatial or relational select and join are encompassed in what we term **extended operator**.

#### 4.3 Extended Operators and Spatial Relations :

We now address some implementation issues in hybrid system. We saw that standard database operations such as joins and selection as well as spatial operations need to be redefined when viewed in the context of the hybrid data architecture. The reason is that we must maintain consistency among separate but related structures; mainly the relation describing the nonspatial aspects of homogeneous objects and the spatial data structure describing the spatial aspect of the objects.

Now we introduce the notion of extended operators as an implementation tool that encapsulated spatial relation. We redefine spatial and nonspatial operations using extended operators . Extended operators provide a uniform interface that masks the difference between spatial and nonspatial operations. They also serve as a way to keep both the spatial and nonspatial components of a spatial relation consistence.

We assume that we have a collection of homogeneous objects  $O$  referred to by the pair  $\langle R, S \rangle$ , where  $R$  is a relation that stores the non-spatial attributes information of  $O$  and  $S$  is a spatial data structure that stores the spatial attributes information of  $O$ . Again ,we assume that the set  $O$  is described by only one spatial attribute and hence has only one spatial data structure associated with it. Relaxing this assumption is easy. We use the notation "op" ( $U$  where  $U$  is either  $R$  or  $S$  or the

pair  $\langle R, S \rangle$  (depending on the context) to indicate that operation "op" is performed on U.

Extended operators are classified into relational and spatial-based operators;  $x\_op_r$  and  $x\_op_s$ , respectively (r for relational and s for spatial). They are defined using the operators `sp_extract` and `db_extract`, as building blocks in addition to the corresponding un-extended operators.

The Extended Select Operator:

The extended select operator can be either spatial or relational. The extended relational select, referred to as  $x\_op_r$ , takes as its arguments the pair  $\langle R, S \rangle$  and returns the pair  $\langle T, U \rangle$ . Relation T is formed by applying the un-extended relational select operator on R, i.e.  $T = op_r(R)$ . Next, operator `sp_select` is executed to form the resulting spatial data structure U. The spatial object corresponding to the tuple in the resulting relation T are stored into U. The spatial and non-spatial components of each resulting object are then linked. The extended spatial select operator  $x\_op_s$  behaves analogously. First, the spatial select operator `op` is performed and the selected spatial objects are then stored into a resulting spatial data structure U. Operator `db_extract` is then used to extract the tuples, corresponding to the spatial objects stored in U, to build an output relation. In summary,

$$\begin{aligned}x\_op_r(\langle R, S \rangle) &= \langle op_r(r), sp\_extract(op_r(r), S) \rangle, \\x\_op_s(\langle R, S \rangle) &= \langle db\_extract(R, OP_s(s)) \rangle.\end{aligned}$$

The Extended Join Operation:

The extended relation join  $\langle R_3, (S'_1, S'_2) \rangle = x\_op_{rj}(\langle R_1, S_1 \rangle, \langle R_2, S_2 \rangle)$  ( $rj$  denotes relational join) first applies the relational join operator  $op_{rj}$  on  $R_1$  and  $R_2$ . This results in a join relation  $R_3$  which has two spatial attributes  $R_{3s_1}$  and  $R_{3s_2}$ , where  $s_1$  and  $s_2$  are the spatial attributes that originally belonged to  $R_1$  and  $R_2$  respectively. Operator  $sp\_extract$  is executed twice, once for each spatial attribute to build the corresponding data structure  $S'_1$  and  $S'_2$ . In summary,

$$x\_op_{rj}(\langle R_1, S_1 \rangle, \langle R_2, S_2 \rangle) = \langle R_3, (S'_1, S'_2) \rangle, \text{ where}$$

$$R_3 = op(R_1, R_2),$$

$$S'_1 = op\_extract(S_1, R_{3s_1}), \text{ and}$$

$$S'_2 = sp\_extract(S_2, R_{3s_2}).$$

The extended spatial join  $\langle R_3, (S'_1, S'_2) \rangle = x\_op_{sj}(\langle R_1, S_1 \rangle, \langle R_2, S_2 \rangle)$  ( $sj$  denotes spatial join) first applies the spatial join operator on  $S_1$  and  $S_2$ . This results in two data structures,  $S'_1 \subseteq S_1$  and  $S'_2 \subseteq S_2$ .  $S'_1$  and  $S'_2$  represents the complete description of the spatial object participating in the spatial join from each side. In some cases  $op_{sj}$  may result in just one data structure,  $S'_3$  that carries some combined spatial information. We also use an operator that we term as  $db\_j\_extract$  which operates on the output spatial data structure ( $S'_1$  and  $S'_2$  or on  $S'_3$ ) to produce the join relation,  $R_3$ . In summary,

$$\begin{aligned}
x_{op_{s_1}}(\langle R_1, S_1 \rangle, \langle R_2, S_2 \rangle) &= \langle R_3, (S'_1, S'_2) \rangle, \text{ OR} \\
&= \langle R_3, S'_3 \rangle, \text{ WHERE} \\
(S_1, S_2) &= op_{s_1}(S_1, S_2), \text{ AND} \\
R_3 &= db\_j\_extract(R_1, R_2 \{S'_1, S'_2\}), \text{ OR} \\
S'_3 &= op_{s_1}(S_1, S_2), \text{ AND} \\
R_3 &= db\_j\_extract(R_1, R_2, S'_3).
\end{aligned}$$

#### 4.4 SPECIAL FEATURES OF A HYBRID SYSTEM

In summary we can give the special features of a hybrid system as follows:

A hybrid system features a dual architecture where spatial data is stored separately from non spatial data. This may not permit merging of spatial and non-spatial data into common multiple attribute indexes. However, we believe that this feature is too unnecessary and of less practical use.

Spatial data is stored in disk-based spatial data structures that apply some buffering mechanism and are suitable for the type of data and required operations. This matches with the fact that spatial data is voluminous. It also avoids the need for down or up-loading of spatial data from an off-line representation into on-line representation. Considering the large size of spatial data the down-/up-loading approach seems impractical.

We maintain bi-directional links between the spatial and non-spatial description of a spatial object. Although maintaining these links imposes some overhead it provides flexibility in browsing between the spatial and non-spatial side of the system, thereby permitting operations to be performed in any arbitrary order chosen by the optimizer. Maintaining only uni-directional links imposes some restrictions on the optimizer so that the optimizer is forced to favour one aspect of the data over the other.

Since spatial data is stored in separate structures we avoid the duplication of spatial data when the objects are referenced more than once. Consider the relation resulting from joining two relations each having one spatial attribute. If data were stored in textual form inside a tuple, every time this tuple participates in the join the textual description of the spatial object needs to be duplicated as well. However in hybrid system we just duplicate the spatial index referring to the spatial description and not the whole description.



## CHAPTER 5

### QUERY PROCESSING AND OPTIMIZATION

#### 5.1 QUERY PROCESSING

Query processing stage is very important for the optimization as they affect the optimization process to a great extent. To illustrate this I will take an example and will show the different SQP formed and how the selection of a plan can affect the query processing. The use of extended operators has been illustrated.

#### THE EXAMPLE

Let us take the example of finding all the parks within ten kilometers of the university named "JNU" in Delhi.

The extended query will look like as follows:

```
SELECT    L.name L.usage
FROM      land-use L,land-use U
WHERE     U.usage ="university"
AND       U.name  = "JNU"
AND       L.usage = "park"
AND       With_in(U.location,L.location,10)
AND       in_window(L.location,w)
```

## 5.2 ANALYSIS OF EXAMPLE

By using extended operators we have a homogeneous interface. Otherwise, each extended operator results in a relation and one or more spatial data structure. This allows permuting the operation in any desired order. Also, the flexibility of hybrid system is demonstrated.

First logical transformation will make such transformations which will make the circles to be drawn with "JNU" as the center and not that the circles from different parks will be checked for "JNU" to fall in it. Many other transformations will be made which will help in faster search.

The decomposition process will separate the spatial and aspatial parts so that different processes can effectively be used and they may also be used in proper way to minimize the search process at plan formulation and query processing stage.

Now, first several different execution plans for the above query will be shown, next few possible optimization will be shown which may occur. The following shorthand notations are used.

$x_{db_p}$  :select L.name, L.usage (db-projection)

$x_{db_{s_1}}$  :U.usage = "university" (db-selection)

$x_{db_{s_2}}$  :L.usage = "park" (db-selection)

$x_{sp_{w_1}}$  :within (U.location, L.location, 10)

$x_{sp_{w_2}}$  :in\_window (L.location, w)

Now consider the following different orders of query execution which are called as the SQP (query execution plan)

```

query plan 1 : x_dbs1 , x_dbs2 , x_spw1 , x_spw2 , x_dbp
query plan 2 : x_dbs1 , x_dbp , x_spw2 , x_dbs2 , x_spw1
query plan 3 : x_spw2 , x_dbs2 , x_dbs1 , x_spw1 , x_dbp
query plan 4 : x_dbs2 , x_spw2 , x_dbs1 , x_spw1 , x_dbp
query plan 5 : x_spw2 , x_dbs1 , x_dbs2 , x_spw1 , x_dbp

```

### 5.3 DESCRIPTION OF PLAN 1

We describe query plan 1 in more detail. The execution plans are presented using the most general forms of extended operators as described previously in chapter 3. Here more clarification is done :

```
x_db : U.usage = "university"
```

Relation land-use is searched (possibly through an index on attribute usage) and the qualified tuples are selected. The result of the selection operation is stored in a temporary relation  $T_{S_1}$ . S-extract traverses  $T_{S_1}$  and extracts from S, the spatial data structure storing instances of the attribute land-use. Location of type region, the regions (universities) corresponding to the tuples in relation. The extracted regions are stored in a temporary spatial data structure  $S_{S_1}$ . Notice that the database selection is performed by the DBMS and the spatial extract operation is performed by the spatial process.

```
x_dbs2 : L.usage = "park"
```

This operation is executed in the same way as operation  $x_{db_{S_1}}$ . It operates on the original relation land-use but outputs a temporary relation  $T_{S_2}$  and a temporary spatial data structure  $S_{S_2}$ .

$x_{sp} : \text{within} (T . \text{location} T . \text{location}, 10)$

Given relation  $T_{S_1}$  and  $T_{S_2}$  spatial data structures  $S_{S_1}$  and  $S_{S_2}$  and a radius of expansion with value 10, the spatial operator  $\text{within}$  first operates on  $S_{S_1}$  and  $S_{S_2}$  to generate the regions in  $S_{S_2}$  that are within 10 units of distance from regions in  $S_{S_1}$ . The result is another temporary spatial data structure  $S_{S_3}$ . By using the operator  $\text{db\_extract}$  and  $S_{S_3}$ , the spatial join relation is built consisting of tuples from  $T_{S_1}$  and  $T_{S_2}$  that are within the given distance from each other. The result of the spatial join is stored in temporary relation  $T_{S_3}$  that has two spatial attributes  $\text{location1}$  and  $\text{location3}$  of type region corresponding to the data structures  $S_{S_1}$  and  $S_{S_3}$  respectively. For referencing purposes only, all the attributes joined from relation  $T_{S_1}$  are postfixed by the letter 1 (e.g.  $\text{usage1}$ ), while the attributes joined from relation  $T_2$  are postfixed by the letter 3, (e.g.  $\text{usage3}$ ).

$x_{sp_{W_2}} : \text{in\_window} (T_{S_3} \text{location3}, W)$

The spatial selection operation  $\text{in\_window}$  performs the window operation on  $S_{S_3}$ , the data structure corresponding to spatial attributes  $\text{location 3}$ . This results in yet another temporary data structure  $S_{S_4}$ . The tuples in  $T_{S_3}$  corresponding to

the spatial objects in  $S_{S4}$  (i.e. inside the window  $W$ ) are then extracted using operation `db_extract` into the temporary relation  $T_{S4}$ .

```
x_dbp: select TS4. name3 ,TS4. usage3
```

Attributes `name3` and `usage3` are projected from  $T_{S4}$  resulting in relation  $T_{S5}$ . Another attributes including their corresponding spatial data structures, if any are deleted. Below is shown a summary of the above execution plan.

```
x_dbS1 (<U , S >) == > <TS1 , SS1 >
x_dbS2 (<L , S >) == > <TS2 , SS2 >
x_spW1 (<TS1 , SS1> , < TS2 , SS2> , 10) == > <TS3 , SS3 >
x_spW2 (<TS3 , SS3>) == > <TS4 , SS4 >
x_dbp (<TS4 , SS4>) == > <TS5 , SS5 >
```

#### 5.4 OTHER POSSIBILITIES

Extended operators can be executed in any desirable order because of their uniform interface. This gives us flexibility in the sense that it enables us to perform query optimization. In addition, once candidate execution plan is selected for execution more optimization can take place. We start with a general and complete execution plan of the query using extended operators, then apply some transformation rules to enhance the performance of the execution plan while still producing the same correct results as the original execution plan. In other words, depending on the context, we can select

less general and yet simpler forms of extended operators that are necessary to execute the given query with better performance. Below, we only list a range of these rules.

Rearranging the order of the above operations may result in better performance. For example, in the above plan, performing the spatial selection, `in_window` before the spatial join "within" could have resulted in better performance. Also, performing the database projection earlier reduces the I/O overhead. Query plan 2, given above, reflects both of these enhancements. In this case, the spatial query optimizer stops maintaining the spatial data structure of the dropped attributes as early as possible. This is also applicable when the user does not select relational attributes in the select clause (e.g. `select road_coords from..`) Applying this rule to query plan 1 results in the query plan summarized in the below given expressions. Notice that relation U and L are projected at the same time that the selection operation `x_db` and `x_db` are performed on U and L, respectively.

$$\begin{aligned}
 x\_db_{S_1, \rho}(\langle U, S \rangle) &== \langle T'_{S_1}, S_{S_1} \rangle \\
 x\_db_{S_2, \rho}(\langle L, S \rangle) &== \langle T'_{S_2}, S_{S_2} \rangle \\
 x\_db_{W_1}(\langle T'_{S_1}, S_{S_1} \rangle, \langle T'_{S_2}, S_{S_2} \rangle, 10) &== \langle T'_{S_3}, S_{S_3} \rangle \\
 x\_sp_{W_2}(\langle T'_{S_3}, S_{S_3} \rangle) &== \langle T_{S_5}, \emptyset \rangle
 \end{aligned}$$

In addition, several optimization steps can be performed within each operation sequence (application plan). As an example, we can avoid the creation, writing, an subsequent

reading of several temporary relations and spatial data structures (e.g.  $S_{s4}$  is built and stored but was deleted after the projection step).

As another example, if we cascade two spatial operations, we can defer the `db_extract` operator until both spatial operators are performed. This means that we can build the relational part at the end using one `db_extract` operator. This will save one execution of the `db_extract` operator as illustrated by the following example. Suppose we wish to perform the operation  $x_{op_{s2}}(x_{op_{s1}}(\langle R, S \rangle))$  where  $x_{op_{s1}}$  and  $x_{op_{s2}}$  are extended spatial operators. One way to perform this operation is as follows. Let  $S_1$  be  $op_{s1}(S)$  and  $R_1$  be `db_extract` ( $R S$ ) Then,

$$x_{op}(x_{op}(\langle R, S \rangle)) == \langle db\_extract(R_1, op_{s2}(S_1)), op_{s2}(S_1) \rangle.$$

The above formula uses the standard definition of extended operators. However, a better way to perform the same operation is to use  $R$  instead  $R_1$ , in the above formula. The formula then is:

$$x_{op_{s2}}(x_{op_{s1}}(\langle R, S \rangle)) == \langle db\_extract(R, op_{s2}(S_1)), op_{s2}(S_1) \rangle.$$

The new formula is functionally equivalent to the former one but is more efficient. It uses just one execution of the `db_extract` operator. This optimization saves execution time since it avoids creating, writing, and then reading an intermediate temporary relation. Other possible optimization

strategies (e.g. intersecting spatial or tuple pointers, and simplified versions of spatial operations) are also present and we expect more domain specific rules to emerge if we begin the implementation phase of hybrid system.



## CHAPTER 6

### CONCLUSION

An optimization model for augmented SQL has been proposed in this dissertation. Many of the techniques employed by the proposed extended optimizer are based on existing methods used in current optimizer. Although, the strategy is proposed for a particular extension of SQL, we believe the same method can be used for other languages that are based on SQL.

The major contribution of the dissertation is a global optimization strategy for extension of SQL, which requires additional indexing structures to materialize the additional relationships. The proposed strategy does not require extensive modifications of existing DBMSs. The optimization strategy consists of the following modules: extended logical transformation, decomposition, the generation of subquery sequences, and the selection of the best subquery sequence. Queries are transformed into logically equivalent queries that are more efficient to evaluate. As the SQL backend is not capable of evaluating the spatial components of the extended languages, the decomposition breaks a query into several subqueries so that the SQL backend can be used for SQL (UA) subqueries, and a spatial processor for evaluating spatial selection criteria, processes, spatial subqueries. With an unordered set of subqueries, all plausible different arrangements of subqueries

are considered. This involves the generation of plans and the heuristic pruning of the search space so that only plausible plans are examined.

While such optimizer may not produce the best strategy, it can however produce a reasonable strategy. The approach is simple. It will not involve a substantial increase in the costs. The extension to the DBMS is minor: the SQL parser is modified to parse a larger set of predicates and the extended optimizer is built on top of the existing optimizer.

The implementation confirmed that the rewriting rules are correct and that for each hybrid query, all sequences produced by the decomposition algorithm and SQP formulation algorithm gave the same answer. Simple queries that involve three result rewriting rules were used for the test. The results obtained by different SQPs were exactly the same.

The major limitation with the extended optimization model is that we must be able to access the existing DBMS at more than the user's level. The unnested aspatial subqueries formed by the decomposition strategy are syntactically and semantically correct. It would be very inefficient to submit each query individually and then make use of the temporary results as part of the execution strategy. We need to control the buffer management and to access temporary storage without having to

submit the decomposed unnested aspatial queries at the user's level. Using SQL queries with INTO clauses introduces more problems than just creating temporary tables, expensive rereading operations are also required.

## REFERENCES

1. Aho, A. V. and Ullman, J. D. 1979. "Universality of data retrieval languages". In proceedings of the 6th ACM Symposium on Principles of Programming Languages (San Antonio, Tex., Jan. 29-31). ACM, New York, pp. 110-120.
2. ANSI/X3/SPARC 1975. Study Group on DBMS Interim Report. SIGMOD FDT Bull. 7,2,1975.
3. Bancilhon, F. 1978. "On the completeness of query language for relational database". In proceedings of the 7th Symposium on Mathematical Foundations of Computer Science. Springer-Verlag, Berlin and New York, pp. 112-123.
4. Chandra, A. K. and Harel, D. 1980. "Computable queries for relational databases". In proceedings of the 11th ACM Symposium on Theory of Computing (Atlanta, Ga., Apr. 30-May 2 ). ACM, New York, pp. 303-318.
5. Chandra, A. K. and Harel, D. 1980. "Structure and Complexity of relational queries". In Proceedings of 21st IEEE Symposium on Theory of Computing (Syracuse, N. Y., Oct.). IEEE, New York, pp. 333-347.
6. Chang, C. L. and Lee, R. C. T. 1973. Symbolic Logic and Mechanical Theorem Proving. Academic Press, New York.

7. Codd, E. F. 1970. " A relational model of data for large shared data banks". In Communication of the ACM, Vol. 13, No. 6, pp. 377-387.
8. Codd, E. F. 1979. "Extending the relational database model to capture more meaning". In ACM Transaction on Databases System, Vol. 4, No. 4, pp. 397-434.
9. Cooper, E. C. 1980. "On the expressive power of query languages for relational databases". In Technical Reports 14-80, Computer Science Deptt., Harvard Univ., Cambridge, Mass.
10. Dahl, V. 1982. "On databases systems development through logic". In ACM Transaction on Database Systems, Vol. 7, No. 1, pp. 102-123.
11. Date, C. J. 1977. "An Introduction to Database Systems". Addison-Wesley, Reading, Mass.
12. Enderton, H. B. 1972. "A Mathematical Introduction to Logic". Academic Press, New York.
13. Hammer, M. T. and Zdonik, S. P., Jr. 1980. "Knowledge based query processing". In Proceedings of the 6th International Conference on Very Large Databases.

14. King, J. J., 1981, "QUIST : A system for semantic query optimization in relation databases". In Proceedings of the 7th International Conference on Very Large Databases (Cannes, France, Sept. 9-11). IEEE, New York, pp. 510-517.
15. Kowalski, R. 1974. "Predicate logic as a programming language". In DCL memo #75, Deptt. of Computational Logic, Edinburgh University, Edinburgh.
16. Kowalski, R. 1981. "Logic as a database language". In Proceedings of the Advanced Seminar on Theoretical Issues in Databases ( Centraro, Italy, Sept.).
17. Kuhns, J. L. 1967. "Answering questions by computers --A logical study". In Rand Memo RM 5428 PR, Rand Corp. Santa Monica, California.
18. McSkimin, J. and Minker, J. 1977. "The use of a semantic network in a deductive question answering system". In Proceedings of the 5th International Joint Conference on Artificial Intelligence (Cambridge, Mass., Aug.), pp. 50-58.
19. Ooi, B. C. 1988. "Efficient Query processing for Geographic Information System". In Proceedings of the 11th IEEE.

20. Ooi, B. C. and McDonnell, K. J. 1987. "Spatial kd tree and indexing mechanism for Spatial Databases". In Proceedings of 11th IEEE.
21. Pirotte, A. 1978. "High level database query languages". In Logic and Databases . Plenum, N.Y., pp. 409-436.
22. Prade, H. and Testemale, C. (1984). "Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries". Inform. Sci. 34, 1984, pp. 115-143.
23. Raju, K.V.S.V.N. and Majumdar, A.K. (1988). "Fuzzy functional dependencies and Lossless Join decomposition". ACM Trans. Database Syst. 13, 2, (June, 1988), pp. 129-166.
24. Saxena, P.C. and Tripathy, R.C. "Cancellation law and a complete axiomatization of functional dependencies in database relations". Computer and Artificial Intelligence, 8(4), (1989), pp. 347-356.
25. Vassiliou, Y. (1980). "Functional dependencies and incomplete information". In Proceedings of 6th International Conf. on Very Large Databases, IEEE Comput. Soc., Los Angeles, 1986, pp. 260-269.

26. Wong, E. and Youssfi, K.,1976. "Decomposition of a strategy for query processing", ACM Transactions on Database System. 31 YAO. S.B. 1979,"Optimization of query evaluation algorithms",ACM Transactions on Database Systems 4,2 pp. 133-155.
27. Zadeh, L.A. "Fuzzy sets", Inform. and Control, 8, 1965, pp. 338-353.
28. Zadeh, L.A. "Fuzzy sets as a basis of theory of possibility". Fuzzy sets and Systems 1, 1, 1978, pp. 3-28.
29. Zadeh, L.A. "PRUF : A meaning representation language for natural languages". Internat. J. Man-Machine Studies 10, 1978, pp. 395-460.
30. Zaniolo, C. (1984). "Database relations with null values". J. Comput. System Sc. 28, pp. 142-160.