An Intelligent Framework For Distributed Query Optimization Of Spatial Data In Geographic Information Systems

Prashanta Kumar Patra[1] Chittaranjan Pradhan[2] Animesh Tripathy[2]

[1]Department of Computer Science & Engineering, College of Engineering & Technology, BPUT, Bhubaneswar, Orissa-751003 India [2]Department of CSE, SOT, KIIT University, Bhubaneswar, Orissa, India [2]Department of CSE, SOT, KIIT University, Bhubaneswar, Orissa, India

Summary
The Geographic Information System (GIS) uses the spatial database for its data storing purposes. As the spatial database takes huge space, the size and data retrieval cost of database increases. That's why we have to use some optimized technique to retrieve the data from the database. Also, we can apply the distributed database concept to the spatial database to achieve better performance. After using the optimization technique and the distributed database concept we can achieve better query processing at affordable cost.

Key words:
GIS, Distributed Database, Query Optimization, Spatial Database, Fragments..

1. Introduction

A GIS is a system for capturing, storing, analyzing, and managing data and associated attributes which are spatially referenced to the earth. That means, it is a computer system capable of integrating, storing, editing, analyzing, sharing and displaying geographically-referenced information. Geospatial data separate GIS from other information Systems. For ex: Take the example of roads. To describe a road, we refer to its location (i.e. where it is) and its characteristics (e.g. length, name, speed limit and direction). The location (or geometry or shape) represents Spatial Data, where as the characteristics are attribute data. Thus, any geospatial data has 2 components: • Spatial data • Attribute data.

Spatial data: It describes the locations of spatial features, which may be discrete or continuous. Discrete features are individually distinguishable features that do not exist between observations. This includes Points (e.g. wells), lines (e.g. roads) and areas (e.g. land use types). Continuous features are features that exist spatially between observations. Examples of continuous features are: elevation and precipitation. The data models used for the spatial data are:
Vector Data Model: Data are composed of points, lines, and polygons. These features are the basic features in a vector-based GIS, such as ArcGIS9. The basic spatial data model is known as "arc-node topology." One of the strengths of the vector data model is that it can be used to render geographic features with great precision. However, this comes at the cost of greater complexity in data structures, which sometimes translates to slow processing speed.

Raster Data Model: Raster datasets are composed of rectangular arrays of regularly spaced square grid cells. Each cell has a value, representing a property or attribute of interest. While any type of geographic data can be stored in raster format, raster datasets are especially suited to the representation of continuous, rather than discrete, data.

Attribute Data: Attribute data describe the characteristics of spatial features. For raster data, each cell has a value that corresponds to the attribute of the spatial feature at that location. A cell is tightly bound to its cell value. For vector data, the amount of attribute data to be associated with a spatial feature can vary significantly.

Data Layers: In most GIS software data is organized in themes as data layers. This approach allows data to be input as separate themes and overlaid based on analysis requirements. This can conceptualized as vertical layering the characteristics of the earth's surface. The overlay concept is so natural to cartographers and natural resource specialists that it has been built into the design of most CAD vector systems as well.

2. GIS Data Management:

Traditionally map data have been recorded in the form of lines and symbols on paper, and descriptive data or attributes have been restored in written form on file cards and various documents. These traditional data documentations are organized in various systems of filing cabinets and drawers, and each data repository may be regarded as a "library" or "bank" from which users may retrieve information. A data bank may either be available

Manuscript received May 5, 2008
Manuscript revised May 20, 2008

to a wide range of users or restricted to only a few authorized users. In addition, the data deposited may be in the form of one or more files.
However, there is no single or a blue print to manage GIS database, but a logically organized structure for the data management system is clearly important. Ideally, implementing GIS database management can be viewed as a process that begins with needs assessment, continues through data acquisition and analysis, interpretation, data archiving and data sharing with various users and organizations *[Fig 1]*.

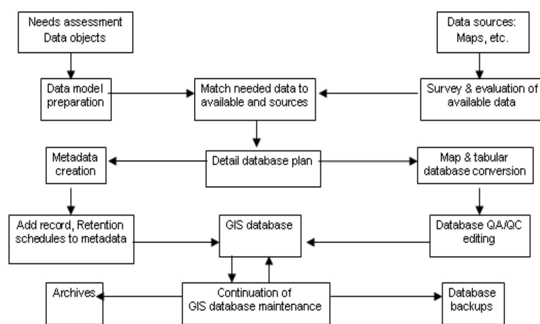


Figure 1 Lifecycle of a GIS database

The processes of database management system can be categorized into six components as follows: a) An inventory of existing data and resources will have to be compiled, and priorities for implementation set. b) Data will have to be designed and organized by establishing structure within and among data sets that will facilitate their storage, retrieval and manipulation. c) Procedures will be required for data acquisition and quality assurance and quality control (QA/QC). d) Data set documentation protocols, including the adoption or creation of metadata content standards and procedure for recording metadata, will need to be developed. e) Procedures for data archival storage and maintenance of printed and electronic data will have to be developed, and f) An administrative structure and procedures will have to be developed so responsibilities are clearly defined.
GIS Data are divided logically into two categories: geometric data and attribute data. They can also be stored physically different although the relationships between the two categories of data must be preserved regardless of whether the division is physical or logical. This can be satisfied through one of the following four approaches:

• Two separate database systems, one for geometrical data and one for attribute data;

• A single database system storing both categories of data;

• One database for geometrical data connected to several different databases for attribute data;

• Several databases for geometrical data and attributes joined into one system.

A Hybrid system stores geometrical data and attribute data in two separate databases *[figure 2]*. Commercially available relational DBMS is generally employed for attribute data for this purpose. The system may include both raster and vector data. To increase the search and record speed, the geometrical data are often split into different layers and aerial units (i.e. map sheets, etc.) for storage.

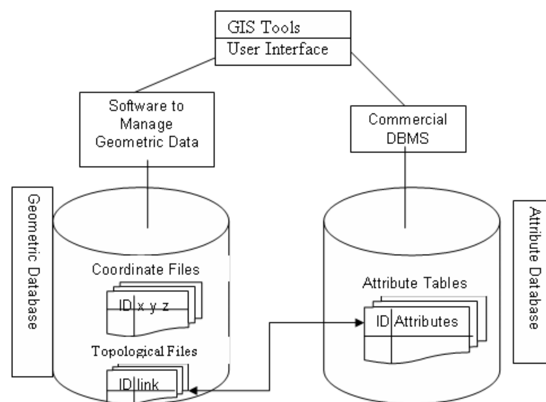


Figure 2 Hybrid system of database management

A spatial data structure is associated with each spatial attribute in the schema and is used to store all data instances of that spatial attribute over the set of homogeneous objects. The spatial data structure is used as an index for spatial objects as well as a medium for performing spatially- related operations. Depending on the attribute's spatial data type (e.g. region, line, or point), a spatial data structure is selected for handling.
The data instances of the set of non-spatial attributes are stored in database relations. Each tuple in the relation corresponds to one object. The following figure illustrates how we link spatial and non-spatial attribute values of an object. In particular we maintain two logical links between the spatial and non-spatial data instances of an object: forward and backward links.
Forward links are used to retrieve the spatial information of an object given the object's non-spatial information. On the other hand, backward inks are used to retrieve the non-spatial information of an object given the object's spatial information. Maintaining forward and backward links between the spatial and non-spatial aspects of a set of

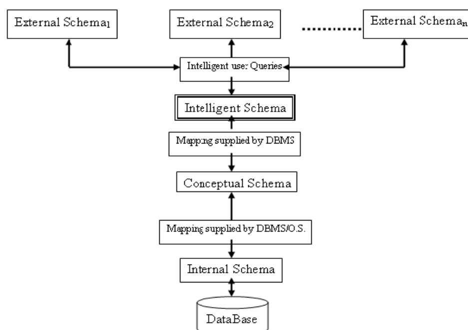
objects facilitates browsing in the two parts as well as permits efficiently query processing.
A commonly used view of data approach is the three-level architecture suggested by ANSI/SPARC (American National Standards Institute/Standards Planning and Requirements Committee). The three views are:

- The external level is the view that the individual user of the database has. This view is often a restricted view of the database and the same database may provide a number of different views for different classes of users.
- The conceptual view is the information model of the enterprise and contains the view of the whole enterprise without any concern for the physical implementation. This view is normally more stable than the other two views.
- The internal view is the view about the actual physical storage of data. It tells us what data is stored in the database and how. At least the following aspects are considered at this level

The challenges in the standard database model are:

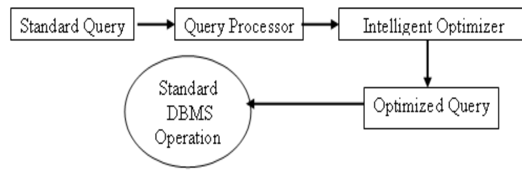
- Optimization is generally done on spatial data and not on attribute data.
- Attribute data constitutes a major portion of a spatial query. There fore, optimization on spatial data takes more time as compared to RDBMS.
- Huge databases require proper optimization for efficient search.

To cover the above challenges of the standard database architecture, we proposed new database architecture [figure 3] by introducing an extra schema "Intelligent Schema" in between the External schema and the Conceptual Schema.



Proposed Database Architecture

The advantages of the Model we have proposed are: Intelligent preprocessing and Optimization of attribute data, which are shown in the following model:



3. Query Optimization:

Query optimization is one of the most important tasks of a relational DBMS. One of the strengths of relational query languages is the wide variety of ways in which a user can express and thus the system can evaluate a query. A given query can be evaluated in many ways and the difference in cost between the best and worst plans is our main theme of discussion. Realistically, we cannot expect to always find the best plan, but we expect to consistently find a plan that is quite good.
Queries are parsed and then presented to a query optimizer, which is responsible for identifying an efficient execution plan. The optimizer generates alternative plans and chooses the plan with the least estimated cost.
The space of plans considered by a typical relational query optimizer can be understood by recognizing that a query is essentially treated as a $\sigma - \Pi - \infty$ algebra expression, with the remaining operations (if any) carried out in the result of the $\sigma - \Pi - \infty$ expression. Optimizing such a relational algebra expression involves two basic steps:

- Enumerating alternative plans for evaluating the expressions. Typically, an optimizer considers a subset of all possible plans because the number of possible plans is very large.
- Estimating the cost of each enumerated plan and choosing the plan with the lowest estimated cost.

A query evaluation plan (or simply plan) consists of an extended relational algebra tree, with additional annotations at each node indicating the access methods to use for each table and the implementation method to use for each relational operator.

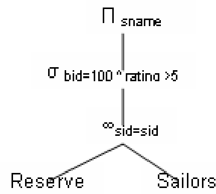
Consider the following SQL query:

SELECT S.sname FROM Reserves R, Sailors S WHERE R.sid=S.sid AND R.bid=100 AND S.rating >5

This query can be expressed in relational algebra as follows:

$\Pi_{sname}(\sigma_{bid=100 \wedge rating>5}$ (Reserves $\infty_{sid=sid}$ Sailors))

This expression is shown in the form of a tree as:



The algebra expression partially specifies how to evaluate the query – we first compute the natural join of Reserves and Sailors, then perform the selections, and finally project the sname field.

4. Query Optimization in spatial data:

Consider the following two schemas for the further discussion.

CREATE TABLE **LAND_USE** (NAME CHAR(40), ADDRESS CHAR(100), LOCATION REGION, USAGE CHAR(40), ZIP NUMBER, IMPORTANCE NUMBER);

CREATE TABLE **ROADS** (ROAD_ID NUMBER, ROAD_NAME CHAR(30), ROAD_TRAFFICABILITY NUMBER, ROAD_LANES NUMBER, ROAD_COORDS LINE_SEGMENT);

Example 1: *Find all roads other than "Route 1" that pass through a given window w.*

SELECT ALL FROM ROADS WHERE IN_WINDOW (ROAD_COORDS, w) AND ROAD_NAME! ="ROUTE 1";

Below, we give several strategies for processing this query as well as others.

Plan 1- Un-optimized:

$<T_1,S_1> \leftarrow$ x_sp_window(<R,S>,w)

$<T_2,S_2> \leftarrow$ x_db_select($<T_1,S_1>$,db_cond)

Plan 1 uses the notation of extended operators without any further optimizations. We can rephrase plan 1 as:

$S_1 \leftarrow$ sp_window(S,w)
$T_1 \leftarrow$ db_extract(R,S_1)
$T_2 \leftarrow$ db_select(T_1,db_cond)
$S_2 \leftarrow$ sp_extract(S_1,T_2)

This helps to clarify the optimization steps demonstrated in the following plans. A reordering of the operations is also possible as given by:

$T_1' \leftarrow$ db_select(R,db_cond)
$S_1' \leftarrow$ sp_extract(S,T_1')
$S_2' \leftarrow$ sp_window(S_1',w)
$T_2' \leftarrow$ db_extract(T_1',S_2')

Plan 2- Further Reorderings:

$S_1 \leftarrow$ sp_window(S,w)
$T_1 \leftarrow$ db_select(R,db_cond)
$<T_2, S_2> \leftarrow$ merge(T_1,S_1)

This gives an alternative reordering of the operators in plan 1. Here, the database process and the spatial process each work independently on a different portion of the input data. The results are merged at a later step. The purpose of the merging step is to find the objects that exist in both the input spatial data structure, say S_1, and the input relation, say T_1, and generates an output pair, say $<T_2, S_2>$, that contains all these common objects.

Basically, there are two ways of performing a merge: spatial-driven (sp_merge) and relational-driven(db_merge). sp_merge traverses the spatial data structure S and for each spatial object that it encounters, say o, sp_merge tries to retrieve o's corresponding tuple, say t, through the tuple-id stored with o. If t is found, then t and o are stored in T and U, respectively. Otherwise, o is not part of the result –i.e., it is skipped. A schemating listing of sp_merge is given by:

sp_merge(R,S)

```
/* Merge relation R with spatial data structure S based on the common objects in time. The results are stored in relation T and spatial data structure U.*/
Begin
  initialize relation T,U;
  traverse S;
  for each spatial object o in S do
   begin
    tid:=get o's tuple_id;
    if tid in R then
     begin
      retrieve tid's tuple t from R;
      append t into T;
      insert o into U;
     end;
    end;
 end;
```

db_merge is the same as sp_merge except that the relation R is traversed and the spatial objects(if any) that correspond to the tuples in R are retrieved and stored into the output spatial data structure U. Tuples with no corresponding spatial objects are discarded, while tuples with matching spatial objects are stored into the output relation T. A schematic listing of db_merge is given by:

db_merge(R,S)

```
/* Merge relation R with spatial data structure S based on the common objects in time. The results are stored in relation T and spatial data structure U.*/
Begin
  initialize relation T,U;
  traverse R;
  for each tuple t in R do
   begin
    sid:=get t's spatial_id;
    if sid in S then
     begin
```

```
retrieve sid's spatial object o from S;
insert o into U;
append t into T;
end;
end;
end;
```

Plan 3- Interaction of pointers:

A third method of merging the results of conjunctive selections, in addition to the spatial-based merging and relation- based merging described in plan 2 above, is by intersection of pointers.

This is a well-known technique for answering conjunctive selections where the tuple-ids resulting from each selection are intersected. Intersecting pointers is possible if the selections that are performed generate tuple-ids. This situation can arise when the attributes that comprise the selection condition can be accessed via a secondary index that contains the associated tuple-ids.

We have two types of object-ids, namely tuple-ids and spatial-ids. In addition, the conjunctive selections may be spatial, relational, or both. Incorporating the intersection of pointers technique can be done in two different ways, depending on whether we intersect the tuple-ids or the spatial-ids. This is illustrated by the following example.

Example 2: Find all 4-lane roads that are within r miles of point (x, y).

SELECT ROAD_NAME FROM ROADS, LAND_USE WHERE IN_CIRCLE (ROAD_COORDS,x,y,r) AND ROAD_LANES=4;

1- *Intersection of tuple-ids:* If a secondary index is present on the attribute road_lanes, then when performing the selection based on road_lanes (i.e. road_lanes=4) would generate a set of tuple-ids. The spatial selection in_circle generates the spatial objects that lie inside the specified circle and stores them in a temporary spatial data structure. The result of the intersection is then materialized both from the relational as well as the spatial side. Note that the operation in_circle can generate the list of tuple-ids directly without the need of an extra traversal of the spatial data structure. The resulting plan is given by:

$$L_{sp} \leftarrow sp_in_circle_tid(S,c)$$
$$L_{db} \leftarrow db_select_tid(R,db_cond)$$
$$<T_2, S_2> \leftarrow list_intersect_tid(L_{db},L_{sp},R,S)$$

Operator sp_in_circle_tid is a simplified version of the operator in_circle which returns just the backward link information (tuple-ids) of the selected spatial objects. db_select_tid is a secondary index selection that returns the tuple-ids. Operation list_intersect_tid is given by:

list_intersect_tid(Lo,Lr,R,S)

```
/*Intersect lists Lo and Lr where each list contain tuple_ids and retrieve the tuples and spatial objects corresponding to the common tuple_ids. The results are stored in relation T and spatial data structure U.*/
begin
 initialize T,U;
 I:=intersect Lr and Lo;
 for each tuple_ids tid in I do
  begin
   retrieve tid's tuple t from R;
   append t into T;
   sid:=get t's spatial_id;
   retrieve sid's spatial object o from S;
   insert o into U;
  end;
 end;
```

2- *Intersection of spatial-ids:* The same strategy can be applied when we consider intersecting spatial-ids instead of tuple-ids. The corresponding plan is:

$$L_{sp} \leftarrow sp_in_circle_sid(S,c)$$
$$L_{db} \leftarrow db_select_sid(R,db_cond)$$
$$<T_2, S_2> \leftarrow list_intersect_sid(L_{db},L_{sp},R,S)$$

Operation sp_in_circle_sid(S,c) is a simplified version of operation sp_in_circle which returns just the spatial-ids of the qualified objects(i.e., the ones lying inside the circle c). In order to return the spatial-ids as a result of the database selection, we need to return the value of the spatial attribute for each qualifying tuple in the selection. The operation list_intersect_sid(L_{db},L_{sp},R,S) intersects the two spatial-ids lists resulting from the spatial and relational selections. Then, it retrieves the spatial and non-spatial description of the objects in the intersection. Operation list_intersect_sid is given by:

list_intersect_sid(Lo,Lr,R,S)

```
/*Intersect lists Lo and Lr where each list contain spatial_ids and retrieve the tuples and spatial objects corresponding to the common spatial_ids. The results are stored in relation T and spatial data structure U.*/
begin
 initialize T,U;
 I:=intersect Lr and Lo;
 for each spatial_ids sid in I do
  begin
   retrieve sid's spatial object o from S;
   insert o into U;
   tid:=get o's tuple_id;
   retrieve tid's tuple t from R;
   append t into T;
  end;
 end;
```

Plan 4 - Pushing spatial operations into sp_extract:

Consider the plan 1 again. In this plan, spatial data structure S_1' is written by operator sp_extract and then read by operator sp_window. To avoid an extra traversal of S_1' as well as the read/write overhead, we can perform some spatial conditions on the fly along with operator sp_extract. This technique may be desirable under some but not all circumstances. For example, if the cardinality of the spatial objects is low and if the spatial test to be performed is relatively simple, then it is indeed more economical to perform this spatial test on the fly along with the sp_extract operator instead of storing the result and then retraversing the whole structure. The resulting plan is: T_1'←db_select(R,db_cond)

S_2'←sp_extract_f(S,T_1',in_window(w))

T_2'←db_extract(T_1',S_2')

Note that the use of the new operator sp_extract_f (f denotes filter) which has one additional argument over sp_extract. This argument serves as a spatial selection condition. All the spatial objects extracted should satisfy this condition. Also note that plan 4 uses only one temporary spatial data structure, namely S_2', which is also the output data structure while plan 1 uses two temporary data structures, namely S_1 and S_2 is also the output data structure.

Plan 5 - Pushing database selection into db_extract:

Consider the plan 1again. The relation T1 is written by operator db_extract and then read by operator db_select. This is one form of the use of the pipelining technique to save from creating needless temporary relations and to save on traversal time. The modified plan is:

S_1←sp_window(S,w)

T_2←db_extract_f(R,S_1,db_cond)

S_2←sp_extract(S_1,T_2)

Note the new operator db_extract_f (f denotes filter) which has one additional argument over db_extract. This argument serves as a relational selection condition. Also this plan uses only one temporary relation, namely T_2, which is also the output relation while plan 1 uses two temporary relations, namely T_1 and T_2, where T_2 is also the output relation.

5. Distributed Database:

A distributed database is a collection of data which belong logically to the same system, but are spread over the sites of a computer network. There are two important aspects of a distributed database:

- Distribution: The data are not resident at the same site or processor.
- Logical Correlation: The data have some properties which tie them together so that, we can distinguish a distributed database from a set of local databases which are resident at different sites of a computer network.

The main issues of distributed database we have considered are:

Fragment: It means we are dividing the whole database into different groups called fragments. As a result our database size at one site reduces.

Replicas: It means we putting copies of the fragments at different sites so that if any one fails to work, then its replica works for it.

Concurrency: It means as we are storing the replicas, any modification done at one site that should be reflected at all the replicas present.

A fragment can be defined by an expression in a relational language which takes global relations as operands and produces the fragment as result.

The rules which must be followed when defining fragments are:

- *Completeness Condition:* All the data of the global relation must be mapped into the fragments.
- *Reconstruction Condition:* It must always be possible to reconstruct each global relation from its fragments.
- *Disjointness Condition:* The fragments should be disjoint so that the replication of data can be controlled explicitly at the allocation level. This condition is useful mainly with horizontal fragmentation.

Fragmentation is of three types. They are discussed below as:

Horizontal Fragmentation: It consists of partitioning the tuples of a global relation into subsets; this is clearly useful in distributed databases, where each subset can contain data which have common geographical properties. It can be defined by expressing each fragment as a selection operation on the global relation.

Vertical Fragmentation: It is the subdivision of the attributes of the global relation into groups; fragments are obtained by projecting the global relation over each group. This can be useful in distributed databases where each group of attributes can contain data which have common geographical properties.

Mixed Fragmentation: It is of two types:

- Horizontal followed by Vertical
- Vertical followed by Horizontal

6. Distributed Spatial Query Optimization:

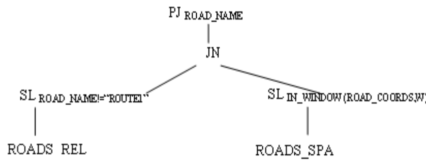
There are four rules for converting a global query into a fragment query and there by optimizing the query. Query optimization uses the Natural Join. For example, take the following query:

Query: Find the roads other than “Route1” that pass through a given window ‘w’

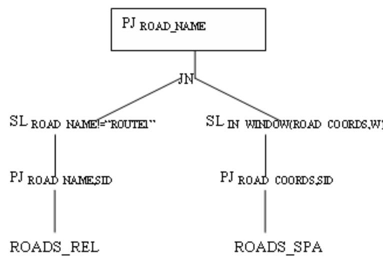
Solution:

SELECT ROAD_NAME FROM ROADS WHERE IN_WINDOW(ROAD_COORDS,W) AND ROAD_NAME!="ROUTE1";

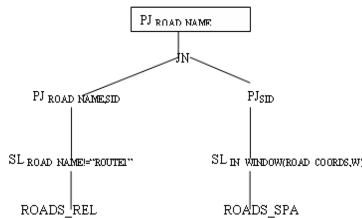
The operator tree for this query is shown as:



Rule 1: *Use Idempotence of the fragment queries using Selection and Projection to generate the fragment queries.* Applying this rule, we get the modified operator tree as:



Rule 2: *Push Selection and Projections down in the tree as far as possible,* i.e. push joins up in the tree as far as possible. Applying this rule, we get the operator tree as:



Rule 3: *Use appropriate Selection and Projection for generating equivalence of fragment relations,* i.e. create fragments of each table.

Let we have made the horizontal fragments.

ROADS_REL=ROADS_REL1+ROADS_REL2+ROADS_REL3

ROADS_REL1=SL ROAD_NAME="ROUTE1" ROADS_REL

ROADS_REL2=SL ROAD_NAME="ROUTE2" ROADS_REL

ROADS_REL3=SL ROAD_NAME="ROUTE3" ROADS_REL

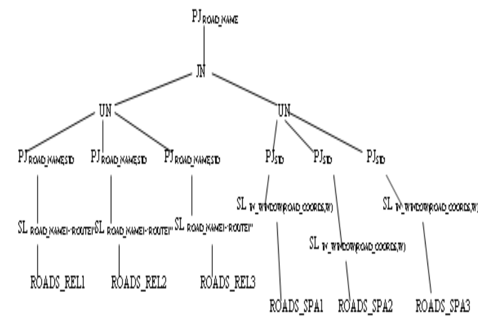
ROADS_SPA=ROADS_SPA1+ROADS_SPA2+ROADS_SPA3

ROADS_SPA1=SL REGIONS="NORTH" ROADS_SPA

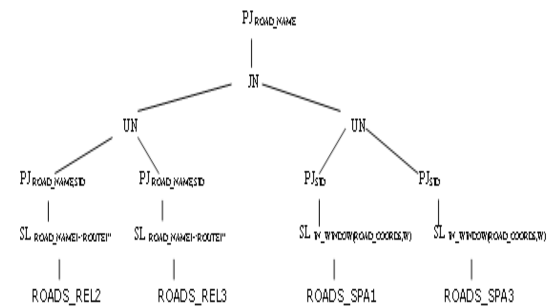
ROADS_SPA2=SL REGIONS="SOUTH" ROADS_SPA

ROADS_SPA3=SL REGIONS="CENTRE" ROADS_SPA

The modified operator tree after applying rule 3 is shown as:



Rule 4: *Replace the branch with an empty relation for the appropriate Selection and Projection if the condition is contradictory,* i.e. if the result of a fragment is null, then remove it. Applying the rule 4, we get the modified tree as:



7. Future Work:

When this proposed model with the distributed database concept will be applied in the general spatial database, then the people can access the spatial data all over the world at an affordable cost. There are so many researchers are working on the management of spatial data. In the near future, this proposed model will be validated.

8. References:

[1] *J. Ronald Eastman, Michele Fulk, and James Toledano: The GIS Handbook. Clark University, 1993*

[2] *ESRI White Paper: GIS Topology, July 2005*

[3] *Ralf Hartmut Güting: An Introduction to Spatial Database Systems, FernUniversität Hagen, Special Issue on Spatial Database Systems of the VLDB Journal (Vol. 3, No. 4), Germany 1994*

[4] *Hanan Samet: Spatial Data Structures, Institute of Advanced Computer Studies And Center for Automatic Research, University of Maryland*

[5] *Petr Kuba: Data structures for spatial data mining, Masaryk*

[6] *University Brno, Czech Republic, September 2001*

[7] *Shashi Shekhar, Pusheng Zhang, Yan Huang, Ranga Raju Vatsavai: Trends in Spatial Data Mining, Department of Computer Science and Engineering, University of Minnesota*

[8] *THE NATIONAL SPATIAL DATA INFRASTRUCTURE, Jerzy Albin,SURVEYOR GENERAL OF POLAND, 10th EC GI & GIS Workshop, ESDI State of the Art, Warsaw, Poland, 23-25 June 2004*

[9] *Database Management Systems(COP 725),Markus SchneiderDepartment of Computer and Information Science and Engineering,CSE Building, Room E450,University of Florida*

[10] *Spatial SQL: A Query and Presentation Language; Max J. Egenhofer, National Center for Geographic Information and Analysis and Department of Surveying Engineering, University of Maine, Orono, ME 04469, USA, MAX@MECAN1.bitnet; IEEE Transactions on Knowledge and Data Engineering 6(1):85-95,1994*

[11] *Spatial SQL: A Query and Presentation Language, Max J. Egenhofer, Member, IEEE*

[12] *Distributed Database Management Systems in the Modern Enterprise, Jason C. Stollings, Bowie State University, Distributed Database Management Systems in the Modern Enterprise*

[13] *Query Optimization, Yannis E. Ioannidis, Computer Sciences Department, University of Wisconsin, Madison, WI 53706.*

Prashanta Kumar Patra received Bachelor degree in Engineering. in Electronics from SVRCET (NIT), Surat, India M.Tech in Computer Engg from I.I.T., Kharagpur and Ph. D. in Computer Science from Utkal University, India. He is presently working as Professor & Head of the Department of Computer Science & Engg , College of Engg & Tech, a constituent college of Biju Patnaik University of Technology, Orissa, India. He has published many papers at National/International journals/ Conferences in the areas of Soft Computing, Image processing, Pattern recognition and Bioinformatics which are the subjects of his research interest.

Animesh Tripathy is currently pursing his Research in Intelligent Database Systems. He has completed his Bachelor of Engineering in Computer Engineering and Master of Technology in Computer Science & Engineering from University of Calcutta. Presently he is working as Asst. Professor in Computer Science Department, KIIT UNIVERSITY, Bhubaneswar, Orissa, India. He has published some innovative research papers in International Journals & Conferences. His major strength lies in GIS, Image Analysis & Intelligent Database Systems.

Chittaranjan Pradhan is a Research Student in Computer Science Engineering at KIIT University, India. His Mater Thesis in Intelligent Information Retrieval Systems. He has published his work in various National Level Conferences and was highly acclaimed. He has completed his B.Tech in Computer Science Engineering. He is currently the Top Rank Holder of the University in his discipline.