Contents

* Overview
* Problem with B+ Trees in Spatial Domain
* Requirements from a Spatial Indexing Structure
* Approaches
* SQL/MM Standard
* Current Issues
Overview

• What is a Spatial Data?
• Who requires Spatial Data?
• What are types of Spatial Data?
• What are types of Spatial Data Queries?
• Spatial Databases and their Applications
Overview

What is Spatial Data?

- Information representing geometry - location - topology
  - 2d - Planar | Surface
  - 3D - Volumetric
  - 4D - Spatio-Temporal

CREATE TABLE CITY (  
  NAME VARCHAR(30),  
  POPULATION INTEGER,  
  CITY_PARKS VARCHAR(30) ARRAY[10],  
  LOCATION ST_GEOMETRY )

We can determine the area of San Francisco by executing a query like this:

SELECT location.area  
FROM CITY  
WHERE name = 'San Francisco'
Overview

Who requires Spatial Data?

* **Local Governments:** City Planning | Traffic Management | Accident Investigation

* **State | Provincial Governments:** Highway Planning | Natural Resource Management

* **National Governments:** Defense | Border Control

* **Extractive Industries:** Mineral | Water Location

* **Farming:** Plot Allocation
Overview

What are types of Spatial Data?

* **Point Data**
  
  * Points in m-Dimensional space.
  
  * No Space | No Area | No Volume.
  
  * Raster Point Data: Each Pixel with added measured value

* **Region Data**
  
  * Objects having a spatial extent.
  
  * Location: A fixed point [Centroid]
  
  * Boundary: A geometric approximation [Vector Data] constructed using Polygons
Overview

What are types of Spatial Data Queries?

* Point Queries
* Spatial-Range Queries
* Nearest-Neighbor Queries
* Spatial-Join Queries *expensive*
* Similarity Queries
Overview

Spatial Databases

* normal databases understand numeric | character types.
* need to make them understand geometry and features
* *SimpleFeatures* specification by OGC
  * Point | LineString | Polygon | Multipoint | Multipolygon
Problem with B+ Trees in Spatial Domain

- Approaches to Adapt to Spatial Domain -
  - Composite B+ Trees
  - Multiple B+ Trees
Problem with B+ Trees in Spatial Domain

Composite B+ Trees

- It linearizes 2D space
- [-] Spatial Proximity is lost.
  - Close in nature => Close on Disk
Problem with B+ Trees in Spatial Domain

Multiple B+ Trees

- Selects too much un-needed data
- Index space grows with increasing dimensions

```sql
select * from R where \( a_0 < A < a_1 \) and \( b_0 < B < b_1 \)
```

Several conventional indexes:
- read tuple with \( a_0 < A < a_1 \)
- read tuple with \( b_0 < B < b_1 \)
- intersect

wanted:
- read only tuples with \( a_0 < A < a_1 \) and \( b_0 < B < b_1 \)
Requirements from a Spatial Indexing Structure

- Should be scalable to n-Dimensions
- Should have symmetric behavior in all Dimensions
- Should support Insert | Delete gracefully
- Should support non-point data
Approaches

* GRIDs
* Quad Trees
* Z-Curves
* X-Trees
* GiST
* R-Trees *popular*
**Approaches**

**GRIDs**

- 2D surface divided into contiguous cells
- Each cell is assigned unique ID and used for indexing
- *Equal Angle GRIDs*
- *Equal Area GRIDs*
- Based on *Space Driven Indexing Method*
Approaches

Quad Trees

- Region-Based Quad Trees
- Point-Based Quad Trees
Approaches

Quad Trees: Oct Trees
Approaches

Quad Trees: kD Trees

- Always Binary
- Always split along a point
Approaches
Z-Curves

* A space-filling curve
* Good locality-preserving behavior
* Z-value is interleaved the binary representation of coord values
* UB-Tree: B+Tree with records stored in Z-Order
Approaches

X-Trees

* Improves upon the R* Tree
* Provides a *overlap free* split and *supernode* mechanism
Approaches

GiST

* Generalized Search Tree
* is both a data structure and an API to build upon.
* R-Trees and X-Trees are built upon using GiST.
* PostgreSQL has an off-the-shelf GiST implementation.
Approaches
R-Trees

* leaf: \(<n\text{-Dim box}, \text{RID}>\)
* non-leaf: \(<n\text{-Dim box}, \text{Ptr to Child}>\)
Current nod := root;
1. If current node is non-leaf:
   for each entry <E, ptr>:
      if box E overlaps Q
      then search subtree identified by ptr;
2. If current node is leaf:
   for each entry <E, rid>:
      if E overlaps Q
      then
         rid identifies an object that might overlap Q.
Approaches

R-Trees | Insert

Current node := root;
1. go down to “best-fit” leaf L
   find child whose box needs least enlargement to cover B; resolve
ties by going to smallest-area child
2. If best-fit leaf L has space,
   then insert entry and stop;
Otherwise,
   split L into L1 and L2
   Adjust entry for L in its parent so that box now covers (only) L1
   Add entry (in the parent node of L) for L2
   // this step could cause parent node to recursively split

* Node Splitting Heuristic: min [area(L1) + area(L2)]
Approaches

R-Trees | Delete

1. search entry to be deleted
2. remove it
3. if node becomes under-full
   then
   delete node
   re-insert remaining entries
Approaches

R-Trees | Properties

* The tree is balanced!
* Inexact Match is a good thing!
  * Match bounding boxes and exact match later.
* Good on Average-Case
* Poor in Worst-Case | Use Priority R-Tree
Approaches

R-Trees | Optimizations

* Store boxes as *Approximate Regions*
  * compact boxes
  * fast interval checking
* use *Convex Polygons*
  * reduces overlap
  * reduces complexity
Approaches

R-Trees | Variants

* R* Tree
  * minimize *box perimeter* than *box area*
  * forced re-inserts*

* R+ Tree
  * insert objects in multiple leaves | reduces overlap
  * search now takes place a single path to leaves.
SQL/MM

* defines how to store, retrieve and process spatial data using SQL
* defines functions to convert, compare, and process this data.
* only supports up to 2D data
* commercial implementations
  * IBM DB2 Spatial Extender
  * IBM Informix Dynamic Server
  * Oracle9 Spatial Product

- SQL/MM Part 1: Framework
- SQL/MM Part 2: Full Text
- SQL/MM Part 3: Spatial
- SQL/MM Part 5: Still image
- SQL/MM Part 6: Data mining
Figure 1: OpenGIS Geometry Class Hierarchy
Current Issues

* Sequential Scan > R-Tree when Dimension > 12
* Nearest Neighbor Queries are meaningful only at Low Contrast
  * need to empirically test the database