Spatial Database & Spatiotemporal Database

Chen Zhutian
Lin Dandan
Outline

Spatial Data Type (Data model)
Spatial Operation
Spatial Index
Spatiotemporal database
Q & A
What is and Why Spatial Database
Spatial Database

What?

- Spatial RDBMS is an RDBMS that can process spatial data.

Why?

- To process spatial data types efficiently.
Spatial Database

Spatial RDBMS
- Standard DBMS + Spatial Subsystem

Data
- RDBMS Data Type + Spatial Data Type

Operation
- SQL + Spatial Operators & Spatial Functions

Index
- B-Tree or B*Tree + Spatial Index
Spatial Data Model
Spatial Data Model

What’s data model?

- Specify structure or schema of a data set
- Document description of data
- Facilitates early analysis of some properties
<table>
<thead>
<tr>
<th>Example</th>
<th>City</th>
<th>River</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Point</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Curve</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Polygon</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Spatial Data Model

Each rectangle shows a distinct spatial object type
Spatial Relationships

Direction relationship  
*e.g. above, below, north of, etc*

Metric relationship  
*e.g. distance < 100*

Topological relationship

Egenhofer, M. Reasoning about Binary Topological Relations.
Spatial Operation
Spatial Operation

- Spatial Measurements
- Spatial Functions
- Spatial Predicates
- Geometry Constructors
- Observer Functions
Spatial Query

- **Spatial selection**
  - E.g. “Find all cities in Bavaria”
  - E.g. “Find all big cities no more than 100 kms from HK”

- **Spatial join**
  - E.g. “For each river, find all cities within less than 50 kms.”

- **Spatial function application**
  - E.g. “For each river going through HK, return the name, the part of its geometry lying inside Bavaria, and the length of that part.”

- **Other set operations.**
  - E.g. Overlay, Fusion, Voronoi, etc.
Query Processing

Minimum bounding rectangle
Query Processing
Two Steps:
1. Filter: Test MBR to acquire candidates
2. Refinement: Get the final results

Figure 1.8: The filter-refine strategy for reducing computation time
Spatial Index
Spatial Index

- Grid (spatial index)
- Z-order (curve)
- Quadtree
- Octree
- UB-tree
- R-tree
- R+ tree
- R* tree
- Hilbert R-tree
- X-tree
- kd-tree
- m-tree
- Point access method
- Binary space partitioning (BSP-Tree)
R tree

- A multi-way external memory tree
- Index nodes and data (leaf) nodes
- All leaf nodes appear on the same level
- Every node contains between m and M entries
- The root node has at least 2 entries (children)
R tree

(a) Data and MBRs
R tree
Search

(a) Data and MBRs

(b) The structure
Variants – R* tree

Weakness of R tree:
• Certain types of data may create small areas but large distances which will initiate a bad split.
• If one group reaches a maximum number of entries, the rest of assigned without consideration of their geometry.
Variants – R* tree

1. Area covered by a rectangle should be minimized
2. Overlap should be minimized
3. The sum of the lengths of the edges (margins) should be minimized
4. Storage utilization should be maximized (resulting in smaller tree height)

Figure 2: The right MBR is preferred
Spatio-Temporal Database
Content

- Introduction of Spatiotemporal Database
- Indices for the historical data retrieval
  - Multi-version B-trees (MVB tree) [Becker et al., VLDB, 1996]
  - Historical R-trees (HR tree) [Nascimento et al., ACM SAC 1998]
  - 3D R-tree
  - MV3R-tree [Tao et al., VLDB 2001]
Introduction

- **Spatiotemporal Database:**
  - Manages spatial object whose geometry change over time
  - Geometry: position and/or extent

(a) (b)

(a) a moving point

(b) a moving and shrinking region
Applications

- Global change data: climate or land cover changes
- Transportation: cars, airplanes
- Animated movies/video DBs
ST Databases

- Extension of existing Spatial Databases
  - Objects change instead of being static
  - At any timestamp it is a conventional Spatial Database

- Extension of existing Temporal Database
  - All the features of existing temporal database
  - Attributes can be spatial also

- New Database type
  - Seldom
  - Because of the need to be consistent with existing solutions
Requirements of Spatiotemporal Database

- Efficient Representation of \textit{Space and Time}
- Data Models
- Query Languages
- Query processing and Indexing
Data Types

- The atomic spatial data types: points, lines and regions
- The temporal data types: events and intervals

In Spatiotemporal Databases

- Moving Points
  - Extent does not matter
  - Each object is modeled as a point (moving vehicles in a GIS based transportation system)

- Moving Regions
  - Extent matters!
  - Each object is represented by an MBR, the MBR can change as the object move (airplanes, storm,...)
Data Types

- To cater for **abrupt changes** and **slow evolution**

- Different Types of changes:
  - Changes are applied **discretely**
    - Urban planning: appearance or dis-appearance of buildings
  - Changes are applied **continuously**
    - Moving objects (e.g. Vehicles)
Temporal Environment

- Two types of environments:
  - **Predicting the future positions:**
    - Each object has a velocity vector. The DB can predict the location at any time $t > t_{\text{now}}$ assuming linear movement.
    - Queries refer to the future
  - **Storing the history:**
    - Queries refer to the past states of the spatial database
The historical spatiotemporal database

Query region $Q$
Retrieval of historical information

- Two types:
  1. **Timestamp queries**: retrieve all objects that intersect a window at a specific timestamp $t$
  2. **Interval queries**: include several consecutive timestamps

- Various indices have been proposed to support those queries
  1. **Multi-version B-trees (MVB tree)** [Becker et al., VLDB, 1996]
  2. **Historical R-trees (HR tree)** [Nascimento et al., ACM SAC 1998]
  3. 3D R-tree
  4. **MV3R-tree** [Tao et al., VLDB 2001]
Multi-version B-trees [VLDB’96]

- Each index entry has the form \(<key, t_{\text{start}}, t_{\text{end}}, \text{pointer}>\)
- For the leaf entry, the pointer points to the actual record with the corresponding key value, while, for the intermediate entries, the pointer points to a next level node.
- \(t_{\text{start}} \text{ and } t_{\text{end}}\) denote the time that the record was inserted and deleted in the database respectively.
- \(t_{\text{end}} = \ast\) means that the entry is currently alive.

![Figure 1: Example of MVB tree](image)
Multi-version B-trees [VLDB’96]

- **Weak version condition**: for each timestamp $t$ and each node except the roots, it is required that either none, or at least $b \cdot P_{version}$ entries are alive at $t$, where $P_{version}$ is a tree parameter and $b$ is the node capacity.

- In the example, $b = 6$ and $P_{version} = 1/3 \rightarrow 2$ entries

- **Weak version underflows**: violates the weak version condition, which occur after deletion at the current time

Figure 1: Example of MVB tree
Multi-version B-trees [VLDB’96]

- How to insert and delete an entry?
  - Insertions and deletions are in a way similar to B-trees except handling the block overflows and weak version underflows.

- **Block overflows**: when an entry is inserted into a full node

- **Version split**: all the live entries of the node are copied to a new node, with their $t_{\text{start}}$ modified to the current time, the value of the $t_{\text{end}}$ of these entries in the original node is changed from * to the current time.
Multi-version B-trees [VLDB’96]

- For example, insert \(<28,4,*>\) at timestamp 4 in the following tree.

- Node A is full. We create a new node D to store all the live entries of A, and change the values of those new and old entries.
Multi-version B-trees [VLDB’96]

- In some cases, the new node may be almost full so that a small number of insertions would cause it to overflow again.
- On the other hand, if it contains too few entries, a small number of deletions will cause it to underflow.

- To avoid these problems, it is required that the number of entries in the new node must be in the range \([b \cdot P_{svu}, b \cdot P_{svo}]\) where \(P_{svu}\) is the parameter for the strong version underflow and \(P_{svo}\) for the strong version overflow.
- A strong version overflow is handled by a key split, a version-independent split according to the key values of the entries in the block.
- The strong version condition is only checked after a version split.
Multi-version B-trees [VLDB’96]

- How to insert and delete an entry?
  - Insertions and deletions are in a way similar to B-trees except handling the block overflows and weak version underflows.

- weak version underflows: violates the weak version condition after deleting an entry

- A merge is attempted with the copy of a sibling node using only its live entries. If the merged node strong version overflows, a key split is performed.
Multi-version B-trees [VLDB’96]

- For example, delete entry <48,1,*> at timestamp 4 from the following tree.

```
Root
<5, 1, *, A>
<43, 1, *, B>
<72, 1, *, C>
```

```
A
<5, 1, *>
<8, 1, *>
<13, 1, *>
<25, 1, 3>
<27, 1, 3>
<39, 1, 3>
```

```
B
<43, 1, *>
<48, 1, *>
<52, 1, 2>
<59, 1, 3>
<68, 1, 3>
```

```
C
<72, 1, *>
<78, 1, *>
<83, 1, *>
<95, 1, 3>
<99, 1, *>
<102, 1, *>
```

- Suppose that $P_{svu} = 1/3$ and $P_{svo} = 5/6$, $b = 6$
- Node B contains only one live entry <43,1,*>. \( \rightarrow \) weak version underflow.
- Suppose we chose C, one sibling of B, and its live entries are copied to a new node C’. The insertion of <48,1,*> causes C’ strong version overflow.
- So we need to do a key split and finally nodes D and E are created.
Multi-version B-trees [VLDB’96]
Historical R-trees (HR tree)
[Nascimento et al., ACM SAC 1998]

- Based on overlapping technique, another framework for transforming a single version data structure into a transaction time access method.
- **An R-tree is maintained for each timestamp in history.** But common branches of consecutive trees are stored only once in order to save space.
- For example, at timestamp 1, we have the following spatial objects
Historical R-trees (HR tree)
[Nascimento et al., ACM SAC 1998]

- But at timestamp 2, the position and extent of object 5 is changed.
Historical R-trees (HR tree)
[Nascimento et al., ACM SAC 1998]

- **Advantages:**
  1. A timestamp query is directed to the corresponding R-tree, search is performed inside this tree only. -> efficiently

- **Disadvantages:**
  1. An interval query should search the corresponding trees of all the timestamps involved. → not efficiently
  2. Expensive space consumption. Extensive duplication of objects even if they do not move
3D R-tree

- To view time as just another dimension and integrate it in the tree construction along with the other dimensions.
- The movements of 2D objects can be modelled as distinct boxes in 3D space.
3D R-tree

- Do not include a mechanism to ensure that each node has a minimum number of live entries at a given timestamp.
- There is a single tree for the whole history, the cost depends on the total number of records, rather than on the number of records alive at the queried timestamps, as in MVB- and HR- trees.
- Those affect performance of timestamp and short-interval queries.
MV3R-tree [Tao et al., VLDB 2001]

- Multi-version 3D R-trees
- Efficiently handle both timestamp and interval queries.
- Combines two structures: a multi-version R-tree (MVR-tree) and a small auxiliary 3D R-tree built on the leaf nodes of the MVR-tree.

Figure 2: Overview of MV3R-tree
**MV3R-tree** [Tao et al., VLDB 2001]

- An MVR-tree contains multiple R-trees (logical trees)
- Each entry has the form `<S, t_{start}, t_{end}, pointer>` where S denotes the spatial minimum bounding rectangle (MBR) as defined in R-trees.
- The **weak version condition**: same as MVB-trees; to guarantee that the number of live entries during a timestamp are either 0 or at least \( b \cdot P_{version} \)
MV3R-tree [Tao et al., VLDB 2001]

- **Insertion and overflow handling**
- Involves distinct insertion policies for leaf and intermediate nodes.
  - For leaf nodes, try to avert version splits which cause redundancy.
  - For intermediate nodes, more redundancy is permitted in order to maintain good performance for timestamp and short-interval queries.
**MV3R-tree** [Tao et al., VLDB 2001]

- Insertion for intermediate nodes.

![Diagram](image)

**Figure 3.3:** Insertion in intermediate nodes

1. Find the node $A$ where the new entry is inserted.
2. If the node $A$ is **full**, a **version split** occurs and all the live entries are copied to a new node $A'$ with their $t_{\text{start}}$ modified to the insertion time, the new entry is inserted in $A'$.
3. If the total number of live entries in $A'$ is greater than $b \cdot P_{svo}$, a **strong version overflow** occurs, triggering a **key split**

- It does not consider the strong version underflows
MV3R-tree [Tao et al., VLDB 2001]

- Insertion for intermediate nodes.

**Figure 3.5: Insertion in leaf nodes**

- In order to avoid version splits, it tries the above alternatives in order
MV3R-tree [Tao et al., VLDB 2001]

- **General key split**
  - Motivated by the fact that in some cases a node to be version split can instead be *key split* without violating the weak version condition
- **Key split** does not generate redundancy

- For example, \( b = 10, P_{\text{version}} = \frac{1}{3} \)
- Insert a new entry \(<S11,3,*>\)
- The entries can be distributed to two nodes so that for each timestamp in the range \([1,*]\) there exist at least \( b \times P_{\text{version}} \) entries alive
- The two new node may have small overlap.
MV3R-tree [Tao et al., VLDB 2001]

- **insert In node after reinserting one of its entries**
- Try to reinsert an existing entry of the node, in order to make room for the new entry.
- Any leaf node can store a re-inserted entry:
  1. Its life span must cover that of the entry
  2. It should be dead if the entry is dead
  3. Its area should not be enlarged much, in order to ensure good performance for timestamp queries.

- **insert new entry into another node**
- Back track to the upper level and try to insert the entry into another branch.
- Only consider branches that will incur small area enlargements
MV3R-tree [Tao et al., VLDB 2001]

- Construction of the auxiliary 3D R-tree
- The number of leaf nodes in an MVR-tree is much lower than the actual number of objects → fairly small compared to a complete 3D R-tree.
- Whenever a leaf node of the MVR-tree is updated, the change is propagated to its entry in the 3D R-tree.
Thanks for Watching

Q & A