Pemanfaatan *Tree* untuk Indexing Basis Data Spasial

Rudolf Rudi Hermanto 13506114
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
rudolf@students.itb.ac.id

***Abstract*—Basis data spasial mendeskripsikan sekumpulan entitas baik yang memiliki lokasi atau posisi yang tetap maupun yang tidak tepat (memiliki kecenderungan untuk berubah, bergerak, atau berkembang). Terdapat indexing dengan memanfaatkan struktur data *tree* pada basis data spasial.**

***Index Terms*—Direct Access, indexing, spatial database, tree**

I. PENDAHULUAN

Dalam kehidupan nyata, ada banyak bidang yang memerlukan penanganan dalam hal geometric, geografis, dan data spasial, yang berarti data-data tersebut akan berhubungan dengan ruang. Ruang yang dimaksud bisa berbentuk dua dimensi ataupun tiga dimensi. Ketika basis data relasional dirasakan sudah tidak lagi bisa memenuhi kebutuhan tersebut, maka basis data alternative untuk menyelesaikan hal tersebut pun mulai dikembangkan, salah satunya adalah basis data spasial.

Definisi basis data spasial sendiri adalah basis data yang mensupport tipe data spasial, seperti point, garis, dan polygon, dalam hal penyimpanan datanya. Basis data spasial juga menyediakan indexing spasial yang diperlukan ketika sistem perlu mengeluarkan data yang diinginkan dari koleksi data yang banyak pada suatu area spesifik tanpa harus memeriksa seluruh data yang ada.

Proses indexing sendiri merupakan pengelompokan data yang berguna untuk mempercepat proses pencarian data. Misalnya, ada sekelompok data dengan huruf pertama yang berbeda-beda dikelompokkan berdasrakan huruf pertamanya. Ketika mesin hendak mengakses data "Rumah", misalnya, mesin tidak perlu menelusuri semua data, cukup kelompok data berawalan "R", sehingga pengaksesan menjadi lebih cepat. Ini disebut *direct access*.

Indexing pada basis data spasial sedikit berbeda karena data yang harus dibuat index adalah tipe data spasial. Berikut akan dijelaskan beberapa tipe indexing pada basis data spasial.

.

II. BASIS DATA SPASIAL

Basisdata Spasial mendeskripsikan sekumpulan entitas baik yang memiliki lokasi atau posisi yang tetap maupun yang tidak tepat (memiliki kecenderungan untuk berubah, bergerak, atau berkembang). Tipe-tipe spasial ini memiliki properties topografi dasar yang memiliki lokasi, dimensi, dan bentuk (shape).

Contoh database spasial meliputi kondisi tekstur tanah, erosi, lereng, ketinggian, jenis tanah, tempat pengambilan sumber bahan bangunan dan penyebaran pemukiman yang dikonstruksikan sebagai ulasan dalam suatu vector. Dimana atribut-atributnya disimpan sebagai database relasional yang bisa diimpor ke model tata ruang.

Karakteristik Pembuatan Database Spasial

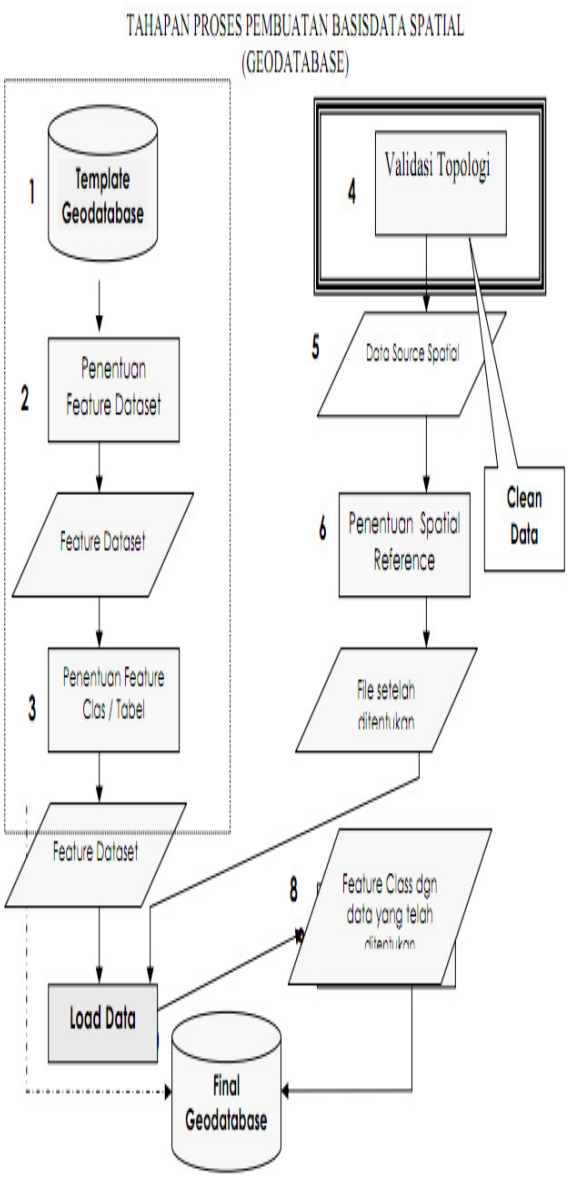
-Data dibuat dalam beberapa tipe yaitu: polygon (area), line (garis) dan point (titik).

-Masing-masing obyek yang dibuat memiliki identifier (ID) / pengenal yang unik (tidak dimiliki oleh obyek lain selain obyek yang sama dengan dirinya sendiri)

Dibandingkan Standard Database, Spasial Database memiliki perbedaan:

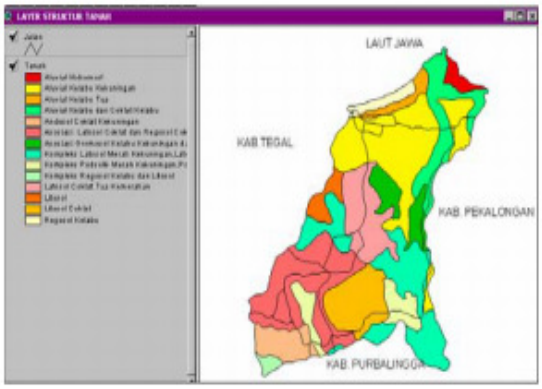
Standard Database	Spasial Database
Memiliki *data types* berupa: varchar, integer, real, date	Memiliki *spasial data types*: point, linestring, polygon, multipoint, multilinestring, multipolygon
Memiliki *one-dimensional indexes*: btree dan hash	Memiliki *spatial indexes*: r-tree, quad tree, grid
Memiliki *function* yang bekerja mengikuti tipe standar. Contohnya: dayofweek(), lower()	Memiliki *spatial functions* yang bekerja mengikuti tipe spasial. Contohnya: ST_Area(geometry), ST_Distance(geometry, geometry), ST_Intersects(geometry, geometry), ST_DWithin(geometry,geometr,radius), ST_Union(geometry,geometry)

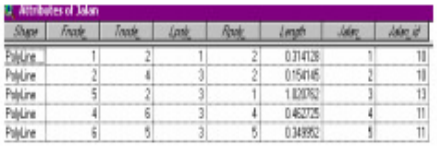
Tahapan proses pembuatan basisdata spasial



Tahapan proses pembuatan basisdata spatial

Data Spasial Sruktur Tanah

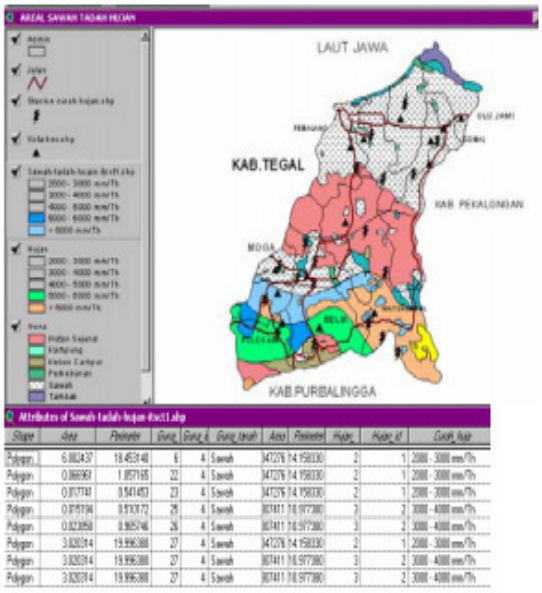




Shape	Fnode_	Tnode_	Lpoly_	Rpoly_	Length	Jalan_	Jalan_id
PolyLine	1	2	1	2	0.314128	1	18
PolyLine	2	4	3	2	0.154145	2	18
PolyLine	5	2	3	1	1.820762	3	13
PolyLine	4	6	3	4	0.462725	4	11
PolyLine	6	5	3	5	0.348952	5	11

Data spatial struktur tanah

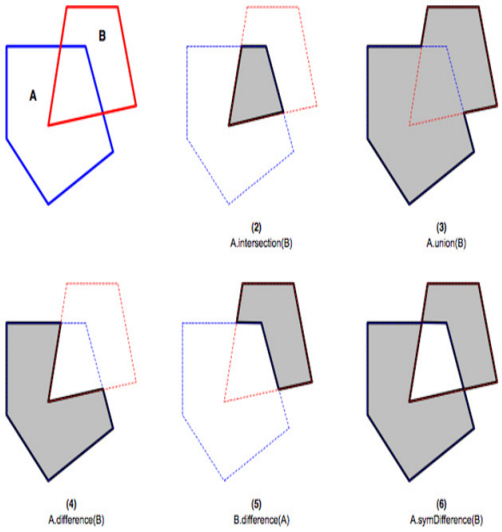
Data Spasial Areal Sawah Tadah Hujan



Shape	Area	Perimeter	Sawah_	Sawah_id	Sawah_kode	Area1	Perimeter1	Hujan_	Hujan_id	Curah_hujan
Polygon	6.882437	19.455148	6	4	Sawah	347276.74	958030	2	1	2000 - 3000 mm/Th
Polygon	0.866963	1.857165	22	4	Sawah	347276.74	958030	2	1	2000 - 3000 mm/Th
Polygon	0.877743	0.941453	23	4	Sawah	347276.74	958030	2	1	2000 - 3000 mm/Th
Polygon	0.875194	0.930572	25	4	Sawah	307411.78	977380	3	2	3000 - 4000 mm/Th
Polygon	0.823058	0.905746	26	4	Sawah	307411.78	977380	3	2	3000 - 4000 mm/Th
Polygon	3.820314	19.996388	27	4	Sawah	347276.74	958030	2	1	2000 - 3000 mm/Th
Polygon	3.820314	19.996388	27	4	Sawah	307411.78	977380	3	2	3000 - 4000 mm/Th
Polygon	3.820314	19.996388	27	4	Sawah	307411.78	977380	3	2	3000 - 4000 mm/Th

Data spatial areal sawah tadah hujan

Contoh-contoh Data Spasial



III. SPASIAL INDEX PADA BASIS DATA SPASIAL

Struktur untuk m*ulti attribute retrieval* berupa Tuple t = (x1,…,xk) point dalam dimensi k adalah grid file, Kd tree, KDB tree dan R Tree.

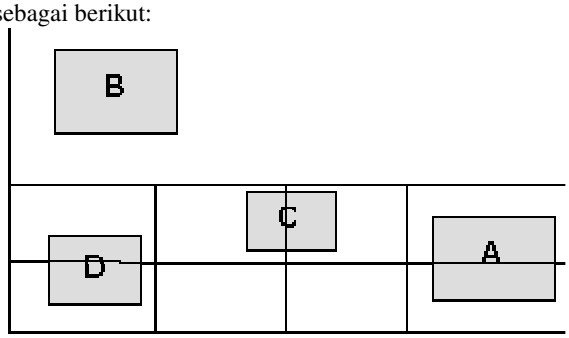
Tree atau pohon adalah graf tak berarah terhubung yang tidak mengandung sirkuit.Pohon merupakan struktur data yang penting dalam ilmu computer.

Jenis-jenis spasial index yang sering digunakan, antara lain :

K-D-Tree

K-d-tree merupakan singkatan dari k-dimensional tree. Disebut k-dimensional tree karena k-d-tree adalah binary tree yang node-nya merupakan k-dimensional point. K dapat bernilai 2 atau lebih.

Cara k-d-tree mengelompokkan data adalah sama dengan binary tree biasa. Suatu region dibagi menjadi dua, kemudian masing-masing dua region tadi dibagi lagi menjadi dua region, demikkian seterusnya hingga tidak dapat dibagi lagi. Gambaran proses partisi ini adalah sebagai berikut:



Gambar 1. K-d-tree partitioning

Pertama-tama, region dibagi menjadi dua: top dan bottom. Karena region top hanya memiliki satu data, maka tidak dibagi lagi. Kemudian region bottom dibagi dua secara vertikal. Karena masih dapat dibagi lagi, maka region ***bottom-left*** dan ***bottom-right*** dibagi lagi menjadi dua region yang lebih kecil. Setelah itu, region tidak dapat dibagi lagi, sehingga proses partisi dihentikan.

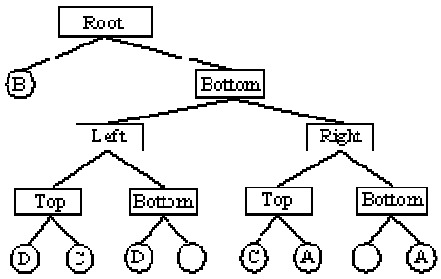
Notasi algoritmik untuk proses partisi k-d-tree adalah:

```
        function kdtree (list of points pointList, int depth)
        {
            if pointList is empty
                return nil;
            else
            {
                // Select axis based on depth so that axis cycles through all valid values
                var int axis := depth mod k;

                // Sort point list and choose median as pivot element
                select median by axis from pointList;

                // Create node and construct subtrees
                var tree_node node;
                node.location := median;
                node.leftChild := kdtree(points in pointList before median, depth+1);
                node.rightChild := kdtree(points in pointList after median, depth+1);
                return node;
            }
        }
```

Pembagian region menghasilkan tree seperti berikut ini:



Gambar 2. K-d-tree

Proses partisi menggunakan k-d-tree dapat menghasilkan balanced tree. Namun pada kebanyakan kasus, balanced tidak begitu dibutuhkan. Meskipun k-d-tree dapat membuat balanced tree dan mempercepat pengaksesan data, namun proses insert dan delete data tidak begitu efisien.

Seperti yang terlihat pada gambar 2, untuk menginsert, mesin harus mencari region yang memiliki ruang kosong. Ruang tersebut tersebar di berbagai tempat, sehingga mesin harus menelusuri banyak node agar menemukan ruang kosong yang cukup untuk menyimpan data yang di-insert. Setelah di-insert, proses partitioning berjalan kembali. Ini berarti, tiap ada proses insert, struktur tree berubah. Ini tentu memakan memori yang cukup besar.

Untuk proses delete, pencarian dilakukan hingga menemukan semua region yang mengandung data yang dicari, kemudian menghapus semua data di setiap region. Misalnya jika kita ingin menghapus A, maka mesin harus menghapus data A pada region ***bottom-right-top*** dan region ***bottom-right-bottom***.

Proses konstruksi k-d-tree memiliki kompleksitas algoritma O($n \log^2 n$) jika proses penentuan median menggunakan algoritma sorting dengan kompleksitas O($n \log n$). Jika pemenuntuan median menggunakan metode linier, maka kompleksitasnya adalah O($n \log n$).

K-D-B Tree

K-D-B Tree merupakan sebuah struktur data baru yang diperkenalkan untuk menangani persoalan pengambilan record multikey melalui indeks yang besar dan dinamis. K-D-B Tree mengkombinasikan K-D tree dengan B-tree. Dengan adanya K-D-B Tree diharapkan bisa diperoleh efisiensi pencarian multidimensional dari K-D Tree dan efisiensi I/O dari B-Tree.

Ruang pencarian dalam K-D-B Tree (subset dari $domain_0$ x $domain_1$ x $domain_{k-1}$) sama dengan metode yang ada didalam K-D Tree. Sebuah ruang pencarian dibagi menjadi dua subruang sesuai dengan perbandingan elemen dari domain tunggal. Untuk pemilihan domain dan elemen dalam domain bisa digunakan berbagai strategi sesuai dengan K-D Tree.

K-D-B Tree seperti B-Tree, pohon dengan banyak cabang dengan jumlah node yang pasti yang selalu sama jumlah cabang ke node daun dari node akar. Jumlah cabang yang harus diakses dari node akar ke node daun akan selalu sama untuk setiap node daun. Tiap node akan disimpan didalam sebuah page. K-D-B Tree terdiri dari sekumpulan *page* dan sebuah variabel root ID yang memberikan ID untuk *root page* pada masing-masing *page*. Terdapat dua jenis page dalam K-D-B Tree,

1. *Region pages*, terdiri dari kumpulan pasangan (*region, page ID*)
2. *Point pages*, terdiri dari kumpulan pasangan (*point, location*), dimana location menunjukkan lokasi dari sebuah record database. Pasangan ini merupakan index dari record.

Sebuah range query bisa diekspresikan dengan menetapkan region, *query region*. Algoritma untuk mendapatkan semua record yang sesuai dengan range query yang ditetapkan oleh query region adalah sebagai berikut,

1. Jika root ID kosong, selesai. Jika tidak, page menjadi root page
2. Jika page merupakan point page, tiap pasangan (point, location) dalam page dimana point anggota dari query region, ambil dan kelarkan dari lokasi record database.
3. Jika page merupakan region page, untuk setiap pasangan (region, child ID) dalam page, jika merupakan perpotongan dari region dan query region tidak kosong, set child ID menjadi page yang dipakai untuk proses kedua.

Ketika ingin memasukkan sebuah record dengan point a dan di location l dalam sebuah pohon dengan root r, pertama dicek jika r NIL, kemudian dibuat sebuah point page p dan dimasukkan record dengan pasangan <a,l> ke dalam p dan mengembalikan p. Jika r tidak NIL, cari a di dalam pohon dengan root t sampai pada sebuah point page, misalkan p. Masukkan record <a,l> ke dalam point page p. Jika ternyata point page p yang akan dimasukkan record tadi sudah terlalu banyak, kemudian cari titik tengah dari point page p dan p dibagi menjadi dua bagian p_{left} dan p_{right}. Jika p merupakan bukan merupakan root, maka root diubah menjadi p dan p_{left} dan masukkan p_{right} ke dalam parent dari p. Jika p merupakan root, buat sebuah root node dengan 2 child, p_{left} dan p_{right}.

Struktur K-D-B Tree tidak memperhatikan mengenai pemanfaatan storage dan point pages yang kosong. Hal tersebut membuat algoritma dasar dari penghapusan sangat sederhana : cari indeks record (point, location) yang sesuai dengan query, hapus (point, location) dari point page.

R-Tree

R-tree merupakan struktur data pohon dengan record index untuk setiap node daunnya yang berisi pointer menunjuk ke objek data. Index ini bersifat dinamik sehingga proses insert dan delete dapat dilakukan bersamaan dengan search dan tidak ada reorganisasi periodic yang perlu dilakukan.

Format record index yang disimpan pada setiap daun adalah

(*I*, tuple-identifier)

Di mana tuple-identifier merujuk kepada sebuah tuple di dalam basis data dan I adalah polygon n-dimensional yang membatasi index objek spasial

$$I = (I_0, I_1, \ldots, I_{n-1})$$

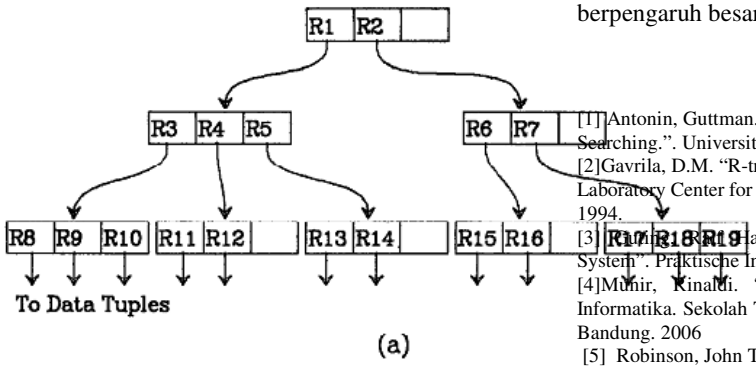
Dengan n adalah jumlah dimensi dan I adalah batasan tertutup untuk interval [a,b] yang mendeskripsikan luas dari sebuah objek bersama dengan dimensinya. Sebagai alternative, I bisa memiliki satu atau kedua endpoint, yang mengindikasikan luas objek. Nodal yang bukan merupakan daun akan memiliki format index

(*I*, child-pointer)

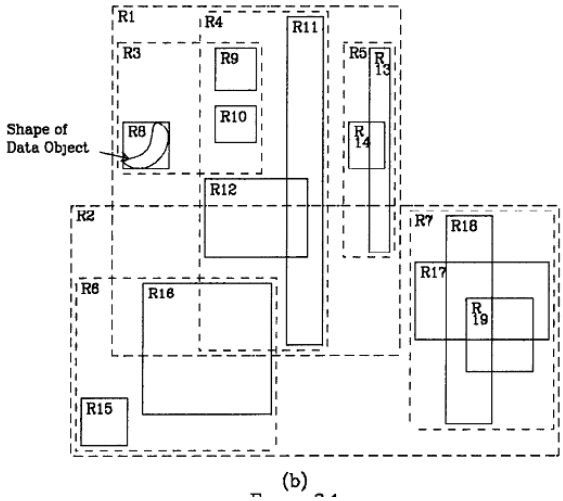
Di mana *child-pointer* adalah alamat dari nodal anak pada R tree tersebut dan I mengcover semua polygon yang ada di nodal-nodal anaknya.

Algoritma pencari akan menyusuri pohon dari akarnya, meskipun demikian ada lebih dari satu subpohon dibawah sebuah nodal yang perlu diperiksa sehingga tidaklah mungkin untuk mengaransi performansi terburuknya. Akan tetapi dengan sebagian besar jenis data, algoritma update akan mengurusi pohon dalam suatu format yang menyebabkan algoritma pencari mengeliminasi area yang tidak relevan pada index dan hanya memeriksa data yang dekat dengan area pencarian saja.

Struktur data R-tree ini berguna dalam pencarian string di database spasial.



Gambar 3. R-tree



Gambar 4. Struktur data objek R-tree

Memasukkan record index untuk tuple data baru adalah dengan memasukkannya ke dalam daun, kemudian nodal yang tertumpuk akan dipisahkan dan pemisahan tersebut akan meluaskan bentuk pohon.

Jika tuple diupdate maka persegi panjang yang mengcovernya akan berubah, sehingga record indexnya harus didelete, diupdate, kemudian dimasukkan kembali sehingga dapat menemukan jalan yang benar di dalam pohonnya.

.

IV. KESIMPULAN

Kesimpulannya, struktur data pohon/*tree* dimanfaatkan dalam indexing basis data spasial. K-d-tree memiliki kelebihan dalam hal pengindeksan dan pengaksesan data namun lemah dalam hal pembentukan tree karena kompleksitas algoritma yang tergolong tinggi. K-D-B Tree sendiri memiliki spesialisasi dalam hal menangani persoalan pengambilan record multikey melalui indeks yang besar dan dinamis. Sedangkan R-tree sangat cocok digunakan dalam pengindexan objek data spasial yang memiliki ukuran besar di mana keseluruhan pohon tidak dapat disimpan didalam *main memory* karena dapat mengurangi biaya I/O hingga hampir bisa diabaikan. Algoritma yang menggunakan node-split juga bagus karena sangat cepat dan kualitas split tidak akan berpengaruh besar terhadap performansi pencarian.

REFERENCES

[1] Antonin, Guttman. "R-trees : A Dynamic Index Structure For Spatial Searching.". University of California.

[2]Gavrila, D.M. "R-tree Index Optimazation." Computer Vision Laboratory Center for Automation Research University of Marylan. 1994.

[3] Güting, Ralf Hartmut. "An Introduction to Spatial Database System". Praktische Informatik IV, Fern Universitat Hagen.1994.

[4]Munir, Rinaldi. "Matematika Diskrit." Program Studi Teknik Informatika. Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung. 2006

[5] Robinson, John T. "The K-D-B-Tree: A Search Structure for Large Multidimensibnal Dynamic Indexes". Department of Computer Science, Carnegie-Mellon University.

.[6]Yao, Bin. "Approximate String Search in Spatial Databases". Computer Science Department, Florida State University Tallahassee, F.L, USA. 2000.

[7] http://www.flegg.net/brett/pubs/spatial/index.html