SỬ DỤNG LƯỚI PHÂN CẤP ĐỂ TẠO INDEX KHÔNG GIAN ĐỘC LẬP CHO LƯU TRỮ VÀ KHAI THÁC DỮ LIỆU ĐỒ HỌA TRONG DBMS

Ngô Quốc Việt [1], Đinh Tiến Sơn [2]
[1] Phân Viện Vật lý Tp. HCM, [2] Công ty DolSoft
(Bài nhận ngày 22 tháng 10 năm 2003)

TÓM TẮT: chúng tôi đưa ra một phương pháp lưu trữ và khai thác dữ liệu không gian trong DBMS với việc tạo index không gian cho các đối tượng trong bảng dữ liệu sao cho chúng hoàn toàn độc lập lẫn nhau. Mục tiêu của phương pháp phân chia không gian với index đối tượng độc lập lẫn nhau trên môi trường đa người dùng sao cho vẫn bảo đảm các vấn đề khai thác. Tính chất chủ đạo trong giải pháp của chúng tôi là việc thêm vào hay xóa đi một đối tượng không ảnh hưởng đến index không gian của các đối tượng khác. Ngoài ra việc phân chia kích thước của lưới của mỗi cấp linh hoạt tùy theo sự phân bố không gian của đối tượng, chứ không cần phải định nghĩa trước theo kinh nghiệm của chuyên gia tạo dữ liệu.

1. Giới thiệu

Việc lưu trữ và khai thác dữ liệu không gian, đặc biệt dữ liệu GIS được quan tâm khá nhiều trong các ứng dụng GIS vì số lượng cũng như tính chất khai thác của lĩnh vực này. Trong thực tế đã có nhiều giải pháp cho vấn đề này, phần lớn các giải pháp đề sử dụng cây phân chia không gian (như Quad-tree hay K-tree) để tạo index cho dữ liệu không gian. Bài viết này chúng tôi đưa ra giải pháp lưu trữ và khai thác dữ liệu GIS trong DBMS vì tính chất phân bố địa lý của các đối tượng trong lĩnh vực này ít có trường hợp overlap lẫn nhau quá nhiều. Phương pháp chỉ dựa vào triết lý cây phân chia không gian để tạo lưới phân lớp, chứ không hòan tòan dựa vào một loại cây (Quad-tree hay K-tree) cụ thể. Chúng tôi đưa ra các giải pháp tạo index không gian độc lập cho mỗi đối tượng.

2. Tạo index không gian thông qua lưới phân cấp

Việc tạo index không gian cho đối tượng dựa vào **minimum bouding rectangle** của mỗi đối tượng. Mỗi một bảng dữ liệu thể hiện một layer bản đồ. Mỗi một đối tượng indexing được thể hiện bằng một mẫu tin trong bảng tương ứng.

Phương pháp Spatial Indexing

Cho một đối tượng bất kỳ, gọi là Obj. Cho Mbr(Obj) = (x_{min}, y_{min}, x_{max}, y_{max}), là hình chữ nhật bao quanh nhỏ nhất của đối tượng.

Đặt $r = \max(x_{max}-x_{min}, y_{max}-x_{min})$, là bán kính của đối tượng.

$xc = (x_{min} + x_{max})/2$.

$yc = (y_{min} + y_{max})/2$. (xc, yc) là tâm của đối tượng.

Spatial Indexing của đối tượng Obj dựa vào bộ ba tham số (r, xc, yc). Tùy theo giá trị của r mà chúng ta sẽ sắp xếp đối tượng Obj vào grid ở cấp tương ứng. Tuy vậy vì cả ba giá trị đều là số double, nên việc tạo index không gian để lưu trữ và khai thác sẽ gặp khó khăn.

Cho giá trị *r* bất kỳ, định nghĩa G_{lr} là grid vô hạn cấp *lr* sao cho kích thước mỗi cell có giá trị

$$lr=log_2 r \qquad (1)$$

Ta thấy rằng, giá trị $log_2 r$ có thể biểu diễn bằng thành phần exponent (bit 52-62) của số double theo chuẩn IEEE.

Như vậy với mỗi đối tượng có bán kính r cho trước ta sẽ tìm được grid với kích thước cell là $log_2 r$ để sắp đối tượng này vào. Vấn đề đến bước này vẫn chưa rõ vì sự sắp xếp các cell của grid chưa rõ ràng. Ngòai ra việc đối tượng được đặt vào cell nào cũng chưa rõ.

Thay vì dùng giá trị double, ta đặt

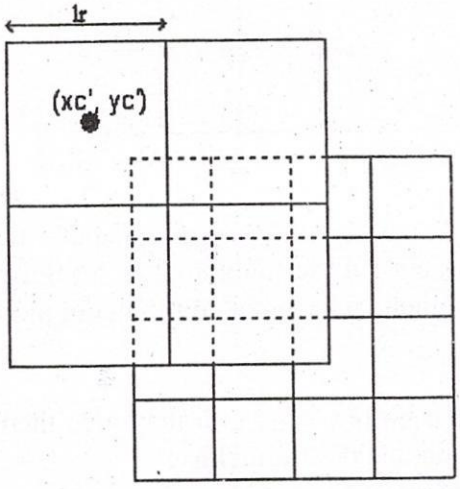
$xc' = [xc/2^{lr}]$. là giá trị nguyên của phân số này. (2)

$yc' = [yc/2^{lr}]$

Giá trị 2^{lr} thể hiện bán kính r của đối tượng như định nghĩa ở trên.

Bây giờ, ta chọn (xc', yc') là tâm của đối tượng Obj đang xét.

Như vậy, ta đã xây dựng được grid với các tham số (kích thước cell, và tâm mỗi cell) là giá trị nguyên. Về cơ bản với Obj đã cho ở trên ta dễ dàng sắp xếp nó vào cell có tâm (xc', yc') của grid G_{lr} theo hình 1 dưới đây.



Hình 1 - Phân bố cell ở cấp *lr*. Phân bố giữa các cấp không hòan tòan theo Quad-Tree.

Ta có thể trộn 03 (lr, xc', yc') giá trị thành một chuỗi trong field DBMS. Đây chính là index không gian của đối tượng hình học. Trong mỗi cấp grid, giá trị *lr* (cấp của grid) giữ nguyên, vì vậy luôn được đặt ở phần đầu của chuỗi index.

Trong thực tế, giá trị xc' và yc' được cộng thêm một gia số để không bao giờ mang giá trị âm, do đó chuỗi index không gian sẽ không xuất hiện dấu trừ. Đọan mã sau mô tả việc tạo chuỗi Spatial Index. Chúng tôi dùng chuỗi spatial index dài 32 bytes.

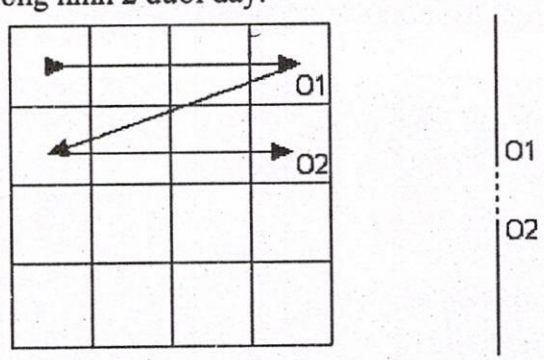
String strSpatialIndex

strSpatialIndex.Format("%6d_%12d_%12d", lr, xc', yc')

Chúng tôi dùng 6 ký tự biểu diễn cấp grid, 12 ký tự cho mỗi thành phần tọa độ tâm.

Phương pháp tạo chuỗi index tuần tự theo hàng

Phương pháp này đơn giản là ghép giá trị xc' theo sau là yc' trong chuỗi spatial index . Minh họa việc kết hợp tâm như trong hình 2 dưới đây.

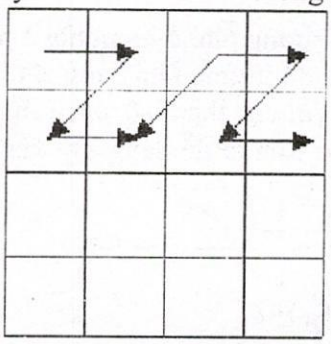


Hình 2 - Phương pháp tạo index không gian tuần tự theo hàng. Đối tượng O1gần O2 trong không gian, nhưng không gần nhau trong bảng dữ liệu.

Lợi điểm của cách tiếp cận này là dễ dàng xây dựng spatial index. Tuy nhiên nhược điểm có thể phát sinh là đôi khi đối tượng trong không gian không được phân bố hợp lý trong bảng dữ liệu, như hình 2 ở trên. Với cách sắp xếp này, có thể cần phải có nhiều query để lấy tập đối tượng cắt hình chữ nhật cho trước, dẫn đến việc truy xuất dữ liệu chậm hơn.

Phương pháp đan chéo

Phương pháp này trộn giá trị xc' và yc' chéo nhau như trong hình 3 dưới đây.



Hình 3 - Phương pháp đan chéo.

Lợi điểm của cách tiếp cận này là tạo điều kiện sao cho nếu đối tượng gần nhau trong thực tế thì dẫn đến spatial index của chúng cũng được lưu gần nhau trong bản dữ liệu. Điều này thuận lợi cho các query không gian. Tuy nhiên việc cài đặt có tương đối phức tạp hơn so với phương pháp tuần tự ở trên.

3. Phương pháp khai thác

Với phương pháp spatial index trình bày ở trên, ta thấy việc thêm, chỉnh sửa hay xóa một đối tượng hoàn toàn không ảnh hưởng đến đối tượng khác.

Phương pháp query không gian

Chúng ta xét query cơ bản nhất là lấy các đối tượng (để hiển thị, phân tích) nằm trong hay cắt một khu vực cho trước (x_1, y_1, x_2, y_2). Chúng ta dùng phương pháp tạo index không gian tuần tự để minh họa cho phép tóan này.

Trước khi trả lời cho query trên, chúng tôi cung cấp các đọan mã để tạo spatial index như sau:

(I)

Đọan mã tính lr = $\log_2$r.

```
GetExponentFromDouble(double r)
//Lấy thành phần exponent của giá trị double
short  wd = 0
long l

Double cD = new Double(db r)

l = cD.doubleToLongBits(db r)
return wd = (short)((l & 0x7ff0000000000000L) >> 52)
```

(II)

Đọan mã tính 2^{lr}. (là bán kính của đối tượng)

```
GetDoubleFromExponent(short value)

long  l = value

l = (l << 52) & (0x7ff0000000000000L)
Double db = new Double(l)
double d = db.longBitsToDouble(l)

return d;
```

(III)

Đọan mã tính xc' hoặc yc'

```
GetCenterFromRadius (double r,  double xc, double yc)
int gridlevel = GetExponentFromDouble (r);
double gridradius = GetDoubleFromExponent (gridlevel)

xc' = (long)(xc/gridradius)          //có thể cộng thêm giá trị để cho không âm
yc' = (long)(yc/gridradius)          //có thể cộng thêm giá trị để cho không âm
```

(IV)

Đoạn mã tạo không gian index của đối tượng có (xc, yc và r).

```
SpatialIndex(double r, double xc, double yc)
CString String;

int gridlevel = GetExponentFromDouble(r)
double gridradius = GetDoubleFromExponent(gridlevel)
long xc_1 = (long)(xc/gridradius)
long yc_1 = (long)(yc/gridradius)

//Cộng thêm giá trị cho xc_1 và yc_1 không mang giá trị âm.

ulong xc_2 = xc_1+bigint
ulong xc_2 =yc_1+bigint

String.Format("%6d_%12d_%12d", gridlevel, xc_2, yc_2)

return String
```

Chúng tôi dùng mã giả để minh họa thực hiện query cơ bản trên. Giả sử index không gian được tạo theo phương pháp tuần tự. Giả sử ta đang cần vẽ lớp bản đồ ở tỉ lệ nhất định, và không vẽ các đối tượng quá nhỏ ở tỉ lệ này trên màn hình – nghĩa là kích thước pixel (nhân kích thước đối tượng với tỉ lệ vẽ nhỏ hơn 2 pixel). Để hiển thị các đối tượng có 'nằm trong màn hình', trước tiên tao cần query trên bảng dữ liệu để lấy các đối tượng thích hợp.

(V)

scale: là tỉ lệ giữa màn hình và kích thước lớp dữ liệu không gian.

maxgridlevel là 'bán kính lớn nhất' của grid có cấp cao nhất (grid chứa đối tượng lớn nhất trong lớp dữ liệu không gian. Giá trị này luôn biết được cho mỗi bảng. Chúng ta dễ dàng lấy được giá trị này khi xét đến 06 ký tự đầu tiên của trường index không gian của mọi record.

minsize Không lấy các đối tượng có *kích thước hiển thị* nhỏ hơn *minsize* pixel

```
minsize= GetExponentFromDouble (2/scale)+3
for(int i = minsize; i <= maxgridlevel; i++)
{
    //xác định kích thước của hình chữ nhật query ở mỗi cấp

    left = x1 - GetDoubleFromExonent(i-3)
    top = y1 - GetDoubleFromExonent (i-3)
    right = x2 + GetDoubleFromExonent(i-3)
    bottom = y2 + GetDoubleFromExonent (i-3)
```

```
        //xác định số hàng và cột của lưới đang xét, cùng cell bắt đầu duyệt
        cellx1 = GetDoubleFromExponent(i)/left
        celly1 = GetDoubleFromExponent(i)/top
        cellx2 = GetDoubleFromExponent(i)/right
        celly2 = GetDoubleFromExponent(i)/bottom

        for(j = cellx1; j <= cellx2; j++)
            for(k =celly1; k <= celly2; k++ )
            {
                CString strSpatialID;
                strSpatialID.Format("%6d_%12d_%12d", i, j+bigint, k+bigint)

                            Thực hiện query trong bảng dữ liệu với trường spatial index bằng
                            với strSpatialID.
            }
    }
```

Trong trường hợp xấu nhất là lưới có kích thước mịn nhất là bằng 1, khi đó việc query không gian theo ví dụ trên sẽ chậm. Trong thực tế ứng dụng, chúng ta có thể không xét đến lưới quá mịn, nghĩa là kích thước cell nhỏ hơn một giá trị nào đó. Chúng ta dùng lưới có kích thước cell lớn hơn chứa đối tượng có cùng kích thước. Khi đó trong đoạn mã (III) ở trên, chúng tôi cộng thêm giá trị 3 vào *gridlevel (hàng thứ hai)*. Điều này có nghĩa là các đối tượng có bán kính '1' sẽ được chuyển lên lưới cấp '3', đối tượng cỡ '2' đặt ở lưới cấp 4, v.v... Ngòai ra để tránh dùng quá nhiều các câu query với điều kiện *bằng*, chúng ta có thể dung bất đẳng thức cho câu query. Với các phân tích như trên chúng tôi xác định cấu trúc bảng để quản lý tương ứng với một lớp dữ liệu không gian như sau.

Tên	Kiểu	Độ dài	Ghi chú
SpatialID	CHAR	32	Gồm ba thành phần, cách nhau bởi ký tự gạch dưới. 6 ký tự đầu biểu diễn cấp grid chứa đối tượng. 12 ký tự kế cho xc' 12 ký tự cuối cùng cho yc'
GuidID	GUID		Kiểu GUID theo chuẩn lưu trữ trong SQLServer
Data	OLEObject		Lưu trữ toàn bộ nội dung của đối tượng không gian
……			Các trường tiện ích khác

4. Kết quả thử nghiệm

Chúng tôi thử nghiệm tốc độ query không gian (dùng cho việc hiển thị và chọn) trên một bảng dữ liệu địa hình của một khu vực rộng 2000km^2 với hơn 1.2 triệu thửa đất, 1.2 triệu chuỗi chú thích, 14 ngàn cung đường, và các dữ liệu thật khác như hệ thống sông, biên giới, các điểm độ cao. Số lượng record khỏang 3 triệu được tạo bằng phương pháp indexing tuần tự . Dung lương dữ liệu trên dĩa xấp xỉ 03 Gigabyte. Sử dụng SQLServer 2000 cài đặt trên PC PIV 1.4 MHz để lưu dữ liệu này . Kết quả thử nghiệm cho thấy tốc độ làm việc rất hiệu quả như trong bảng dưới đây. Kết quả này gần như tương đương với lưu trữ trên tập tin nhị phân, với phương pháp lưu trữ theo cây phân chia không gian Quad-Tree. Chúng tôi cũng đã thử nghiệm trên môi trường nhiều người dung đồng thời và cũng cho ra kết quả tốt như trên.

Thao tác	Thời gian thực hiện (tính theo giây)
Hiển thị toàn bộ trên màn hình	0.47
Phóng to gấp đôi tại tâm	0.44
Tiếp tục phóng to gấp đôi	0.56
Dời khung nhìn sang trái	0.53
Chọn tất cả đối tượng giao với hình tròn	1.78

Tiếp tục phóng to gấp đôi	1.70
Thự hiện phép phân tích không gian (xác định giao , hội với đối tượng cho trước)	1.97
Dời khung nhìn sang phải	1.24
Tiếp tục phóng to gấp đôi (tỉ lệ bây giờ 1:2000)	0.37
Tất cả các thao tác lấy dữ liệu tại tỉ lệ 1:2000	Thời gian thực

5. Kết luận

Giải pháp trên đây đã làm việc rất hiệu quả với số lượng dữ liệu khoảng vài chục triệu record/object cho một lớp bản đồ (hay bảng dữ liệu table), với dung lượng cho 01 layer khoảng 1-10Gigabyte. Các thí nghiệm trên môi trường dùng chung cũng cho thấy hiệu quả. Tuy vậy, chúng tôi chưa thử nghiệm trên những dữ liệu cực lớn (khoảng hàng trăm triệu đến hảng tỉ record cho mỗi bảng). Trong bài viết này chúng tôi chưa đề cập đến các khía cạnh kỹ thuật khác cần giải quyết như Undo/Redo cho nhiều người dùng, các phương pháp indexing khác. Chúng tôi hy vọng sẽ đề cập đến các khía cạnh này trong các bài viết tương lai.

USING HIERARCHICAL GRID TO BUILD INDEPENDENT INDEX FOR SPATIAL DATA IN DBMS

Ngo Quoc Viet, Dinh Tien Son

***ABSTRACT**: We suggest a method to store and explore spatial data using DBMS via creating spatial indices for the 2D real world objects so that they are independent each other. Purpose of space partition with independent spatial is for multi-user environment so that it follow all normal deployment operations. The main idea in our methodology is to create spatial index via aligning object to suitable grid level so that modification (add, update, delete) of one or more objects doesn't affect to spatial index of any object/record in the table. In addition, size and level of grid at each level is determined by the arrangement of objects, not predefined through experiment of the data experts.*

TÀI LIỆU THAM KHẢO

[1] ANSI/IEEE Standard 754-1985,
Standard for Binary Floating Point Arithmetic

[2] Peter van Oosterom., Reactive Data Structures For GeoGraphic Information Systems. Oxford University Press 1993.

[3] Philippe Rigaux, Michel Scholl, Agnes Voisard, "Spatial Databases with application to GIS", Morgan Kaufmann Publishers Academic Press 2002.