Empress Technical News – May 2009

Database Spatial Search Index – Fast Geographical Data Retrieval

Introduction

Empress Ultra Embedded V10.20 offers many additional features to application developers. One of those features is a Spatial Search Index capability for fast geographical data retrieval.

Empress Spatial Index

Empress Ultra Embedded V10.20 Spatial Search Index capability empowers application developers to implement an efficient search using spatial access methods for geographic location data.

The spatial index capability is developed as an additional set of C calls that are used in conjunction with Empress C/C++ Kernel Level API (mr Routines) and a set of SQL commands.

The Empress spatial Index technology is based on an **R* tree** index. **R* tree** is a variant of R tree for indexing spatial information. **R-trees** are tree data structures similar to B-trees (already in use with Empress indexes) but are used for spatial access methods, i.e. for indexing multi-dimensional information; for example, the (X, Y) coordinates of geographical data. A common real-world usage for an R-tree might be: "Find all points of interest within 5 kilometers of my current location".

Empress Spatial Index C API

The Empress spatial search capability uses the following set of C API calls that should be used in conjunction with Empress C/C++ Kernel Level mr Routines:

```
ms_rstree_open
ms_rstree_close
ms_rstree_point_search_in_rectangle
ms_rstree_point_search_in_circle
ms_rstree_rect_search_in_circle
ms_rstree_rect_search_overlap_circle
ms_rstree_rect_search_in_rectangle
ms_rstree_rect_search_overlap_rectangle
ms_rstree_list_free
```

Example Using Spatial Search

In the following source code example ***rstree_select.c*** searches for all the restaurants in the specified rectangle given by its geographical coordinates. Restaurants, stored as points of interest, have the characteristics such as, for example:

```
ID:         4
Name:       Frankie Tomatto's
Address:    7225 Woodbine Avenue, Markham, ON L3R 1A3
Phone:      (905) 940-1900
Latitude:   43 degrees 49 minutes 09.51 seconds North
Longitude:  79 degrees 20 minutes 53.41 seconds West
```

R* tree index expects geographical parameters, such as longitude and latitude, defined as integers or float numbers. Hence, the values expressed in degrees, minutes and seconds need to be converted. For example:

The latitude 43 degrees 49 minutes 09.51 seconds North is converted to:

43 * 360000 + 49 * 6000 + 9.51*100 = **15774951**

The longitude 79 degrees 20 minutes 53.41 seconds West is converted to:

79 * 360000 + 20 * 6000 + 53.41*100 = **28565341**

The following tables give an idea of how distances in meters relates to degrees, minutes and seconds. The figures are taken from the following source**:**

Robinson, Arthur H. et al. *Elements of Cartography,* 5th ed. New York: John Wiley & Sons, 1984. (pp 64-66, Appendix B)

Latitude:

1 degree of latitude	**=**	**1 degree**	**or**	**110,874.4 meters**
1 minute of latitude	**=**	**1/60 of a degree**	**or**	**1,847.91 meters**
1 second of latitude	**=**	**1/60 of a minute**	**or**	**30.7984 meters**
1/10 of a second of latitude	**=**	**.10 of one second**	**or**	**3.0798 meters**
1/100 of a second of latitude	**=**	**.01 of one second**	**or**	**0.3080 meters**

Longitude:

1 degree of longitude	**=**	**1 degree**	**or**	**95,506 meters**
1 minute of longitude	**=**	**1/60 of a degree**	**or**	**1,591.7667 meters**
1 second of longitude	**=**	**1/60 of a minute**	**or**	**26.5294 meters**
1/10 of a second of longitude	**=**	**.10 of one second**	**or**	**2.6529 meters**
1/100 of a second of longitude	**=**	**.01 of one second**	**or**	**0.2653 meters**

Example Source Code

In the example below, integers were used as the coordinates. Float values could have been used as well. When dealing with the circular search in the example the radius of 1500 represents approximately 450 meters.
The source code example ***rstree_select.c is as follows:***

```
#include    <mscc.h>

#define     DATABASE              "testdb"

#define     RTREE_RECT_XMIN       28562500
#define     RTREE_RECT_YMIN       15772000
#define     RTREE_RECT_XMAX       28567500
#define     RTREE_RECT_YMAX       15774000

#define     RTREE_CIRCLE_X        28565200
#define     RTREE_CIRCLE_Y        15773100
#define     RTREE_CIRCLE_RADIUS   1500

int main (int argc, char* argv[])
{
        void*  poi_tabdesc;
        void*  name_attrdesc;
        void*  id_attrdesc;
        void*  x_attrdesc;
        void*  y_attrdesc;
        void*  poi_recdesc;
        void*  qual;
        void*  retrieve_desc;
        void*  rshandle;
        char*  name_value;
        char*  id_value;
        char*  x_value;
        char*  y_value;
        long*  rec_list;
        long   rect_buf[4]={RTREE_RECT_XMIN, RTREE_RECT_YMIN, RTREE_RECT_XMAX,
RTREE_RECT_YMAX};
        long   circle_buf[3]={RTREE_CIRCLE_X, RTREE_CIRCLE_Y, RTREE_CIRCLE_RADIUS};


        msinit ();

          poi_tabdesc = mropen (DATABASE, "poi", 'r');

        poi_recdesc = mrmkrec (poi_tabdesc);
        name_attrdesc = mrngeta (poi_tabdesc, "name");
        id_attrdesc = mrngeta (poi_tabdesc, "id");
        x_attrdesc = mrngeta (poi_tabdesc, "x");
        y_attrdesc = mrngeta (poi_tabdesc, "y");

        name_value = mrspv (name_attrdesc);
        id_value = mrspv (id_attrdesc);
        x_value = mrspv (x_attrdesc);
        y_value = mrspv (y_attrdesc);

        rshandle = ms_rstree_open (poi_tabdesc, x_attrdesc, y_attrdesc, 0);
        if (rshandle == 0)
        {
               fprintf (stderr, "cannot open rstree index \n");
                return (1);
        }

/*** search for points of interest in defined rectangle ***/
        rec_list = ms_rstree_point_search_in_rectangle (rshandle, (long*) rect_buf);

/*** alternatevely search for points of interest in defined circle
        rec_list = ms_rstree_point_search_in_circle (rshandle, (long*) circle_buf);
```

```
***/

        if (rec_list == 0)
                goto done1;

        printf ("Points of Interest inside defined rectangle\n\n");
        printf ("id   name               X coordinate   Y coordinate\n");

          qual = mrqlst (poi_tabdesc, rec_list);

        retrieve_desc = mrgetbegin (qual, poi_recdesc, (void*) 0);
        while (mrget (retrieve_desc))
        {
             mrcopyv (poi_recdesc, name_attrdesc, name_value);
             mrcopyv (poi_recdesc, id_attrdesc, id_value);
             mrcopyv (poi_recdesc, x_attrdesc, x_value);
             mrcopyv (poi_recdesc, y_attrdesc, y_value);
             printf ("%-5s%-20s%-15s%-15s\n", id_value, name_value, x_value, y_value);
        }
        mrgetend (retrieve_desc);
        ms_rstree_list_free (rec_list);


        mrfree (name_value);
        mrfree (id_value);
        mrfree (x_value);
        if (!mrfrrec (poi_recdesc))
        {
               fprintf (stderr, "ERROR in freeing record buffers\n");
               fprintf (stderr, "mroperr='%d' : %s\n", mroperr, mrerrmsg ());
        }

        if (!mrclose (poi_tabdesc))
        {
               fprintf (stderr, "Unable to close the table\n");
               fprintf (stderr, "mroperr='%d' : %s\n", mroperr, mrerrmsg ());
        }
done1:
       ms_rstree_close (rshandle);

       msend ();
       return 0;
}
```

Example – Preparing Data

The following sequence of steps will show how to prepare data in the database for running the example. The latitude and longitude values were real values derived from Google Earth.

To create the database ***testdb*** (if it doesn't exist) run:

```
empmkdb testdb
```

To create the table ***poi*** that contains 'points of interest' locations and insert records with some restaurants information, run:

```
empbatch testdb sql_commands
```

where **sql_commands** file looks as follows:

```
CREATE TABLE poi (
        id              INTEGER     NOT NULL,
        name            VARCHAR(32),
        address         CLOB(64),
        phone           VARCHAR(15),
        x               INTEGER,
        y               INTEGER
);

INSERT INTO poi VALUES ( 1, "Ten23",
"3100 Steeles Avenue East, Markham, ON L3R 8T3",
"(905) 479-6488", 28565120, 15773195 );

INSERT INTO poi VALUES ( 2, "Buffet City",
"3160 Steeles Avenue East, Markham, ON L3R 4G9",
"(905) 474-9899", 28563997, 15773487 );

INSERT INTO poi VALUES ( 3, "Golden Griddle",
"7100 Woodbine Avenue, Markham, ON L3R 5J2",
"(905) 477-6980", 28565768, 15773777 );

INSERT INTO poi VALUES ( 4, "Frankie Tomatto's",
"7225 Woodbine Avenue, Markham, ON L3R 1A3",
"(905) 940-1900", 28565341, 15774951 );

INSERT INTO poi VALUES ( 5, "Phoenix Restaurant",
"7155 Woodbine Avenue, Markham, ON L3R 1A3",
"(905) 940-1113", 28565710, 15774350 );

INSERT INTO poi VALUES ( 6, "Tandoori Queen",
"708 Gordon Baker Road, North York, ON M2H 3B4",
"(416) 492-7993", 28564299, 15771282 );

INSERT INTO poi VALUES ( 7, "Owl & Firkin",
"7181 Woodbine Avenue, Markham, ON L3R 1A3",
"(905) 513-6611", 28565684, 15774552 );

CREATE RSTARTREE INDEX poix ON poi(x,y);
```

Example – Compiling the C Program

To compile an example: go to the directory where the example is copied and run the Empress compiler utility **empcc** command:

```
empcc -o rstree_select rstree_select.c
```

The executable file **rstree_select** will be created in the same directory.

Example – Running the Program Using a Rectangle

When **rstree_select** gets executed it will display the following result.

```
Points of Interest inside defined rectangle

id   name                X coordinate   Y coordinate
1    Ten23                28565120       15773195
2    Buffet City          28563997       15773487
3    Golden Griddle       28565768       15773777
```

Points of interest (i.e. restaurants) that satisfy the criteria for the above search are shown in the **Fig 1**. The letter A in the red balloon on the map below is the starting point. The numbers in the blue balloons on the map below correspond to the "id" of the "Points of Interest".

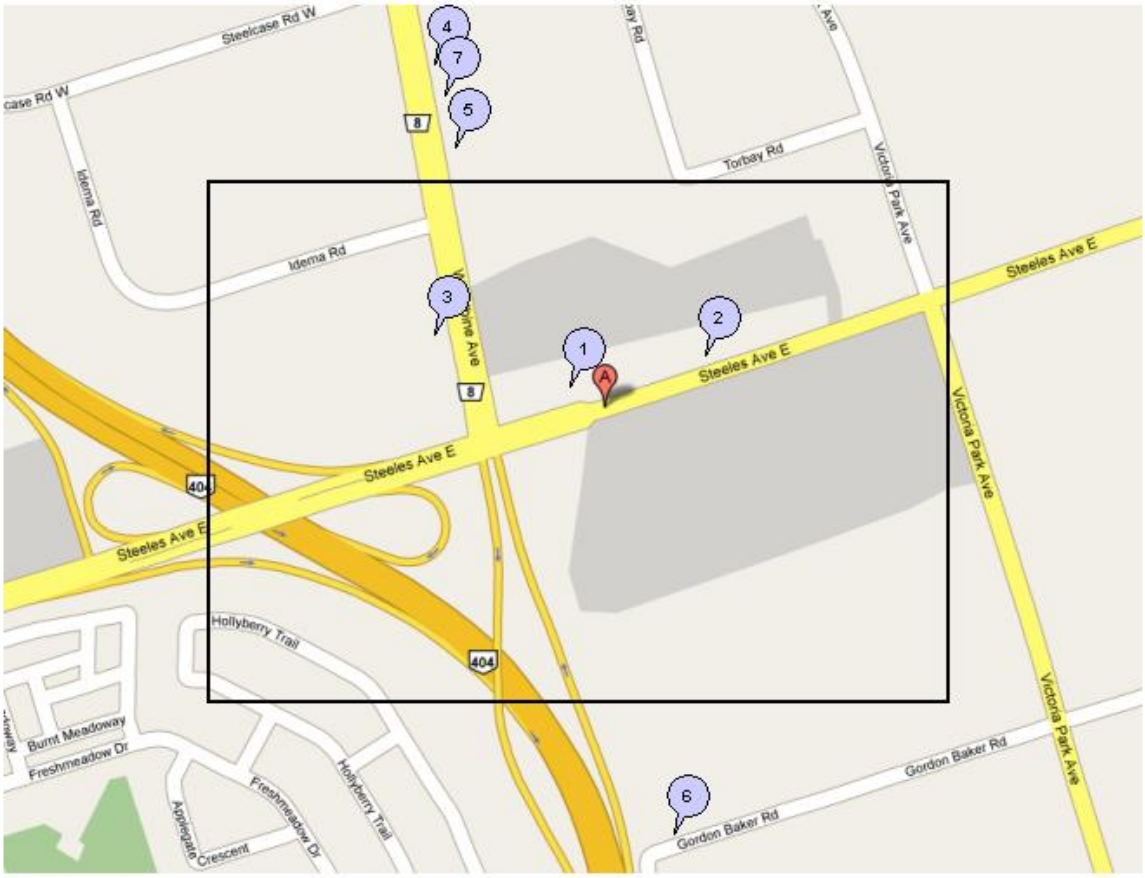

Fig 1: Spatial search for all points of interest inside the defined rectangle

Example – Running the Program Using a Circle

Instead of using rectangle spatial search:

```
ms_rstree_point_search_in_rectangle (rshandle, (long*)
rect_buf);
```

a circle spatial search could be used:

```
ms_rstree_point_search_in_circle (rshandle, (long*)
circle_buf);
```

With this change, when **rstree_select** gets executed it will display the following result.

```
Points of Interest inside defined circle

id    name                X coordinate   Y coordinate
1     Ten23                28565120       15773195
2     Buffet City          28563997       15773487
3     Golden Griddle       28565768       15773777
5     Phoenix Restaurant   28565710       15774350
```

Points of interest (i.e. restaurants) that satisfy the criteria for the above search are shown in the **Fig 2**. The letter A in the red balloon on the map below is the starting point. The numbers in the blue balloons on the map below correspond to the “id” of the “Points of Interest”.

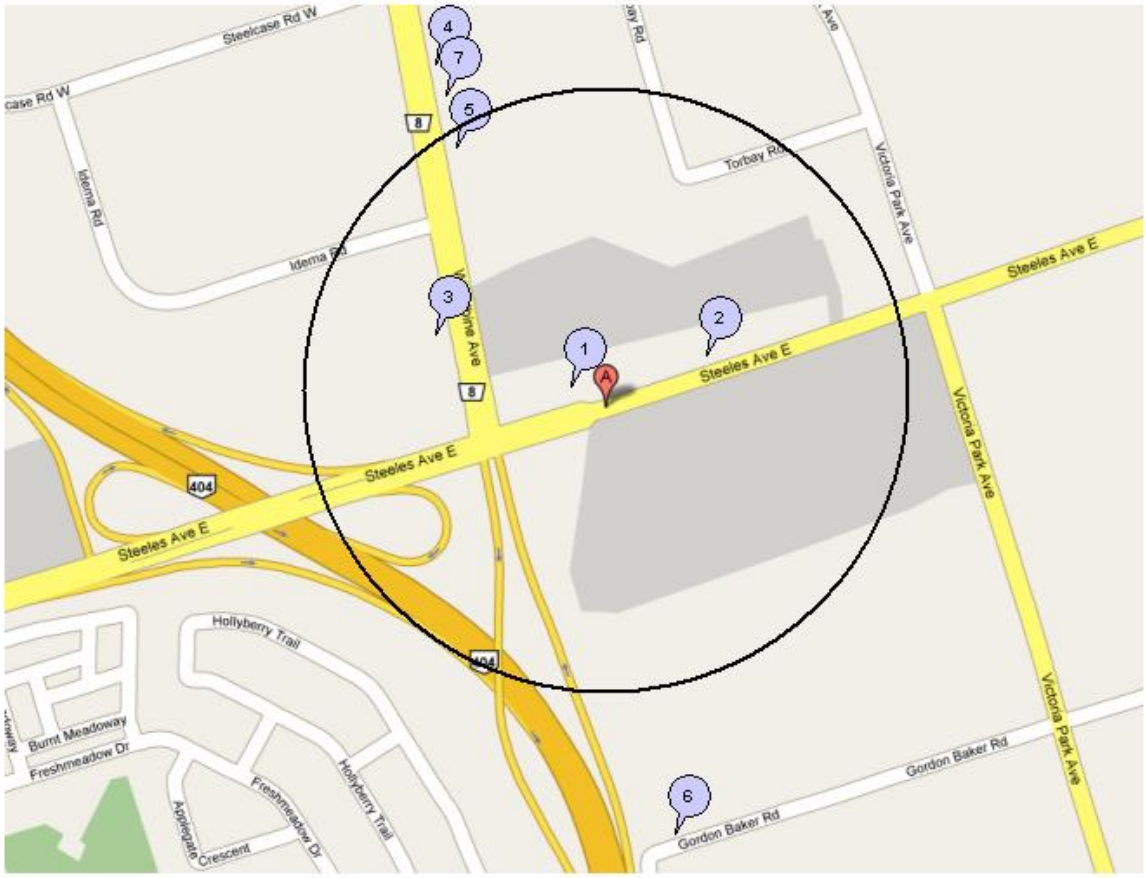

Fig 2: Spatial search for all points of interest inside the defined circle

The rectangle produced 3 points of interest while the circle produced 4 points of interest.

And Furthermore

Empress Spatial index search functionality can be used with other regular Empress query search conditions. In addition, it can be used with special Empress features such as Empress text search and Empress Shiborikomi search.

Empress Software Inc.
www.empress.com

Please email your comments and questions regarding this article to ivy.wong@empress.com.

If you do not wish to receive further emails from Empress Software, please write to unsubscribe@empress.com.

Copyright©2008 Empress Software Inc. All Rights Reserved, the Empress Logo and all trademarks identified by ®, TM or SM are registered trademarks, trademarks, or service marks of Empress Software Inc, and may be registered in certain jurisdictions. All other trademarks are the property of their respective owners.