Applied Databases

Lecture 9
Spacial Queries and Indexes

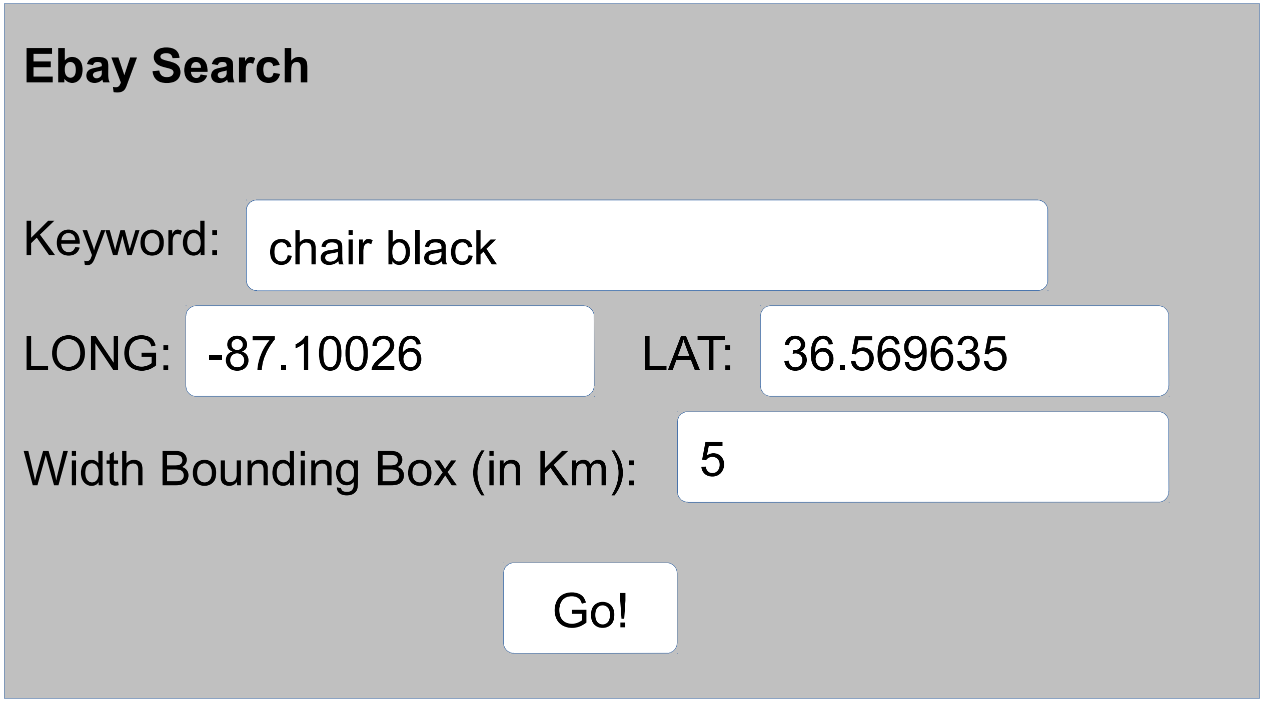
Sebastian Maneth

University of Edinburgh - February 13th, 2017

Outline

1. Assignment 2
2. Spatial Types
3. Spatial Queries
4. R-Trees

Assignment 2



→ simple web page (provided by us)

→ calls your java program with: a string of keywords, and,

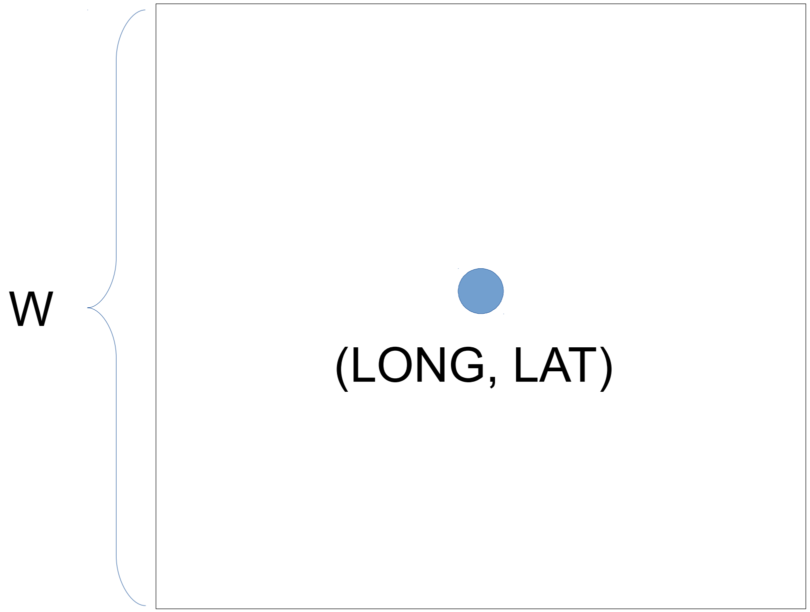
→ optionally also LONG/LAT/Width values.

Bounding Box

→ square box

→ centered at the point (LONG, LAT)

→ of given width W



Assignment 2

Implement three Java functions:

(1) basicSearch(String query, int NumResultsToSkip,
int numResultsToReturn)

(2) spatialSearch(String query, SearchRegion region,
int NumResultsToSkip,
int numResultsToReturn)

(3) getHTMLforItemId(String itemId)

Assignment 2

Implement three Java functions:

(1) basicSearch(String query, int NumResultsToSkip,
int numResultsToReturn)

(2) spatialSearch(String query, SearchRegion region,
int NumResultsToSkip,
int numResultsToReturn)

(3) getHTMLforItemId(String itemId)

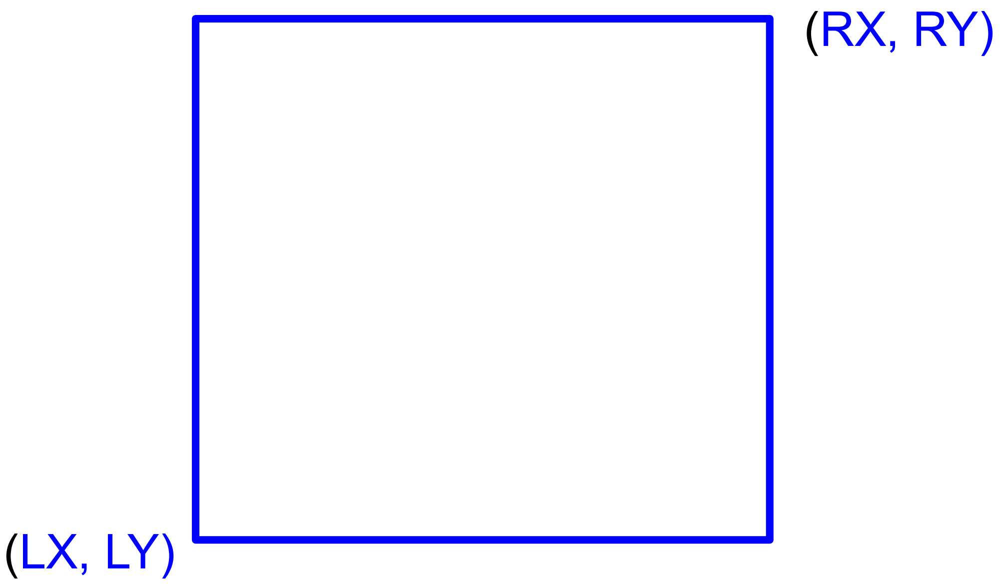
Assignment 2

spatialSearch(SearchRegion region)

→ find all items in a given region

→ a region is a box, given by two points:

- (LX, LY) coordinates of lower left corner
- (RX, RY) coordinates of upper right corner



Assignment 2

spatialSearch(SearchRegion region)

→ find all items in a given region

```
SELECT item_id FROM Item_coordinates
WHERE LX <= latitude
  AND latitude <= RX
  AND LY <= longitude
  AND longitude <= RY;
```

```
+------------+-----------+-----------+
| item_id    | latitude  | longitude |
+------------+-----------+-----------+
| 1043374545 | -87.10026 | 36.569635 |
```

Assignment 2

spatialSearch(SearchRegion region)

→ find all items in a given region

```
SELECT item_id FROM Item_coordinates
WHERE LX <= latitude
  AND latitude <= RX
  AND LY <= longitude
  AND longitude <= RY;
```

range queries (intersected)

→ you would create B+tree indexes on longitude and latitude

→ CREATE INDEX long on Item_coordinates(longitude);

→ CREATE INDEX lat on Item_coordinates(latitude);

Assignment 2

spatialSearch(SearchRegion region)

→ find all items in a given region

```
SELECT item_id FROM Item_coordinates
WHERE LX <= latitude
  AND latitude <= RX
  AND LY <= longitude
  AND longitude <= RY;
```

range queries (intersected)

→ you would create B+tree indexes on longitude and latitude

→ CREATE INDEX long on Item_coordinates(longitude);

→ CREATE INDEX lat on Item_coordinates(latitude);

→ On large data, this would be **very slow**!

2. Spatial Types

MySQL

→ provides spatial indexes (an R-tree index, in particular)

→ spatial index can be created for spatial attributes

Spatial Types

→ POINT
→ LINESTRING
→ POLYGON
→ GEOMETRY

→ MULITPOINT
→ MULTILINESTRING
→ MULTIPOLYGON
→ GEOMETRYCOLLECTION

2. Spatial Types

POINT

→ 2-dimensional
→ X-coordinate and Y-coordinate
→ is a zero-dimensional GEOMETRY

point type

```
CREATE TABLE Item_xy (item_id PRIMARY KEY, xy POINT);

INSERT INTO Item_xy VALUES (123456, POINT(-5.03434, 19.999));
```

conversion function from pair of doubles to point.

2. Spatial Types

POINT

→ 2-dimensional
→ X-coordinate and Y-coordinate
→ is a zero-dimensional GEOMETRY

point type

```
CREATE TABLE Item_xy (item_id PRIMARY KEY, xy POINT);

INSERT INTO Item_xy VALUES (123456, POINT(-5.03434, 19.999));
```

conversion function from
pair of doubles to point.

```
SELECT * FROM Item_xy;
+---------+-------------------------+
| item_id | xy                      |
+---------+-------------------------+
```

2. Spatial Types

POINT

→ 2-dimensional
→ X-coordinate and Y-coordinate
→ is a zero-dimensional GEOMETRY

point type

```
CREATE TABLE Item_xy (item_id PRIMARY KEY, xy POINT);

INSERT INTO Item_xy VALUES (123456, POINT(-5.03434, 19.999));
```

conversion function from pair of doubles to point.

→ POINT has no display function
→ either use X(.) and Y(.) to obtain X- and Y-values
→ or use AsText(.)

```
SELECT item_id,X(xy),Y(xy) FROM Item_xy;
+---------+----------+--------+
| item_id | X(xy)    | Y(xy)  |
+---------+----------+--------+
|  123456 | -5.03434 | 19.999 |
+---------+----------+--------+
```

2. Spatial Types

POINT

→ 2-dimensional
→ X-coordinate and Y-coordinate
→ is a zero-dimensional GEOMETRY

point type

```
CREATE TABLE Item_xy (item_id PRIMARY KEY, xy POINT);

INSERT INTO Item_xy VALUES (123456, POINT(-5.03434, 19.999));
```

conversion function from pair of doubles to point.

→ POINT has no display function
→ either use X(.) and Y(.) to obtain X- and Y-values
→ or use AsText(.)

```
SELECT item_id,AsText(xy) FROM Item_xy;
+---------+----------+-------------+
| item_id | AsText(xy)             |
+---------+----------+-------------+
|  123456 | POINT(-5.03434 19.999) |
+---------+----------+-------------+
```

2. Spatial Types

LINESTRING

→ string of lines connecting a list of one ore more points
→ is a one-dimensional GEOMETRY

```
> SET @ls = GeomFromText('LineString(1 1,2 2,3 3)');

> SELECT NumPoints(@ls);
+-----------------+
| NumPoints(@ls) |
+-----------------+
|               3 |
+-----------------+
>
SELECT AsText(PointN(@ls,3));
+------------------------+
| AsText(PointN(@ls,3)) |
+------------------------+
| POINT(3 3)             |
+------------------------+
```

```
> SELECT GLength(@ls);
+---------------------+
| GLength(@ls)        |
+---------------------+
| 2.8284271247461903 |
+---------------------+

> SELECT IsClosed(@ls);
+----------------+
| IsClosed(@ls) |
+----------------+
|              0 |
+----------------+
```

equals 2*sqrt(2)

StartPoint(@ls)
=EndPoint(@ls)?

2. Spatial Types

LINESTRING

→ string of lines connecting a list of one ore more points
→ is a one-dimensional GEOMETRY

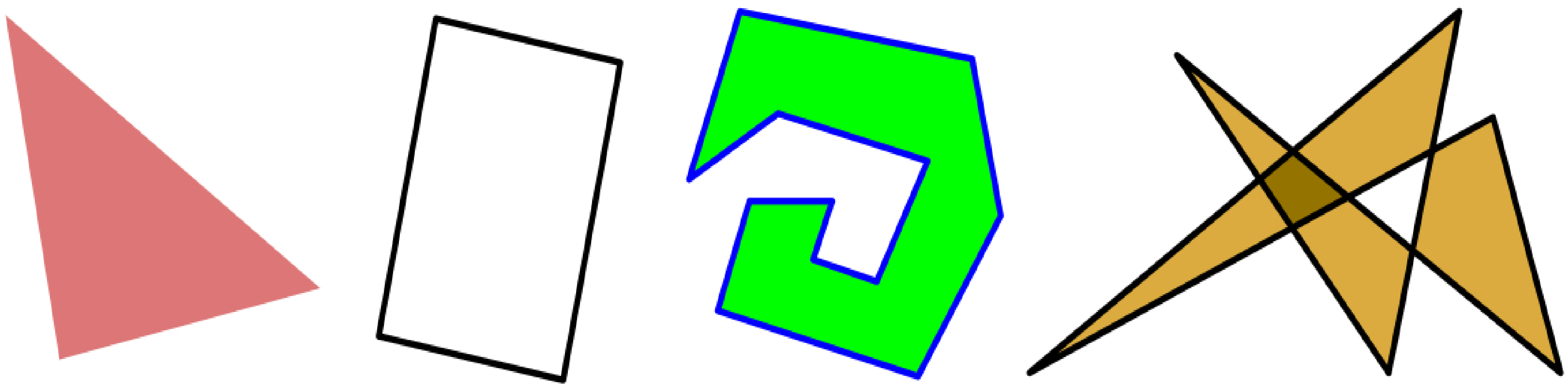
```
> SET @ls = GeomFromText('LineString(1 1,2 2,3 3 1 1)');

> SELECT IsClosed(@ls);
+---------------+
| IsClosed(@ls) |
+---------------+
|             1 |
+---------------+
```

StartPoint(@ls)
=EndPoint(@ls)?

2. Spatial Types

POLYGON

→ closed linestring, describes a planar surface
→ is a *two-dimensional* GEOMETRY



Polygons of different kinds:
→ open (excluding its boundary)
→ bounding circuit only (ignoring its interior)
→ closed (both)
→ self-intersecting with varying densities of different regions.

2. Spatial Types

POLYGON

```
> SET @poly = GeomFromText('Polygon(0 0,0 3,3 0,0 0)');

> SELECT Area(@poly);
+-------------+
| Area(@poly) |
+-------------+
|         4.5 |
+-------------+

> SELECT AsText(Centroid(@poly));
+-------------------------+
| AsText(Centroid(@poly)) |
+-------------------------+
| POINT(1 1)              |
+-------------------------+
```

2. Spatial Types

POLYGON

```
> SET @poly = GeomFromText('Polygon(0 0,0 3,3 0,0 0)');

> SELECT Area(@poly);
+-------------+
| Area(@poly) |
+-------------+
|         4.5 |
+-------------+

> SET @poly = GeomFromText('Polygon((0 0,0 3,3 0,0 0),
                                    (1 1,1 2,2 1,1 1))');
> SELECT Area(@poly);
+-------------+
| Area(@poly) |
+-------------+
|           4 |
+-------------+
```

excluded

2. Spatial Types

```
> SET @poly = GeomFromText('Polygon(0 0,0 3,3 0,0 0)');

> SELECT Area(@poly);
+-------------+
| Area(@poly) |
+-------------+
|         4.5 |
+-------------+
> SET @poly = GeomFromText('Polygon((0 0,0 3,3 0,0 0),
                                    (1 1,1 2,2 1,1 1))');
> SELECT Area(@poly);
+-------------+
| Area(@poly) |
+-------------+
|           4 |
+-------------+
> SET @poly = GeomFromText('MultiPolygon( ((0 0,0 3,3 0,0 0)),
                                          ((1 1,1 2,2 1,1 1)) )');
> SELECT Area(@poly);
+-------------+
| Area(@poly) |
+-------------+
|           5 |
+-------------+
```

added

2. Spatial Types

POLYGON

```
> SET @poly = GeomFromText('Polygon(0 0,0 3,3 0,0 0)');

> SELECT Area(@poly);
+-------------+
| Area(@poly) |
+-------------+
|         4.5 |
+-------------+
> SET @poly = GeomFromText('Polygon((0 0,0 3,3 0,0 0),
                                    (1 1,1 2,2 1,1 1))');
> SELECT Area(@poly);
+-------------+
| Area(@poly) |
+-------------+
|           4 |
+-------------+
> SELECT AsText(Centroid(@poly));
+----------------------------------------------+
| AsText(Centroid(@poly))                      |
+----------------------------------------------+
| POINT(0.9583333333333334 0.9583333333333334) |
+----------------------------------------------+
```

2. Spatial Types

POLYGON

```
> SET @poly = GeomFromText('Polygon((0 0,0 3,3 0,0 0),
                                   (1 1,1 2,2 1,1 1))');
> SELECT NumInteriorRings(@poly);
+-------------------------+
| NumInteriorRings(@poly) |
+-------------------------+
|                       1 |
+-------------------------+

> SELECT AsText(InteriorRingN(@poly,1));
+--------------------------------+
| AsText(InteriorRingN(@poly,1)) |
+--------------------------------+
| LINESTRING(1 1,1 2,2 1,1 1)    |
+--------------------------------+

> SELECT AsText(ExteriorRing(@poly));
+-----------------------------+
| AsText(ExteriorRing(@poly)) |
+-----------------------------+
| LINESTRING(0 0,0 3,3 0,0 0) |
+-----------------------------+
```

2. Spatial Types

POINT, LINESTRING, and POLYGON
are subtypes of GEOMETRY

→ a GEOMETRYCOLLECTION gc contains any number
of geometries, i.e., of points, linestrings, and polygons

→ NumGeometries(gc) – the number of geometries in collection gc

→ GeometryN(gc, n) – the geometry number n

Spatial Indexes

MySQL provides SPATIAL INDEXES (R-trees) for spatial types

→ but only for tables of certain types:
- MyISAM Storage Engine
- InnoDB Storage Engine

```
CREATE TABLE Item_xy (item_id PRIMARY KEY, xy POINT);

CREATE SPATIAL INDEX xy_index ON Item_xy (xy);
ERROR 1464 (HY000): The used table type doesn't support
SPATIAL indexes
```

Spatial Indexes

MySQL provides SPATIAL INDEXES (R-trees) for spatial types

→ but only for tables of certain types:
- MyISAM Storage Engine
- InnoDB Storage Engine

```
CREATE TABLE Item_xy (item_id PRIMARY KEY, xy POINT)
ENGINE=MYISAM;

CREATE SPATIAL INDEX xy_index ON Item_xy (xy);
ERROR 1252 (42000): All parts of a SPATIAL index must be NOT
NULL
```

Spatial Indexes

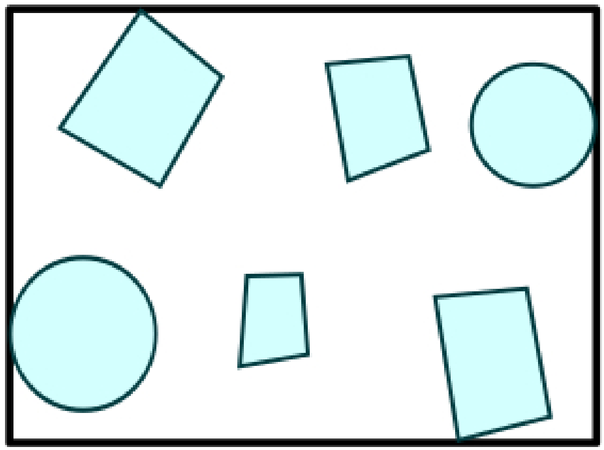
MySQL provides SPATIAL INDEXES (R-trees) for spatial types

→ but only for tables of certain types:
- MyISAM Storage Engine
- InnoDB Storage Engine

```
CREATE TABLE Item_xy (item_id PRIMARY KEY, xy POINT NOT
NULL) ENGINE=MYISAM;

CREATE SPATIAL INDEX xy_index ON Item_xy (xy);
Query OK
```

3. Spatial Queries

→ MySQL lets you test geometric relationships, based on Minimum Bounding Rectangles (aka Bounding Boxes or Envelops)



A set of geometric shapes enclosed by its minimum bounding rectangle

3. Spatial Queries

→ MySQL lets you test geometric relationships, based on Minimum Bounding Rectangles (aka Bounding Boxes or Envelops)

g1,g2: geometries (such as point, line, or polygon)

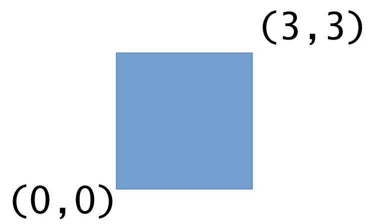
→ MBRContains(g1,g2)
→ MBRCoveredBy(g1,g2)
→ MBRCovers(g1,g2)
→ MBRDisjoint(g1,g2)
→ MBREqual(g1,g2)
→ MBRIntersects(g1,g2)
→ MBROverlaps(g1,g2)
→ MBRTouches(g1,g2)
→ MBRWithin(g1,g2)

3. Spatial Queries

→ mySQL lets you test geometric relationships, based on Minimum Bounding Rectangles (aka Bounding Boxes or Envelops)

```
> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');

> SET @g2 = GeomFromText('Point(1 1)');


> SELECT MBRContains(@g1,@g2), MBRWithin(@g2,@g1);
+----------------------+--------------------+
| MBRContains(@g1,@g2) | MBRWithin(@g2,@g1) |
+----------------------+--------------------+
|                    1 |                  1 |
+----------------------+--------------------+
```

3. Spatial Queries

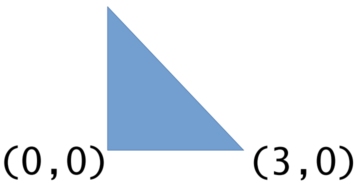


→ select all points within a given MBR

```
> SET @poly = GeomFromText('Polygon(0 0,0 3,3 3,3 0,0 0)');

> SELECT AsText(p) from Points where MBRWithin(p,@poly);
+-----------------+
| AsText(p)       |
+-----------------+
| POINT(0 0)      |
| POINT(0.5 0.5)  |
| POINT(1 1)      |
| POINT(1.5 1.5)  |
| POINT(2 2)      |
| POINT(2.5 2.5)  |
| POINT(3 3)      |
+-----------------+
```

points on the diagonal
(0 0) (.5 .5) etc

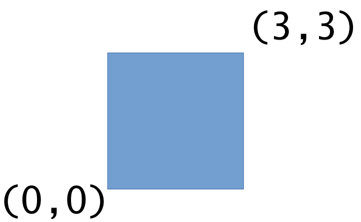
3. Spatial Queries



→ select all points within a given MBR

```
> SET @poly = GeomFromText('Polygon(0 0,0 3,3 0,0 0)');

> SELECT AsText(p) from Points where MBRWithin(p,@poly);

??
```

3. Spatial Queries



→ select all points within a given MBR

```
> SET @poly = GeomFromText('Polygon(0 0,0 3,3 3,3 0,0 0)');

> SELECT AsText(p) from Points where MBRWithin(p,@poly);
+-----------------+
| AsText(p)       |
+-----------------+
| POINT(0 0)      |
| POINT(0.5 0.5)  |
| POINT(1 1)      |
| POINT(1.5 1.5)  |
| POINT(2 2)      |
| POINT(2.5 2.5)  |
| POINT(3 3)      |
+-----------------+
```

→ how can you sort these points wrt to the distance to another point?

3. Spatial Queries

```
> SELECT AsText(p) from Points;
+-----------------+
| AsText(p)       |
+-----------------+
| POINT(0 0)      |
| POINT(0.5 0.5)  |
| POINT(1 1)      |
| POINT(1.5 1.5)  |
| POINT(2 2)      |
| POINT(2.5 2.5)  |
| POINT(3 3)      |
| POINT(3.5 3.5)  |
| POINT(4 4)      |
+-----------------+
```

```
> SELECT AsText(p),(2-x(p))*(2-x(p))+
(2-y(p))*(2-y(p)) AS sqdist FROM Points;
+-----------------+--------+
| AsText(p)       | sqdist |
+-----------------+--------+
| POINT(0 0)      |      8 |
| POINT(0.5 0.5)  |    4.5 |
| POINT(1 1)      |      2 |
| POINT(1.5 1.5)  |    0.5 |
| POINT(2 2)      |      0 |
| POINT(2.5 2.5)  |    0.5 |
| POINT(3 3)      |      2 |
| POINT(3.5 3.5)  |    4.5 |
| POINT(4 4)      |      8 |
+-----------------+--------+
```

squared Eucledian distance
from the point (2,2)

3. Spatial Queries

```
> SELECT AsText(p) from Points;
+-----------------+
| AsText(p)       |
+-----------------+
| POINT(0 0)      |
| POINT(0.5 0.5)  |
| POINT(1 1)      |
| POINT(1.5 1.5)  |
| POINT(2 2)      |
| POINT(2.5 2.5)  |
| POINT(3 3)      |
| POINT(3.5 3.5)  |
| POINT(4 4)      |
+-----------------+
```

```
> SELECT AsText(p),(2-x(p))*(2-x(p))+
(2-y(p))*(2-y(p)) AS sqdist FROM Points
ORDER BY sqdist;
+-----------------+--------+
| AsText(p)       | sqdist |
+-----------------+--------+
| POINT(2 2)      |      0 |
| POINT(1.5 1.5)  |    0.5 |
| POINT(2.5 2.5)  |    0.5 |
| POINT(1 1)      |      2 |
| POINT(3 3)      |      2 |
| POINT(0.5 0.5)  |    4.5 |
| POINT(3.5 3.5)  |    4.5 |
| POINT(0 0)      |      8 |
| POINT(4 4)      |      8 |
+-----------------+--------+
```

sorted
squared Eucledian distance
from the point (2,2)

3. Spatial Queries

```
> SELECT AsText(p) from Points;
+-----------------+
| AsText(p)       |
+-----------------+
| POINT(0 0)      |
| POINT(0.5 0.5)  |
| POINT(1 1)      |
| POINT(1.5 1.5)  |
| POINT(2 2)      |
| POINT(2.5 2.5)  |
| POINT(3 3)      |
| POINT(3.5 3.5)  |
| POINT(4 4)      |
+-----------------+
```

```
> SELECT AsText(p),SQRT((2-x(p))*(2-x(p))+
(2-y(p))*(2-y(p))) AS dist FROM Points
HAVING dist<=2
ORDER BY dist;
+-----------------+---------------------+
| AsText(p)       | dist                |
+-----------------+---------------------+
| POINT(2 2)      |                   0 |
| POINT(1.5 1.5)  | 0.707106781186547 6 |
| POINT(2.5 2.5)  | 0.7071067811865476  |
| POINT(1 1)      | 1.4142135623730951  |
| POINT(3 3)      | 1.4142135623730951  |
+-----------------+---------------------+
```

→ why can you not use WHERE here?

sorted
squared Eucledian distance from the point (2,2)

3. Spatial Queries

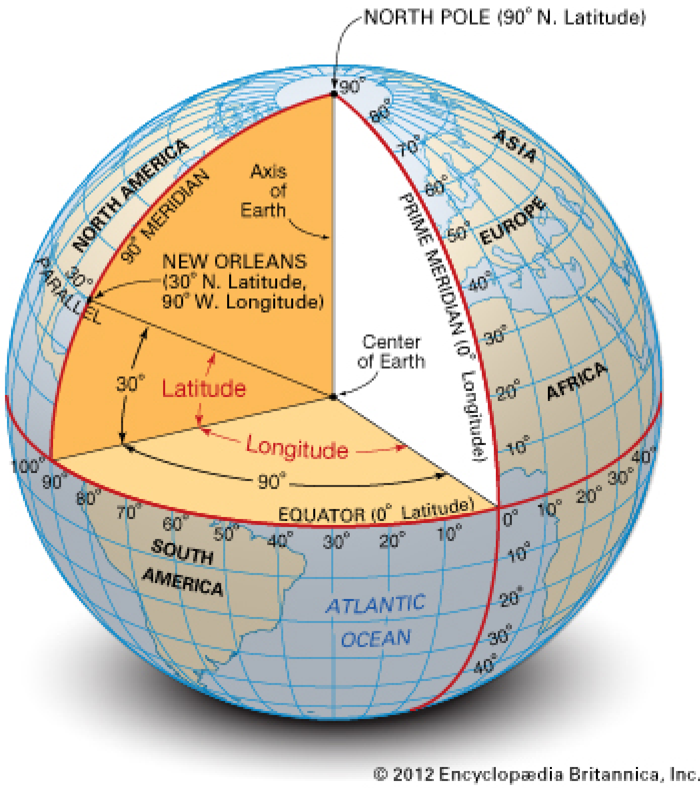
```
> SELECT AsText(p) from Points;
+-----------------+
| AsText(p)       |
+-----------------+
| POINT(0 0)      |
| POINT(0.5 0.5)  |
| POINT(1 1)      |
| POINT(1.5 1.5)  |
| POINT(2 2)      |
| POINT(2.5 2.5)  |
| POINT(3 3)      |
| POINT(3.5 3.5)  |
| POINT(4 4)      |
+-----------------+
```

```
> SELECT AsText(p),SQRT((2-x(p))*(2-x(p))+
(2-y(p))*(2-y(p))) AS dist FROM Points
HAVING dist<=2
ORDER BY dist;
+-----------------+---------------------+
| AsText(p)       | dist                |
+-----------------+---------------------+
| POINT(2 2)      |                   0 |
| POINT(1.5 1.5)  | 0.707106781186547 6 |
| POINT(2.5 2.5)  | 0.707106781186547 6 |
| POINT(1 1)      | 1.4142135623730951  |
| POINT(3 3)      | 1.4142135623730951  |
+-----------------+---------------------+
```

Cave

→ longitude and latitude are on a sphere
→ **cannot** use Eucledian distance as shown here!

sorted
squared Eucledian distance from the point (2,2)



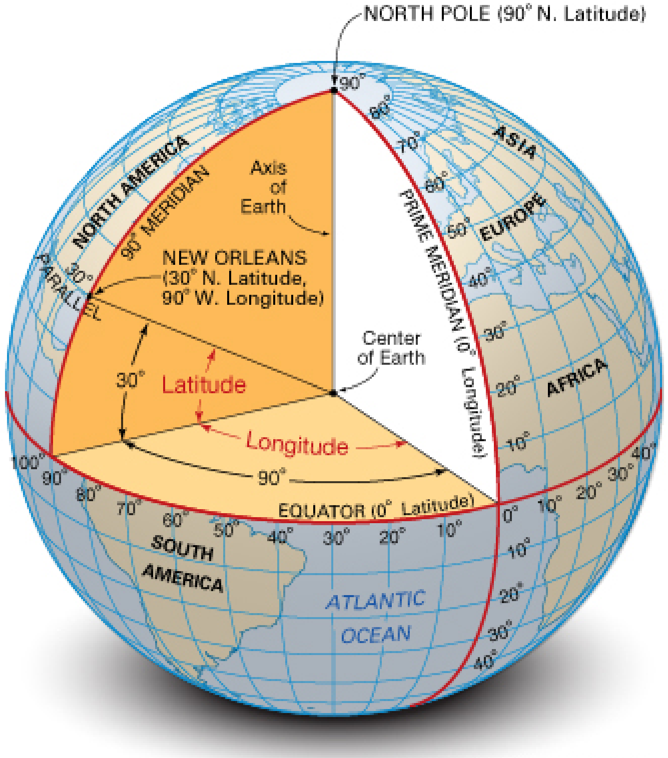

```
+------------+-----------+-------------+
| item_id    | latitude  | longitude   |
+------------+-----------+-------------+

...

| 1043747228 | 45.580557 | -122.374776 |
| 1043749860 | 40.578996 |   -74.27987 |
```

west coast

east coast

```
CREATE FUNCTION
get_distance_in_miles_between_geo_locations(
x1 decimal(10,6), y1 decimal(10,6), x2 decimal(10,6), y2 decimal(10,6))
returns decimal(10,3)
DETERMINISTIC BEGIN
return((ACOS( SIN(x1*PI()/180)*SIN(x2*PI()/180)+
              COS(x1*PI()/180)*COS(x2*PI()/180)*COS((y1-y2)*PI()/180))
       *180/PI())*60*1.1515);
END
```

Rounded! (→ what's this?)



© 2012 Encyclopædia Britannica, Inc.

```
+------------+-----------+-------------+
| item_id    | latitude  | longitude   |
+------------+-----------+-------------+

...

| 1043747228 | 45.580557 | -122.374776 |
| 1043749860 | 40.578996 |   -74.27987 |
```

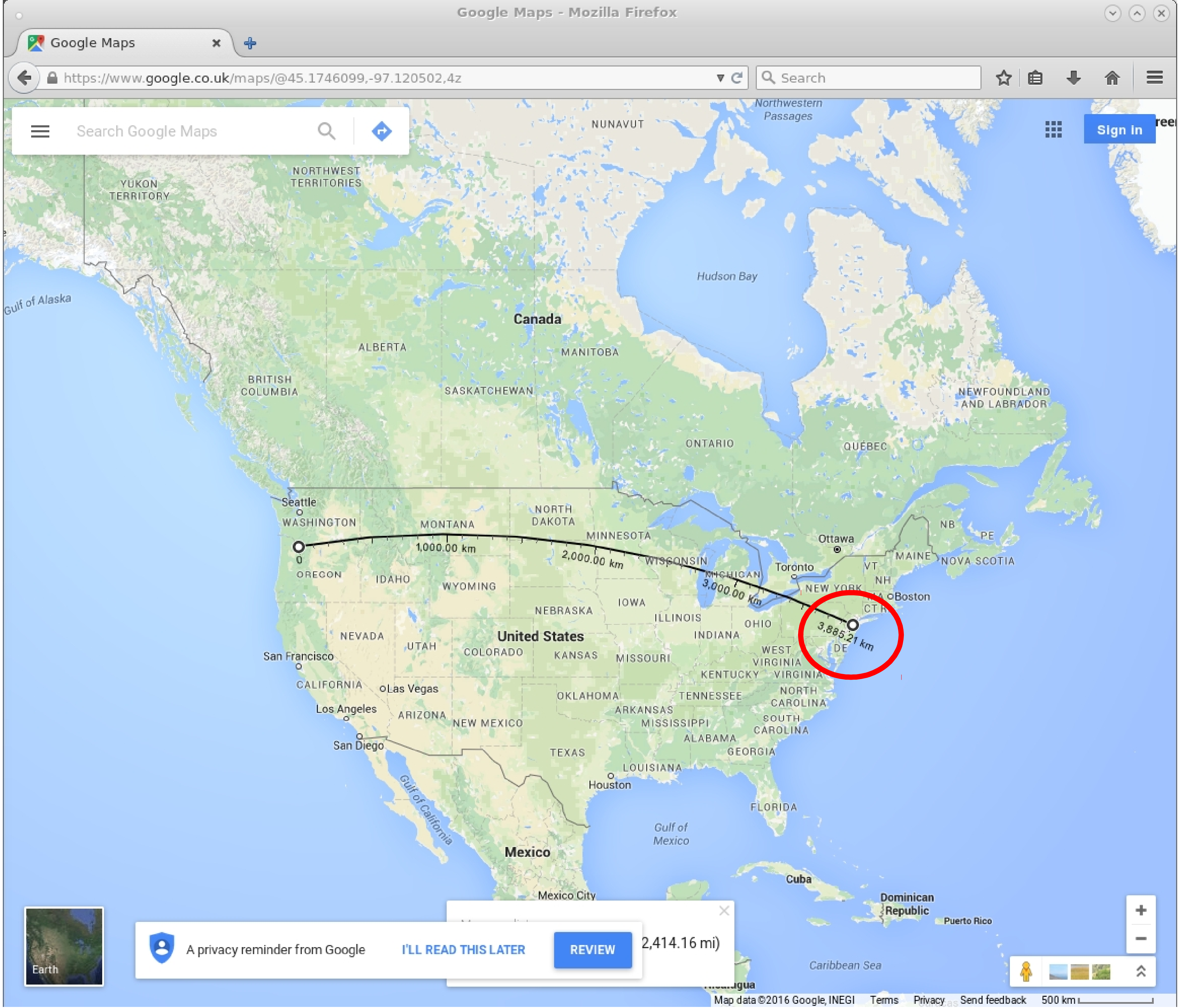
west coast

east coast

```
SELECT
get_distance_in_miles_between_geo_locations(
    45.580557, -122.374776,
    40.578996, -74.27987) as dist;
+----------+
| dist     |
+----------+
| 2414.696 |
+----------+
```

3885.24

Miles to Km
(*1.609)


Google Maps - Mozilla Firefox
Google Maps
https://www.google.co.uk/maps/@45.1746099,-97.120502,4z
Search
Search Google Maps
Sign in
NUNAVUT
Northwestern Passages
YUKON TERRITORY
NORTHWEST TERRITORIES
Gulf of Alaska
Hudson Bay
Canada
ALBERTA
MANITOBA
BRITISH COLUMBIA
SASKATCHEWAN
NEWFOUNDLAND AND LABRADOR
ONTARIO
QUÉBEC
Seattle
WASHINGTON
MONTANA
NORTH DAKOTA
MINNESOTA
Ottawa
NB
PE
MAINE
NOVA SCOTIA
0
1,000.00 km
2,000.00 km
WISCONSIN
MICHIGAN
Toronto
VT
NH
OREGON
IDAHO
WYOMING
3,000.00 km
NEW YORK
Boston
NEBRASKA
IOWA
ILLINOIS
OHIO
3,885.21 km
NEVADA
UTAH
United States
COLORADO
KANSAS
MISSOURI
INDIANA
WEST VIRGINIA
DE
San Francisco
CALIFORNIA
Las Vegas
KENTUCKY
VIRGINIA
OKLAHOMA
TENNESSEE
NORTH CAROLINA
Los Angeles
ARIZONA
NEW MEXICO
ARKANSAS
MISSISSIPPI
SOUTH CAROLINA
San Diego
ALABAMA
GEORGIA
TEXAS
LOUISIANA
Houston
Gulf of California
FLORIDA
Gulf of Mexico
Mexico
Cuba
Mexico City
Dominican Republic
Puerto Rico
Caribbean Sea
Earth
A privacy reminder from Google
I'LL READ THIS LATER
REVIEW
2,414.16 mi)
Map data ©2016 Google, INEGI
Terms
Privacy
Send feedback
500 km

4. R-Trees

→ R-Trees can organize any-dimensional data

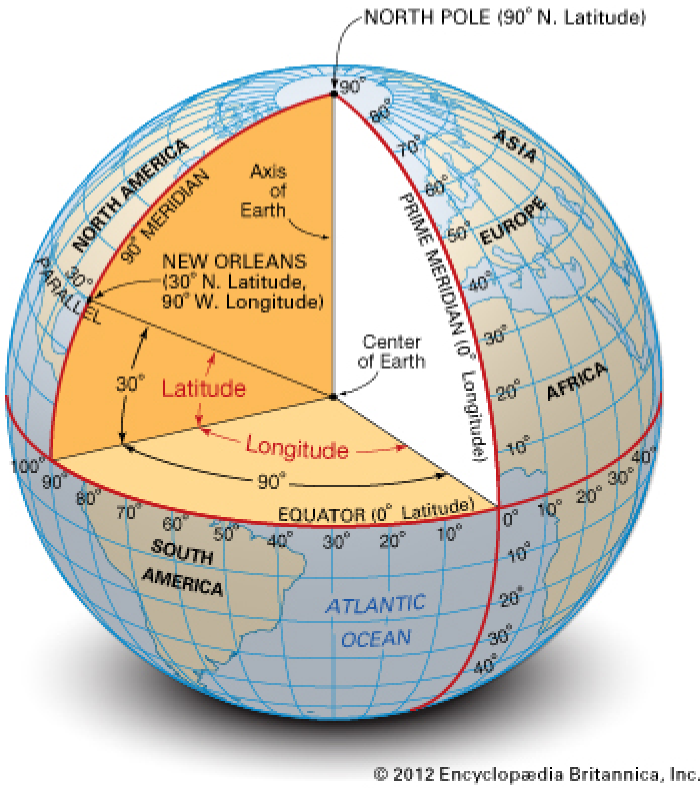
→ data represented by minimum bounding boxes

→ each node bounds it’s children

→ a node can have many objects in it

→ the leaves point to the actual objects (stored on disk probably)

→ the height is always log n (it is height balanced)




```
+------------+-----------+-------------+
| item_id    | latitude  | longitude   |
+------------+-----------+-------------+

...

| 1043747228 | 45.580557 | -122.374776 |
| 1043749860 | 40.578996 |   -74.27987 |
```

west coast

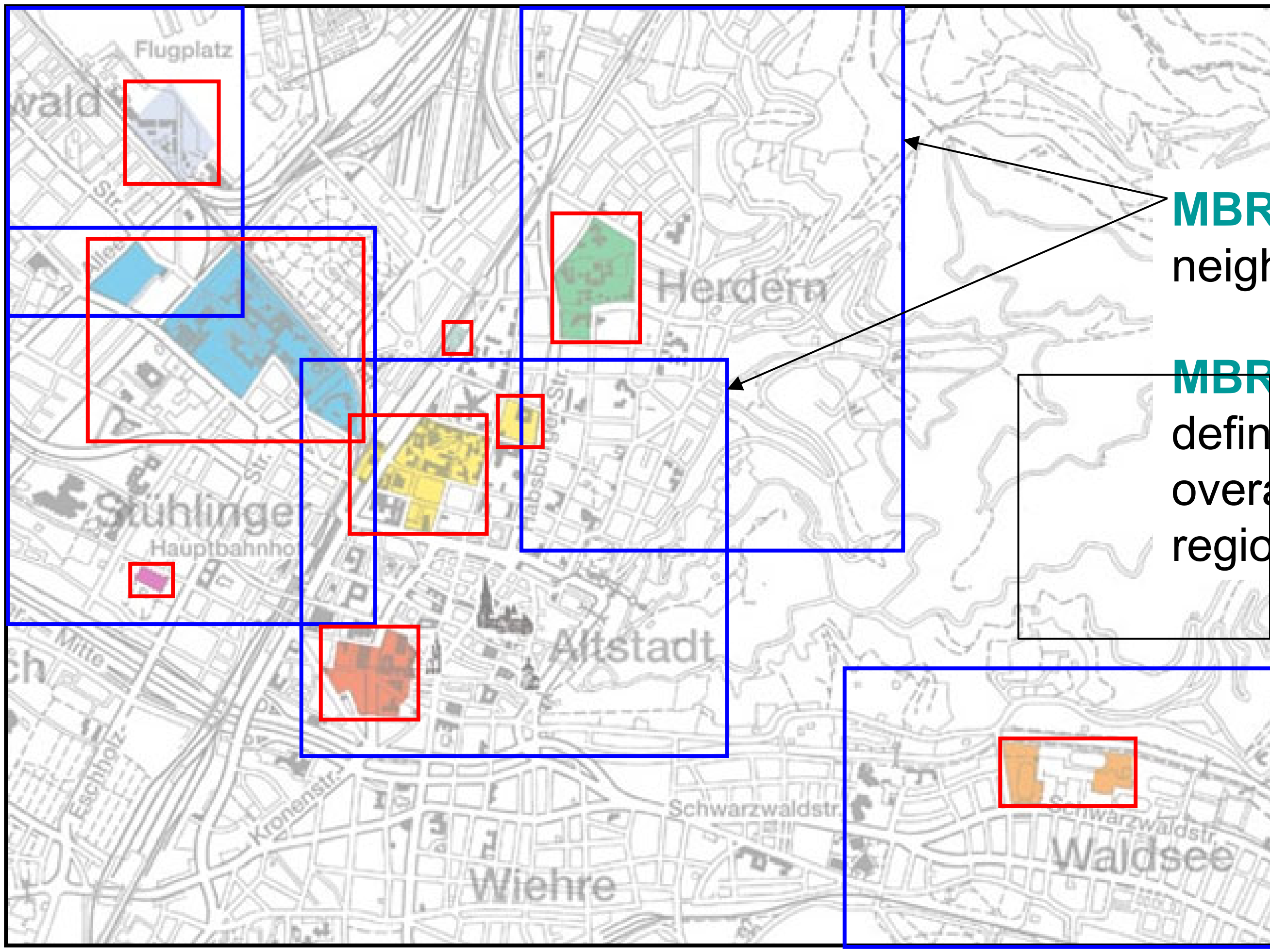
east coast

What you should do for Assignment 2:

→ calculate box **large enough** to include actual values wrt LONG/LAT
→ within the box filter out wrong items using exact formula

4. R-Trees

Consider: Given a city map, ‘index’ all university buildings in an efficient structure for quick relational-topological search.



MBR of the city neighbourhoods.

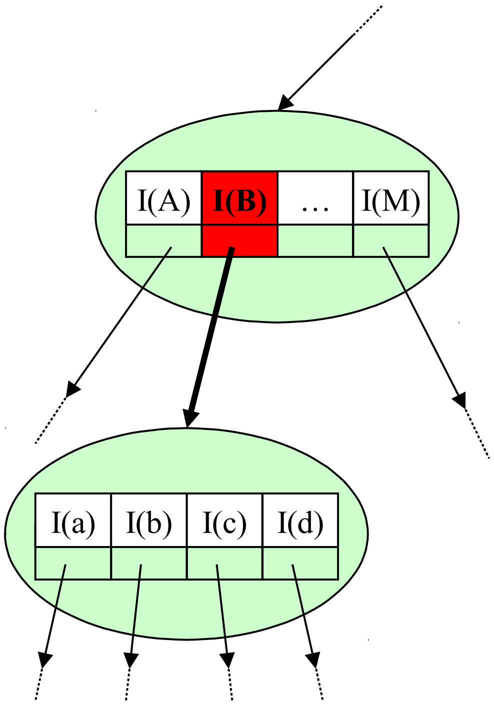
MBR of the city defining the overall search region.

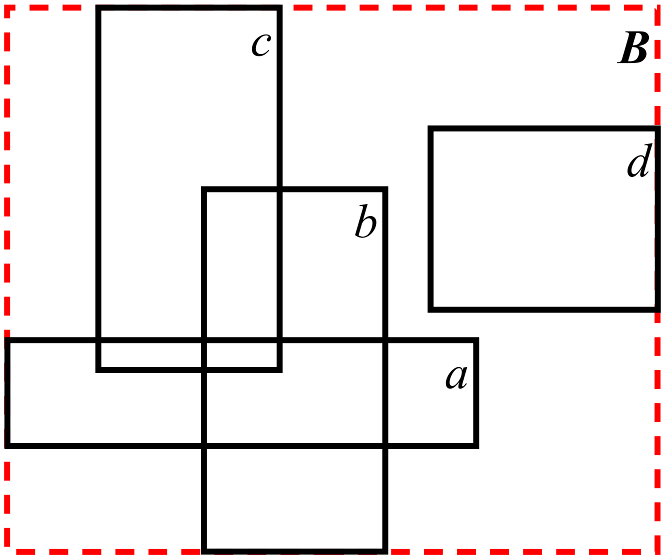
4. R-Trees

An entry *E* in a *non-leaf* node is defined as:

E = (*I*, *child-pointer*)

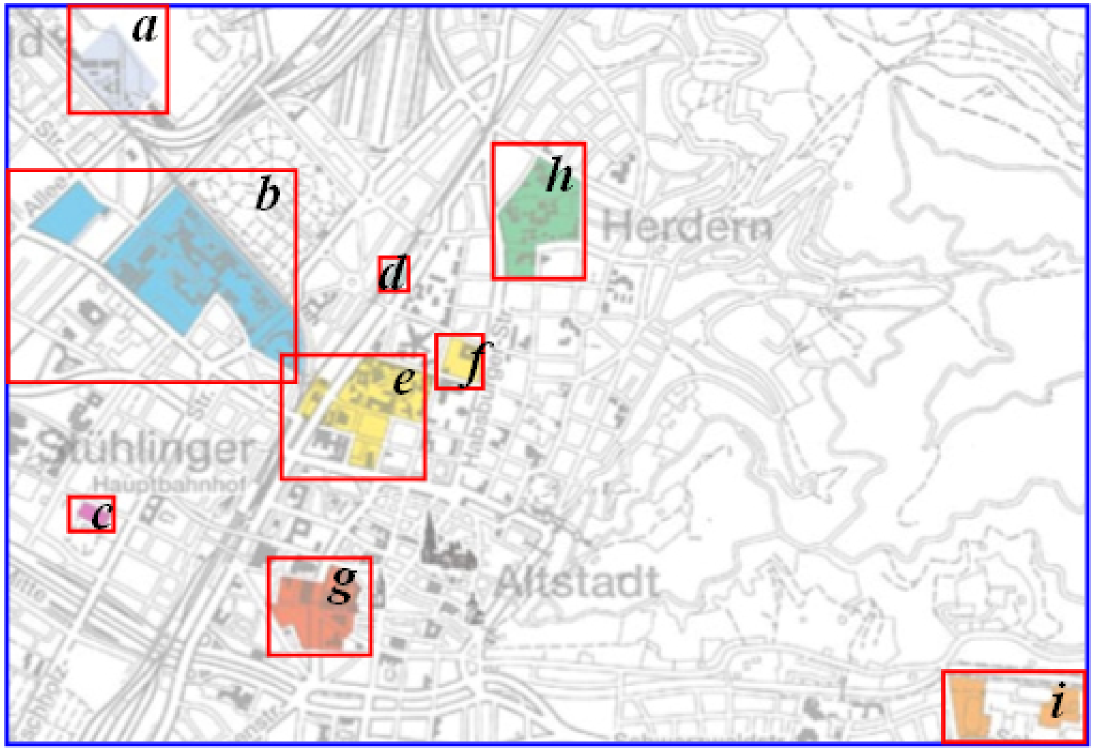
where the *child-pointer* points to the child of this node, and *I* is the MBR that encompasses all the regions in the child-node's pointer's entries.





4. R-Trees

Typical query: Find and report all university building sites that are within 5km of the city centre.



Key:
a: FAW
b: Uni-Klinikum
c: Psychologie
d: Biotechnology
e: Institutsviertel
f: Rektorat
g: Uni-zentrum
h: Biology / Botanischer
i: Sportszentrum

Approach:

i. Build the R-Tree using rectangular regions *a*, *b*, … *i*.
ii. Formulate the query range *Q*.
iii. Query the R-Tree and report all regions overlapping *Q*.

END
Lecture 9