Temporal and Spatial Data Management
Fall 2017

Spatial Databases
SL05

- Introduction
- Modeling spatial concepts
- Organizing the underlying space
- Spatial data types, algebras, and relationships
- Integrating geometry into DBMS data model
- Querying spatial databases
- Spatial indexes
- System architecture

Literature and Acknowledgments

The slides were developed in collaboration with Johann Gamper from the Free University of Bozen-Bolzano and are based on the following material:

- R. H. Güting, An Introduction to Spatial Database Systems, VLDB Journal 3 (1994), 357-399.
- `http://dna.fernuni-hagen.de/gueting/home.html` (cf. tutorials).

Introduction/1

- The **goal** of spatial database research is to extend DBMS data models and query languages to be able to manage geometries/geometric objects.
  - Data structures for geometric shapes
  - Algorithms for performing geometric computations
  - Indexing techniques for multi-dimensional space
  - Extensions of query optimizers
- The main **motivation** for spatial databases in the past was to support geographic information systems (GIS)
  - Early GIS made limited use of DBMS technology
    - Only non-spatial data were stored in a DB; geometries were managed in separate files
  - Modern GIS systems are built on top of DBMS
    - All major commercial systems offer spatial extensions (e.g., Oracle, IBM DB2, Informix)

Introduction/2

- Spatial databases have a **wider scope** and are able to represent other spaces (beside geographical spaces in GIS systems)
  - The layout of a VLSI design
  - A 3D model of the human body
  - A protein structure studied in molecular biology
- An important distinction has to be made between **image databases** and **spatial databases**
  - Image databases manage raster images of space
  - Spatial databases manage objects (with clear location and extent) in space
  - Feature extraction can be used to identify spatial entities within an image that can be stored in a spatial database.

Introduction/3

- **Definition:** A **spatial DBMS**
  - is a DBMS with additional capabilities for handling spatial data
  - offers **spatial data types** in its data model and query language
    - Structure in space, e.g., point, line, region
    - Relationships among them, e.g., intersection
  - supports spatial data types in its implementation
    - Providing at least **spatial indexing** (retrieving objects in particular area without scanning the whole space)
    - Efficient algorithms for **spatial join** (not simply filtering the Cartesian product)

Introduction/4

- Oracle:
  - Oracle Spatial is a **set of functions and procedures** that enables spatial data to be stored, accessed, and analyzed.
  - Oracle Spatial provides a SQL schema (MDSYS) that prescribes the storage, syntax, and semantics of supported geometric data types.
- PostgreSQL, PostGIS:
  - PostGIS adds types (geometry, geography, raster) to the PostgreSQL database. It adds functions, operators, and index enhancements that apply to these spatial types.
- Standard:
  - The OGC (Open Geospatial Consortium) is an organization that works on open standards for the geospatial community.
  - There is a SQL/MM Spatial standard
  - The SQL Multimedia `ST_GEOMETRY` type and the Oracle Spatial `SDO_GEOMETRY` type are essentially interoperable.

Modeling Spatial Concepts/1

- The entities to be represented in a spatial database include anything that might appear on a paper map
- Two alternative views about what needs to be represented
  - **Objects in space:** Distinct entities arranged in space, each of which has its own geometric description
    - e.g., cities, rivers, highway networks, forests, etc.
  - **Space:** Space itself, i.e., to say something about every point in space
    - e.g., thematic maps, land use, partition of a country into districts
- To model these diverse entities the following classes of concepts are distinguished
  - **Single objects**
  - **Spatially related collections of objects**

Modeling Spatial Concepts/2

- Three fundamental abstractions of **single objects**
  - **Point:** An object for which only its location but not its extent is relevant
    - e.g., cities on a large-scale map, hospitals, subway stations
  - **Line (curve):** An entity moving through space or a connection in space
    - e.g., rivers, highways, telepone cables
  - **Region:** An entity that has an extent in the 2D space
    - e.g., countries, forests, lakes
    - May in general have holes and consist of several disjoint pieces

Spatial Types in PostgreSQL

- PostgreSQL offer various geometric types:
  - point: (x,y)
  - lseg: ((x1,y1),(x2,y2))
  - box: ((x1,y1),(x2,y2))
  - path: [(x1,y1),...]
  - polygon: ((x1,y1),...)
  - circle: <(x,y),r>

```
CREATE TABLE cities (
  name VARCHAR(80), location point );

INSERT INTO cities
VALUES ('San Francisco', '(-194.0, 53.0)');

CREATE TABLE t (id INT, area polygon);

INSERT INTO t
VALUES (1, '((2,2),(3,4),(3,6),(1,1))');
```

Spatial Types in PostGIS

- PostGIS is a spatial database extender for PostgreSQL.
- PostGIS adds support for geographic objects.
- PostGIS adds a universal geometry data type.
- Do not mix PosgreSQL and PostGIS.

```
CREATE TABLE shapes (name VARCHAR, geom geometry);

INSERT INTO shapes VALUES
  ('Point', 'POINT(0 0)'),
  ('Linestring', 'LINESTRING(0 0, 1 1, 2 2)'),
  ('Polygon', 'POLYGON((0 0, 1 0, 1 1, 0 0))'),
  ('PolyHole', 'POLYGON((0 0, 10 0, 10 10, 0 0),
                        (1 1, 1 2, 2 2, 1 1))');
```

Spatial Types in Oracle

- Oracle provides a SDO_GEOMETRY object type for spatial objects.
- A geometry is stored as an object, in a single row, in a column of type SDO_GEOMETRY.
- A geometry is an ordered sequence of vertices that are connected by straight line segments or circular arcs.

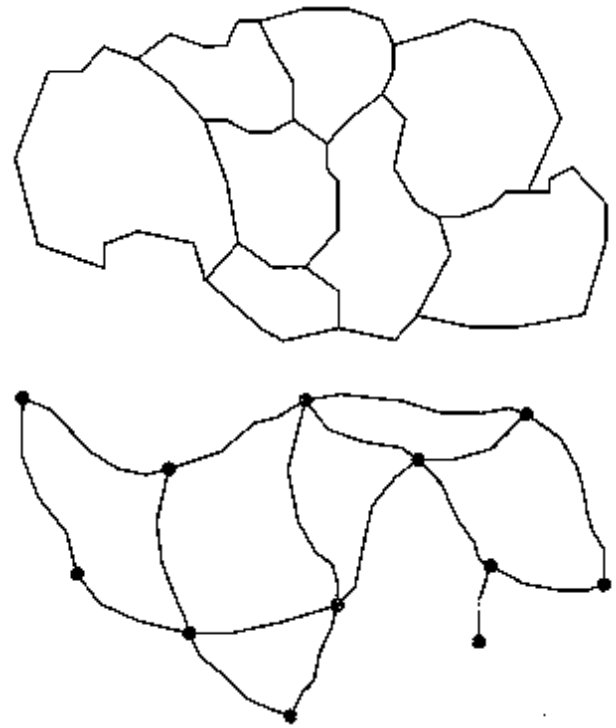
```
CREATE TABLE cola_markets (
  mkt_id NUMBER PRIMARY KEY, name VARCHAR2(32),
  shp SDO_GEOMETRY );

INSERT INTO cola_markets VALUES (
  1, 'cola_a',
  SDO_GEOMETRY(
    2003, -- 2D polygon
    NULL, NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3), -- rectangle (1003 = exterior)
    SDO_ORDINATE_ARRAY(1,1, 5,7) -- LL and UR corner
  ) );

CREATE INDEX cola_spatial_idx
ON cola_markets(shp)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Modeling Spatial Concepts/3

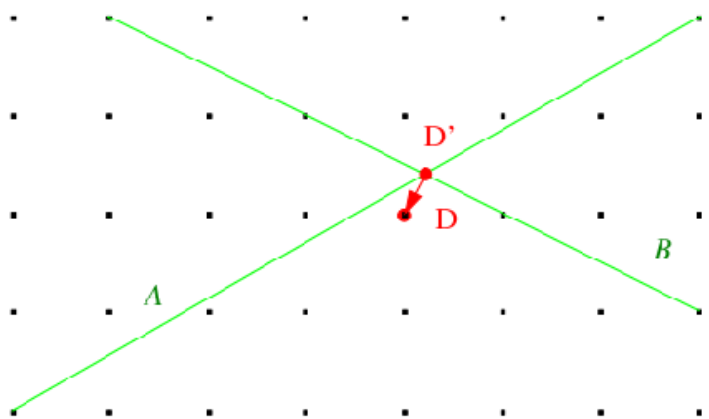
- The two most important abstractions for **spatially related collections of objects**
  - **Partition:** A set of region objects that are required to be disjoint
    - The adjacency relationship is of particular interest, i.e., common boundary of region objects
    - e.g., land use, districts, land ownership
  - **Network:** A graph embedded into the plane, consisting of a set of point objects forming its nodes and a set of line objects describing the geometry of the edges
    - e.g., highways, public transport, power supply lines



- Other intersting abstractions include **nested partition**, e.g., a country partitioned into provinces partitioned into districts, etc.

Organizing the Underlying Space/1

- Euclidean geometry is not suitable as a basis for modeling in spatial databases
  - Euclidean space is **continuous**, i.e., points are represented by a pair of real numbers $p = (x, y) \in \mathbb{R}^2$
  - But computer numbers are **discrete**
    - $\Rightarrow$ space is represented by a discrete raster
- **Example:** Intersection of two lines
  - Intersection point is rounded to the nearest grid point
  - A subsequent test to determine whether the intersection point is on one of the lines yields a false result



Organizing the Underlying Space/2

- Experiment by Cotelo-Lema and Luaces (cf. paper *DualgridFF: a Robust, Consistent and Efficient Physical Data Model for Spatial Databases*, in ACMGIS 2010)
- Evaluation of correctness of set operations in Postgis v1.0.2 (v1.5.1.1).
- N/A indicates a topological exception (Postgis v1.0.2)

	Test	% Wrong answers (Original Postgis)
1	$(A \cap B) - A = \emptyset$	N/A (2.25%)
2	$(A \cap B) - B = \emptyset$	N/A (2.37%)
3	$(A \cap B) \subseteq A$	5.66%
4	$(A \cap B) \subseteq B$	5.97%
5	$A \cup (A \cap B) = A$	N/A (2.25%)
6	$B \cup (A \cap B) = B$	N/A (2.37%)
7	$(A \cap B) \subseteq (A \cup B)$	0%
8	$A \subseteq (A \cup B)$	2.31%
9	$B \subseteq (A \cup B)$	2.31%
10	$disjoint((A - B), (A \cap B))$	0%
11	$disjoint((B - A), (A \cap B))$	0%
12	$disjoint((A - B), B)$	3.48%
13	$disjoint((B - A), A)$	2.31%
14	$(A \cup B) - B = (A - B)$	N/A (3.48%)
15	$(A \cup B) - A = (B - A)$	N/A (2.31%)
16	$(A - B) \subseteq A$	3.08%
17	$B - A \subseteq B$	2.31%
18	$(A \cup B) - (A \cap B) = symdiff(A, B)$	0%
19	$(A - B) \cup (B - A) = symdiff(A, B)$	0%

Organizing the Underlying Space/3

- Approach 1: Definition of a **discrete geometric basis** for modeling as well as for implementation.
  - The goal is to not compute new intersection points within geometric operations.
  - Two approaches for a geometric basis:
    - **Simplicial complexes** (based on combinatorial topology) (Frank & Kuhn 86, Egenhofer, Frank & Jackson 89)
    - **Realms** (Güting & Schneider 93, Schneider 97)
- Approach 2: Deal with numeric imprecision in applications and/or in spatial operations.
  - Rewrite equality on points to distance comparisons with threshold.
  - Oracle uses a tolerance with many spatial operators.

Spatial Data Types and Algebras

- The basic spatial abstractions can be embedded into an existing DBMS data model by using abstract data types
- **Spatial (abstract) data types** encapsulate
  - the structure of a spatial object, e.g., region
  - and operations on it, e.g., predicates, functions, construction operators
- **Spatial algebra:** A collection of spatial data types with related operations.
  - Important properties of an algebra are **closure** under operations and **completeness**.

Spatial Queries in PostgreSQL

- Operators and functions in PostgreSQL:
  - +, translation, box '((0,0),(1,1))' + point '(2.0,0)'
  - #, intersection, '((1,-1),(-1,1))' # '((1,1),(-1,-1))'
  - @>, contains?, circle '((0,0),2)' @> point '(1,1)'
  - #, number of points, # '((1,0),(0,1),(-1,0))'
  - @@, center, @@ circle '((0,0),10)'
  - area(object), area, area(box '((0,0),(1,1))')
  - center(object), point center, center(box '((0,0),(1,2))')
  - circle(box), box to circle, circle(box '((0,0),(1,1))')

```
SELECT *
FROM r
WHERE poly @> '(2, 8)';

SELECT point(area)
FROM t;
```

Spatial Queries in PostGIS

```
SELECT id, ST_AsText(poly)
FROM r
WHERE ST_Contains(poly, ST_GeomFromText('POINT(9 2)'));

SELECT ST_Contains(
         ST_GeomFromText('POLYGON((0 0, 10 10, 10 0, 0 0))'),
         ST_GeomFromText('POINT(0 0)'));
```

```
SELECT name, beer_price,
       distance(location,
                GeometryFromText('POINT(1195722 383854)',2167))
FROM pubs;
```

```
SELECT COUNT(*) AS cnt, p.id AS id
FROM polys p JOIN circles c
ON ST_Contains(c.geom, ST_PointOnSurface(p.geom))
GROUP BY p.id
```

Spatial Queries in Oracle

```
SELECT SDO_GEOMETRY('POINT(-79 37)')
FROM DUAL;
```

```
SELECT name, SDO_GEOM.SDO_AREA(shp, 0.005)
FROM cola_markets;
```

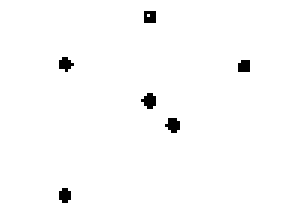
```
SELECT SDO_GEOM.SDO_DISTANCE(b.shp, d.shp, 0.005)
FROM cola_markets b, cola_markets d
WHERE b.name = 'cola_b'
AND d.name = 'cola_d';
```

```
SELECT *
FROM cola_markets
WHERE SDO_OVERLAPS(shp, SDO_GEOMETRY('POINT(-79 37)')) = 'TRUE';
```

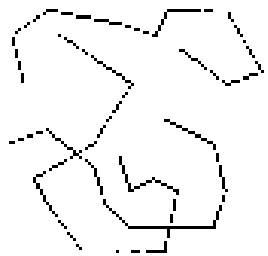
```
SELECT c.shp.Get_Dims()
FROM cola_markets c
WHERE c.name = 'cola_b';
```

ROSE Algebra/1

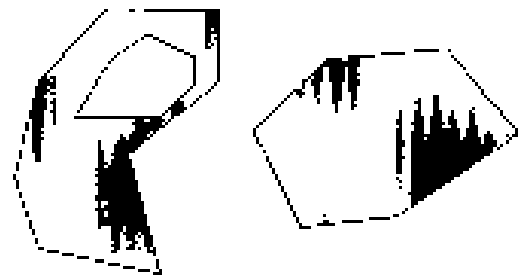
- **ROSE Algebra** (RObust Spatial Extension, Güting & Schneider 95)
  - A spatial algebra with realm-based spatial data types (i.e., objects composed from realm elements)
- **ROSE data types**: points, lines, regions



a points value



a line value



a region value

ROSE Algebra/2

- **ROSE operations**: ROSE provides precisely defined operations
  - Let $GEO = \{points, lines, regions\}$, $EXT = \{lines, regions\}$ and $geo \in GEO$, $ext \in EXT$
- Spatial predicates for topological relationships

inside :	$geo \times regions$	$\rightarrow bool$
intersect, meets :	$ext \times ext$	$\rightarrow bool$
adjacent, encloses :	$regions \times regions$	$\rightarrow bool$

- Operations returning atomic spatial data types

intersection :	$lines \times lines$	$\rightarrow points$
intersection :	$regions \times regions$	$\rightarrow regions$
plus, minus :	$geo \times geo$	$\rightarrow geo$
contour :	$regions$	$\rightarrow lines$

ROSE Algebra/3

- Spatial operators returning numbers

$$\begin{aligned} \textbf{dist}: & \quad geo \times geo \rightarrow real \\ \textbf{perimeter, area}: & \quad regions \rightarrow real \end{aligned}$$

- Spatial operations on set of objects

$$\textbf{sum}: \quad set(obj) \times (obj \rightarrow geo) \rightarrow geo$$

  - `sum` is a spatial aggregate function: Takes a set of objects together with a spatial attribute of the objects of type *geo* and returns the geometric union of all attribute values
  - e.g., form the union of a set of provinces to determine the area of a country

$$\textbf{closest}: \quad set(obj) \times (obj \rightarrow geo) \times geo \rightarrow set(obj)$$

  - The `closest` operator determines within a set of objects those whose spatial attribute value has minimal distance from some other geometric object

Spatial Relationships

- **Spatial relationships** (predicates) are the most important operations offered by a spatial DBMS
  - **Topological relationships:** adjacent, inside, disjoint, etc.
  - **Direction relationships:** above, below, north_of, etc.
  - **Metric relationships:** distance < 100
- Enumeration of all possible topological relationships between two simple regions (no holes, connected)
  - Based on comparing two objects boundaries (δA) and interiors ($\circ A$)
  - 8 of the 16 possible combinations are not valid

$\delta A_1 \cap \delta A_2$	$\delta A_1 \cap \circ A_2$	$\circ A_1 \cap \delta A_2$	$\circ A_1 \cap \circ A_2$	Relationship name
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	A_1 *disjoint* A_2
$\emptyset$	$\emptyset$	$\neq \emptyset$	$\neq \emptyset$	A_2 *in* A_1
$\emptyset$	$\neq \emptyset$	$\emptyset$	$\neq \emptyset$	A_1 *in* A_2
$\neq \emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	A_1 *touch* A_2
$\neq \emptyset$	$\emptyset$	$\emptyset$	$\neq \emptyset$	A_1 *equal* A_2
$\neq \emptyset$	$\emptyset$	$\neq \emptyset$	$\neq \emptyset$	A_1 *cover* A_2
$\neq \emptyset$	$\neq \emptyset$	$\emptyset$	$\neq \emptyset$	A_2 *cover* A_1
$\neq \emptyset$	$\neq \emptyset$	$\neq \emptyset$	$\neq \emptyset$	A_1 *overlap* A_2

Integrating Geometry into the DBMS Data Model

- Spatial datatypes can be embedded in any data model, e.g., the relational data model
- DBMS data model must be extended with SDTs at the level of atomic data types (integer, string)
- **Basic idea**
  - Represent **spatial objects** by objects (of the DBMS data model) with **at least one SDT attribute**.
    - Relational data model: spatial objects are tuples with at least one SDT attribute
    - **Example:** Relational tables to store cities, rivers, and countries

      ```
      cities(name: string, pop: int, loc: points)
      rivers(name: string, route: lines)
      highways(name: string, route: lines)
      states(name: string, area: regions)
      ```
  - Representation of **spatially related collections of objects** (e.g., partitions) as a set of objects with a region attribute
    - Loses some information, e.g., regions are disjoint

Querying Spatial Databases/1

- Connect operations of a spatial algebra to the facilities of a DBMS query language
- Two main issues must be considered
  - Fundamental operations for manipulating sets of database objects
    - spatial selection, spatial join, etc.
  - Graphical input and output

Querying Spatial Databases/2

- **Spatial selection:** Select those objects that satisfy a *spatial predicate* with the query object
  - *All cities in Bavaria?*

```
select sname
from cities c
where c.center inside Bavaria.area
```

  - *All rivers intersecting a query window?*

```
select *
from rivers r
where r.route intersects Qwindow
```

  - *All big cities no more than 100 km from Hagen?*

```
select cname
from cities c
where dist(c.center,Hagen.center) < 100
and c.pop > 500k
```

Querying Spatial Databases/3

- **Spatial join:** Compares any two joined objects based on a predicate on their *spatial attribute values*.
  - *For each river passing through Bavaria, cities within less than 50 km?*

```
select r.rname, c.cname
from rivers as r, cities as c
where r.route intersects Bavaria.area
and dist(r.route,c.area) < 50 km
```

  - *Make a list, showing for each country the number of its neighbor countries?*

```
select s.name, count(*)
from states as s, states as t
where s.area adjacent t.area
group by s.name
```

Querying Spatial Databases/4

- **Spatial function application:** Apply (spatial) functions to each member of a set.
  - *Return the part of river Rhine that is within Germany?*

```
select intersection(r.route, s.area)
from rivers as r, states as s
where r.name = 'Rhine'
and s.name = 'Germany
```

Querying Spatial Databases/5

- **Graphical input and output**
  - While traditional DBMS deal with alphanumeric data (types), some data (types) in spatial DBMS require a graphical representation
    - Input: How to determine "Qwindow" or "Bavaria" in the previous examples?
    - Output: How to show "intersection(route,Bavaria.area)" or "r.route"?
  - The final information to be retrieved is often the result of several queries, the result of which should be graphically overlayed on top of a map
    - Graphical combination of several query results (e.g., add/remove layers, change order of layers)
    - Display of context (e.g., show background such as a raster image or boundary of states)

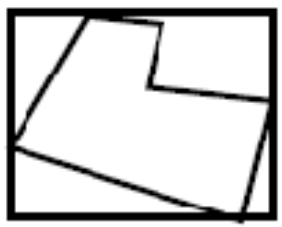
Spatial Indexing/1

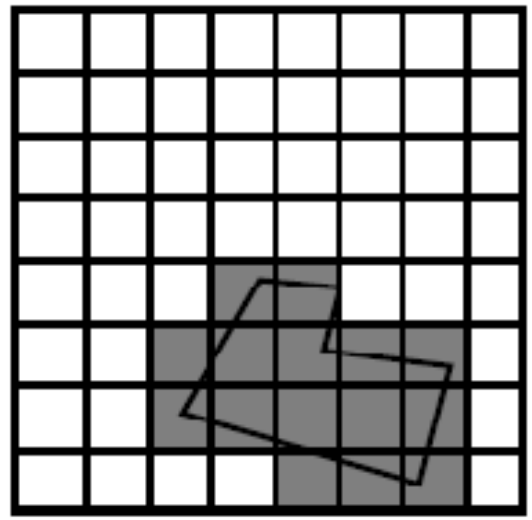
- Mainly used to support spatial selection
  - but supports also other operations, e.g., spatial join or finding the closest object
- A spatial index organizes space and the objects in it in some way so that only parts of the space and a subset of the objects need to be considered to answer a query

- Two main approaches:
  - Map spatial objects to a 1-D space and utilize standard indexing techniques, e.g., B-tree
  - Dedicated spatial index data structures, e.g., R-tree

Spatial Indexing/2

- A fundamental idea of spatial indexing is the use of **approximations**
  - Bounding box approximation
  - Grid approximation





- Leads to a **filter and refine** strategy for query processing
  1. Filter: Returns a set of candidate objects which is a superset of the objects fulfilling a predicate
  2. Refine: For each candidate, the exact geometry is checked

Spatial Indexing/3

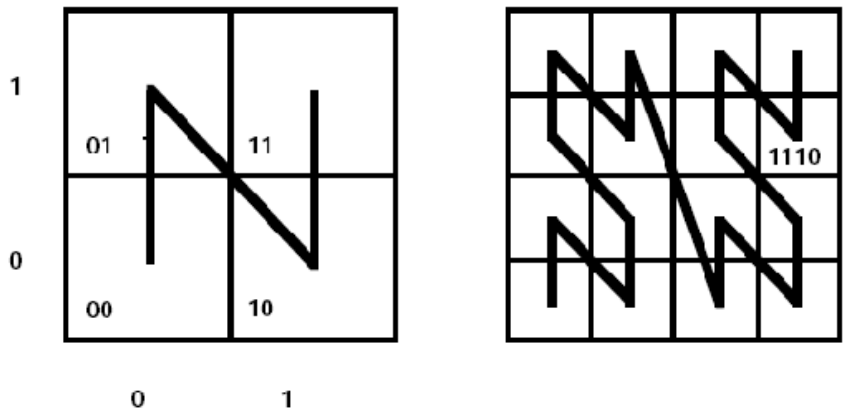
- Most spatial index structures are designed to either store **points** (for point values) or **rectangles** (for line and region values)
- Operations on those structures: insert, delete, check membership
- Typical query types
  - for points:
    - *Range query*: all points within a query rectangle
    - *Nearest neighbor*: point closest to a query point
    - *Distance scan*: enumerate points in increasing distance from a query point
  - for rectangles:
    - *Intersection query*
    - *Containment query*

Spatial Index Structures for Points/1

- **Spatial index structures for points**
  - Data structures for representing points in k dimensions (multi-attribute) have a long tradition, e.g., a tuple $t = (x_1, \ldots, x_k)$
  - Can be used to store geometrical points
  - Problem: index must cluster points that are close to each other
- **1-D embedding: basic idea**
  1. Find a linear order for points such that points close together in space are also close to each other in the linear order
     - Maintains locality
  2. Define this order recursively for a hierarchical subdivision of the space at different granularities.

Spatial Index Structures for Points/2

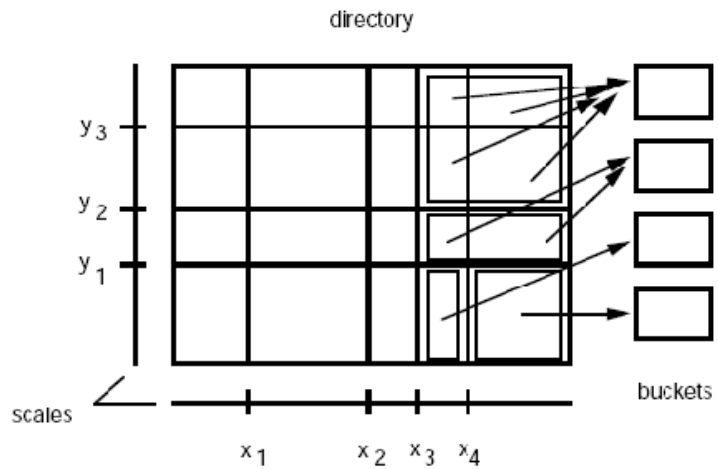
- **Bit-interleaving** is the most popular such order (Morton 1966)
  - Later also called **z-order** (Orenstein and Manola, 1988)



  - Each cell at each level of the hierarchy has an associated bit string whose length corresponds to the level to which the cell belongs.
    - e.g., the top-right cell in the left diagram has bit string 11, on the right-side cell 1110 is shown.
    - The bit-string 1110 is obtained by choosing 11 at the top level, and then 10 within the top level quadrant.
  - The order which is imposed on all cells of a hierarchical subdivision is given by the *lexicographical order of the bit strings.*

Spatial Index Structures for Points/3

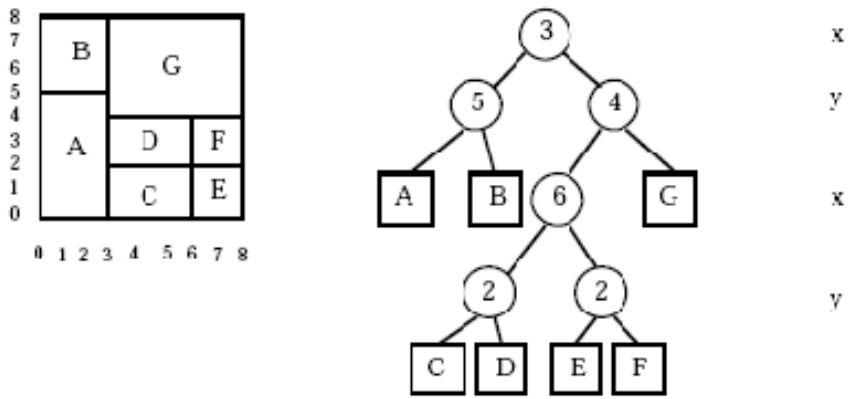
- **GRID index**: Spatial index structure for points (Nievergelt, Hinterberger, and Sevcik 84)
  - The following example partitions the data space into *cells* by an irregular grid



  - The directory is a k-dimensional array whose entries are logical pointers to buckets.
  - All points in a cell are stored in the bucket pointed to by the correpsonding directory entry.
  - The scales are small and are kept in main memory; the directory is on the disk.

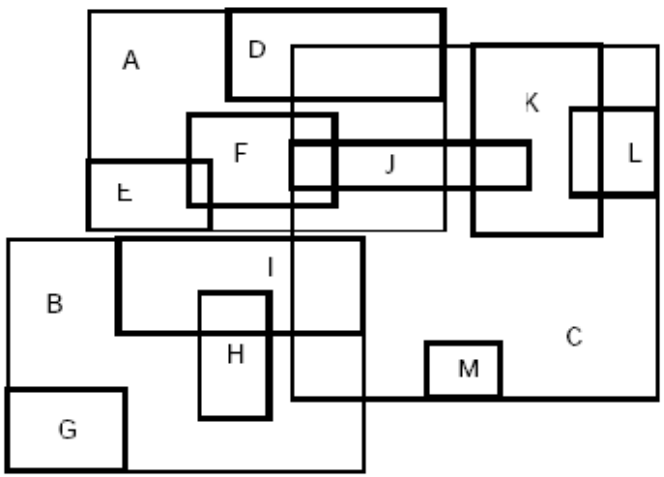
Spatial Index Structures for Points/4

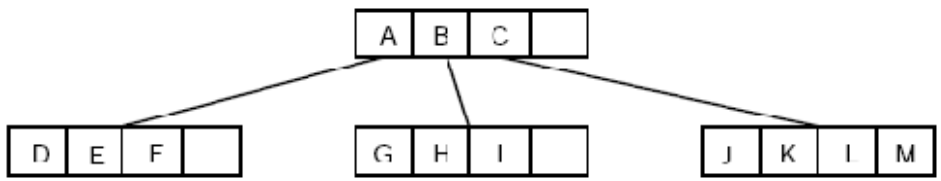
- **kd-Tree** (Bentley 75)
  - Binary tree where each internal node contains a key drawn from one of the k dimensions
  - The key in the root node (level 0) divides the data space with respect to dimension 0, the keys in its children (level 1) divide the two subspaces with repsect to dimension 1, and so forth, up to dimension $k-1$, after which cycling through the dimensions restarts.
  - Leaves contain the points to be stored



Spatial Index Structures for Rectangles/1

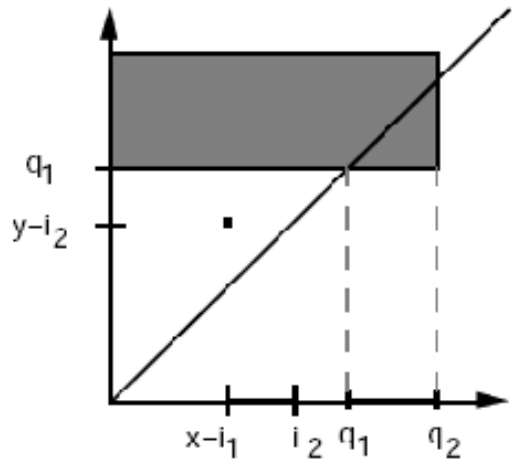
- The **R-tree** (Guttmann 84) groups objects and encloses them into minimum bounding rectangles (MBRs).
- MBRs are organized hierarchically (as a tree) and may overlap.
- Advantage: Spatial object (or key) is in a single bucket
- Disadvantage: Multiple search paths due to overlapping bucket regions





Spatial Index Structures for Rectangles/2

- **Transformation approach**
  - Idea: Consider simple geometric shapes as points in higher dimensional space (the parameter space)
  - k-dimensional rectangles are transformed into $2k$-dimensional points, and a point data structure is used.
  - Rectangle (x_l, x_r, y_b, y_t) can be viewed as a point in 4-D space
- **Example**: Interval $i = [i_1, i_2]$ is mapped to a point (x, y) in 2-D:



  - An intersection query with an interval $q = [q_1, q_2]$ translates to a condition: Find all points (x', y') s.t. $x' < q_2$ and $y' > q_1$.
  - All intervals instersecting q are in the shaded area

Spatial Join/1

- Very active research area in the last few years
- Traditional join methods such as hash join or sort/merge join are not applicable
- Filtering Cartesian product is expensive
- Proposed methods can be classified along the following criteria:
  - Grid approximation/bounding box
  - None/one/both operands are represented in a spatial index structure
- **Grid approximations** with an overlap predicate
  - A parallel scan of two sets of z-elements corresponding to two sets of spatial objects is performed
  - Similar to a merge join

Spatial Join/2

- **Bounding boxes**: For two sets of rectangles R and S all pairs (r, s), $r \in R$ and $s \in S$ such that r intersects s:
  - *No spatial index on R and S*: rectangle intersection algorithm uses a computational geometry algorithm to detect rectangle intersection; external divide and conquer algorithm that is similar to external merge sorting
  - *Spatial index on either R or S*: index join scans the non-indexed operand and for each object, the bounding box of its SDT attribute is used as a search argument on the indexed operand (efficient if non-indexed operand is not too big)
  - *Both R and S are indexed*: synchronized traversal of both structures so that pairs of cells of their repsective partitions covering the same part of space are encountered together.

System Architecture/1

- **Requirements**
  - Representations for the data types of a spatial algebra
  - Procedures for atomic operations (e.g., overlap)
  - Spatial index structures
  - Access operations for spatial indices
  - Filter and refine techniques
  - Spatial join algorithms
  - Cost functions for all these operations (for query optimizer)
  - Statistics for estimating selectivity of spatial selection and join
  - Spatial data types and operations within data definition and query language
  - User interface extensions to handle graphical representation and input of SDT values

System Architecture/2

- Spatial DBMS prototypes built on extensible systems
  - SECONDO (Güting 04)
  - PostgreSQL with PostGIS
  - MONET (Boncz et al. 96)
  - PROBE (Orenstein 86)
- Commercial extensible DBMS with spatial extensions
  - Oracle Spatial
  - IBM DB2 (with Spatial Extender)
  - Informix Universal Server (with Geodetic DataBlade)
  - SQL Server Spatial

Summary

- A spatial DBMS offers capabilities for dealing with spatial data, it offers spatial data types, and it supports spatial data types in its implementation (spatial indexing and spatial join)
- The most important spatial data types are *points*, *lines* and *regions*.
- The issues that arise from the precision of geometric operations must be handled in a principled way.
- Along with spatial data types comes a spatial algebra (i.e., operations on spatial data types) and spatial relationships.
- Spatial indexes are a crucial part of any database systems that supports geographical information: mapping to lower dimensional space, grid file, kd tree, R* tree
- Spatial data types must be integrated into the DBMS at the level of standard data types. This is best achieved with an extensible DBMS.
- In order to query spatial databases we need support for graphical input and output.