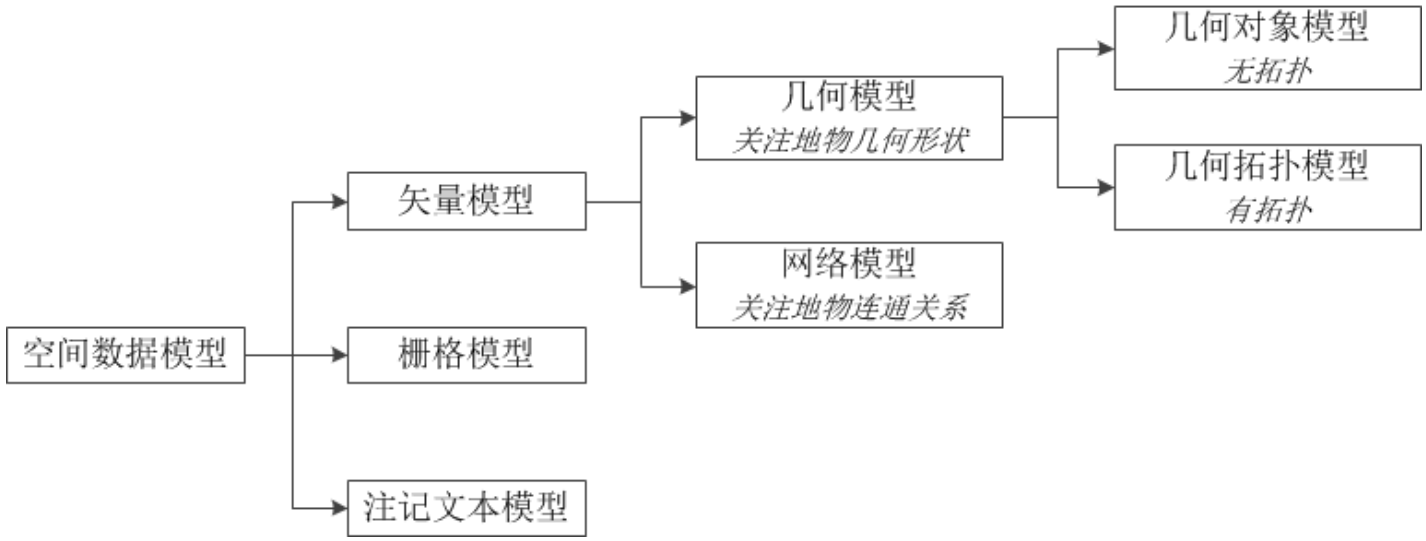
第十二章 空间数据模型

陶煜波
计算机科学与技术学院

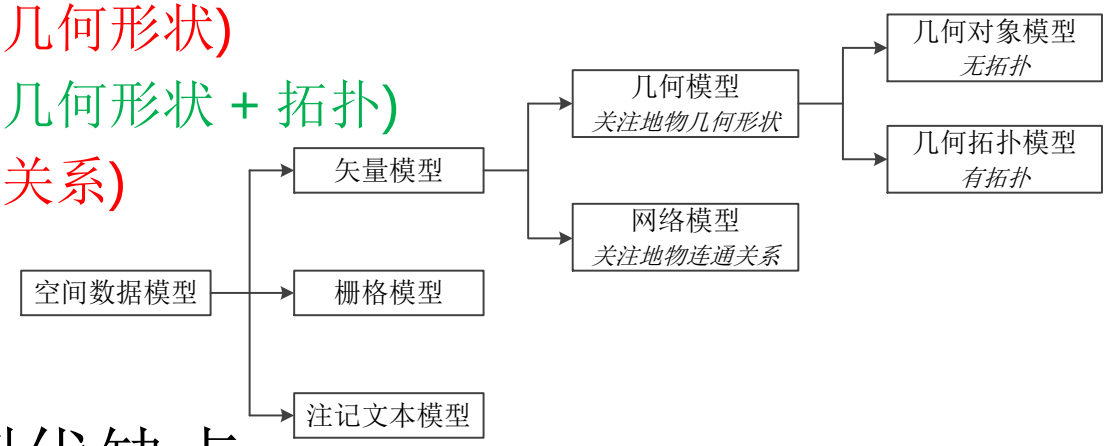
第十二章 空间数据模型

- 12.1 几何对象模型
- 12.2 几何拓扑模型
- 12.3 空间网络模型
- 12.4 基于几何对象模型构建空间网络模型
- 12.5 栅格数据模型 (自学)
- 12.6 注记文字模型



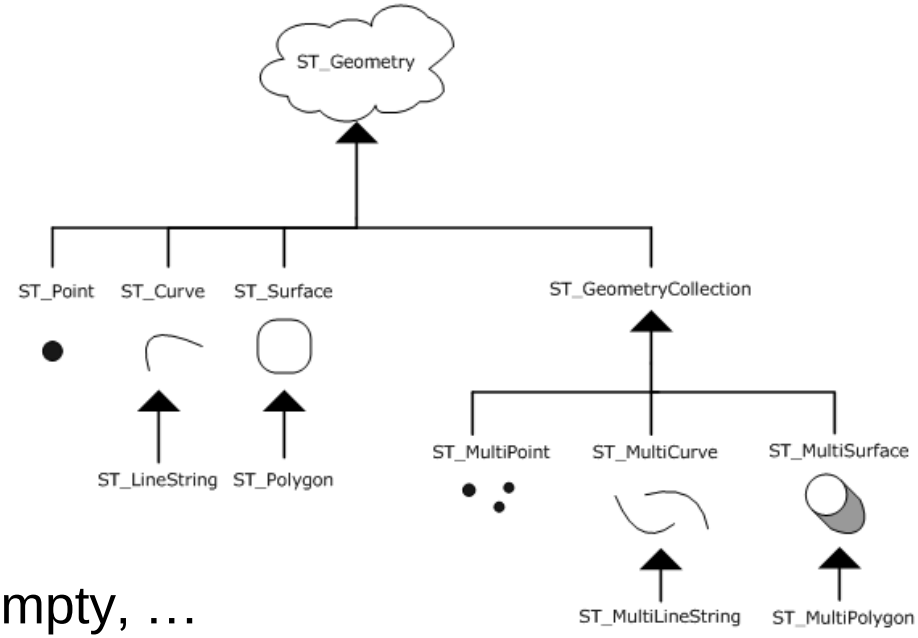
12.1 几何对象模型

- 空间数据模型分类
  - 矢量模型
    - 几何对象模型 (地物几何形状)
    - 几何拓扑模型 (地物几何形状 + 拓扑)
    - 网络模型 (地物连通关系)
  - 栅格模型 (自学)
  - 注记文本模型
- 矢量模型和栅格模型优缺点
- 概念模型 → 逻辑模型 → 物理模型



12.1 几何对象模型

- 对象关系数据库：数据+方法（C++中的类）
- 几何对象层次关系
  - 坐标维数和几何维数
  - 边界、内部、外部
  - 九交矩阵
- 几何对象方法
  - 常规方法 (12种)
    - Dimension, Boundary, IsEmpty, …
  - 常规GIS分析方法 (7种)
    - Distance, Buffer, ConvexHull, Intersection, Union, Difference, SymDifference
  - 空间查询方法 (11种)
    - 包含于(within): 若$a \cap b = a$，且$\mathrm{I}(a) \cap E(b) = \emptyset$，则a包含于b内



12.1 几何对象模型

空间关系	定义	九交矩阵
Equals	$a \subseteq b$，$a \supseteq b$	TFFFTFFFT
Disjoint	$a \cap b = \emptyset$	FF*FF****
Touches	$\mathrm{I}(a) \cap I(b) = \emptyset$，$a \cap b \neq \emptyset$	FT******* F**T***** F***T****
Crosses	$\mathrm{I}(a) \cap I(b) \neq \emptyset$ ， $a \cap b \neq a$， $a \cap b \neq b$	T*T****** (点/线, 点/面, 线/面) 0******** (线/线)
Within	$a \cap b = a$，$\mathrm{I}(a) \cap E(b) = \emptyset$	T*F**F***
Overlaps	Dim(I(a)) = Dim(I(b)) = Dim(I(a) ∩ $I(b)$)，$a \cap b \neq a$，$a \cap b \neq b$	T*T***T** (点/点, 面/面) 1*T***T** (线/线)
Contains	a.Contains(b) ←→ b.Within(a)	?
Intersects	a.Intersects(b) ←→ !b.Disjoint(b)	?

12.1 几何对象模型

- 空间查询方法 (11种)
  - Equals, Disjoint, Intersects, Touches, Crosses, Within, Contains, Overlaps, Relates
  - LocateAlong, LocateBetween
- 空间关系 (8种)
  - 相离(disjoint), 相交(intersects), 相等(equals)
  - 交叠(overlaps), 包含于(within), 包含(contains)
  - 相接(touches), 穿越(crosses)
- Geometry类
  - 几何类构造函数，如ST_GeometryFromText，输入文本，返回几何类，或ST_MakePoint，输入数值，返回几何类
  - 空间关系函数的输入参数为Geometry类

12.1 几何对象模型

- 逻辑模型
  - 基于预定义数据类型的实现
    - numeric和BLOB
  - 基于扩展几何类型的实现
    - Geometry类
- 表模式
  - 系统表
    - GEOMETRY_COLUMNS和SPATIAL_REF_SYS
  - 用户表
    - Feature和Geometry
- 物理模型
  - WKB和WKT

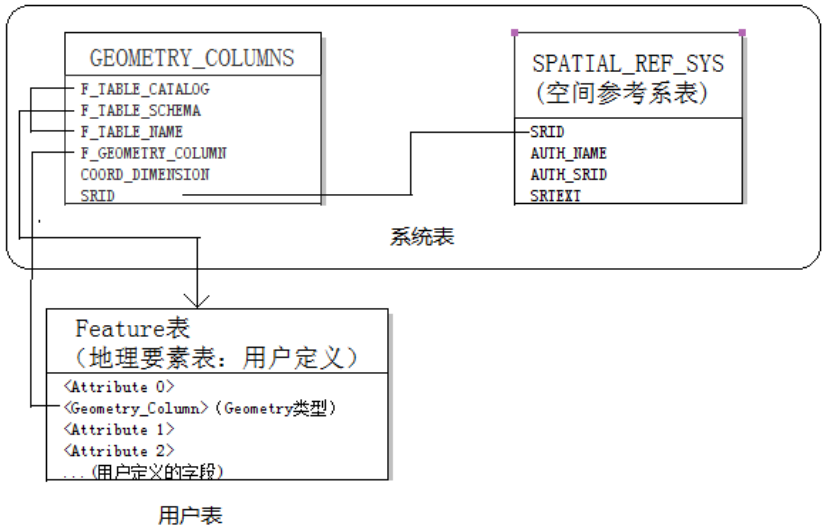


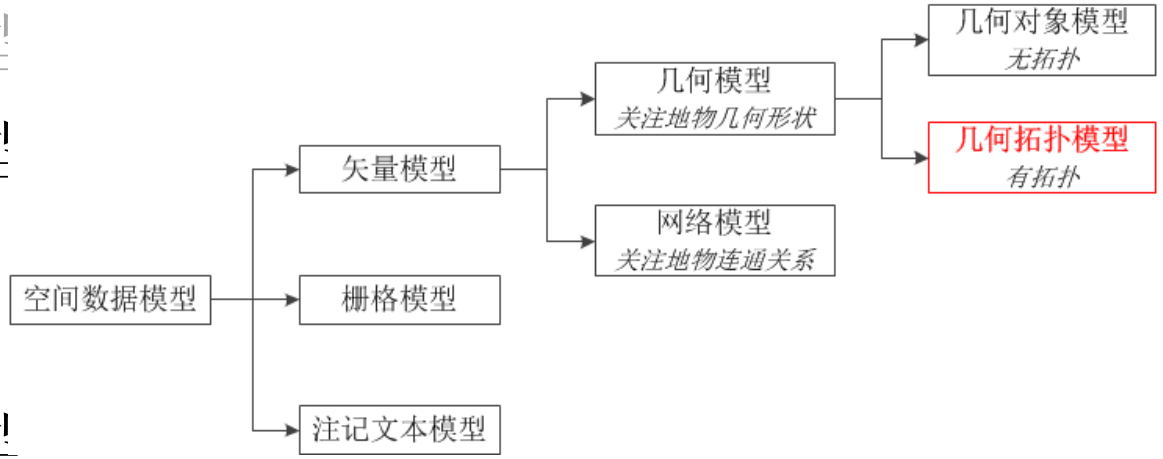
图3-16 基于扩展Geometry类型的要素表模式

12.1 几何对象模型

- 空间数据库 = 对象关系/关系数据库 + 空间扩展
  - Oracle + Oracle Spatial
  - SQL Server + SQL Server Spatial
  - PostgreSQL + PostGIS
  - MySQL + MySQL Spatial
  - SQLite + SQLite Spatialite
- PostGIS
  - 提供了空间数据类型、空间函数和空间索引
  - Geometry (Point/Line/Polygon/Multixxx, 空间参考系)
  - ST_XXX
  - GiST

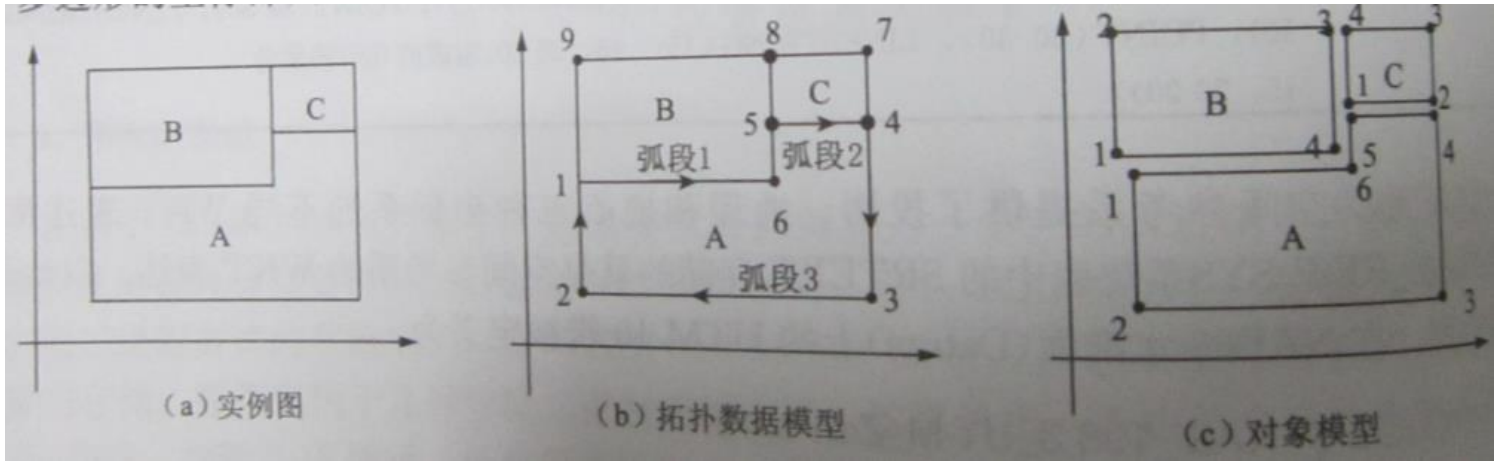
第十二章 空间数据模型

- 12.1 几何对象模型
- 12.2 几何拓扑模型
  - 12.2.1 概念模型
  - 12.2.2 逻辑模型
- 12.3 空间网络模型
- 12.4 基于几何对象模型构建空间网络模型
- 12.5 栅格数据模型 (自学)
- 12.6 注记文字模型



12.2 几何拓扑模型

- 现实世界中，多边形的边界可以用线来描述，而线又可以用点的集合来描述
- 拓扑关系数据模型是以拓扑关系为基础组织和存储各个几何对象
  - 特点是以点、线、多边形间的拓扑连接关系为中心，他们的坐标存储具有依赖关系



12.2 几何拓扑模型

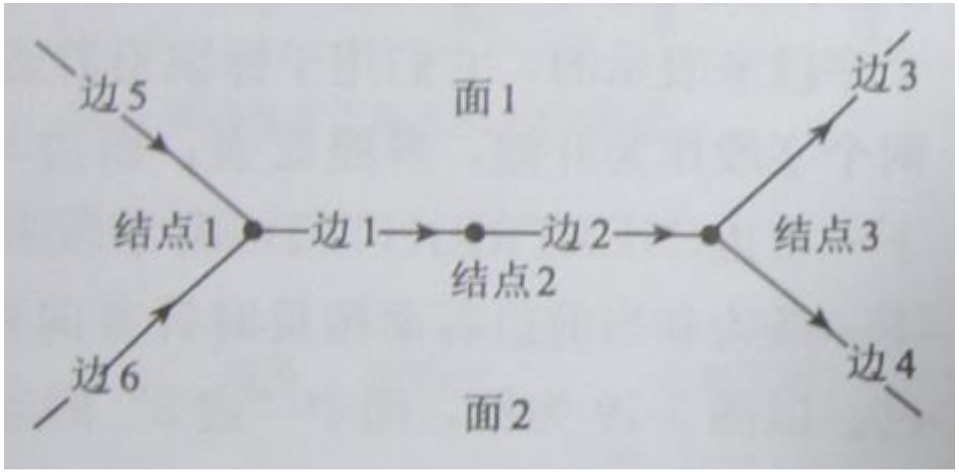
- 优点
  - 数据结构紧凑
  - 拓扑关系明晰
  - 拓扑查询和网络分析效率高
- 数据库厂商通过预先存储的几何对象的拓扑数据结构，提高系统空间查询和分析效率

12.2.1 几何拓扑概念模型

- OGC的抽象数据标准对拓扑概念模型做了定义，但SFA SQL尚未涉及相关定义
- SQL/MM根据ISO的思想定义了几何对象的拓扑数据模型，且Oracle Spatial实现了该拓扑数据模型

12.2.1 几何拓扑概念模型

- 最小拓扑(minitopo)模型
  - 仅采用其中的3个拓扑要素，面(face)、边(edge)和结点(node)



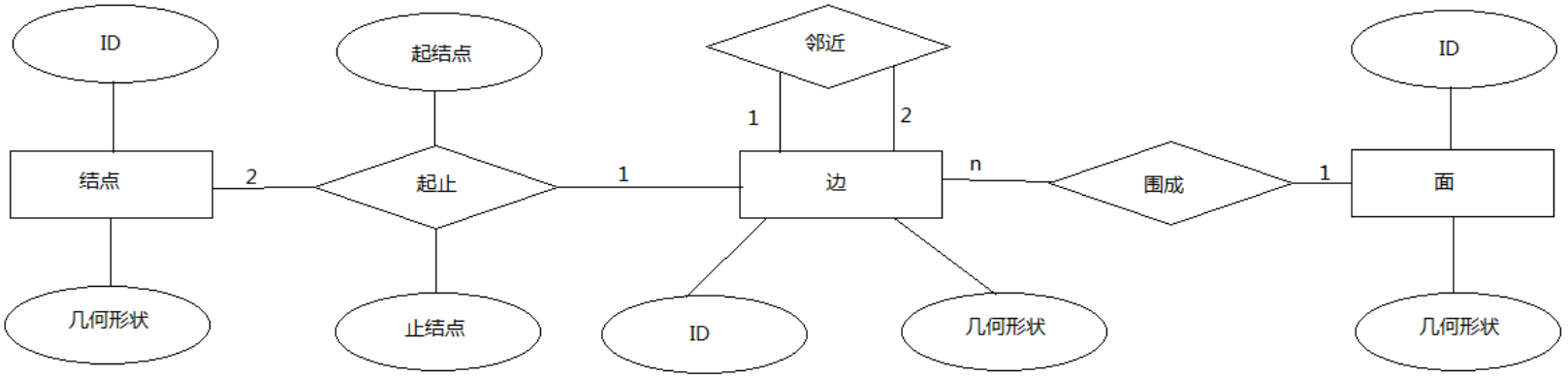
拓扑模型中面、边、结点的概念及关系示意图

12.2.1 几何拓扑概念模型

- 最小拓扑(minitopo)模型
  - 结点：零维拓扑的单形(primitive)，是两个或两个以上的边交汇处的空间点
  - 边：一维拓扑的单形，几何实现是曲线。SQL/MM的边是有向的，其方向由坐标点存储顺序决定
  - 面：二维拓扑单形，几何实现是曲面
  - 上述是相对独立的实体，拥有各自的唯一标识和几何形状
  - 边是联系面和结点的纽带
    - 边和结点是起止关系，面是由边围城

12.2.1 几何拓扑概念模型

- 拓扑模型的E-R图



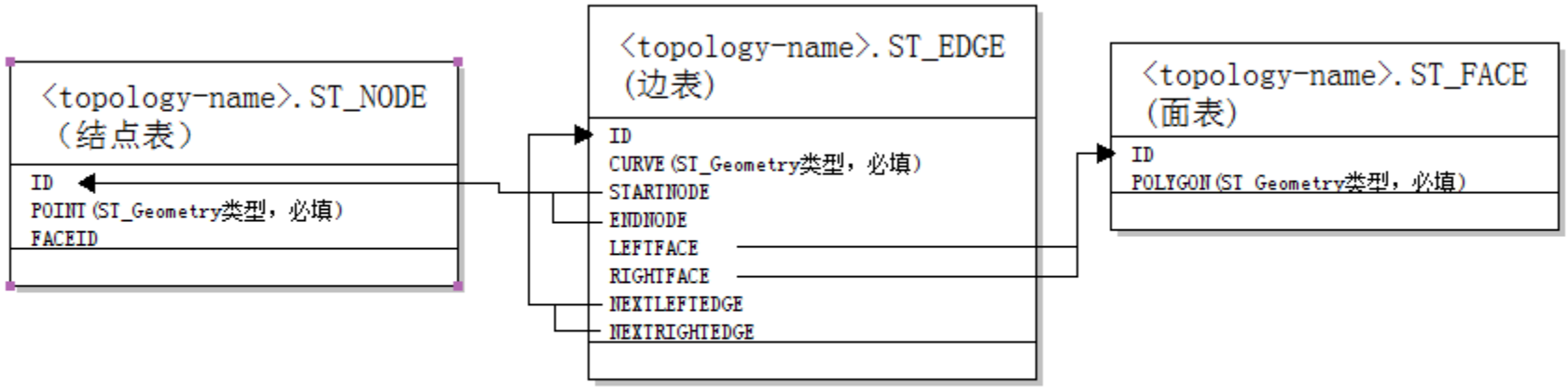
拓扑模型E-R图

12.2.2 几何拓扑逻辑实现

- SQL/MM中所有的表名和几何类型都是冠以”ST_”的前缀
  - 尽管SFA SQL中的表名、类型和函数没有明确冠以”ST_”，但也支持以”ST_”开头的表、类型和函数

12.2.2 几何拓扑逻辑实现

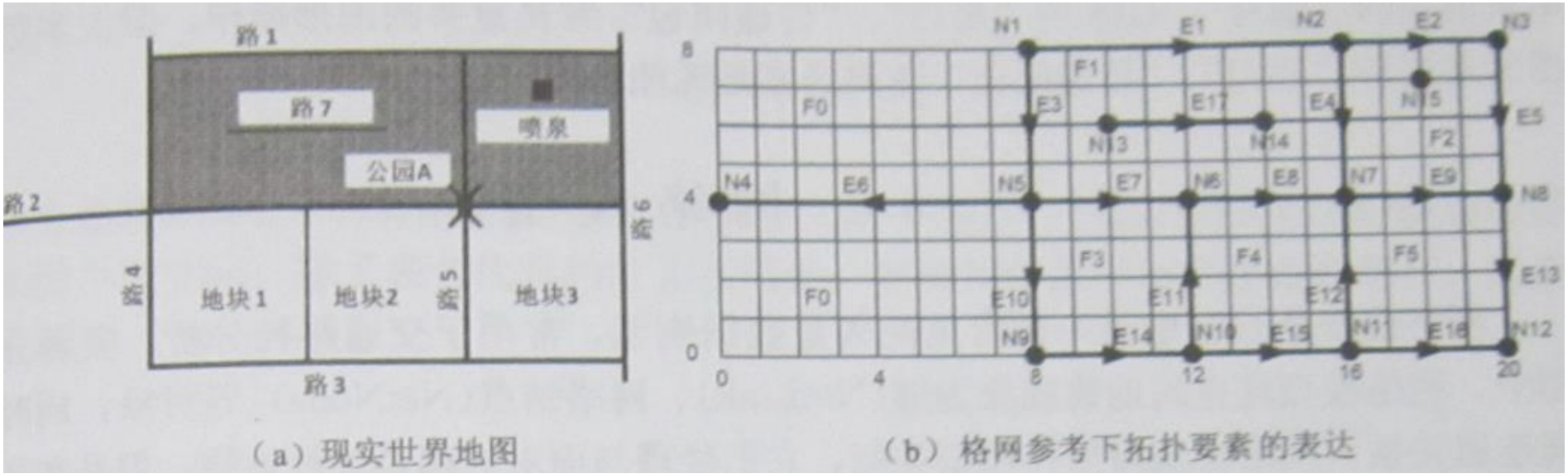
- 结点表ST_<topology-name>_NODE
  - 几何类型是ST_Point，相当于SFA SQL的Point
- 边表的几何类型是ST_Curve
- 面表的几何类型是ST_Polygon



拓扑模型表模式

12.2.2 几何拓扑逻辑实现

- 举例：



（a）现实世界地图　　（b）格网参考下拓扑要素的表达

12.2.2 几何拓扑逻辑实现

- 举例：Node

编号	面编号	几何形状
1		Point(8,8)
2		Point(16,8)
3		Point(20,8)
4		Point(0,4)
5		Point(8,4)
6		Point(12,4)
⋮	⋮	⋮
13		Point(10,6)
14		Point(14,6)
15	2	Point(18,7)

(b)〈topology-name〉.ST_NODE表

12.2.2 几何拓扑逻辑实现

- 举例：Edge

编号	起结点	止结点	下一左边	下一右边	左面	右面	几何形状
1	1	2	2	3	0	1	Line(8 8,16 8)
2	2	3	5	4	0	2	Line(16 8,20 8)
3	1	5	7	1	1	0	Line(8 8,8 4)
4	2	7	9	−1	1	2	Line(16 8,16 4)
5	3	8	13	−2	0	2	Line(20 8,20 4)
6	5	4	−6	−3	0	0	Line(8 4,0 4)
7	5	6	8	10	1	3	Line(8 4,12 4)
8	6	7	−4	−11	1	4	Line(12 4,16 4)
9	7	8	−5	−12	2	5	Line(16 4,20 4)
10	5	8	14	6	3	0	Line(8 4,8 0)
11	10	6	−7	15	3	4	Line(12 0,12 4)
12	11	7	−8	16	4	5	Line(16 0,16 4)
13	8	12	−16	−9	0	5	Line(20 4,20 0)
14	9	10	11	−10	3	0	Line(8 0,12 0)
15	10	11	12	−14	4	0	Line(12 0,16 0)
16	11	12	−13	−15	5	0	Line(16 0,20 0)
17	13	14	−17	17	1	1	Line(10 6,14 6)

(a) (topology-name).ST_EDGE 表

12.2.2 几何拓扑逻辑实现

- 举例：Face

编号	几何形状
0	Polygon((0 0,20 0,20 8,0 8,0 0))
1	Polygon((8 4,16 4,16 8,8 8,8 4))
2	Polygon((16 4,20 4,20 8,16 8,16 4))
⋮	⋮
5	Polygon((16 0,20 0,20 4,16 4,16 0))

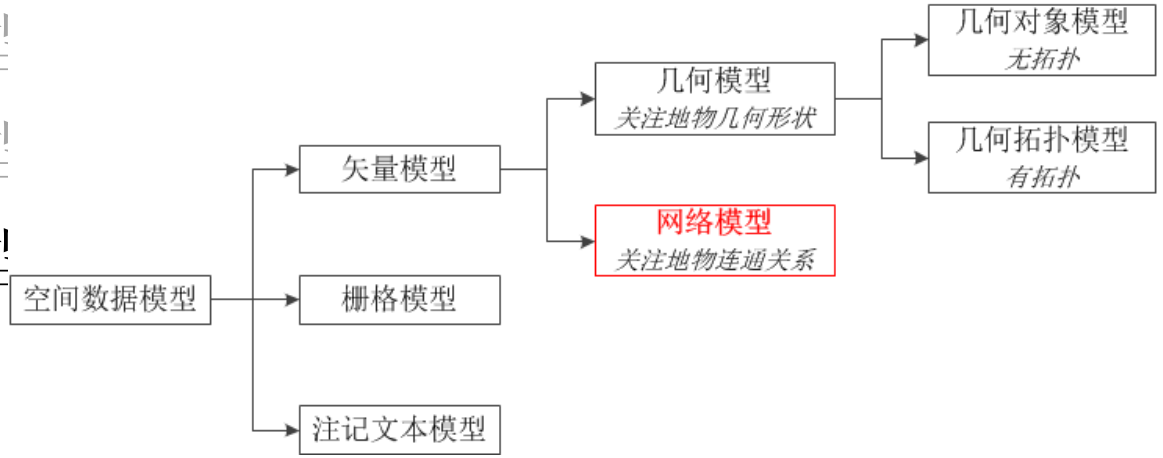
(c)〈topology-name〉. ST_FACE 表

12.2.2 几何拓扑逻辑实现

- 冗余信息处理
  - 维护数据的一致性
  - 如面中加边AddEdgeModFace，拆分边ModEdgeSplit，改变边的几何形体ChangeEdgeGeom
  - 获取拓扑信息，如获得构成某面的所有边GetFaceEdges，获得该面的几何形体GetFaceGeometry
- 自参考不良情况
- 无法实现GIS中“递归”和“传递闭包”查询

第十二章 空间数据模型

- 12.1 几何对象模型
- 12.2 几何拓扑模型
- 12.3 空间网络模型
  - 12.3.1 概念模型
  - 12.3.2 逻辑模型
- 12.4 基于几何对象模型构建空间网络模型
- 12.5 栅格数据模型 (自学)
- 12.6 注记文字模型

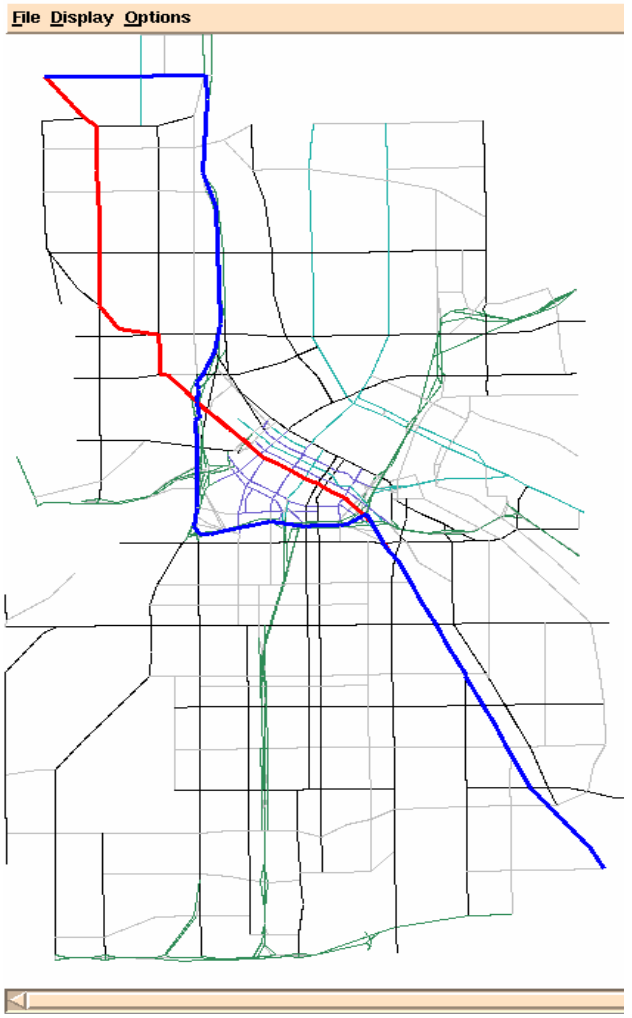


12.3 空间网络模型

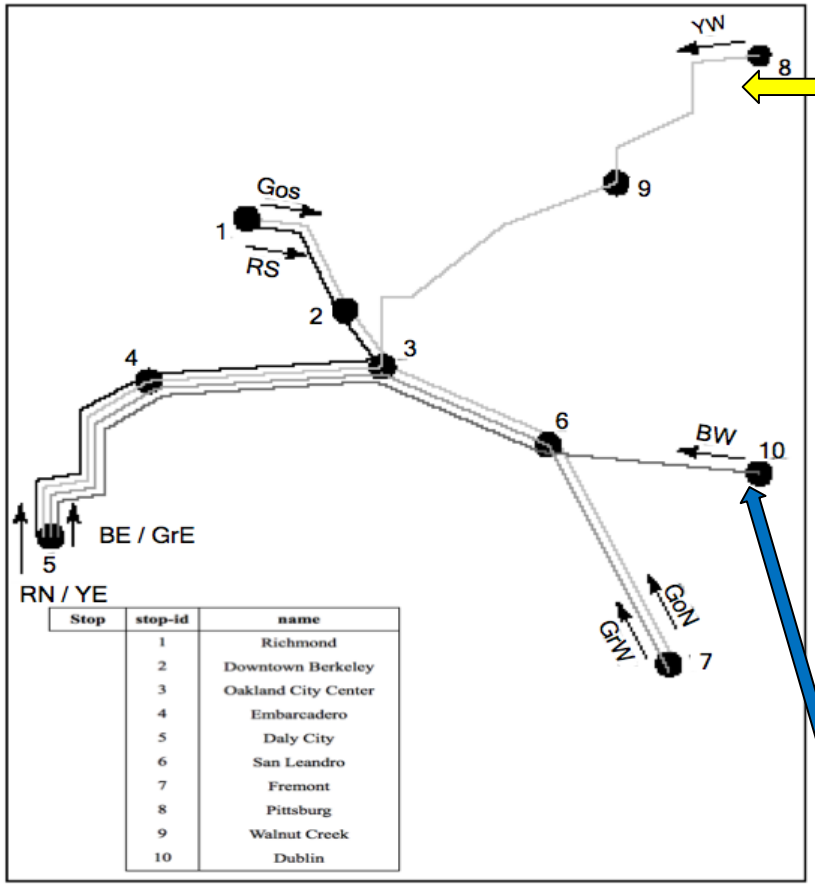
- Location Based Services
  - Location: Where am I ?
  - Directory: What is around me?
  - Routes: How do I get there?
- Spatial Network Analysis
  - Route (A start-point, Destination(s))
  - Allocation (A set of service centers, A set of customers)
  - Site Selection (A set of customers, Number of new service centers)
- Spatial Network Examples
  - Road, Railway, River

Spatial Network Query Example

- Find shortest path from a start-point to a destination
- Find nearest hospital by driving distance
- Find shortest route to deliver packages to a set of homes
- Allocate customers to nearest service center



Railway Network & Queries



Stop	stop-id	name
	1	Richmond
	2	Downtown Berkeley
	3	Oakland City Center
	4	Embarcadero
	5	Daly City
	6	San Leandro
	7	Fremont
	8	Pittsburg
	9	Walnut Creek
	10	Dublin

- Find the number of stops on the Yellow West (YW) route
- List all stops which can be reached from Downtown Berkeley (2)
- List the routes numbers that connect Downtown Berkeley (2) & Daly City (5)
- Find the last stop on the Blue West (BW) route

River Network & Queries

- List the names of all direct and indirect tributaries (支流) of Mississippi river
- List the direct tributaries of Colorado
- Which rivers could be affected if there is a spill in North Platte river

(b) River Network

Data Models of Spatial Networks

- Conceptual Model
  - Information Model: Entity Relationship Diagrams
  - Mathematical Model: Graphs
    - Road (turns), River, Railway
- Logical Data Model
  - Abstract Data types - Transitive Closure
  - Custom Statements in SQL
    - SQL2 - CONNECT clause in SELECT statement
    - SQL3 - WITH RECURSIVE statement
- Physical Data Model
  - Storage-Structures, File-Structures
    - Adjacency matrix, adjacency list, normalized/denormalized tables
    - Minimize CRR
  - Algorithms for common operations

Query Processing for Spatial Networks

- Main memory
  - Connectivity: Breadth first search, depth first search
  - Shortest path: Dijkstra's algorithm, A*
    - Iterate
      - Expand most promising descent node
        - » Dijkstra's: try closest descendent to self
        - » A* : try closest descendent to both destination and self
      - Update current best path to each node, if a better path is found
    - Till destination node is expanded
- Disk-based
  - Connectivity strategies are in SQL3
  - Shortest path - Hierarchical routing algorithm

12.3 空间网络模型

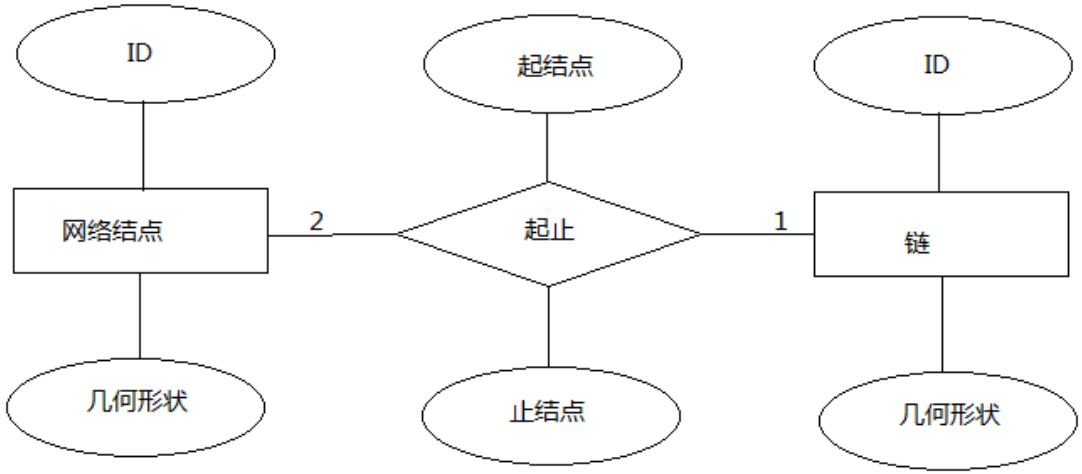
- 网络模型是GIS中另一种常见的矢量数据类型
  - 常用于交通路径分析、资源分配等
  - 将空间地物抽象为链(NetLink)、网络结点(NetNode)等对象，关注其连通性
  - 与拓扑模型相似，处理矢量地物
    - 拓扑模型中地物的精确形状是必不可少的组成部分
    - 网络模型中具体地物之间距离或者阻力的度量是必不可少的组成部分，但地物精确形状则是可选项

12.3 空间网络模型

- 典型应用
  - 交通网络（如陆上、海上及航空线路）
  - 管线与隧道中分析水、汽油及电力的流动
- 实际应用中，用户需要采用面向对象的视角来管理其中的基础设施，又希望做一些网路分析的应用
  - 此时，为了更好地服务用户，几何对象模型和网络模型可能需要同时存在

12.3.1 空间网络概念模型

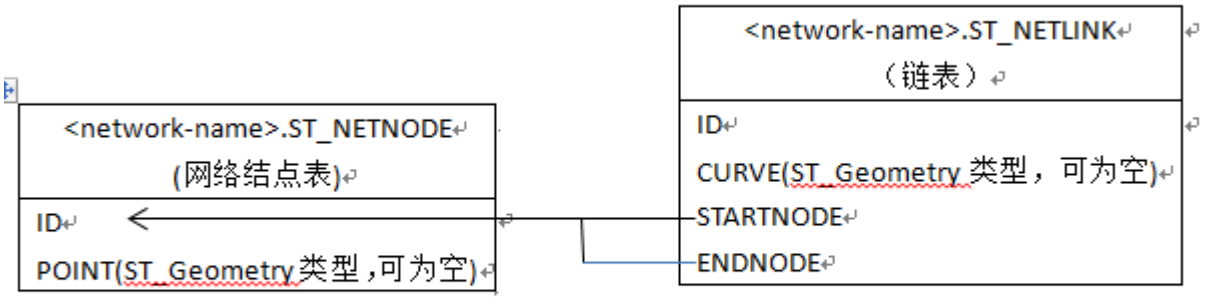
- 网络模型中的链与网络结点之间的连通关系，类似于拓扑模型中边与结点之间的关系
  - 网络模型的几何形状属性是可选项
  - 网络模型中“链”已经是最顶级的单体对象，没有拓扑模型中面实体，以及面边之间的围成关系和边边之间邻接关系



网络模型E-R图

12.3.2 空间网络模型的逻辑实现

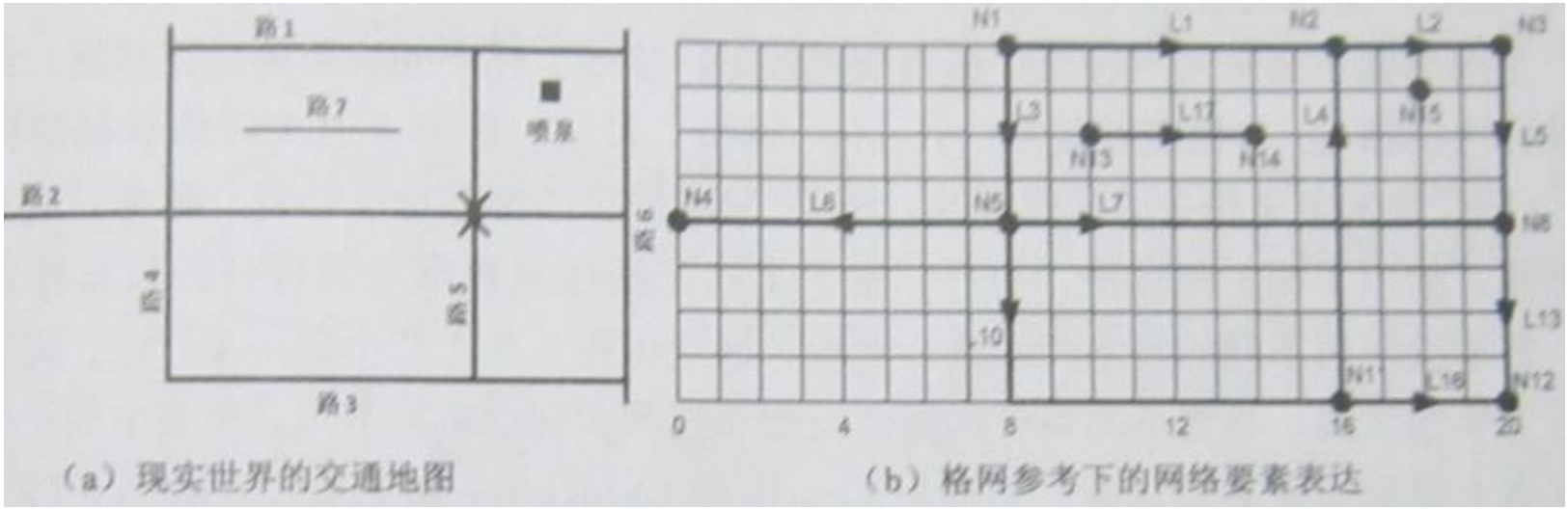
- SQL/MM中网络模型的逻辑实现
  - ST_NETNODE
    - ID
    - Point (ST_Geometry)
  - ST_NETLINK
    - ID
    - Curve (ST_Geometry)
    - StartNode
    - EndNode



网络模型表模式

12.3.2 空间网络模型的逻辑实现

- 交通信息举例：



（a）现实世界的交通地图　　（b）格网参考下的网络要素表达

12.3.2 空间网络模型的逻辑实现

- 交通信息举例：EndNote, NetLink

编号	几何形状
1	Point (8, 8)
2	Point (16, 8)
3	Point (20, 8)
4	Point (0, 4)
5	Point (8, 4)
8	Point (20, 4)
11	Point (16, 0)
12	Point (20, 0)
13	Point (10, 6)
14	Point (14, 6)
15	Point (18, 7)

(a) (network-name).ST_NETNODE 表

编号	起结点	止结点	几何形状
1	1	2	LineString (8 8, 16 8)
2	2	3	LineString (16 8, 20 8)
3	1	5	LineString (8 8, 8 4)
4	11	2	LineString (16 0, 16 8)
5	3	8	LineString (20 8, 20 4)
6	5	4	LineString (8 4, 0 4)
7	5	8	LineString (8 4, 20 4)
10	5	11	LineString (8 4, 8 0, 16 0)
13	8	12	LineString (20 4, 20 0)
16	11	12	LineString (16 0, 20 0)
17	13	14	LineString (10 6, 14 6)

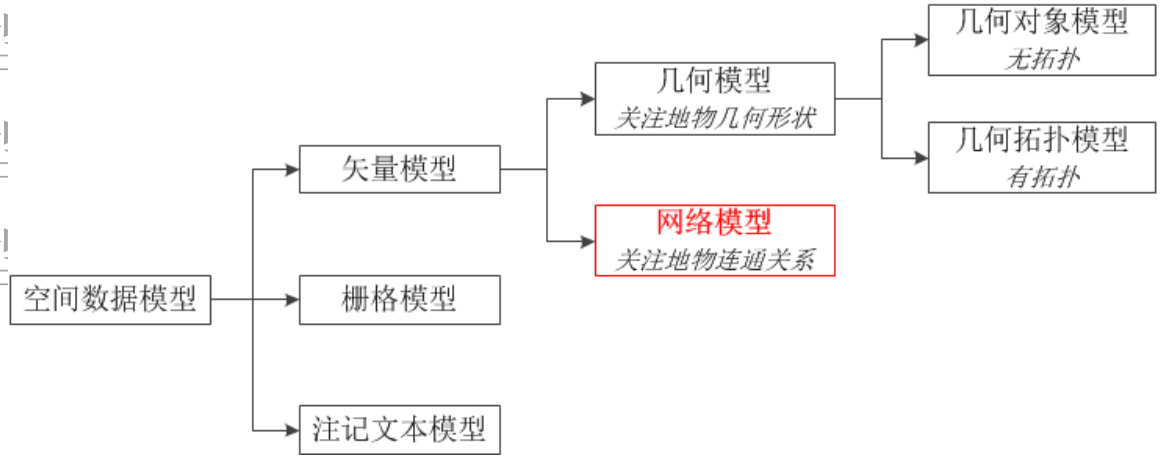
(b) (network-name).ST_NETLINK 表

12.3.2 空间网络模型的逻辑实现

- 网络模型表达和处理能力不足
  - 存在一定的冗余信息，需提供相关函数
    - 初始化网络InitTopoNet
    - 逻辑网络有效性检查ValidLogicalNet
    - 移动结点MoveNode
    - 加链AddLink
    - 改变链的几何形体ChangeLinkGeom
  - 需提供最短路径等基础网络分析函数
    - 无向最短路径
    - 有向最短路径

第十二章 空间数据模型

- 12.1 几何对象模型
- 12.2 几何拓扑模型
- 12.3 空间网络模型
  - 12.3.1 概念模型
  - 12.3.2 逻辑模型



- 12.4 基于几何对象模型构建空间网络模型
- 12.5 栅格数据模型 (自学)
- 12.6 注记文字模型

pgRouting 2.6

- How to create a database and load pgrouting
  - createdb mydatabase
  - psql mydatabase -c “create extension postgis”
  - psql mydatabase -c “create extension pgrouting”
- How to load some data
  - osm2pgrouting - load OSM data into postgresql with pgRouting requirements
  - shp2pgsql - the postgresql shapefile loader
  - ogr2ogr - a vector data conversion utility
  - osm2pgsql - load OSM data into postgresql

http://docs.pgrouting.org/

pgRouting 2.6

- How to build a topology
  - Ends of an edge will be connected to a unique node
  - Select pgr_createTopology('myroads', 0.000001)
- How to check your graph for errors
  - Select prg_analyzeGraph('myroads', 0.000001);
  - Select prg_analyzeOneway('myroads', s_in_rules, s_out_rules, t_in_rules, t_out_rules, direction);
- How to renode if there is any error
  - pgr_nodeNetwork
- How to compute a route
  - Select pgr_<algorithm> (<SQL for edges>, start, end, <additional options>
  - pgr_costResult[] and pgr_geomResult[]

pgRouting Data Types

- pgr_costResult[]
  - seq integer -- sequential ID indicating the path order
  - id1 integer -- typically the node id
  - id2 integer -- typically the edge id
  - cost float8   -- cost attribute
- pgr_costResult3[]
  - seq integer -- sequential ID indicating the path order
  - id1 integer -- typically the path id
  - id2 integer -- typically the node id
  - id3 integer -- typically the edge id
  - cost float8   -- cost attribute

pgRouting Data Types

- pgr_geomResult[]
  - seq integer -- sequential ID indicating the path order
  - id1 integer -- generic name, to be specified by the func
  - id2 integer -- generic name, to be specified by the func
  - geom geometry

pgRouting Topology Functions

- Topology of a network
  - An edge table with source and target attributes
  - A vertices table associated with the edge table
- pgr_createTopology
  - Build a network topology based on the geometry information
  - Parameters
    - edge_table text - Newtork table name
    - tolerance float8 - Snapping tolerance of disconnected edge
    - the_geom text - Geometry colum name of the network table
    - id text - Primary key column name of the network table
    - source text - Source column name of the network table
    - target text - Target column name of the network table
    - rows_where text - Condition to SELECT a subset or rows

pgRouting Topology Functions

- pgr_createTopology
  - Builds a network topology based on the geometry information
  - The edge_table will be affected
    - The source column values will change
    - The target column values will change
    - An index will be created, if it doesn't exists, to speed up the process to the following columns
      - id, the_geom, source, target
  - OK after the network topology has been built
    - Creates a vertices table: <edge_table>_vertices_pgr
    - Fills id and the_geom columns of the vertices table
    - Fills the source and target columns of the edge table referencing the id of the vertices table

pgRouting Topology Functions

- pgr_createTopology
  - Builds a network topology based on the geometry information
  - The edge_table will be affected
  - OK after the network topology has been built
  - FAIL when the network topology was not built due to an error
    - A required column of the Network table is not found or is not of the appropriate type
    - The condition is not well formed
    - The names of source, target or id are the same
    - The SRID of the geometry could not be determined

pgRouting Topology Functions

- pgr_createTopology
  - The Vertices Table
    - id bigint - Identifier of the vertex.
    - cnt integer - Number of vertices in the edge_table that reference this vertex
    - chk integer - Indicator that the vertex might have a problem
    - ein integer - Number of vertices in the edge_table that reference this vertex AS incoming
    - eout integer - Number of vertices in the edge_table that reference this vertex AS outgoing
    - the_geom geometry - Point geometry of the vertex
- prg_createVerticesTable
  - Reconstructs the vertices table based on the source and target information

pgRouting Topology Functions

- pgr_analyzeGraph
  - Analyze the network topology
  - The edge table to be analyzed must contain
    - a source column and a target column filled with id's of the vertices of the segments
    - the corresponding vertices table <edge_table>_vertices_pgr that stores the vertices information

```
varchar pgr_analyzeGraph(text edge_table, double precision tolerance,
                  text the_geom:='the_geom', text id:='id',
                  text source:='source',text target:='target',text rows_where:='true')
```

- pgr_analyzeOneway
  - Analyzes oneway streets and identifies flipped segments

pgRouting Topology Functions

- pgr_analyzeGraph
  - Analyzes the network topology
    - edge_table: text Network table name. (may contain the schema name as well)
    - tolerance: float8 Snapping tolerance of disconnected edges. (in projection unit)
    - the_geom: text Geometry column name of the network table. Default value is the_geom
    - id: text Primary key column name of the network table. Default value is id
    - source: text Source column name of the network table. Default value is source
    - target: text Target column name of the network table. Default value is target
    - rows_where: text Condition to select a subset or rows. Default value is true to indicate all rows

pgRouting Topology Functions

- pgr_analyzeGraph
  - Analyzes the network topology
  - OK after the analysis has finished
    - Uses the vertices table: <edge_table>_vertices_pgr
    - Fills compeletly the cnt and chk columns of the vertices table
    - Returns the analysis of the section of the network defined by rows_where
  - FAIL when the analysis was not completed due to an error
    - The vertices table is not found
    - A required column of the Network table is not found or is not of the appropriate type
    - The condition is not well formed
    - The names of source, target or id are the same
    - The SRID of the geometry could not be determined

pgRouting Topology Functions

- pgr_nodeNetwork
  - Node an network edge table
  - This function reads the edge_table table, that has a primary key column id and geometry column named the_geom and intersect all the segments in it against all the other segments and then creates a table edge_table_noded
  - It uses the tolerance for deciding that multiple nodes within the tolerance are considered the same node

pgRouting Topology Functions

- pgr_nodeNetwork
  - Node an network edge table
  - edge_table: text Network table name. (may contain the schema name as well)
  - tolerance: float8 tolerance for coincident points (in projection unit)
  - id: text Primary key column name of the network table. Default value is id
  - the_geom: text Geometry column name of the network table. Default value is the_geom
  - table_ending: text Suffix for the new table's. Default value is noded

pgRouting Topology Functions

- pgr_nodeNetwork
  - Nodes an network edge table
  - The output table will have for edge_table_noded
  - id: bigint Unique identifier for the table
  - old_id: bigint Identifier of the edge in original table
  - sub_id: integer Segment number of the original edge
  - source: integer Empty source column to be used with pgr_createTopology function
  - target: integer Empty target column to be used with pgr_createTopology function
  - the_geom: geometry Geometry column of the noded network

pgRouting Routing Functions

- pgr_apspJohnson
  - All Pairs Shortest Path, Johnson's Algorithm
- pgr_apspWarshall
  - All Pairs Shortest Path, Floyd-Warshall Algorithm
- pgr_astar
  - Shortest Path A*
- pgr_bdAstar
  - Bi-directional A* Shortest Path

pgRouting Routing Functions

- pgr_bdDijkstra
  - Bi-directional Dijkstra Shortest Path
- pgr_dijkstra
  - Shortest Path Dijkstra
- pgr_kDijkstra
  - Mutliple destination Shortest Path Dijkstra
- pgr_ksp
  - K-Shortest Path
- pgr_tsp
  - Traveling Sales Person
- pgr_trsp
  - Turn Restriction Shortest Path (TRSP)

pgRouting With Driving Distance Enabled

- pgr_drivingDistance
  - Returns the driving distance from a start node
- pgr_alphashape
  - Core function for alpha shape computation
- pgr_pointsAsPolygon
  - Draws an alpha shape around given set of points

Dijkstra最短路径算法

- Dijkstra 1 to 1
  - pgr_dijkstra(text edges_sql, bigint start_vid, bigint end_vid, boolean directed:=true)
- Dijkstra many to 1
  - pgr_dijkstra(text edges_sql, array[ANY_INTEGER] start_vids, bigint end_vid, boolean directed:=true)
- Dijkstra 1 to many
  - pgr_dijkstra(text edges_sql, bigint start_vid, array[ANY_INTEGER] end_vids, boolean directed:=true);
- Dijkstra many to many
  - pgr_dijkstra(text edges_sql, array[ANY_INTEGER] start_vids, array[ANY_INTEGER] end_vids, boolean directed:=true)

Dijkstra最短路径算法

- Description of the parameters the SQL query
  - edges_sql: a SQL query
  - select id, source, target, cost [, reverse_cost] from edge_table
    - id: ANY-INTEGER identifier of the edge
    - source: ANY-INTEGER identifier of the first end point vertex of the edge
    - target: ANY-INTEGER identifier of the second end pont vertex of the edge
    - cost: ANY-NUMERICAL weight of the edge (source, target), if negative: edge (source, target) does not exist, therefore it's not part of the graph
    - reverse_cost: ANY-NUMERICAL (optional) weight of the edge (target, source), if negative: edge (target, source) does not exist, therefore it's not part of the graph

Dijkstra最短路径算法

- Description of the parameters of the signatures
  - edges_sql: a SQL query
  - start_vid: BIGINT identifier of the starting vertex of the path
  - start_vids: array[ANY-INTEGER] array of identifiers of starting vertices
  - end_vid: BIGINT identifier of the ending vertex of the path
  - end_vids: array[ANY-INTEGER] array of identifiers of ending vertices
  - directed: boolean (optional). When false the graph is considered as Undirected. Default is true which considers the graph as Directed

Dijkstra最短路径算法

- Description of the return values

Returns set of (seq [, start_vid] [, end_vid] , node, edge, cost, agg_cost)

- seq: INT isequential value starting from 1
- path_seq: INT relative position in the path. Has value 1 for the begining of a path
- start_vid: BIGINT id of the starting vertex. Used when multiple starting vetrices are in the query
- end_vid: BIGINT id of the ending vertex. Used when multiple ending vertices are in the query
- node: BIGINT id of the node in the path from start_vid to end_v.
- edge: BIGINT id of the edge used to go from node to the next node in the path sequence. -1 for the last node of the path
- cost: FLOAT cost to traverse from node using edge to the next node in the path sequence
- agg_cost: FLOAT total cost from start_v to node

Dijkstra最短路径算法

- pgr_costResult[] pgr_dijkstra(text sql, interger source, integer target, boolean directed:=true)

```
SELECT * FROM pgr_dijkstra(
        'SELECT id, source, target, cost, reverse_cost FROM edge_table',
        2, 3
);
 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
   1 |        1 |    2 |    4 |    1 |        0
   2 |        2 |    5 |    8 |    1 |        1
   3 |        3 |    6 |    9 |    1 |        2
   4 |        4 |    9 |   16 |    1 |        3
   5 |        5 |    4 |    3 |    1 |        4
   6 |        6 |    3 |   -1 |    0 |        5
(6 rows)


SELECT * FROM pgr_dijkstra(
        'SELECT id, source, target, cost, reverse_cost FROM edge_table',
        2, 5
);
 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
   1 |        1 |    2 |    4 |    1 |        0
   2 |        2 |    5 |   -1 |    0 |        1
(2 rows)
```

Example

- create table Road (
  id serial primary key,
  name text,
  geom geometry(LineString, 4326));

- insert into Road(name, geom) values
- ('A', ST_GeomFromText('LineString(0 20, 10 20)', 4326));
- ('B', ST_GeomFromText('LineString(10 20, 10 2)', 4326));
- ('C', ST_GeomFromText('LineString(10 2, 20 2)', 4326));
- ('D', ST_GeomFromText('LineString(0 5, 20 5)', 4326));
- ('E', ST_GeomFromText('LineString(20 4.9999, 20 2.0001)', 4326));

Example

- create table Road_network (
  id serial primary key,
  name text,
  source int,
  target int,
  geom geometry(LineString, 4326),
  len float);

- insert into Road_network(name, geom) select name, geom from Road;

- select pgr_createTopology(‘road_network', 0.00001, 'geom', 'id', 'source', 'target', 'true');

- select pgr_analyzeGraph(‘road_network', 0.00001, 'geom', 'id', 'source', 'target', 'true');

Example

- select pgr_nodeNetwork(‘road_network', 0.00001, the_geom:='geom', id:='id', table_ending:='1');
- select pgr_createTopology(‘road_network_1', 0.001, 'geom', 'id', 'source', 'target', 'true');
- select pgr_analyzeGraph(‘road_network_1', 0.001, 'geom', 'id', 'source', 'target', 'true');

Example

- create table temp (
  id serial primary key,
  name text,
  source int,
  target int,
  geom geometry(LineString, 4326));

- insert into temp(name, source, target, geom)
  select O.name, N.source, N.target, N.geom from Road_network O, Road_network_1 N where O.id = N.old_id order by O.name;

- delete from Road_network;

- insert into Road_network(name, source, target, geom)
  select name, source, target, geom from temp;

Example

- update Road_network
  set len = ST_LENGTH(geom);

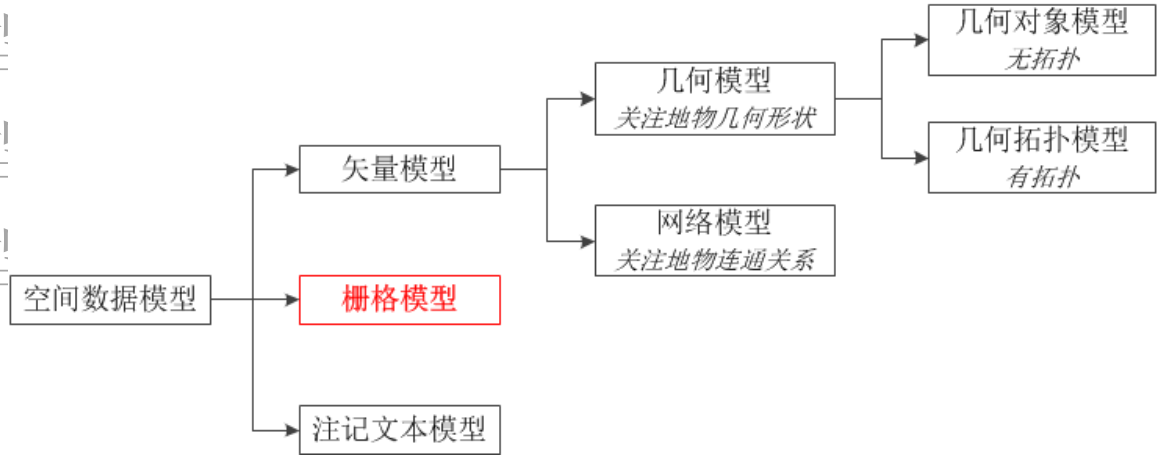
- SELECT seq, id1 AS node, id2 AS edge, cost
  FROM pgr_dijkstra(
  'SELECT id, source, target, len as cost FROM Road_network',
  5, 3, false
  );

Example

- 扩展
  - 新增加路F，从(0,5)到(10,20)
  - 查询(12,6)到(10,20)的最短距离
  - 查询(12,6)到(18,4)的最短距离
  - 限制单行，只能从(10,5)到(0,5)

第十二章 空间数据模型

- 12.1 几何对象模型
- 12.2 几何拓扑模型
- 12.3 空间网络模型
  - 12.3.1 概念模型
  - 12.3.2 逻辑模型



- 12.4 基于几何对象模型构建空间网络模型
- 12.5 栅格数据模型 (自学)
  - 12.5.1 概念模型
  - 12.5.2 逻辑与物理模型
  - 12.5.3 影像金字塔技术
- 12.6 注记文字模型

12.5 栅格数据模型

- 栅格数据
  - 采用一组笛卡尔平面(一个二维数组)来描述空间对象的性质
- 栅格数据分类
  - GIS栅格分析中能够涉及的栅格数据
  - 遥感影像数据
  - 图片数据

12.5 栅格数据模型

- 栅格数据分类
  - GIS栅格分析中能够涉及的栅格数据
    - 如土壤pH分布数据，数字高程模型(DEM)数据
    - 每个栅格单元仅有对应的一项属性值，整数或浮点数
  - 遥感影像数据
    - 数字正投影图(DOM)
    - 每个栅格单元通常对应多个属性值，记录现实世界中对应区域对不同波段光谱的反射值，整数或浮点数
  - 图片数据
    - 数码照片，数字栅格地图(DRG)
    - 每个栅格单元通常对应3个属性值，值为0-255

12.5 栅格数据模型

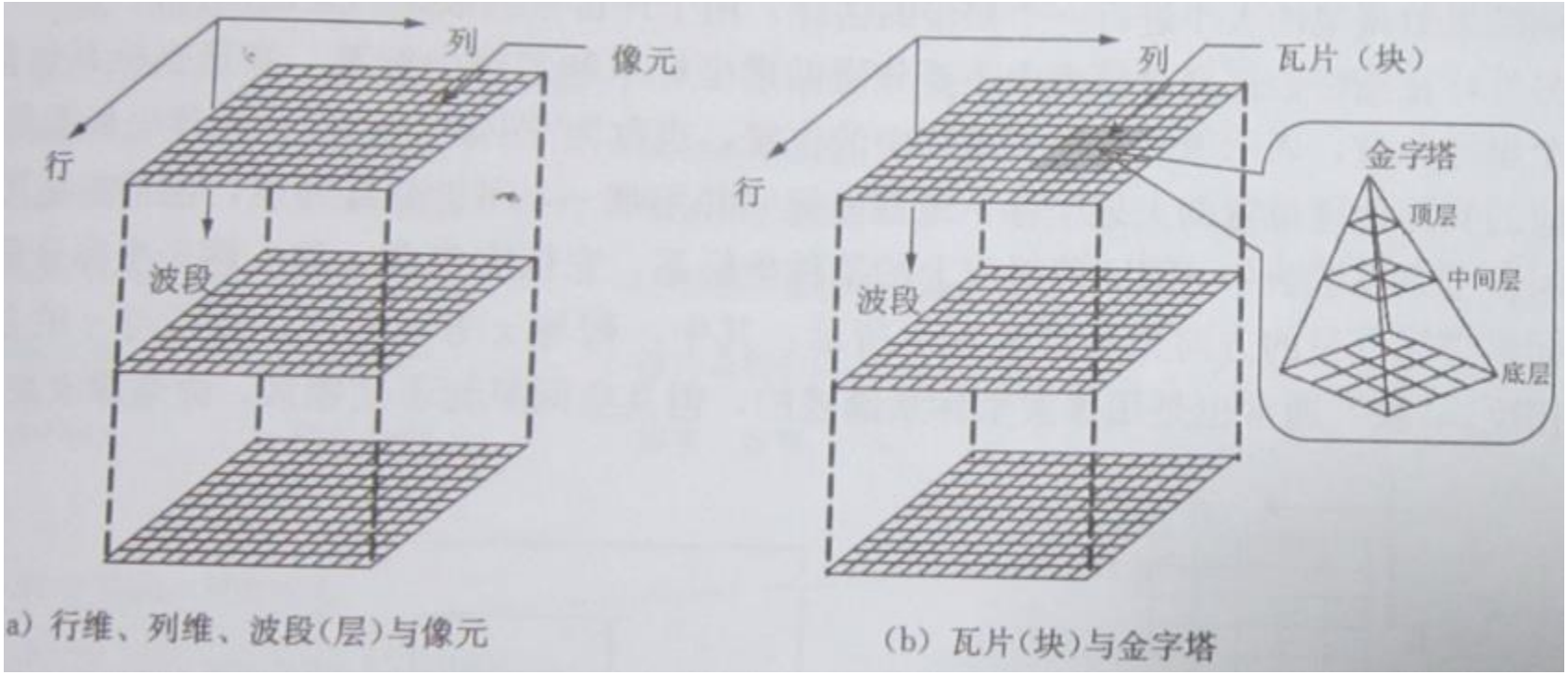
- 空间数据库采用一种统一栅格数据模型，描述上述不同种类的栅格数据
  - 也称为网格(grid)模型或覆盖(coverage)模型
- 栅格数据标准
  - OGC尚未定义相关的实现标准
  - SQL/MM的第四部分也仅针对上述第三类数据——静态图像，给出了相关标准
  - 以Oracle的GeoRaster和PostGIS的WKTRaster为例，介绍基本概念及其存在的差异或争议

12.5.1 栅格数据概念模型

- 栅格数据可以用行维、列维和波段(层)维进行统一描述
  - 层通常为逻辑上的概念，而波段则视为物理上的概念
  - 行有行宽，列有列宽，波段则有波段数目
  - 波段表示的是多维栅格像元矩阵的一个物理维度，单一波段内数据独自称为一个二维像元矩阵

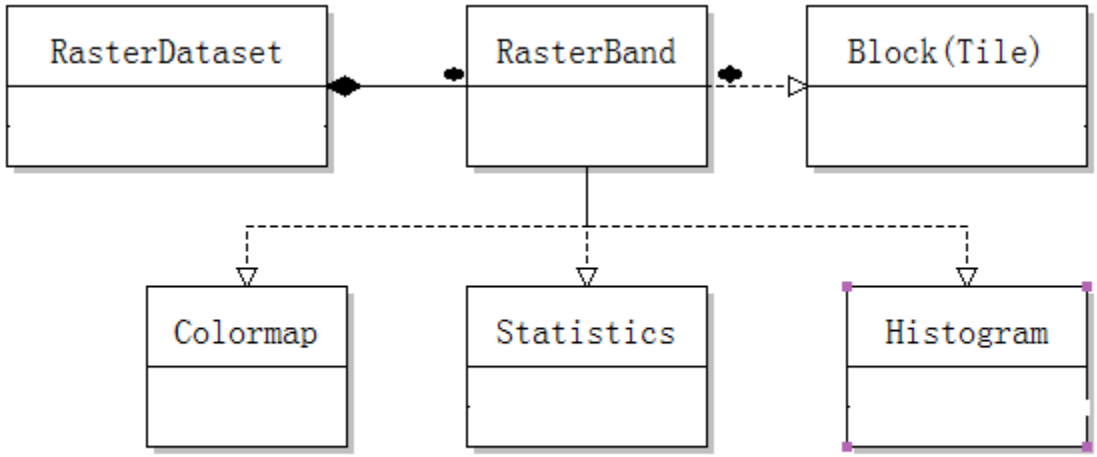
12.5.1 栅格数据概念模型

- 行维，列维，波段，像元
- 瓦片（块），金字塔



12.5.1 栅格数据概念模型

- 栅格数据的最小管理单元
  - 像元，数量太大，影响栅格数据的绘制效率
  - 瓦片tile或块block，m×n的像元区域
  - 为块建立不同分辨率的可视化表达，形成金字塔，用户请求时，按需传输块



栅格数据概念模型

12.5.1 栅格数据概念模型

- 概念模型
  - RasterDataSet代表栅格数据，由不同波段组成
  - 每个波段RasterBand都依赖颜色表Colormap、统计信息Statistics和直方图Histogram等用于描述栅格元数据的对象类，同时依赖若干承载栅格数据的像素块Block(Tile)
  - Colormap用于描述波段中不同像元的属性值在可视化时用何种颜色显示
  - Statistics用于描述波段中像元属性的统计值，如最大、最小和均值
  - Histogram用于描述波段中像元属性值的分布情况
  - Block(Tile)用于描述波段内某块中的像素值，提供一些访问块内像素信息的基本方法，甚至金字塔的访问

12.5.1 栅格数据概念模型

- 像素坐标系和地理坐标系
  - 像素坐标系包含行、列两个坐标分量，表示像素在像素矩阵中的位置
  - 地理坐标系是与像素对应的实际地理位置的大地坐标
- 块通常用像素坐标系描述

12.5.2 栅格模型的逻辑与物理实现

- 基于扩展数据类型的实现
  - 在空间数据库中扩展栅格数据类型及其相关函数的定义(如SDO_Raster, WKTRaster)
  - 将块中的栅格数据存储在栅格数据类型中，由数据库复制栅格数据的解释和管理
  - 用户可以通过扩展的SQL语句访问栅格数据
- Oracle的GeoRaster和PostGIS的WKTRaster

Oracle Spatial的GeoRaster

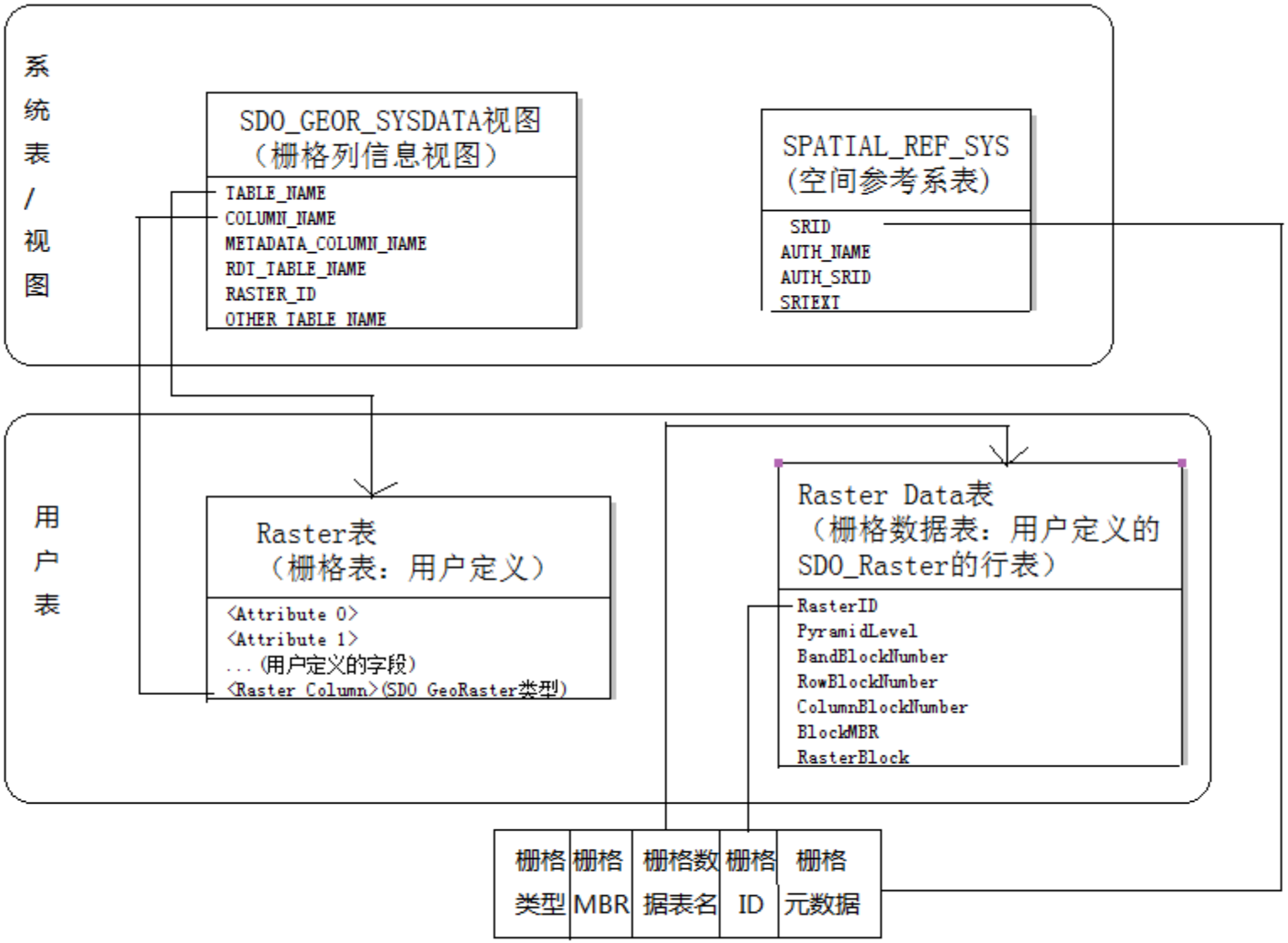
Oracle Spatial扩展了SDO_GeoRaster和SDO_Raster两个数据类型

```
--SDO_GeoRaster对象结构
Create Type SDO_GeoRaster As Object (
RasterType        Number,          --栅格数据类型
SPatialExtent     SDO_GEOMETRY,    --栅格数据范围
RasterDataTble VARCHAR2(32),--存储栅格数据的数据表名
RasterID          Number,          --栅格数据的ID
Metadata          XMLType          --栅格元数据
);
```

Oracle Spatial的GeoRaster

Oracle Spatial扩展了SDO_GeoRaster和SDO_Raster两个数据类型

```
--SDO_Raster对象结构
Create Type SDO_Raster As Object (
RasterID            Number,          --栅格数据ID
PyramidLevel        Number,          --金字塔级别数
BandBlockNumber     Number,          --块的波段号
RowBlockNumber      Number,          --块的行号
ColumnBlockNumber Number,            --块的列号
BlockMBR            SDO_GEOMETRY,    --块的最小边界矩形
RasterBlock         BLOB             --块内的栅格单元数据
);
```



GeoRaster的逻辑实现

Oracle Spatial的GeoRaster

- SDO_GeoRaster
  - 实际应用中的一个栅格数据集，描述了栅格数据的类型、栅格数据的范围、存储栅格数据的数据表名、栅格数据的ID以及栅格数据的元数据(XML类型)
  - 栅格数据的元数据
    - 对象信息(Object Info)：包括栅格类型、栅格ID、是否为空、空单元的值以及版本信息等信息
    - 栅格信息：包含单元值的类型、维数、分块、金字塔类型、重采样算法等信息
    - 空间参照系统信息：包括空间参考系ID以及地理定位所需的仿射变换信息

Oracle Spatial的GeoRaster

- 栅格数据的元数据
  - 对象信息(Object Info)：包括栅格类型、栅格ID、是否为空、空单元的值以及版本信息等信息
  - 栅格信息：包含单元值的类型、维数、分块、金字塔类型、重采样算法等信息
  - 空间参照系统信息：包括空间参考系ID以及地理定位所需的仿射变换信息
  - 时间参照系统信息：包括起止时间、时间分辨率等信息
  - 波段参照系统信息：包括对个光谱波段的描述信息
  - 每层的层信息：包括尺度因子、颜色相关信息(灰度级)、柱状图以及其他用于管理和使用所必需的基于层次的属性

Oracle Spatial的GeoRaster

- SDO_Raster
  - 按块的方式管理和存储栅格单元数据，包括栅格数据ID、金字塔级别数、块的波段号、块的行号、块的列号、块的MBR、块内的栅格单元数据等信息
  - 前5项标识唯一栅格数据
  - 数据存储在RasterBlock中
  - 最小边界矩形存于BlockMBR中

Oracle Spatial的GeoRaster

- GeoRaster的实现模型
  - 栅格表(raster table)和栅格数据表(rastet data table RDT)两个用户表
  - 栅格表
    - 采用SDO_GeoRaster扩展类型作为栅格列，其他字段为用户自定义属性。栅格表的一行对应一个栅格数据，其栅格列存储了该栅格数据的类型、MBR、存储栅格单元值的RDT表名、栅格ID、栅格元数据

Oracle Spatial的GeoRaster

- GeoRaster的实现模型
  - 栅格数据表
    - 由SDO_Raster扩展类型定义的行对象表，包括栅格ID、金字塔级别数、块的波段号、块的行号、块的列号、块的MBR、块内的栅格单元数据等信息
    - 前5项组成该表的联合主码
    - 栅格表和栅格数据表通过SDO_GeoRaster类型中的栅格数据表名和栅格ID关联

Oracle Spatial的GeoRaster

- GeoRaster的实现模型
  - 栅格系统数据视图 SDO_GEOR_SYSDATA View
    - 记录了数据库中用户定义的栅格表的表名、列名、栅格数据表名和栅格等
    - 目前其元数据列名和其他表名暂时无用
  - 空间参考系表 SPATIAL_REF_SYS
    - 系统表，记录了该空间数据库所支持的所有空间参考系的列表

Oracle Spatial的GeoRaster

- GeoRaster逻辑实现中的注意事项
  - 栅格表的栅格列仅存储了栅格数据的摘要和元数据信息，RDT表用于存储栅格单元数据。栅格数据中的一条记录对应RDT表中的多条记录
  - GeoRaster不限定栅格表和RDT表之间的对应关系。同一个栅格表中的不同记录对应的栅格数据可以存储在同一个RDT表中，也可以存储在不同的RDT表中。
  - RDT表的一行对应栅格数据的某行、某列、某波段所在块的某金字塔级别的栅格单元数据，但实现上，经常把所有波段联合存在一条记录的RasterBlock字段中

Oracle Spatial的GeoRaster

- GeoRaster逻辑实现中的注意事项
  - 栅格数据的空间参考、各波段的直方图和颜色表等信息存储在XML类型的栅格元数据中，它们将按照XML数据库的方式进行查询和检索。其中基于XML类型的栅格元数据具有较好的扩展性
  - 波段和层的区别。波段是物理上的概念，层是逻辑上的概念。一个栅格数据在逻辑上可以被划分为若干个层，每个层可能会有多个波段。若一个层对应一个时态，GeoRaster就为用户提供了一种时态数据的组织方式

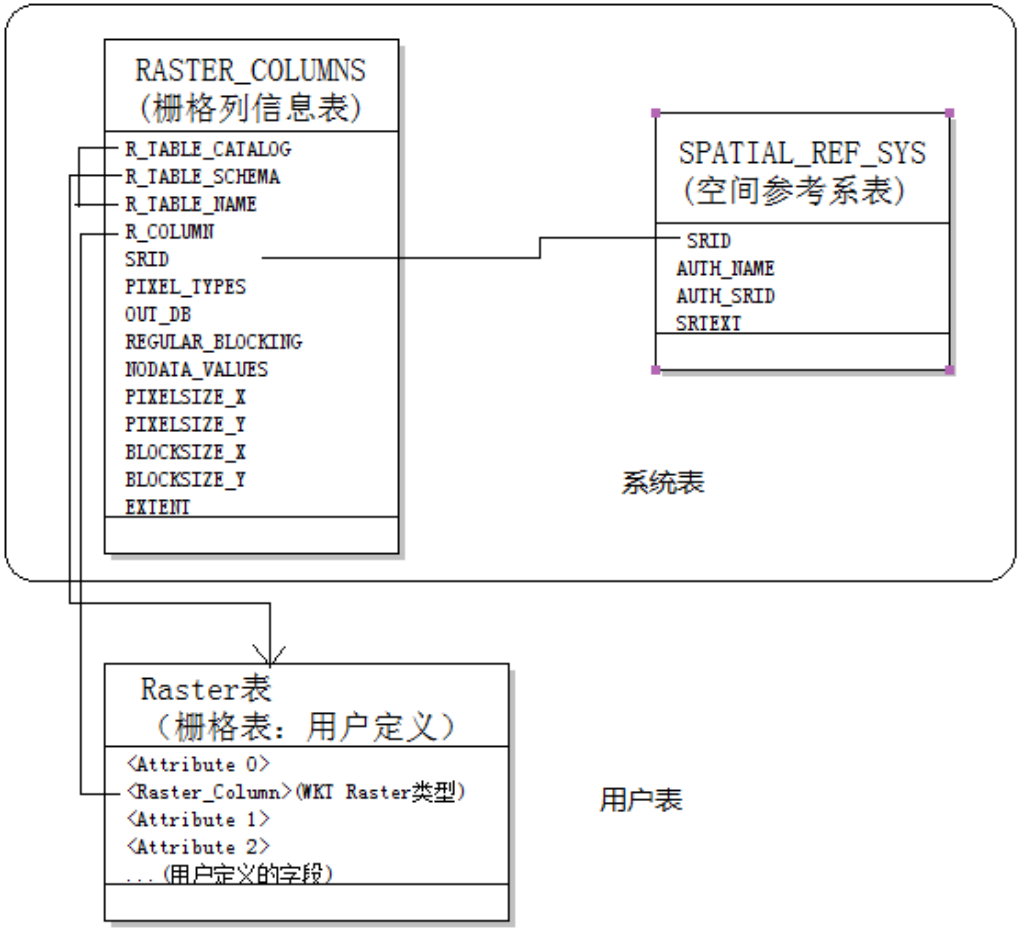
在组织方面具有较强的灵活性，但这些概念增加了用户理解和操作的难度

PostGIS的WKTRaster

- PostGIS先后出现过PGCHIP, PGRaster, WKTRaster三种栅格管理模块
  - PGCHIP使用CHIP类型扩展存储栅格数据，元数据与实际数据混合存储在CHIP结构中，构成了raster字段，实现了GDAL库的PostGIS栅格驱动
  - PGCHIP处于GDAL和PostGIS之间，栅格驱动函数不在PostGIS内部实现，而是扩展GDAL的源码，故不属于扩展数据类型的实现
  - PGRaster主要参考了Oracle Spatial的GeoRaster的设计，实现了PostgreSQL环境下Geotiff文件格式的导入导出

PostGIS的WKTRaster

- WKTRaster目的是实现矢量层和栅格层的无缝一体化操作，例如两者之间进行叠加操作
  - 将栅格数据视为一种新的WKT/WKB几何类型
  - 其逻辑实现也以几何对象模型的逻辑实现相似



WKT Raster的逻辑模型

PostGIS的WKTRaster

- 栅格表 – 用户表
  - 记录一组具有相同属性和行为的栅格数据集合，不同的行代表不同的栅格数据
  - Raster_Column列是数据库扩展的栅格数据类型
  - 其他列是用户定义的属性列
- 空间参考系(SPATIAL_REF_SYS) –系统表
  - 记录了该空间数据库所支持的所有空间参考系的列表

PostGIS的WKTRaster

- 栅格列信息(RASTER_COLUMNS) – 系统表
  - 记录了数据库中所有的栅格数据表及其栅格列的属性
  - R_TABLE_CATALOG, R_TABLE_SCHMEA, R_TABLE_NAME分别用来记录某Raster表所在的CATALOG名，模式名和该表的表名，用来唯一标识一个栅格表
  - R_COLUMN用于记录由前三列值确定的Raster表中Raster列的名字
  - SRID用于描述某栅格表的空间参考系，作为外码参考SPATIAL_REF_SYS
  - PIXELTYPE用于描述栅格数据的像素类型，例如1BB, 4BUI, 8BSI, 32BF等，分别表示1位布尔类型，4位无符号整数，8位有符号整数，32位浮点数

PostGIS的WKTRaster

- 栅格列信息(RASTER_COLUMNS) – 系统表
  - OUT_DB是布尔类型的数组，标志栅格数据每个波段在存储数据库内(in_db)的，还是存储在数据库外(out_db)
  - NODATA_VALUES是双精度类型的数据，用于记录每个波段中代表无数据的像素单元的值
  - PIXELSIZE_X和PIXELSIZE_Y分别描述了像素单元x和y方向的长度
  - BLOCKSIZE_X和BLOCKSIZE_Y分别描述了像素块x和y方向的长度
  - EXTENT描述了该表中所有栅格行的四至

PostGIS的WKTRaster

- WKT Raster与Geometry数据类型的实现方案相似
  - 采用单个的扩展栅格数据类型和单栅格表来存储和管理栅格数据
  - Oracle Spatial采用SDO_GEORASTER和SDO_RASTER两个栅格类型和两个表来存储和管理
- 由于不同应用对栅格数据的理解和组织各有不同，WKT Raster不限定栅格类型存储的内容
  - 一条记录可以存一幅图像，也可以存储其中一块

PostGIS的WKTRaster

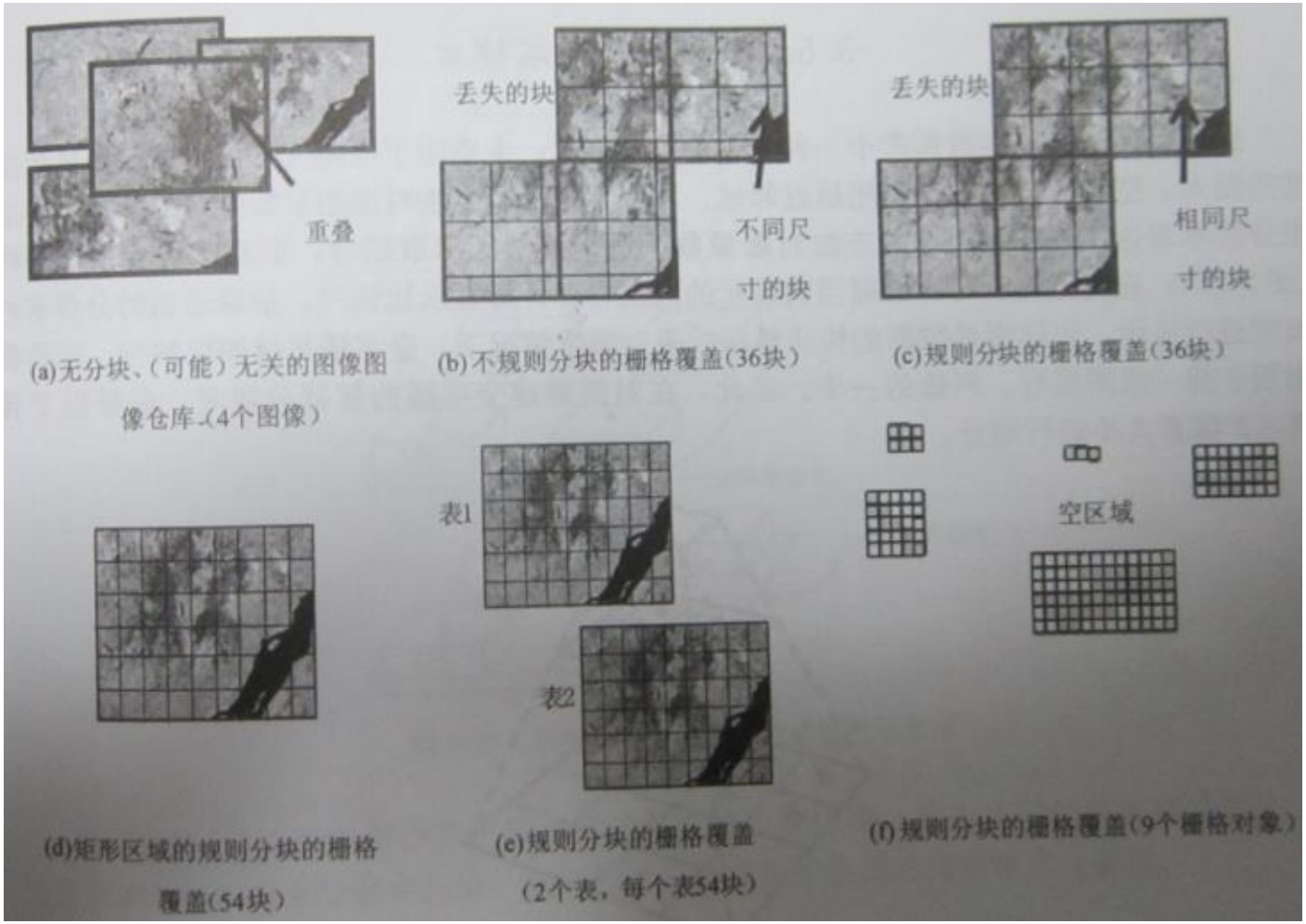
- 6种不同应用模式及其可能的表组织
  - 无分块、(可能)无关的图像仓库
    - 不要求图像一定有地理空间位置
    - 对于有地理空间位置的图像，他们可能重叠，也可能不重叠
  - 不规则分块的栅格覆盖
    - 可能不覆盖一个矩形区域，可能丢失一些块，不同的块可能有不同的尺寸
  - 规则分块的栅格覆盖
    - 可能不覆盖一个矩形区域，可能丢失一些块，块的尺寸均相同

PostGIS的WKTRaster

- 6种不同应用模式及其可能的表组织
  - 矩形区域的规则分块的栅格覆盖
    - 必须覆盖一个矩形区域，可以有丢失的块，块的尺寸必须相同
  - 平铺的图像
    - 覆盖区域必须是矩形，不能有丢失的块，块的尺寸相同
  - 栅格对象覆盖
    - 栅格数据层与矢量数据层进行一定的空间操作后，形成一些栅格化的结果集，WKT Raster可将其按栅格对象的形式存于栅格表中

PostGIS的WKTRaster

- 6种不同应用模式及其可能的表组织



PostGIS的WKTRaster

- WKTRaster不仅支持影像数据存储在栅格字段中，也支持将其存储在文件中

--存储在栅格列中

RasterFromText('RASTER'(2, 8, 30.0, 3, 22165.382558570, 785, 0, 0, -22165.382558570815, -545856.650, 7086694.1733, BAND (8BUI, (0, 3, 7, 6, 8, 9, 5, 5, 6, 2, 2, 4, 4)), BAND (16BF, (0.0, 1.2, 1.2, 2.6, 2.6, 3.4, 3.4, 4.0, 4.0, 5.6, 5.6, 6.4, 6.4, 6.8, 6.8, 8.6, 8.6))', [<srid>])

PostGIS的WKTRaster

- WKTRaster不仅支持影像数据存储在栅格字段中，也支持将其存储在文件中

--存储在文件中

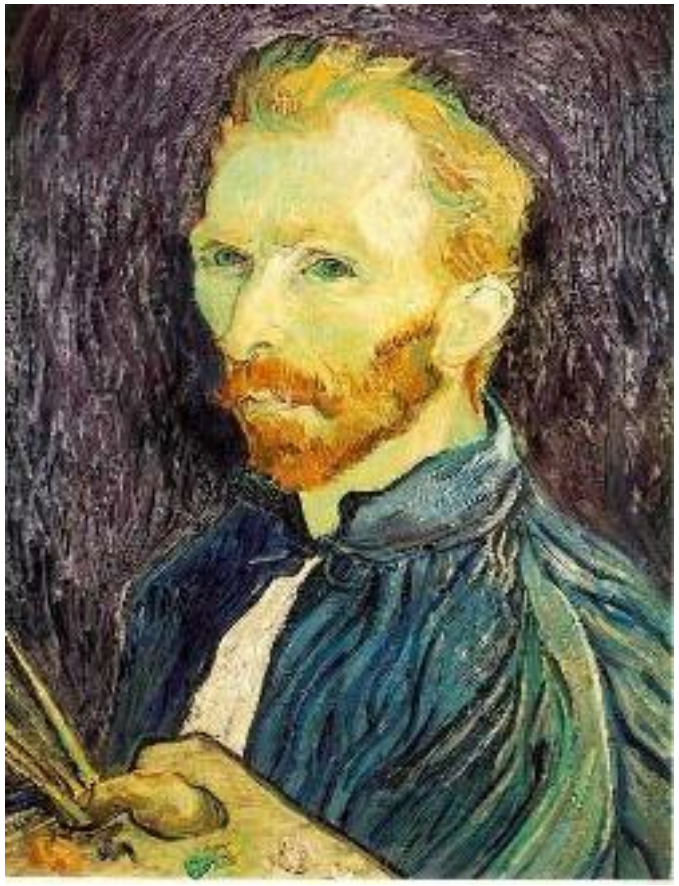
RasterFromText('RASTER'(2, 8, 30.0, 3, 22165.382558570, 785, 0, 0, -22165.382558570815, -545856.650, 7086694.1733, BAND (PT_EXT, 0, C:/data/landsat/01b1.tif), BAND (PT_EXT, 0.0, C:/data/landsat/01b2.tif)', [<srid>])

12.5.3 影像金字塔技术

- 影像金字塔是空间数据库中一种常见的管理技术，提高大影像数据浏览显示速度
  - 最近邻域、双线性插值等方法对原始影像进行重采样，形成粗分辨率表达，利用同样的方法对影像数据进行逐级构造，形成多级金字塔结构
    - 通常行列数减半，因此按$2^n \times 2^n$像素大小进行分块
  - 浏览显示时，根据当前显示的范围自动计算显示比例尺，抽取适当的分辨率的级别进行显示，以达到影像数据的快速显示效果

12.5.3 影像金字塔技术

- 最近邻域采样 – 存在失真情况



1/2

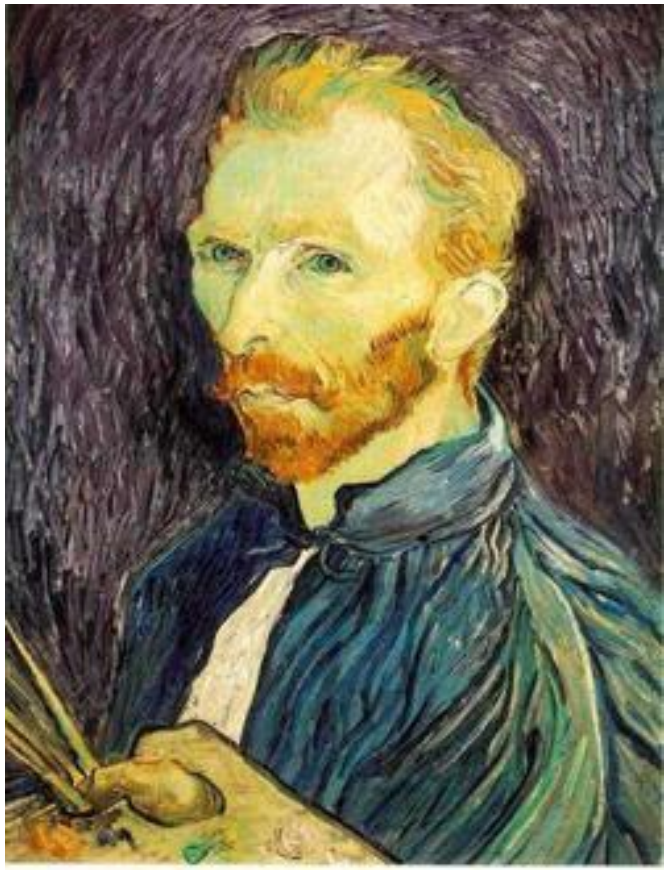
1/4

1/8

思考：Sampling and the Nyquist rate?

12.5.3 影像金字塔技术

- 双线性插值



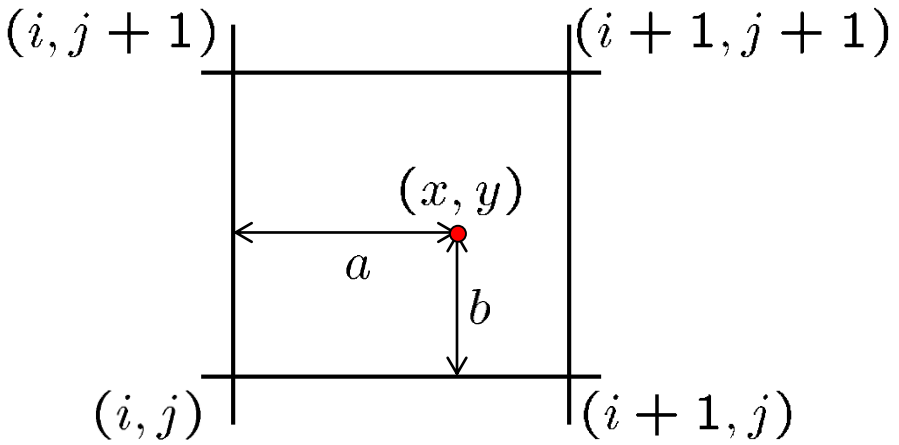
Bilinear 1/2

BL 1/4

BL 1/8

12.5.3 影像金字塔技术

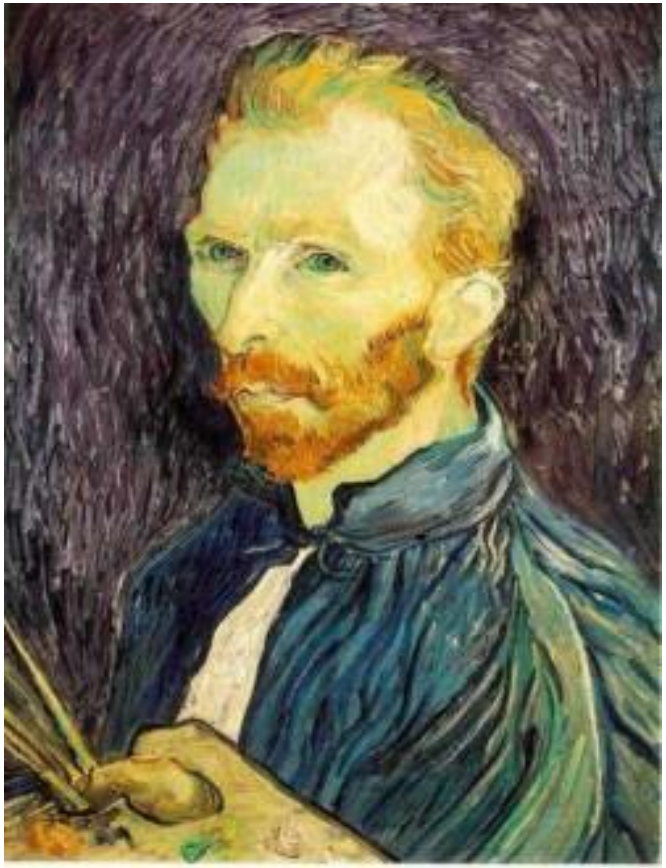
- A common method for resampling images



$$
\begin{aligned}
F(x,y) = \quad & (1-a)(1-b) && F(i,j) \\
& +a(1-b) && F(i+1,j) \\
& +ab && F(i+1,j+1) \\
& +(1-a)b && F(i,j+1)
\end{aligned}
$$

Subsampling with Gaussian pre-filtering

- Solution: filter the image, *then* subsample



Gaussian 1/2

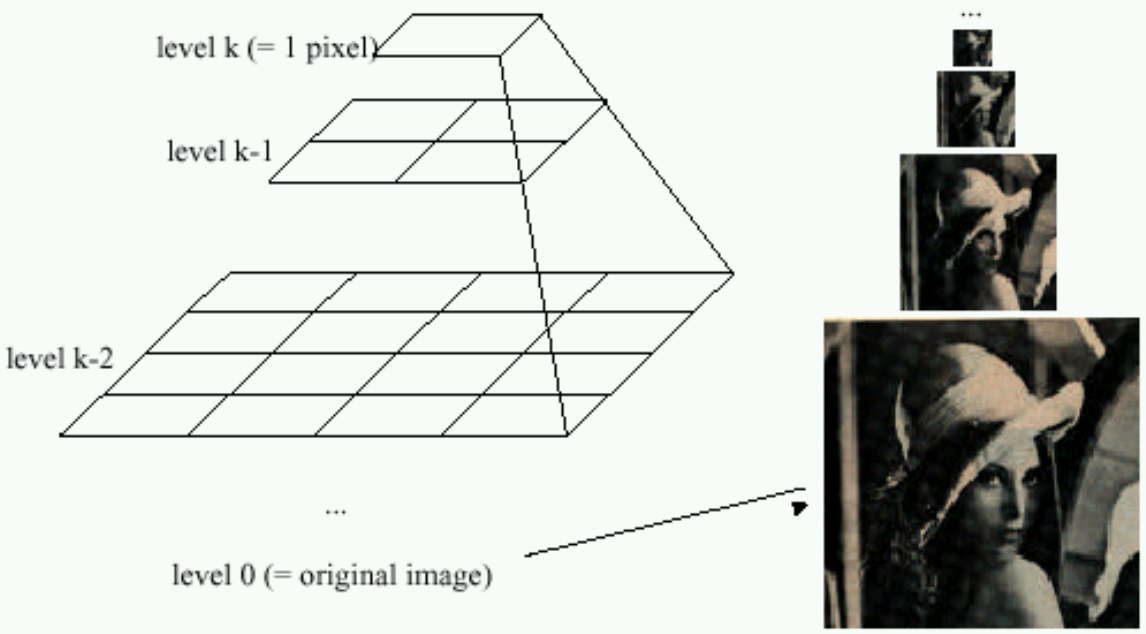
G 1/4

G 1/8

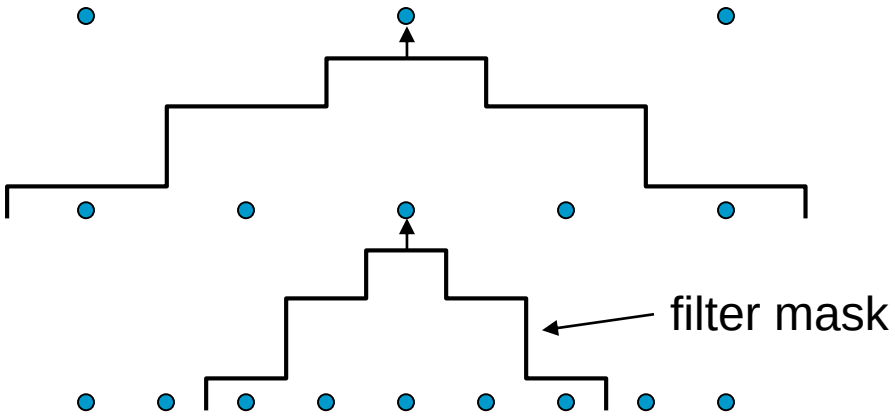
思考： Filter size should double for each ½ size reduction?

高斯金字塔

Idea: Represent NxN image as a "pyramid" of
1x1, 2x2, 4x4,…, 2^kx2^k images (assuming N=2^k)



高斯金字塔



Repeat

- Filter
- Subsample

Until minimum resolution reached

- can specify desired number of levels (e.g., 3-level pyramid)

The whole pyramid is only 4/3 the size of the original image!

小波金字塔

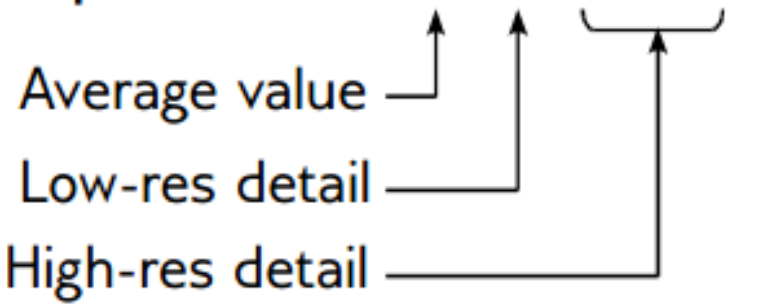
- 小波Wavelet: Haar
  - “Store the difference and pass the sum”
  - Represent every two successive values A, B by
    - (A + B) / 2 (average)
    - (A – B) / 2 (detail)
  - Allows perfect reconstruction
  - A sequence of n values becomes two sequences of n/2 values each

小波金字塔

- 举例: [9 7 3 5]

Resolution	Averages	Detail coefficients
4	[9 7 3 5]	
2	[8 4]	[1 −1]
1	[6]	[2]

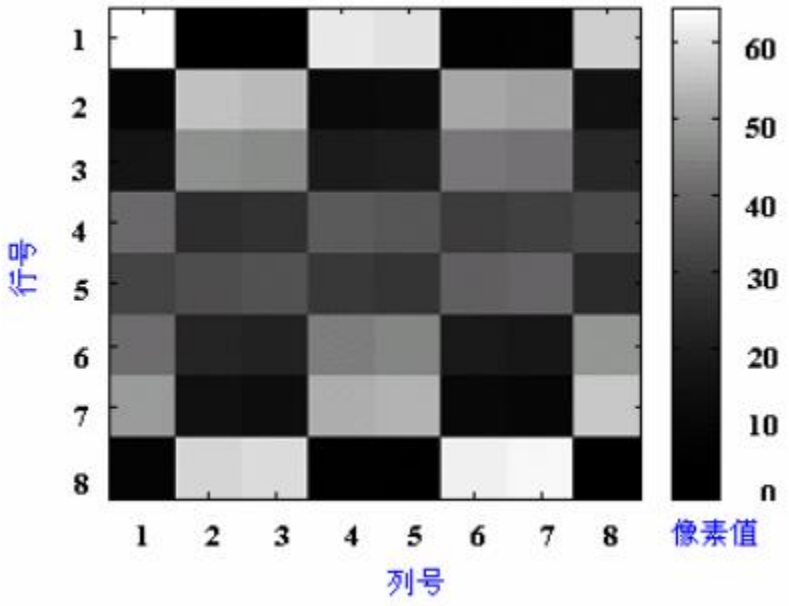
- Wavelet transform of sequence = [6 2 1 -1]



小波金字塔

- 举例：二维
  - 先按行小波变换，再按列小波变换

$$A=\begin{bmatrix} 64 & 2 & 3 & 61 & 60 & 6 & 7 & 57 \\ 9 & 55 & 54 & 12 & 13 & 51 & 50 & 16 \\ 17 & 47 & 46 & 20 & 21 & 43 & 42 & 24 \\ 40 & 26 & 27 & 37 & 36 & 30 & 31 & 33 \\ 32 & 34 & 35 & 29 & 28 & 38 & 39 & 25 \\ 41 & 23 & 22 & 44 & 45 & 19 & 18 & 48 \\ 49 & 15 & 14 & 52 & 53 & 11 & 10 & 56 \\ 8 & 58 & 59 & 5 & 4 & 62 & 63 & 1 \end{bmatrix}$$



小波金字塔

- 举例：二维
  - 先按行小波变换，再按列小波变换

64	2	3	61	60	6	7	57
9	55	54	12	13	51	50	16
17	47	46	20	21	43	42	24
40	26	27	37	36	30	31	33
32	34	35	29	28	38	39	25
41	23	22	44	45	19	18	48
49	15	14	52	53	11	10	56
8	58	59	5	4	62	63	1

原始图像

32.5	0	0.5	0.5	31	-29	27	-25
32.5	0	-0.5	-0.5	-23	21	-19	17
32.5	0	-0.5	-0.5	-15	13	-11	9
32.5	0	0.5	0.5	7	-5	3	-1
32.5	0	0.5	0.5	-1	3	-5	7
32.5	0	-0.5	-0.5	9	-11	13	-15
32.5	0	-0.5	-0.5	17	-19	21	-23
32.5	0	0.5	0.5	-25	27	-29	31

按行小波变换

32.5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	4	-4	4	-4
0	0	0	0	4	-4	4	-4
0	0	0.5	0.5	27	-25	23	-21
0	0	-0.5	-0.5	-11	9	-7	5
0	0	0.5	0.5	-5	7	-9	11
0	0	-0.5	-0.5	21	-23	25	-27

再按列小波变换

小波金字塔

- 举例：二维
  - 先按行小波变换，再按列小波变换
  - 压缩(< 5 ➔ 0)

$$\begin{bmatrix} 32.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & -4 & 4 & -4 \\ 0 & 0 & 0 & 0 & 4 & -4 & 4 & -4 \\ 0 & 0 & 0.5 & 0.5 & 27 & -25 & 23 & -21 \\ 0 & 0 & -0.5 & -0.5 & -11 & 9 & -7 & 5 \\ 0 & 0 & 0.5 & 0.5 & -5 & 7 & -9 & 11 \\ 0 & 0 & -0.5 & -0.5 & 21 & -23 & 25 & -27 \end{bmatrix}$$

小波变换结果

$$\begin{bmatrix} 32.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 27 & -25 & 23 & -21 \\ 0 & 0 & 0 & 0 & -11 & 9 & -7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & -9 & 11 \\ 0 & 0 & 0 & 0 & 21 & -23 & 25 & -27 \end{bmatrix}$$

小波变换结果压缩

小波金字塔

- 举例：二维
  - 先按行小波变换，再按列小波变换
  - 压缩(< 5 ➔ 0)
  - 重构

$$\begin{bmatrix} 64 & 2 & 3 & 61 & 60 & 6 & 7 & 57 \\ 9 & 55 & 54 & 12 & 13 & 51 & 50 & 16 \\ 17 & 47 & 46 & 20 & 21 & 43 & 42 & 24 \\ 40 & 26 & 27 & 37 & 36 & 30 & 31 & 33 \\ 32 & 34 & 35 & 29 & 28 & 38 & 39 & 25 \\ 41 & 23 & 22 & 44 & 45 & 19 & 18 & 48 \\ 49 & 15 & 14 & 52 & 53 & 11 & 10 & 56 \\ 8 & 58 & 59 & 5 & 4 & 62 & 63 & 1 \end{bmatrix}$$

原始图像

$$\begin{bmatrix} 32.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 27 & -25 & 23 & -21 \\ 0 & 0 & 0 & 0 & -11 & 9 & -7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & -9 & 11 \\ 0 & 0 & 0 & 0 & 21 & -23 & 25 & -27 \end{bmatrix}$$

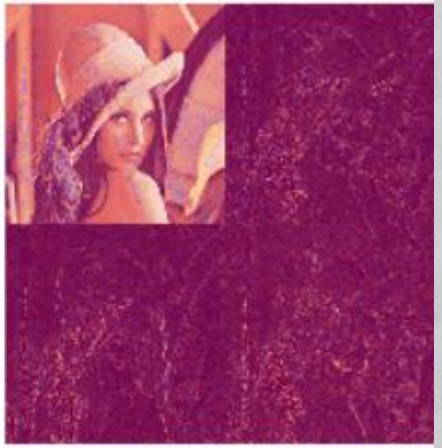
小波变换结果压缩

$$\begin{bmatrix} 59.5 & 5.5 & 7.5 & 57.5 & 55.5 & 9.5 & 11.5 & 53.5 \\ 5.5 & 59.5 & 57.5 & 7.5 & 9.5 & 55.5 & 53.5 & 11.5 \\ 21.5 & 43.5 & 41.5 & 23.5 & 25.5 & 39.5 & 32.5 & 32.5 \\ 43.5 & 21.5 & 23.5 & 41.5 & 39.5 & 25.5 & 32.5 & 32.5 \\ 32.5 & 32.5 & 39.5 & 25.5 & 23.5 & 41.5 & 43.5 & 21.5 \\ 32.5 & 32.5 & 25.5 & 39.5 & 41.5 & 23.5 & 21.5 & 43.5 \\ 53.5 & 11.5 & 9.5 & 55.5 & 57.5 & 7.5 & 5.5 & 59.5 \\ 11.5 & 53.5 & 55.5 & 9.5 & 7.5 & 57.5 & 59.5 & 5.5 \end{bmatrix}$$

重构图像

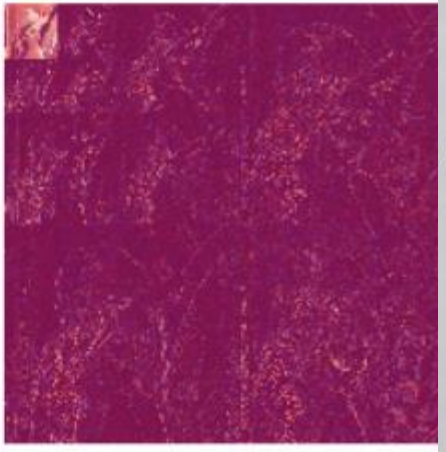
小波金字塔

(a) 原始图像

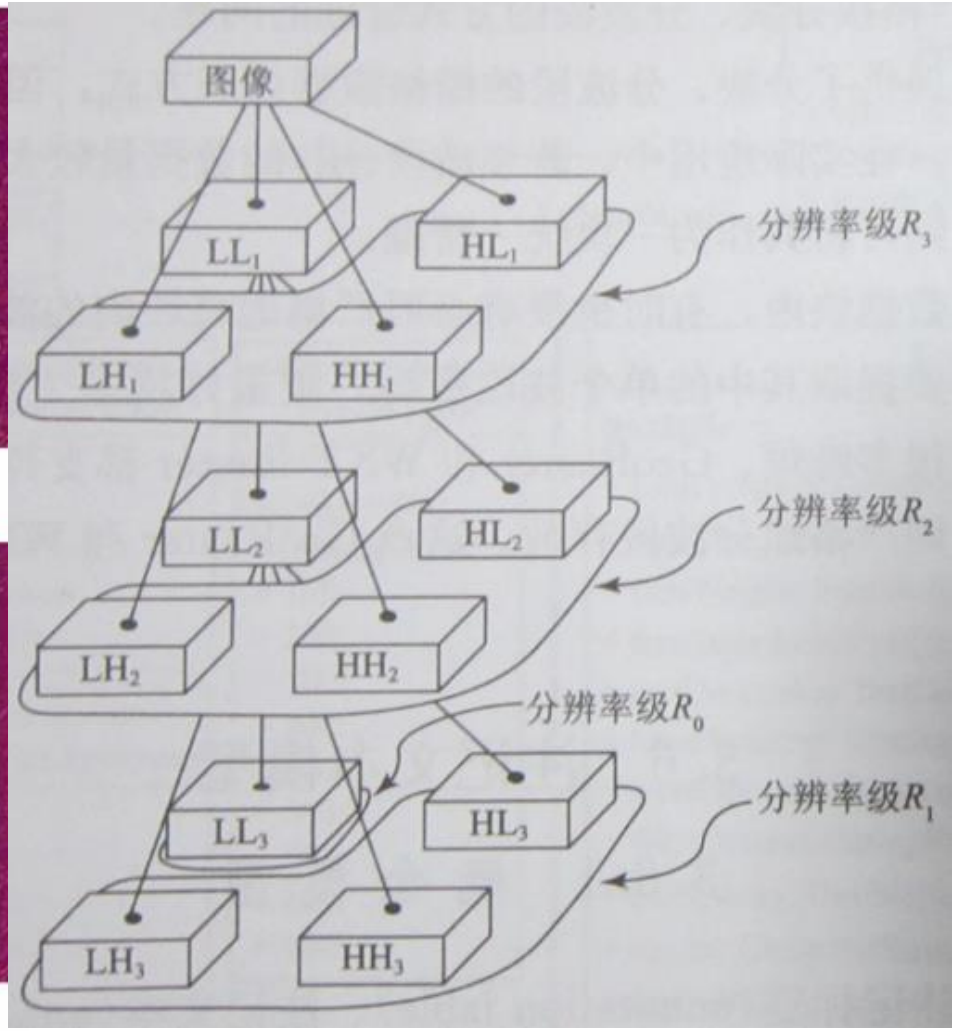


(b) 1/4分辨率图像

(c) 1/8分辨率图像



(d)1/16分辨率图像



12.5.3 影像金字塔技术

- 本质上是用冗余换效率的技术方案
  - 带来较大的数据冗余
  - 保留原始影像数据，称为“无损金字塔”
  - 对数据进行压缩，称为“有损金字塔”

12.5.4 问题与讨论

- 栅格概念模型已取得一致的认识，但实现方面有较大的区别，主要争议在于
  - 栅格数据存储在数据库中，还是文件中
    - 早起空间数据库是将栅格数据以文件的形式存储在磁盘中，数据库中的每条栅格元组记录了其栅格文件的文件路径
    - ArcSDE将栅格数据按BLOB的形式存储在数据库中
    - GeoRaster按扩展数据类型的方式管理栅格数据
    - WKTRaster既支持数据库存储模式，也支持文件存储模式

12.5.4 问题与讨论

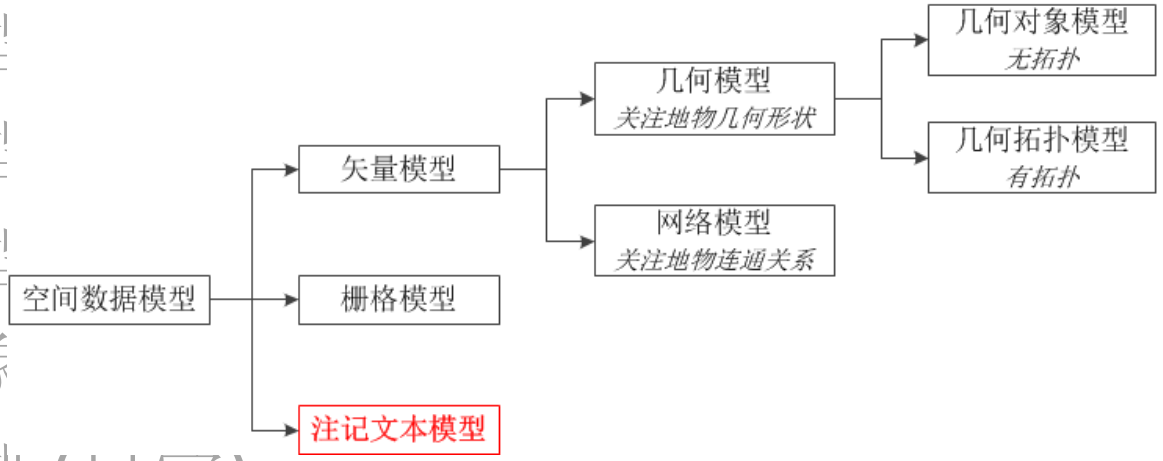
- 结合更新和操作特点选择合适的存储方式
  - 数据量小、更新频率较高或涉及复杂空间分析操作（如某地区的污染扩散场的数据），数据库管理的方式更好
    - 充分利用数据库的并发编辑能力和空间分析功能，甚至可以和矢量数据一起进行分析与处理
  - 数据量大、基本无需更新、查询操作简单的遥感影像数据，采用文件管理的方式更好
    - 数据库无需负担海量影像数据的并发控制、安全管理、灾难恢复等，文件管理具有较高的响应速度
    - Google Earth采用文件系统管理其海量的影像数据

12.5.4 问题与讨论

- 栅格概念模型已取得一致的认识，但实现方面有较大的区别，主要争议在于
  - 栅格数据存储在数据库中，还是文件中
  - 栅格数据是否严格按分块、分波段的方式管理
    - 若某波段的数据量较大，可以采用分块的方式管理
    - 若数据量较小，则可以将其作为一块统一管理
    - GeoRaster和WKT Raster都支持分波段、混合波段两种存储模式
    - ArcSDE则严格地分波段存储

第十二章 空间数据模型

- 12.1 几何对象模型
- 12.2 几何拓扑模型
- 12.3 空间网络模型
- 12.4 基于几何对象
- 12.5 栅格数据模型 (日字)
  - 12.5.1 概念模型
  - 12.5.2 逻辑与物理模型
  - 12.5.3 影像金字塔技术
- 12.6 注记文字模型
  - 12.6.1 概念模型
  - 12.6.2 逻辑模型



12.6 注记文字模型

- GIS注记分为以下3类
  - 注记标签 annotation label
    - 选择要素层中的某个属性值作为标记，附着在各要素的旁边显示，与要素具有正式的连接关系
    - 标签的显示风格与该要素层的文本风格定义一致
    - 在漫游和缩放后按照当前地图比例尺下的最佳位置重叠
  - 注记文本 annotation text
  - 注记尺寸 annotation size

12.6 注记文字模型

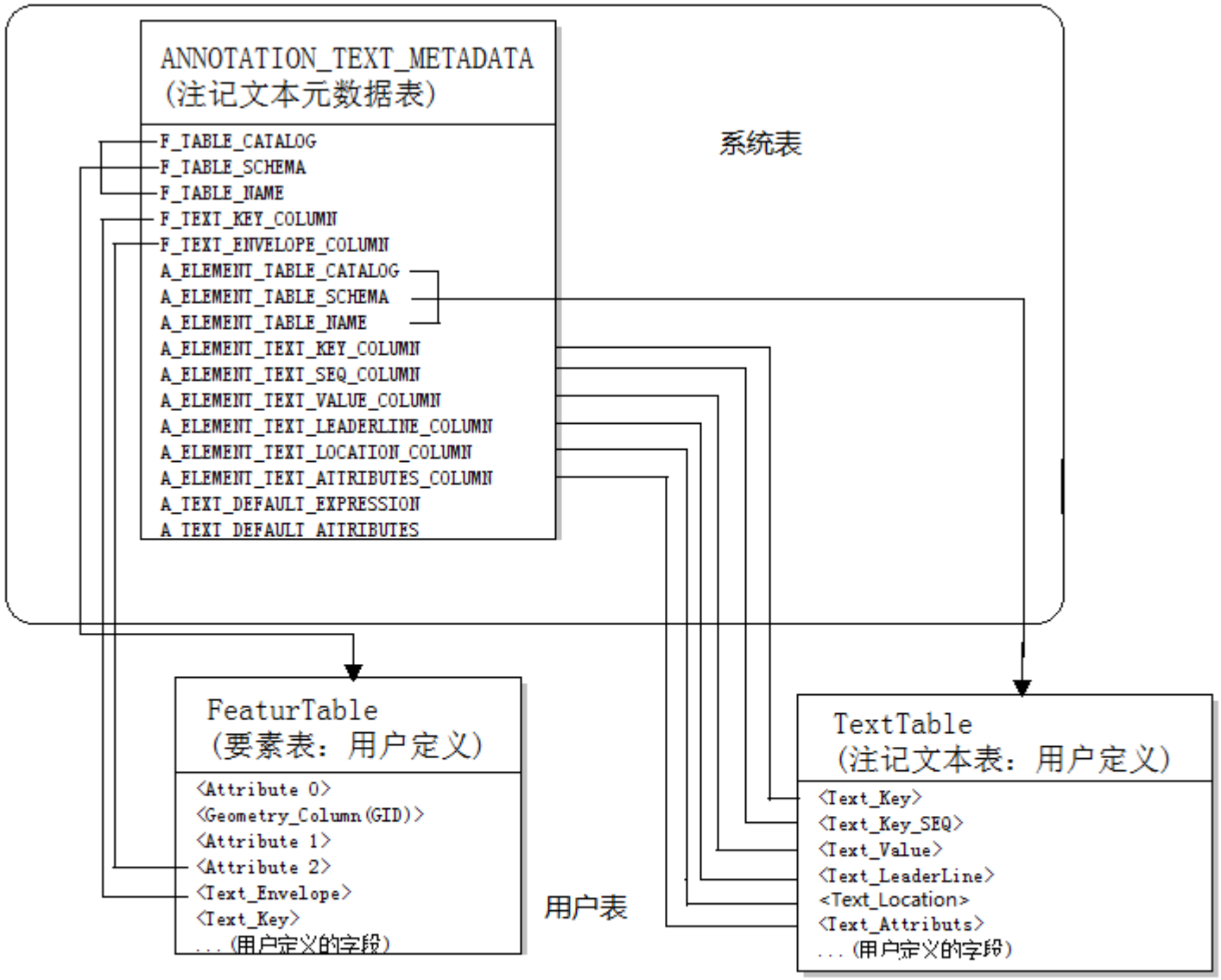
- GIS注记分为以下3类
  - 注记标签 annotation label
  - 注记文本 annotation text
    - 独立于要素层的一个文本数据集，由一些有序的格子独立放置的文本元素组成
    - 这些文本元素可能会沿着地理要素的方向、根据某地理要素的范围进行放置，与地理要素无正式连接
    - 例如地图上用于命名山脉的文本通常是一个标准注记文本
    - 注记文本的字体大小和位置都是固定的，不受地图视窗漫游与缩放的影像
  - 注记尺寸 annotation size

12.6 注记文字模型

- GIS注记分为以下3类
  - 注记标签 annotation label
  - 注记文本 annotation text
  - 注记尺寸 annotation size
    - 用于标注几何体长、宽、高数值的注记
    - 常用于在地块、房屋的测量等应用
    - ESRI的GeoDatabase提供了DimensionFeature用于描述注记尺寸类

12.6 注记文字模型

- 注记标签中的文字是要素的某个字段属性、其显示与该层的本文风格一致，因此无需额外对其进行定义
- 标记文本具有自己地理位置（文本要素的放置方向或范围）和属性（文本要素的文字或显示字样），其将和点、线、多边形一样，是一种类型的要素
- 标注尺寸的文本可能来自要素的某属性，但其有自己的显示模式和风格，地理空间数据对其也有相应的定义，目前标准尚未涉及此部分



基于预定义数据类型的文本标注的逻辑模型

12.6.1 注记文字概念模型

- Text
  - 一个标记文本类，描述了该条标注文本所放置的范围(Envelop)，记录了组成所有文本元素(TextElement)
- TextElement
  - 构成text的基本元素，描述了标记文本中每部分文字的具体情况，如该文字的内容(Value)，放置位置(Location)，带箭头的指引线(Leaderline)等，而对该文字的字体、大小、方向等的描述在TextAttributes中

12.6.1 注记文字概念模型

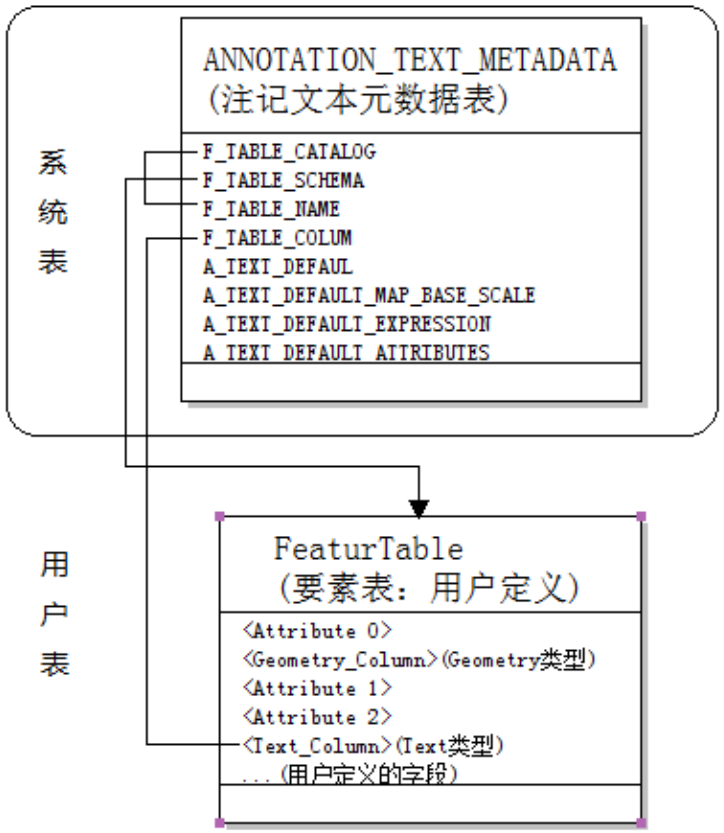
- TextAttributes
  - 详细描述了文字的字体、大小、颜色、旋转角度等信息
  - 每个TextElelement都有自身的属性，但这些属性并不是独立使用
  - 第一个TextElement的属性将会作为后续TextElement的属性，除非后续的TextElement中某个属性相并不与此相同才会额外进行处理
- 字体样式(FontStyle)，文本整饰(Textdecoration)，字体宽(Fontweight)，水平对齐方式(Horizontalalignment), 垂直对齐方式(Verticalalignment)

12.6.2 注记文字逻辑模型

- 基于预定义的数据类型
  - 文本码值(Text_Key)
    - 唯一标识一个Text对象
  - 显示范围(Text_Envelope)
    - Text_Key和Text_Envelope唯一标识一个TextElement对象
  - 文本元素的序号(Text_Key_SEQ)
  - 文本内容(Text_Value)
  - 文本指示线(Text_LeaderLine)
  - 文本放置位置(Text_Location)
  - 文本(Text_Attributes)的注记文本(Text)表
  - 元数据表(ANNOTATION_TEXT_METADATA)

12.6.2 注记文字逻辑模型

- 基于预定义的数据类型
  - 大部分文本的显示属性可能都是一样的，将显示属性存储在元数据表的A_TEXT_DEFAULT_ATTRIBUTES字段中
  - 若某行的属性与其有差异，则可记录在Text表的<Text_Attributes>列中
  - 文本最终的属性是元数据表中的属性与单个行(记录)属性值的叠加



基于扩展Geometry数据类型的文本标注的逻辑模型

12.6.2 注记文字逻辑模型

- 基于扩展Geometry数据类型
  - 一个扩展Text类型的Text_Column字段
  - 一个用于描述标记文本的元数据表(ANNOTATION_TEXT_METADATA)
  - Text类型列的默认地图基本尺度A_TEXT_DEFAULT_MAP_BASE_SCALE，默认表达A_TEXT_DEFAULT_EXPRESSION，显示属性A_TEXT_DEFAULT_ATTRIBUTES

第十二章 空间数据模型

- 12.1 几何对象模型
- 12.2 几何拓扑模型
- 12.3 空间网络模型
- 12.4 基于几何对象模型构建空间网络模型
- 12.5 栅格数据模型 (自学)
- 12.6 注记文字模型