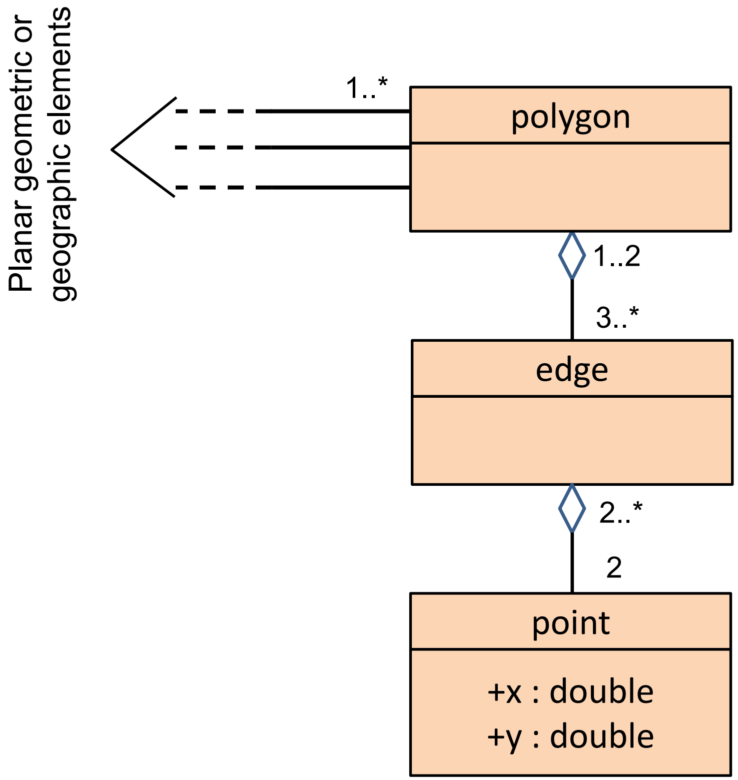
Handling Spatial data

LECTURE 5th: Spatial data in Relational Databases

- **Spatial Data Import and Management**
- **Elementary Cadaster**
- **The need to integrate spatial data management into the Database Systems**

Nikolas Mitrou
Prof. ECE NTUA

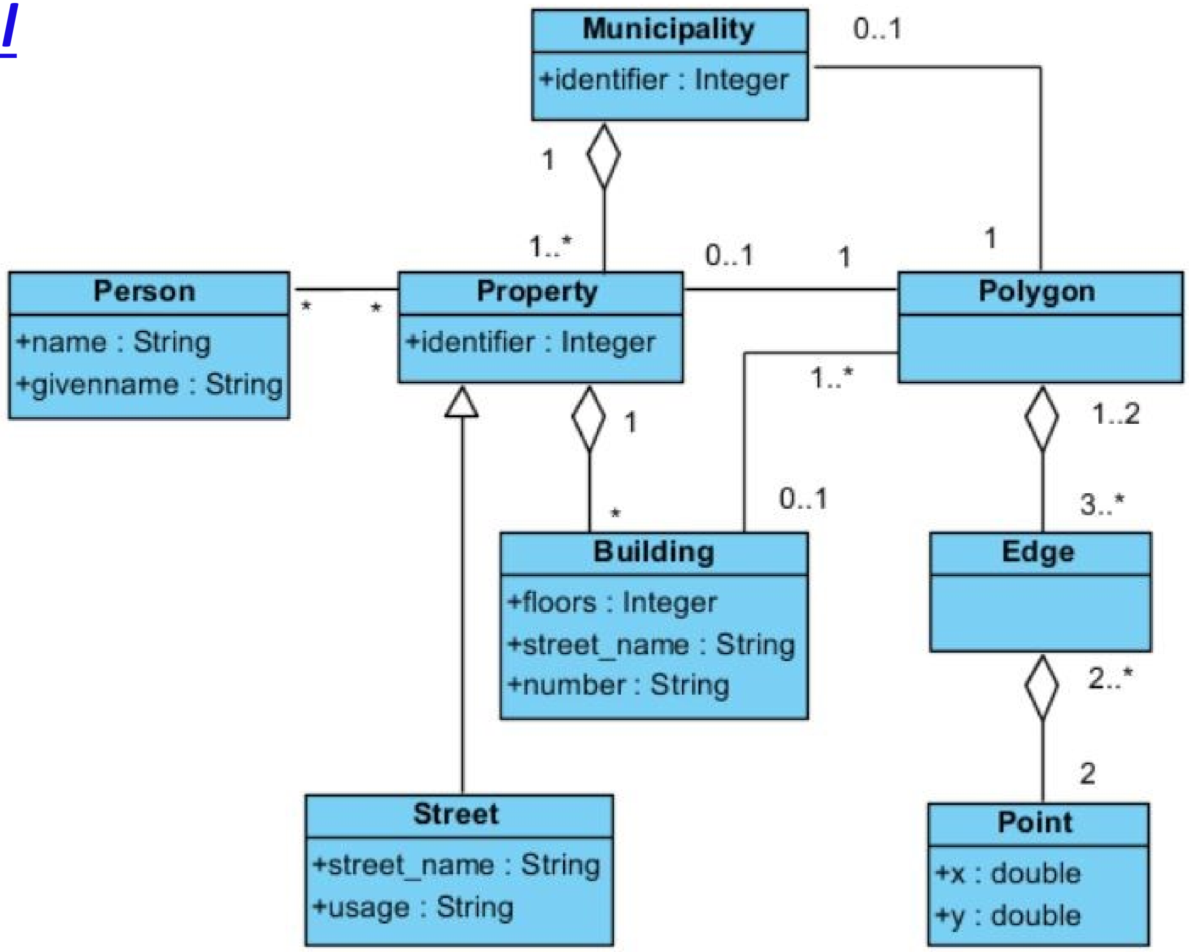
Inserting and handling spatial data



- Planar (2D) geometric or geographic objects related with one or more polygons
- Each polygon consists of a set of edges (three or more). Each edge participates in the perimeter of one or two polygons
- Each edge is defined by exactly two points. Each point participates in the definition of two or more edges.

Example: Elementary Cadaster

UML model



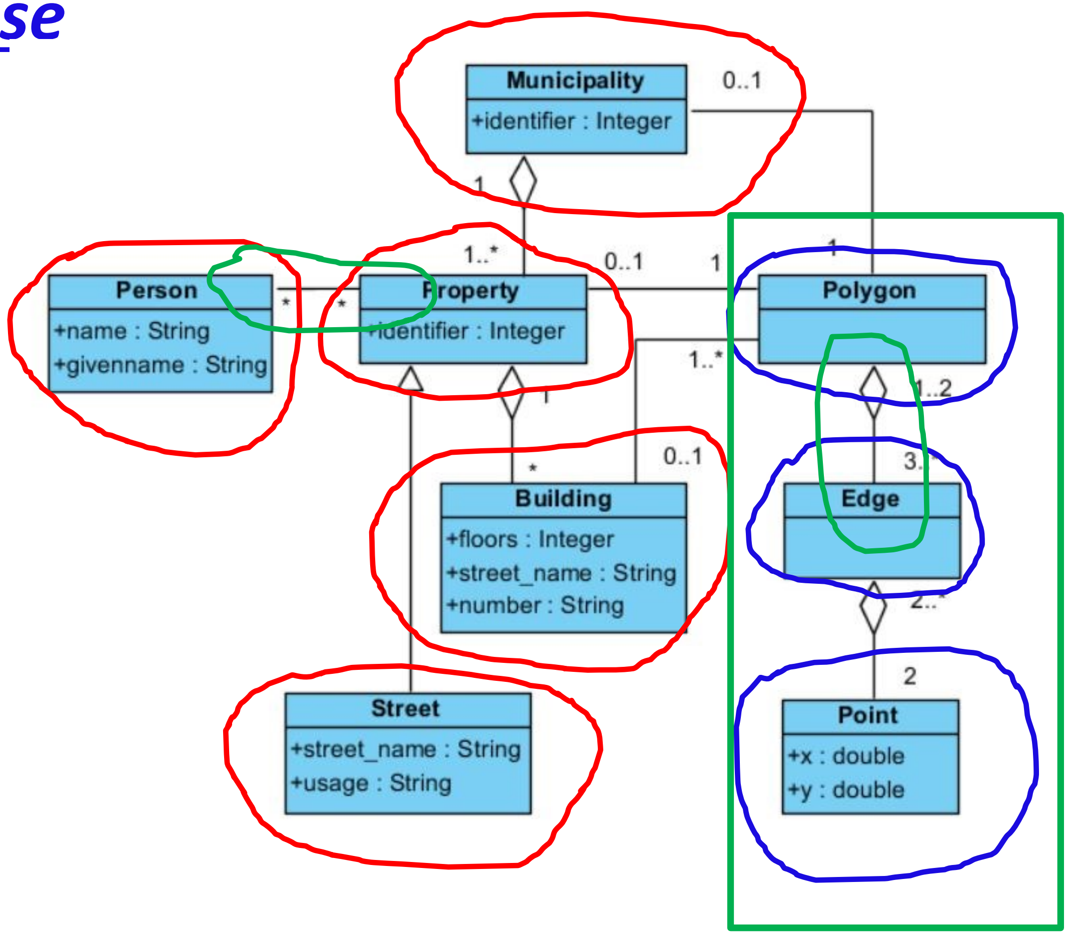
Elementary Cadaster (cont.)

Map to a Database

Creation of 10 tables

- ✓ Municipality
- ✓ property
- ✓ Person
- ✓ Building
- ✓ Street
- ✓ Polygon
- ✓ Edge
- ✓ Point
- ✓ Polygonedge
- ✓ Propertyperson

Χωρικά (Polygon, Edge, Point, Polygonedge)



Why not edgepoint ???

Elementary Cadaster (cont.)

municipality

idmunicipality [PK] integer	polygonid integer
1	1001
2	1002

polygon

idpolygon integer	propertyid integer	buildingid integer
1	101	
2	102	
3	103	
4	104	
5	105	

person

idperson integer	name character varying(50)	giv cha
202219	ΓΙΑΞΟΠΟΥΛΟΥ	ΓΛΥ
202221	ΛΟΥΛΟΥ	ΗΛΕ
202223	ΞΥΛΑΓΓΟΥΡΑ	ΜΑΡ
202225	ΠΡΙΦΤΗ	ΙΩΑ
202227	ΣΠΑΝΑΚΗΣ	ΜΑΡ
202215	ΧΡΗΣΤΟΥ	ΑΝΑ
202201	ΑΝΑΓΝΩΣΤΟΠΟΥΛΟΣ	ΧΡΗ
202203	ΒΑΣΙΛΕΙΑΔΗΣ	ΣΑΒ
202205	ΔΕΜΕΡΤΖΗ	ΙΣΙ
202207	ΛΑΒΙΔΑΣ	ΑΛΕ
202209	ΟΙΚΟΝΟΜΙΔΟΥ	ΧΡΙ
202211	ΤΣΕΛΚΑ	ΙΩΑ
202213	ΦΑΣΟΥΛΗ	ΑΝΝ
202217	ΠΑΝΑΓΙΩΤΟΠΟΥΛΟΥ	ΙΩΑ

propertyperson

propertyid [PK] integer	personid [PK] integer
102	202201
102	202215
103	202203
103	202227

property

idproperty integer	municipalityid integer
101	1
102	1
103	1
104	1
105	2
106	1

polygonedge

idpolygonedge integer	polygonid integer	edgeid integer
26	1	1
30	1	5
29	1	4
28	1	3
27	1	2
31	2	6
34	2	2

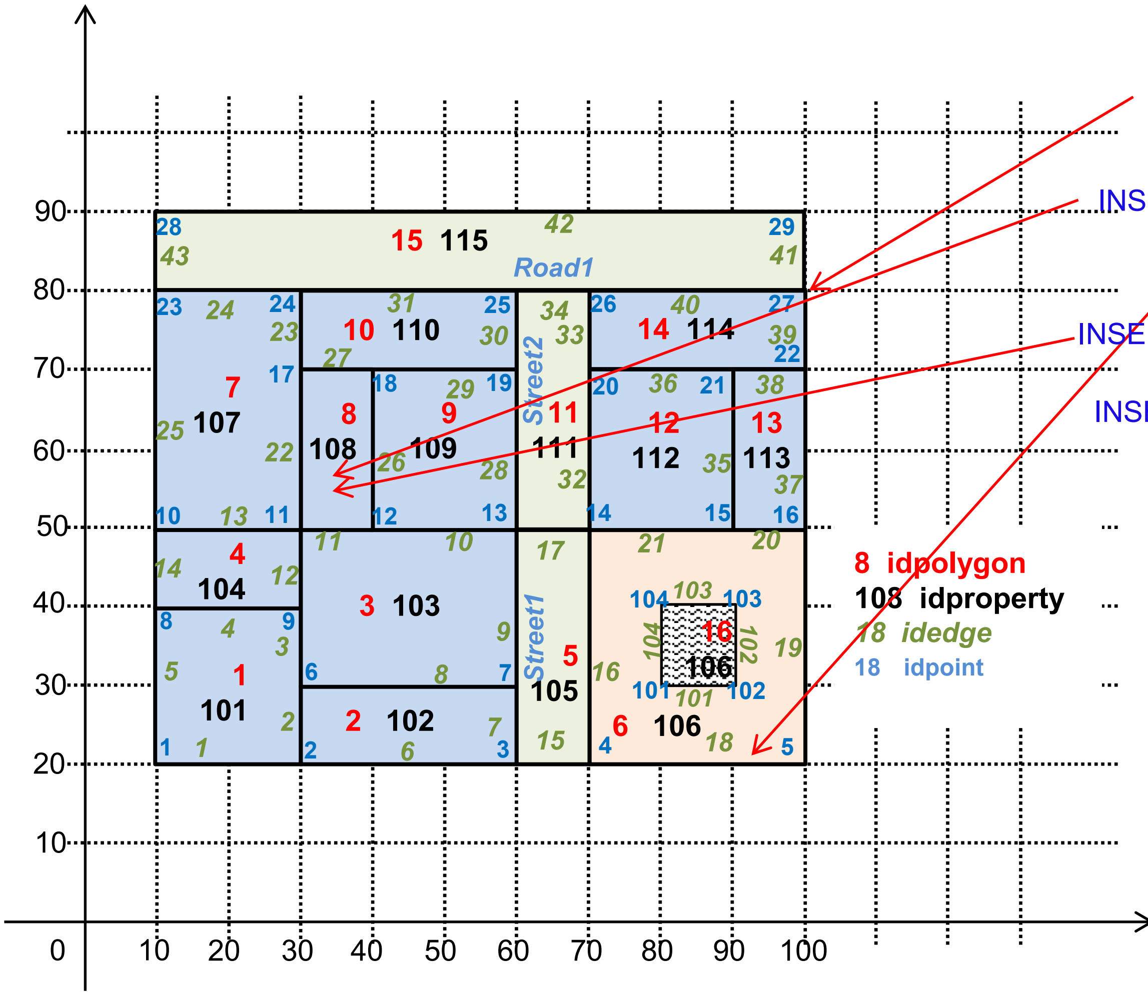
edge

idedge [PK] int	pointid1 integer	pointid2 integer
1	1	2
2	2	6
3	6	9
4	8	9
5	1	8
6	2	3

point

idpoint [PK] int	x numeric(10,3)	y numeric(10,3)
1	10.000	20.000
2	30.000	20.000
3	60.000	20.000
4	70.000	20.000
5	100.000	20.000
6	30.000	30.000

Elementary Cadaster (cont.)




```
INSERT INTO point VALUES (27,100,80);
INSERT INTO polygon VALUES (8, 108, NULL);
INSERT INTO edge VALUES (18,4,5);
INSERT INTO property VALUES (108, 400, 1);
INSERT INTO polygonedge VALUES (60,8, 11);
```

Queries about spatial data

Calculation of edge length

```
SELECT e.idedge,
     ROUND(SQRT( POWER((p1.x - p2.x),2) +
                POWER ((p1.y - p2.y),2)) ,2)
     AS length
FROM exercise1.edge e, exercise1.point p1,
     exercise1.point p2
WHERE  e.pointID1 = p1.IDpoint
AND    e.pointID2 = p2.IDpoint
ORDER BY length, e.IDedge
```

idedge integer	length numeric
[illegible]	[illegible]
33	10.00
34	10.00
38	10.00
39	10.00
41	10.00
43	10.00
101	10.00
102	10.00
103	10.00
104	10.00
1	20.00
4	20.00
5	20.00
9	20.00
10	20.00
13	20.00

Queries about spatial data (cont.)

Find edges per property (polygonedge)

```
SELECT propertyid, edgeid
  FROM exercise1.polygon, exercise1.polygonedge
  WHERE
       polygon.idpolygon=polygonedge.polygonid
  AND polygon.propertyid > 0
```

propertyid integer	edgeid integer
101	1
101	2
101	3
101	4
101	5
102	6
102	7
102	8
102	2
103	8
103	9
103	10
103	11
103	12
103	3
104	4

Queries about spatial data (cont.)

Calculation of property circumference: as an SQL query

```
SELECT C1.propertyid, C1.circumference FROM
 (SELECT propertyedge.propertyid,
      ROUND(SUM(A1.length),2) AS circumference
  FROM
    (SELECT propertyid, edgeid
     FROM exercise1.polygon, exercise1.polygonedge
     WHERE
        polygon.idpolygon=polygonedge.polygonid
     AND polygon.propertyid > 0) propertyedge,

    (SELECT e.idedge, SQRT( POWER((p1.x - p2.x), 2)
              + POWER ((p1.y - p2.y), 2)) AS length
     FROM exercise1.edge e, exercise1.point p1,
          exercise1.point p2
     WHERE
        e.pointID1 = p1.IDpoint AND
        e.pointID2 = p2.IDpoint ) A1
  WHERE propertyedge.edgeid = A1.idedge
  GROUP BY propertyedge.propertyid) C1
ORDER BY C1.circumference, C1.propertyid;
```

propertyid integer	circumference numeric
104	60.00
108	60.00
113	60.00
101	80.00
102	80.00
105	80.00
109	80.00
110	80.00
111	80.00
112	80.00
114	80.00
103	100.00
107	100.00
106	160.00
115	200.00

Queries about spatial data (cont.)

Calculation of polygon circumference II – as a function

```
DROP FUNCTION IF EXISTS exercise1.circumference(integer) CASCADE;
CREATE FUNCTION exercise1.circumference(polygID integer)
         RETURNS NUMERIC AS $$
  DECLARE circ NUMERIC :=0;
    p1ID exercise1.point.IDpoint%TYPE;
    p2ID exercise1.point.IDpoint%TYPE;
    p1x exercise1.point.x%TYPE;
    p1y exercise1.point.y%TYPE;
    p2x exercise1.point.x%TYPE;
    p2y exercise1.point.y%TYPE;
  BEGIN
    FOR p1ID, p2ID IN (SELECT edge.pointID1, edge.pointID2
                       FROM exercise1.edge, exercise1.polygonedge
                       WHERE edge.IDedge=polygonedge.edgeID
                       AND polygonedge.polygonid=polygID)
    LOOP
        p1x = (SELECT x FROM exercise1.point WHERE point.IDpoint=p1ID);
        p1y = (SELECT y FROM exercise1.point WHERE point.IDpoint=p1ID);
        p2x = (SELECT x FROM exercise1.point WHERE point.IDpoint=p2ID);
        p2y = (SELECT y FROM exercise1.point WHERE point.IDpoint=p2ID);
        circ = circ + SQRT( POWER((p1x - p2x), 2) +
                            POWER ((p1y - p2y), 2));
     END LOOP;
     RETURN ROUND(circ,2);
  END;
$$ LANGUAGE plpgsql;
```

Queries about spatial data (cont.)

Calculation of polygon circumference II – (cont)

```
CREATE VIEW  exercise1.circumferences AS
          SELECT IDpolygon AS ΠΟΛΥΓΩΝΟ,
                 exercise1.circumference(IDpolygon) AS ΠΕΡΙΜΕΤΡΟΣ
          FROM exercise1.polygon pol
          WHERE pol.propertyID>0
          ORDER BY ΠΕΡΙΜΕΤΡΟΣ, ΠΟΛΥΓΩΝΟ;
```

```
SELECT * FROM exercise1.circumferences;
```

ΠΟΛΥΓΩΝΟ integer	ΠΕΡΙΜΕΤΡΟΣ numeric
16	40.00
4	60.00
8	60.00
13	60.00
1	80.00
2	80.00
5	80.00
9	80.00
10	80.00
11	80.00
12	80.00
14	80.00
3	100.00
7	100.00
6	120.00
15	200.00

Exercise:

How should we call `circumference` for property circumference calculation, in the case othey consist of multiple polygons;

Check it, after setting, e.g.:

```
UPDATE exercise1.polygon SET propertyID=1
WHERE IDpolygon=6;
```

PL/PgSQL (cont)

Return non-simple types (e.g. records)

- **RETURN NEXT expression;**

```
CREATE TYPE exercise1.neighbours
AS (pers INT, nam VARCHAR, prop INT, border INT);

CREATE OR REPLACE FUNCTION exercise1.find_neighbours(text)
RETURNS SETOF exercise1.neighbours AS $$
DECLARE
    name ALIAS FOR $1;
    ...
    neighbs exercise1.neighbours;
BEGIN
    ...
    FOR neighbs IN (SELECT …  )
    LOOP
       RETURN NEXT neighbs
    END LOOP;
    RETURN;
END;
$$ LANGUAGE plpgsql;
```

PL/PgSQL (συνέχεια)

Return non-simple types (e.g. records) (cont.)

- **RETURN QUERY query;**

```
CREATE TYPE exercise1.neighbours
AS (pers INT, nam VARCHAR, prop INT, border INT);

CREATE OR REPLACE FUNCTION exercise1.find_neighbours(text)
RETURNS SETOF exercise1.neighbours AS $$
DECLARE
    name ALIAS FOR $1;
    ...
BEGIN
    ...
    RETURN QUERY   SELECT personID, name, propertyID, edge
                   FROM …
END;
```

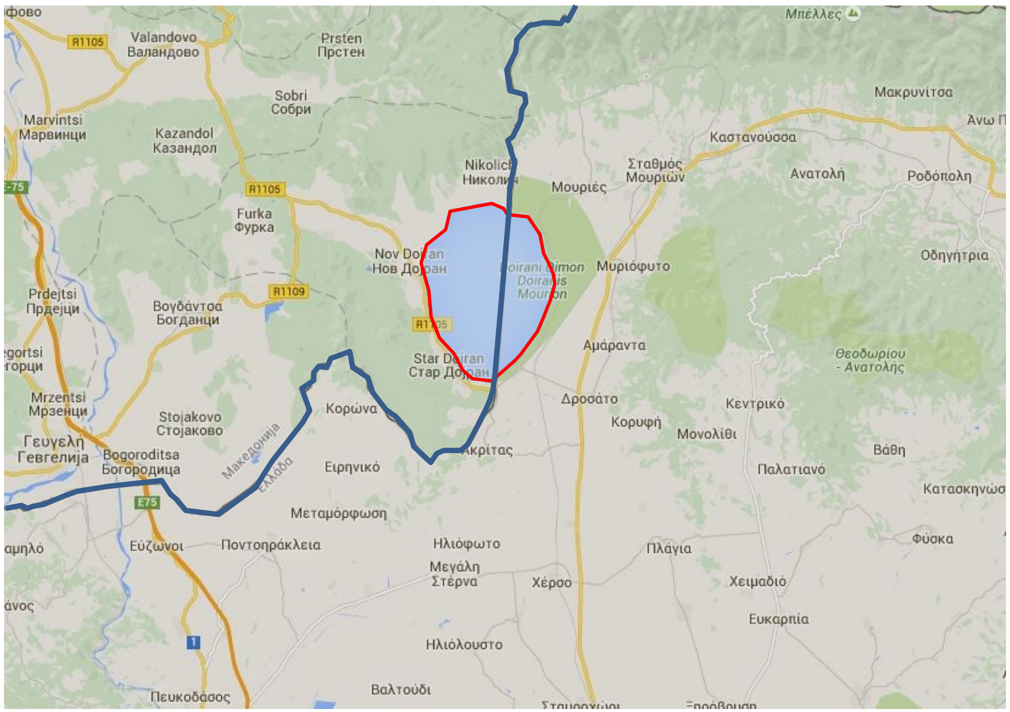
Instead, special "geometry" columns are inserted into descriptive data tables of geon

Integrated spatial data management

- Managing spatial data in DBMS, as conventional arrays of the <polygon><edge><point> hierarchy, as in the previous examples, is complex and error-prone:
  - Update and search queries are complex
  - All calculations must be done in the application program (outside management system)
  - Any reconstruction of the data results in changes to the application programs
- Instead, special "geometry" columns are inserted into descriptive data tables of geometric or geographic entities(abstract data types)
- A rich repertoire of functions is available for the most common operations and necessary calculations within the management system.
- Well-known database management systems (DBMS) offer extensions with such functionality

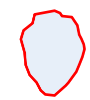
Integrated spatial data management (cont)

Example of abstract data types



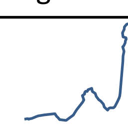

```
CREATE TABLE lakes (
  IDlake VARCHAR(20) PRIMARY KEY,
  name VARCHAR (50));

SELECT AddGeometryColumn('lakes',
  'lake_geom',4326, 'POLYGON', 2);
```

IDlake	name	lake_geom
GR_L021	Δοϊράνη	

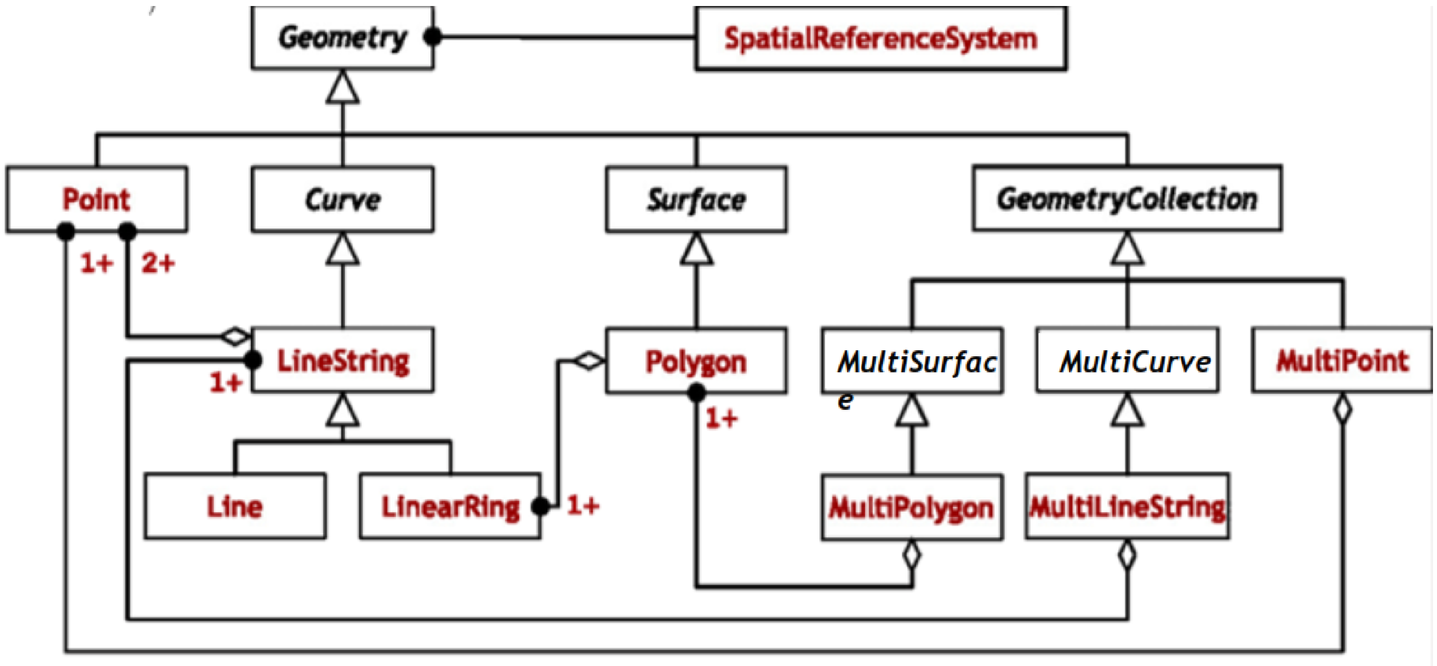
```
CREATE TABLE borders (
IDborders VARCHAR(20) PRIMARY KEY);

SELECT AddGeometryColumn('borders',
'geometry', 4326, 'LINESTRING', 2);
```

IDborders	geometry
GR_B042	

OGC simple features

Simple (geometrical) features hierarchy, according to OGC



Queries for spatial data

Example of getting a geometry

Query

```
SELECT name, ST_astext(lake_geom)  FROM lakes;
WHERE name =  'Δοϊράνη';
```

Result

name	lake_geom
Δοϊράνη	POLYGON ((22.7209 41.2390, 22.7109 41.2324, 22.7082 41.2273, ... 22.7209 41.2390));

Example of a topological relationship

```
SELECT lakes.name
FROM lakes, borders
WHERE CROSSES (lakes.lake_geom, borders.geometry)=1;
```