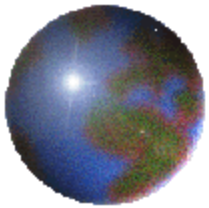
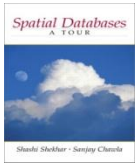


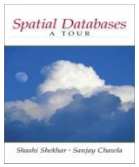
Chapter 3:Spatial Query Languages



Learning Objectives

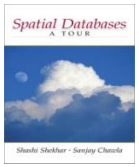
- Learning Objectives (LO)
  - LO1: Understand concept of a query language
  - LO2 : Learn to use standard query language (SQL)
  - LO3: Learn to use spatial ADTs with SQL
    - Learn about OGIS standard spatial data types and operations
    - Learn to use OGIS spatial ADTs with SQL
  - LO4: Learn about the trends in query languages

- Mapping Sections to learning objectives
  - LO2 - 3.2, 3.3
  - LO3 - 3.4, 3.5
  - LO4 - 3.6



3.4 Extending SQL for Spatial Data

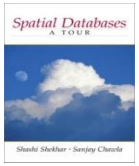
- Motivation
  - SQL has simple atomic data-types, like integer, dates and string
  - Not convenient for spatial data and queries
    - Spatial data (e.g. polygons) is complex
    - Spatial operation: topological, euclidean, directional, metric
- SQL 3 allows user defined data types and operations
  - Spatial data types and operations can be added to SQL3
- Open Geodata Interchange Standard (OGIS)
  - Half a dozen spatial data types
  - Several spatial operations
  - Supported by major vendors, e.g. ESRI, Intergraph, Oracle, IBM,...



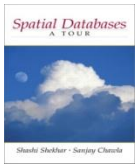
OGIS Spatial Data Model

- Consists of base-class `Geometry` and four sub-classes:
  - `Point, Curve, Surface` and `GeometryCollection`

- Operations fall into three categories:
  - Apply to all geometry types
    - SpatialReference, Envelope, Export,IsSimple, Boundary
  - Predicates for Topological relationships
    - Equal, Disjoint, Intersect, Touch, Cross, Within, Contains
  - Spatial Data Analysis
    - Distance,Buffer,Union, Intersection, ConvexHull, SymDiff
  - Table in next slide details spatial operations

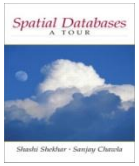


Basic Functions	`SpatialReference()`	Returns the underlying coordinate system of the geometry
	`Envelope()`	Returns the minimum orthogonal bounding rectangle of the geometry
	`Export()`	Returns the geometry in a different representation
	`IsEmpty()`	Returns true if the geometry is a null set.
	`IsSimple()`	Returns true if the geometry is simple (no self-intersection)
	`Boundary()`	Returns the boundary of the geometry
Topological/ Set Operators	`Equal`	Returns true if the interior and boundary of the two geometries are spatially equal
	`Disjoint`	Returns true if the boundaries and interior do not intersect.
	`Intersect`	Returns true if the geometries are not disjoint
	`Touch`	Returns true if the boundaries of two surfaces intersect but the interiors do not.
	`Cross`	Returns true if the interior a surface intersects with a curve
	`Within`	Returns true if the interior of the given geometry does not intersect with the exterior of another geometry.
	`Contains`	Tests if the given geometry contains another given geometry
	`Overlap`	Returns true if the interiors of two geometries have non-empty intersection
Spatial Analysis	`Distance`	Returns the shortest distance between two geometries
	`Buffer`	Returns a geometry that consists of all points whose distance from the given geometry is less than or equal to the specified distance
	`ConvexHull`	Returns the smallest convex geometric set enclosing the geometry
	`Intersection`	Returns the geometric intersection of two geometries
	`Union`	Returns the geometric union of two geometries
	`Difference`	Returns the portion of a geometry which does not intersect with another given geometry
	`SymmDiff`	Returns the portions of two geometries which do not intersect with each other



List of Spatial Query Examples

- Simple SQL SELECT_FROM_WHERE examples
  - Spatial analysis operations
    - Unary operator: Area (Q5, pp.68)
    - Binary operator: Distance (Q3)
  - Boolean Topological spatial operations - WHERE clause
    - Touch (Q1, pp. 67)
    - Cross (Q2, pp. 68)
  - Using spatial analysis and topological operations
    - Buffer, overlap (Q4)
- Complex SQL examples
  - Aggreagate SQL queries
  - Nested queries



Example schema

```
CREATE  TABLE   Country(
        Name    varchar(30),
        Cont    varchar(30),
        Pop     Integer,
        GDP     Number,
        Shape   Polygon);
```

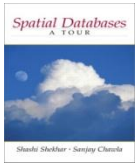
(a)

```
CREATE  TABLE   River(
        Name    varchar(30),
        Origin  varchar(30),
        Length  Number,
        Shape   LineString);
```

(b)

```
CREATE  TABLE   City (
        Name    varchar(30),
        Country varchar(30),
        Pop     integer,
        Shape   Point );
```

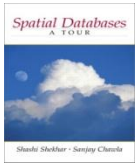
(c)



Query 1

Query: Find the names of all countries which are neighbors of *USA* in the `Country` table.

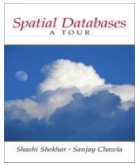
```
SELECT   C1.Name AS "Neighbors of USA"
FROM     Country C1, Country C2
WHERE    Touch(C1.Shape, C2.Shape) = 1 AND
         C2.Name = 'USA'
```



Query 2

Query: For all the rivers listed in the `River` table, find the countries through which they pass.

```
SELECT   R.Name C.Name
FROM     River R, Country C
WHERE    Cross(R.Shape, C.Shape) = 1
```

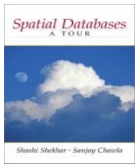


Query 3

Query: For each river, find its closest city.

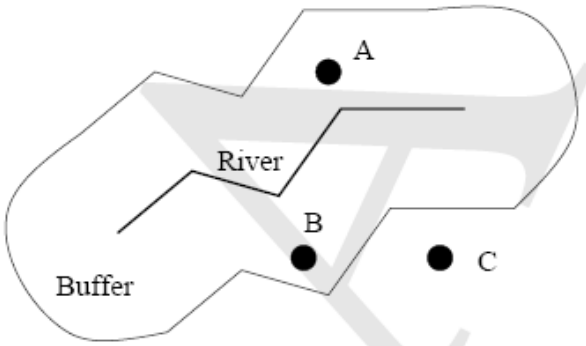
```
SELECT C1.Name, R1.Name
FROM   City C1, River R1
WHERE  Distance (C1.Shape, R1.Shape) <
        ALL (SELECT Distance(C2.Shape, R1.Shape)
             FROM   City C2
             WHERE  C1.Name <> C2.Name
             )
```

Comments: How is Distance computed between line and point?
Operator overloading or multiple redefinitions?

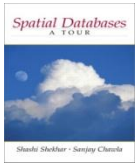


Query 4

Query: The St. Lawrence river can supply water to cities which are within 300 km. List the cities which can use water from the St. Lawrence.




```
SELECT   Ci.Name
FROM     City Ci, River R
WHERE    Overlap(Ci.Shape, Buffer(R.Shape,300)) = 1 AND
         R.Name = 'St.  Lawrence'
```

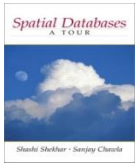


Query 5

Query: List the name, population, and area of each country listed in the `Country` table.

Use: Area(O.Shape)

```
SELECT   C.Name, C.Pop, Area(C.Shape) AS "Area"
FROM     Country C
```



Query 6

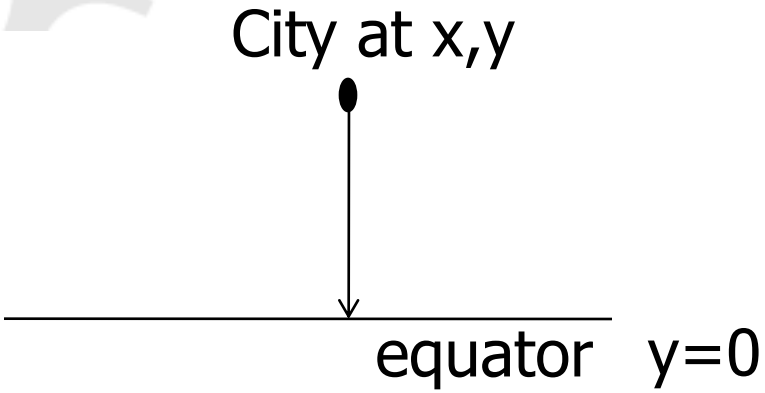
Query: List the length of the rivers in each of the countries they pass through.

```
SELECT   R.Name, C.Name , Length(Intersection(R.Shape, C.Shape))
         AS "Length"
FROM     River R, Country C
WHERE    Cross(R.Shape, C.Shape) = 1
```

Query 7

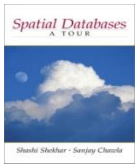
Query: List the GDP and the distance of a country's capital city to the equator for all countries.

Use: Point(x,y) to construct new points and C.x or C.y to get coordinates of points




```
SELECT   Co.GDP, Distance(Point(0,Ci.y),Ci.Shape) AS "Distance"
FROM     Country Co, City Ci
WHERE    Co.Name = Ci.Country AND
         Ci.Capital = 'Y'
```

Co. Name	Co. GDP	Dist-to-Eq (in Km).
Havana	16.9	2562
Washington, D.C.	8003	4324
Brasilia	1004	1756
Ottawa	658	5005
Mexico City	694.3	2161
Buenos Aires	348.2	3854

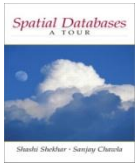


Query 8

Query: List all countries, ordered by number of neighboring countries.

```
SELECT     Co.Name, Count(Co1.Name)
FROM       Country Co, Country Co1
WHERE      Touch(Co.Shape, Co1.Shape)
GROUP BY   Co.Name
ORDER BY   Count(Co1.Name)
```

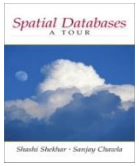
What about countries with no neighbors (Island)?



Query 9

Query: List the countries with only one neighboring country. A country is a neighbor of another country if their land masses share a boundary.

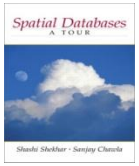
```
SELECT     Co.Name
FROM       Country Co, Country Co1
WHERE      Touch(Co.Shape, Co1.Shape))
GROUP BY   Co.Name
HAVING     Count(Co1.Name) = 1
```



Query 10

Query: Which country has the maximum number of neighbors?

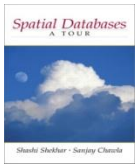
```
CREATE VIEW Neighbor AS
SELECT                  Co.Name, Count(Co1.Name) AS num_neighbors
FROM                    Country Co, Country Co1
WHERE                   Touch(Co.Shape, Co1.Shape)
GROUP BY                Co.Name


SELECT  Co.Name, num_neighbors
FROM    Neighbor
WHERE   num_neighbor = (SELECT Max(num_neighbors)
        FROM Neighbor)
```



Learning Objectives

- Learning Objectives (LO)
  - LO1: Understand concept of a query language
  - LO2 : Learn to use standard query language (SQL)
  - LO3: Learn to use spatial ADTs with SQL
  - LO4: Learn about the trends in query languages
    - Facilities for user defined data types in SQL3

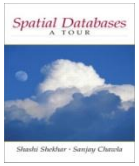
- Mapping Sections to learning objectives
  - LO2 - 3.2, 3.3
  - LO3 - 3.4, 3.5
  - LO4 - 3.6



Defining Spatial Data Types in SQL3

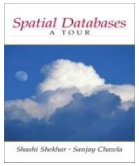
- SQL3 User defined data type - Overview
  - CREATE TYPE statements
  - Defines a new data types
  - Attributes and methods are defined
  - Separate statements for interface and implementation
    - Examples of interface in Table 3.12 (pp. 74)

- Additional effort is needed at physical data model level



Examples (Point)

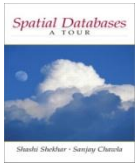
```
CREATE  TYPE Point AS OBJECT (
        x   NUMBER,
        y   NUMBER,
        MEMBER FUNCTION Distance(P2   IN Point) RETURN NUMBER,
        PRAGMA RESTRICT_REFERENCES(Distance, WNDS);

           CREATE  TABLE    City (
                   Name     varchar(30),
                   Pop      int,
                   Capital  char(1),
                   Shape    Point   );

INSERT INTO CITY('Brasilia', 'Brazil', 1.5, 'Y',
                 Point(-55.4,-23.2));
```



Examples (LineString)

```
CREATE TYPE LineType AS VARRAY(500) OF Point;

CREATE  TYPE LineString AS OBJECT (
        Num_of_Points INT,
        Geometry LineType,
        MEMBER FUNCTION Length(SELF IN) RETURN NUMBER,
        PRAGMA RESTRICT_REFERENCES(Length, WNDS);

 CREATE  TABLE   River(
         Name    varchar(30),
         Origin  varchar(30),
         Length  number,
         Shape   LineString   );
```

```
INSERT INTO RIVER('Mississippi', 'USA', 6000,
                  LineString(3, LineType(Point(1,1),Point(1,2),Point(2,3)))
```



Examples (Polygon)

```
CREATE TYPE PolyType AS VARRAY(500) OF Point

CREATE  TYPE Polygon AS OBJECT (
        Num_of_Points INT,
        Geometry PolyType ,
        MEMBER FUNCTION Area(SELF IN) RETURN NUMBER,
        PRAGMA RESTRICT_REFERENCES(Length, WNDS);

CREATE  TABLE     Country(
        Name      varchar(30),
        Cont      varchar(30),
        Pop       int,
        GDP       number,
        Life-Exp  number,
        Shape     LineString  );

INSERT INTO  Country('Mexico', 'NAM', 107.5, 694.3, 1004.0,
             Polygon(23, Polytype(Point(1,1), ..., Point(1,1)))
```