# I sistemi per la gestione delle basi di dati geografiche

*Un modello di riferimento*
*per la progettazione logica*

2007

Alberto Belussi

# Sistemi per la Gestione di Basi di Dati Geografiche

Premessa

- Consideriamo il modello relazionale come modello di riferimento per i sistemi che gestiscono basi di dati tradizionali.
- La componente geometrica del dato geografico non può essere rappresentata nel modello relazionale utilizzando un insieme di attributi di tipo numerico che contengano coordinate, in quanto:
  1. I metodi tradizionali per l'accesso ai dati (indici) non sono adeguati per le interrogazioni spaziali,
  2. l'utente non e' in grado di interagire con un'interfaccia che spezza i valori geometrici in coordinate semplici,
  3. non esiste supporto alle interrogazioni spaziali.

# Sistemi per la Gestione di Basi di Dati Geografiche

- Un sistema per la gestione di basi di dati basato sul paradigma Object-Oriented non può essere adottato per due motivi:
  - le prestazioni di questi sistemi sono ancora insufficienti per questo tipo di applicazioni,
  - l'integrazione con basi di dati relazionali è difficoltosa.

**Soluzione attuale**

Sistemi dedicati al dato geografico (Geo-DBMS) o sistemi tradizionali (DBMS ) con estensioni specifiche per la gestione del dato geografico.

Tali sistemi adottano un modello dei dati geo-relazionale.

# Geo-DBMS

Definizione di Geo-DBMS secondo Güting (1994):

- un sistema per la gestione di dati geografici (Geo-DBMS) e' innanzitutto un sistema per la gestione di dati (DBMS);
- esso offre tipi predefiniti (Spatial Data Types - SDT) per rappresentare il dato geografico nello schema e per trattare il dato geografico nel linguaggio di interrogazione;
- esso offre un'implementazione dei dati geografici che include l'accesso ai dati geometrici attraverso un indice spaziale e un algoritmo efficiente per l'operazione di join spaziale.

# Il modello dei dati di un Geo-DBMS

La maggior parte degli attuali Geo-DBMS adotta un **modello dei dati geo-relazionale**.

Tale modello è un'estensione del modello relazionale che presenta domini e operatori specifici per rappresentare e manipolare il dato geografico.

*Quali estensioni sono necessarie*
*al modello relazionale per trattare il dato geografico?*

1. Introduzione di tipi specifici per rappresentare il dato spaziale.
2. Estensione del linguaggio di interrogazione per esprimere condizioni di selezione basate su proprietà spaziali.

# Modelli dei dati esistenti

- OpenGIS® Simple Features Specification For SQL (attualmente OpenGIS® Simple Features Access – part 2: SQL option)
- Modello dei dati di Oracle 10g
- Modello dei dati di ESRI Shapefile
- Modello dei dati di ESRI GeoDatabase
- Modello dei dati di altri sistemi (MGE Intergraph, MapInfo, Informix)

# OpenGIS® Simple Features Specification For SQL (SFS)

- Hanno contribuito alla sua definizione: ESRI Inc., IBM Corporation, Informix Software Inc., MapInfo Corporation, Oracle Corporation.
- Versione precedente: *OpenGIS Project Document 99-049 Date: May 5, 1999*.
- Attuale versione: *OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option Date: October 5, 2006*

Di seguito descriviamo nel dettaglio il modello.

# SFS :Feature Tables

Simple geospatial feature collections will conceptually be stored as tables with geometry valued columns in a Relational DBMS (RDBMS), each feature will be stored as a row in a table.

The non-spatial attributes of features will be mapped onto columns whose types are drawn from the set of standard SQL92 data types.

The spatial attributes of features will be mapped onto columns whose SQL data types are based on the underlying concept of additional geometric data types for SQL.
A table whose rows represent Open GIS features shall be referred to as a **feature table**.

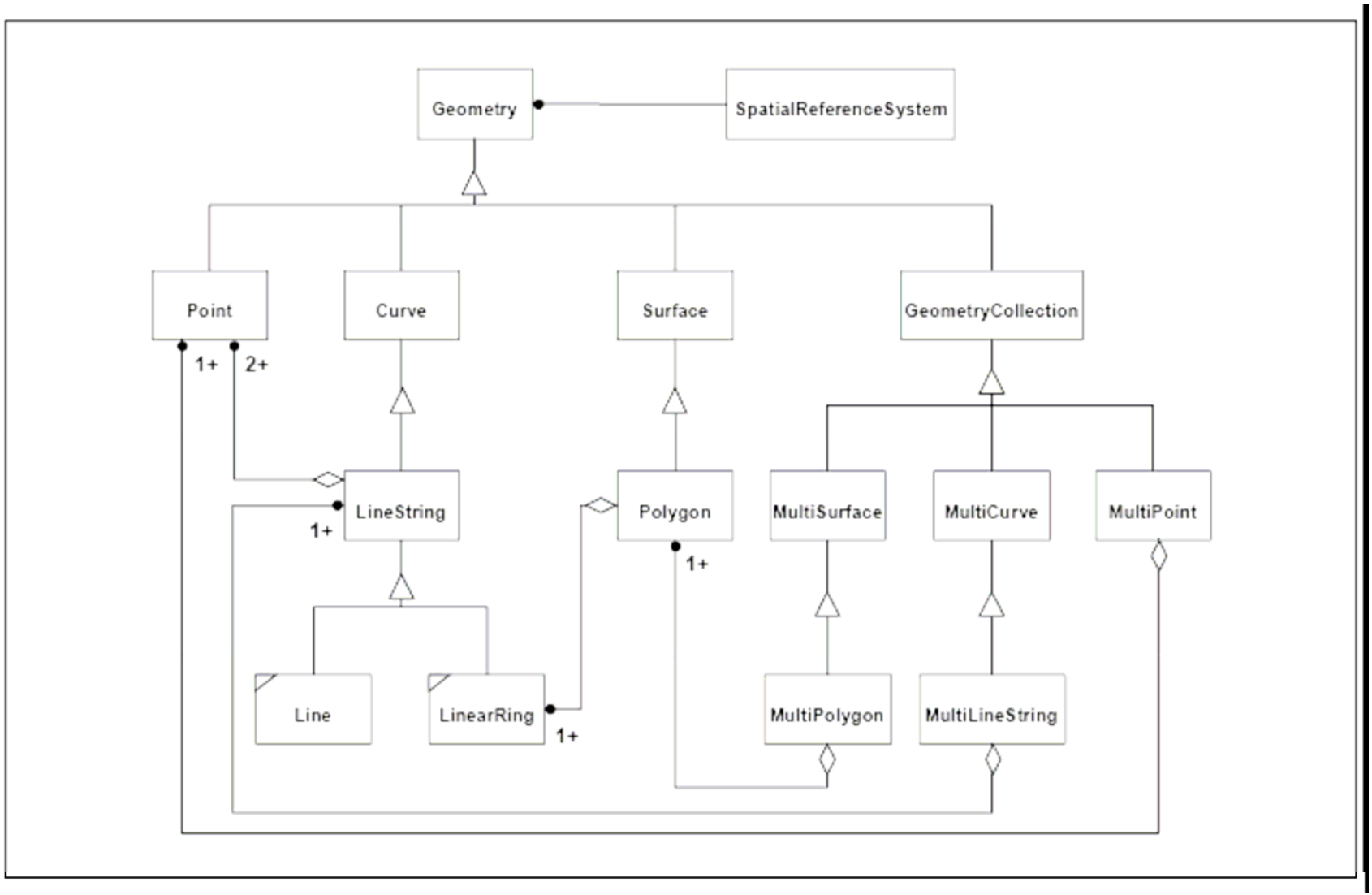
# SFS :Feature Tables

Feature table implementations are described for two target SQL environments: **SQL92** and **SQL92 with Geometry Types.**

We consider only the **SQL92 with Geometry Types** implementation.

The term **SQL92 with Geometry Types** is used to refer to a SQL92 environment that has been extended with a set of Geometry Types.

# SFS: Geometry Object Model



# SFS: Geometry class

Geometry is the root class of the hierarchy. Geometry is an **abstract** (non-instantiable) **class**.

The instantiable subclasses of Geometry are restricted to 0, 1 and two-dimensional geometric objects that exist in two-dimensional coordinate space ($\Re^2$).

All instantiable geometry classes are defined so that valid instances of a geometry class are **topologically closed** (i.e. all defined geometries include their boundary).

# SFS: basic methods on Geometry

**GeometryType** ( ):String —Returns the name of the instantiable subtype of Geometry of which *this* Geometry instance is a member.

**Dimension** ( ):Integer—The inherent dimension of *this* Geometry object, which must be less than or equal to the coordinate dimension.

**SRID** ( ):Integer—Returns the Spatial Reference System ID for *this* Geometry.

**Envelope**( ):Geometry—The minimum bounding box for *this* Geometry.

# SFS: basic methods on Geometry

**AsText**( ):String —Exports *this* Geometry to a specific well-known text representation of Geometry.

**AsBinary**( ):Binary—Exports *this* Geometry to a specific well-known binary representation of Geometry.

**IsEmpty**( ):Integer —Returns 1 (TRUE) if *this* Geometry is the empty geometry . If true, then *this* Geometry represents the empty point set, ∅, for the coordinate space.

**IsSimple**( ):Integer —Returns 1 (TRUE) if *this* Geometry has no anomalous geometric points, such as self intersection or self tangency.

# SFS: basic methods on Geometry

**Boundary**( ):Geometry —Returns the closure of the combinatorial boundary of *this* Geometry.

**Distance**(anotherGeometry:Geometry):Double—Returns the shortest distance between any two points in the two geometries as calculated in the spatial reference system of *this* Geometry.

**Buffer**(distance:Double):Geometry—Returns a geometry that represents all points whose distance from *this* Geometry is less than or equal to distance. Calculations are in the Spatial Reference System of *this* Geometry.

# SFS: basic methods on Geometry

**ConvexHull**( ):Geometry—Returns a geometry that represents the convex hull of *this* Geometry.

**Intersection**(anotherGeometry:Geometry):Geometry—Returns a geometry that represents the point set intersection of *this* Geometry with anotherGeometry.

**Union**(anotherGeometry:Geometry):Geometry—Returns a geometry that represents the point set union of *this* Geometry with anotherGeometry.

**Difference**(anotherGeometry:Geometry):Geometry—Returns a geometry that represents the point set difference of *this* Geometry with anotherGeometry.

**SymDifference**(anotherGeometry:Geometry):Geometry—Returns a geometry that represents the point set symmetric difference of *this* Geometry with anotherGeometry.

# SFS: Geometric collection

A GeometryCollection is a geometry that is a collection of 1 or more geometries.

All the elements in a GeometryCollection must be in the same Spatial Reference. This is also the Spatial Reference for the GeometryCollection.

Methods:

**NumGeometries**( ):Integer—Returns the number of geometries in *this* GeometryCollection.

**GeometryN**(N:integer):Geometry—Returns the Nth geometry in *this* GeometryCollection.

# SFS: Point

A Point is a 0-dimensional geometry and represents a single location in coordinate space. A Point has a x-coordinate value and a y-coordinate value.

The boundary of a Point is the empty set.

Methods:

**X**( ):Double —The x-coordinate value for *this* Point.

**Y**( ):Double —The y-coordinate value for *this* Point.

# SFS: Multipoint

A MultiPoint is a 0 dimensional geometric collection. The elements of a MultiPoint are restricted to Points.

The points are not connected or ordered.

A MultiPoint is simple if no two Points in the MultiPoint are equal (have identical coordinate values).

The boundary of a MultiPoint is the empty set.

# SFS: Curve

A Curve is a one-dimensional geometric object usually stored as a sequence of points, with the subtype of Curve specifying the form of the interpolation between points.

This specification defines only one subclass of Curve, LineString, which uses linear interpolation between points.

Topologically a Curve is a one-dimensional geometric object that is the homeomorphic image of a real, closed, interval

$$D = [a, b] = \{x \in R \mid a <= x <= b\}$$

under a mapping

$$f:[a,b] \rightarrow \Re^2$$

# SFS: Curve properties

- A Curve is **simple** if it does not pass through the same point twice.
- A Curve is **closed** if its start point is equal to its end point.
- The boundary of a closed Curve is empty.
- A Curve that is simple and closed is a **Ring**.
- The boundary of a non-closed Curve consists of its two end points.
- A Curve is defined as **topologically closed**.

# SFS: Curve Methods

- **Length**( ):Double—The length of *this* Curve in its associated spatial reference.
- **StartPoint**( ):Point—The start point of *this* Curve.
- **EndPoint**( ):Point—The end point of *this* Curve.
- **IsClosed**( ):Integer—Returns 1 (TRUE) if *this* Curve is closed (StartPoint ( ) = EndPoint ( )).
- **IsRing**( ):Integer—Returns 1 (TRUE) if *this* Curve is closed (StartPoint ( ) = EndPoint ( )) and *this* Curve is simple (does not pass through the same point more than once).

# SFS: LineString, Line, LinearRing

A **LineString** is a Curve with linear interpolation between points. Each consecutive pair of points defines a **line segment**.

A **Line** is a LineString with exactly 2 points.

A **LinearRing** is a LineString that is both closed and simple.

# SFS: LineString, LinearRing (examples)

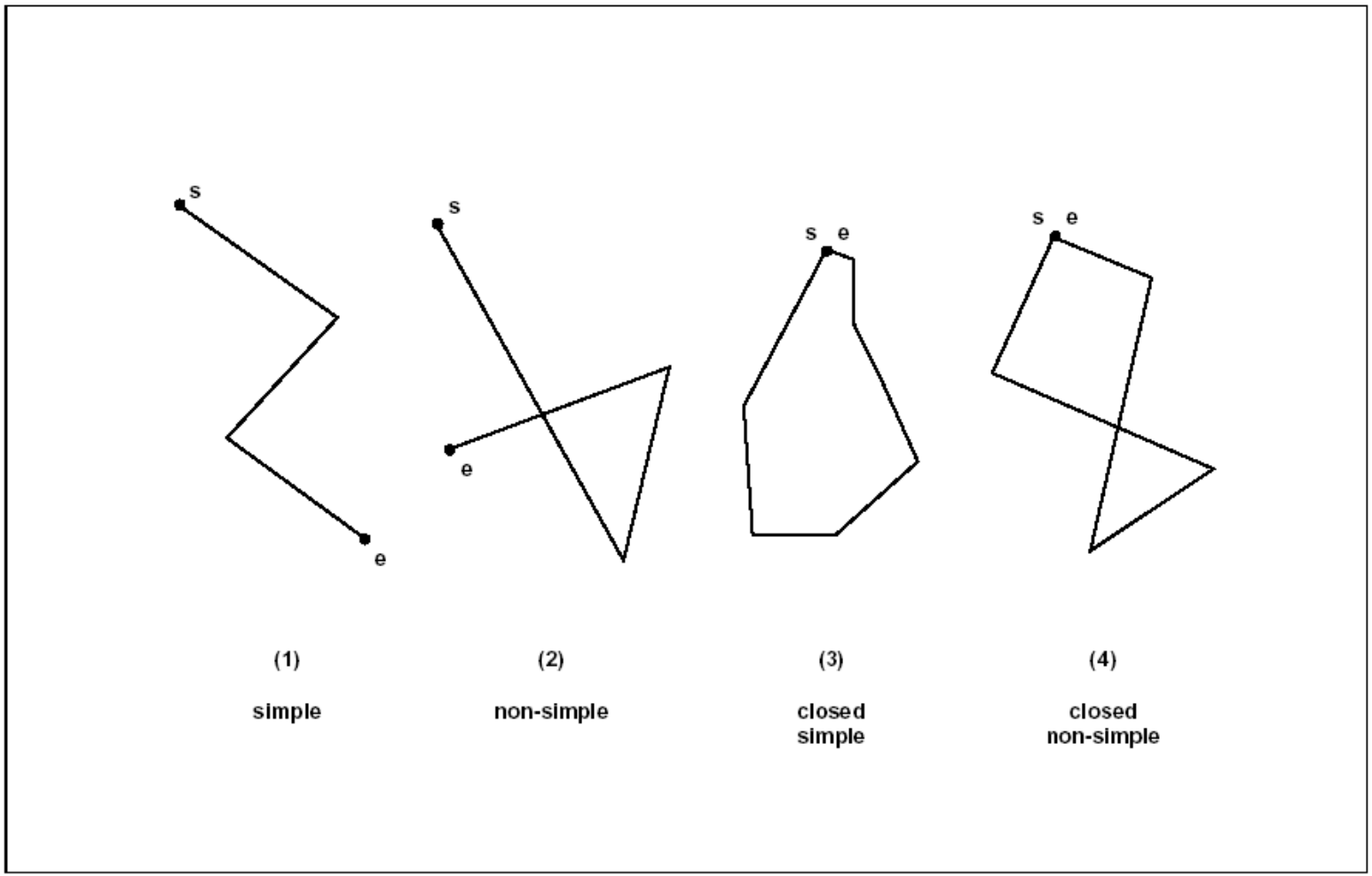



**Figure 2.2—(1) a simple LineString, (2) a non-simple LineString, (3) a simple, closed LineString (a LinearRing), (4) a non-simple closed LineString**

# SFS: LineString methods

- **NumPoints**( ):Integer—The number of points in *this* LineString.
- **PointN**(N:Integer):Point—Returns the specified point N in *this* Linestring.

# SFS: MultiCurve

A MultiCurve is a one-dimensional GeometryCollection whose elements are Curves.

MultiCurve is a non-instantiable class.

A MultiCurve is **simple** if and only if all of its elements are simple and the only intersections between any two elements occur at points that are on the boundaries of both elements.

A MultiCurve is **closed** if all of its elements are closed.

A MultiCurve is defined as *topologically closed*.

# SFS: MultiCurve

The boundary of a MultiCurve is obtained by applying the ‘mod 2’ union rule: a point is in the boundary of a MultiCurve if it is in the boundaries of an odd number of elements of the MultiCurve.

Methods:

**IsClosed**( ):Integer—Returns 1 (TRUE) if *this* MultiCurve is closed (StartPoint ( ) = EndPoint ( ) for each curve in *this* MultiCurve)

**Length**( ):Double—The Length of *this* MultiCurve which is equal to the sum of the lengths of the element Curves.

# SFS: MultiCurve examples

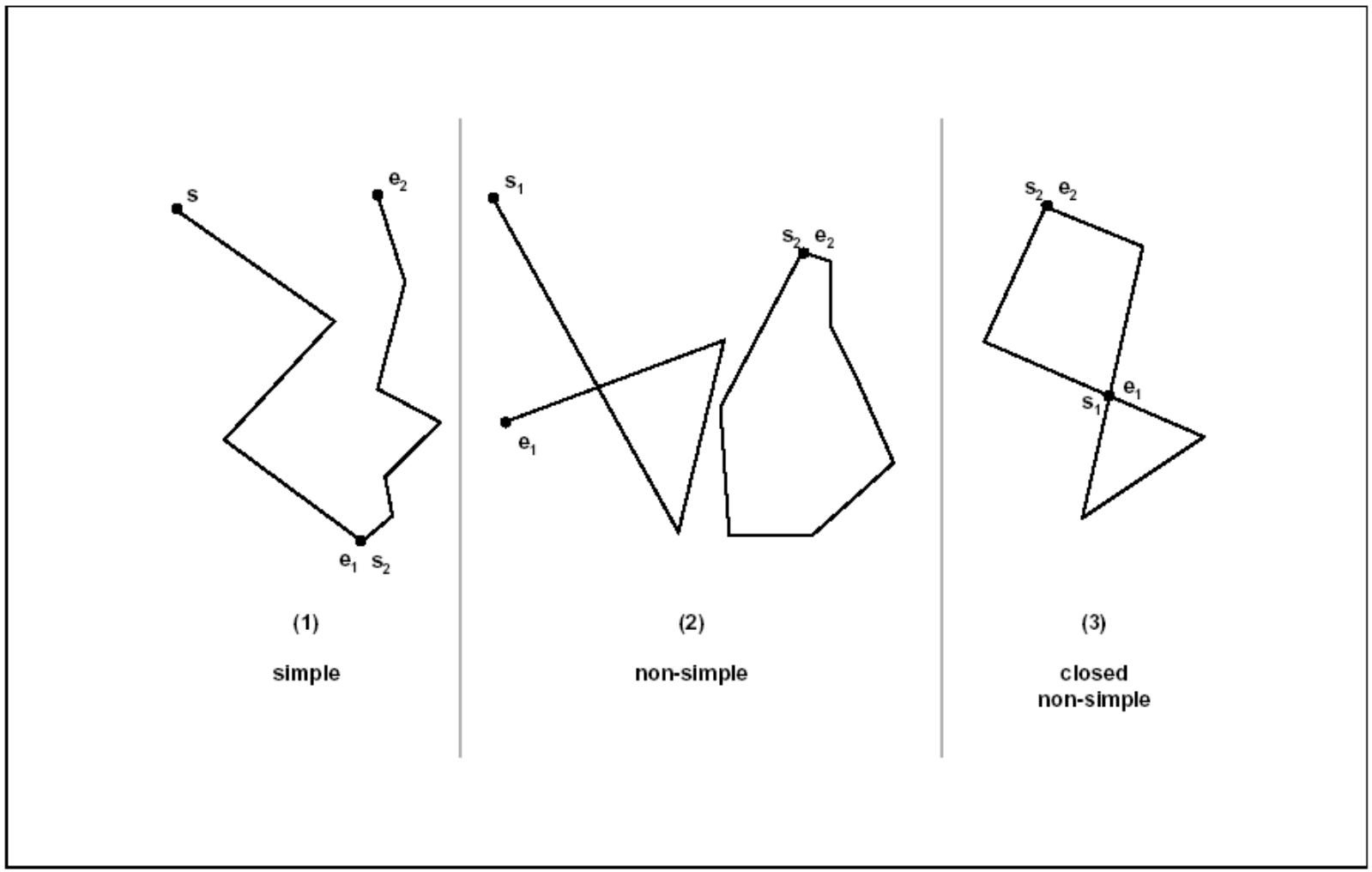



**Figure 2.3—(1) a simple MultiLineString, (2) a non-simple MultiLineString with 2 elements, (3) a non-simple, closed MultiLineString with 2 elements**

# SFS: Surface

A Surface is a two-dimensional geometric object.

The OpenGIS Abstract Specification defines a **simple Surface** as consisting of a single 'patch' that is associated with one 'exterior boundary' and 0 or more 'interior' boundaries.

The boundary of a simple Surface is the set of closed curves corresponding to its 'exterior' and 'interior boundaries.

The only instantiable subclass of Surface defined in this specification, **Polygon**, is a simple Surface that is planar.

# SFS: Surface methods

**Area**( ):Double—The area of *this* Surface, as measured in the spatial reference system of *this* Surface.

**Centroid**( ):Point—The mathematical centroid for *this* Surface as a Point. The result is not guaranteed to be on *this* Surface.

**PointOnSurface**( ):Point—A point guaranteed to be on *this* Surface.

# SFS: Polygon

A **Polygon** is a planar Surface, defined by 1 exterior boundary and 0 or more interior boundaries.

Polygons are topologically closed.

The **boundary** of a Polygon consists of a set of LinearRings that make up its exterior and interior boundaries.

The **interior** of every Polygon is a connected point set.

# SFS: Polygon examples

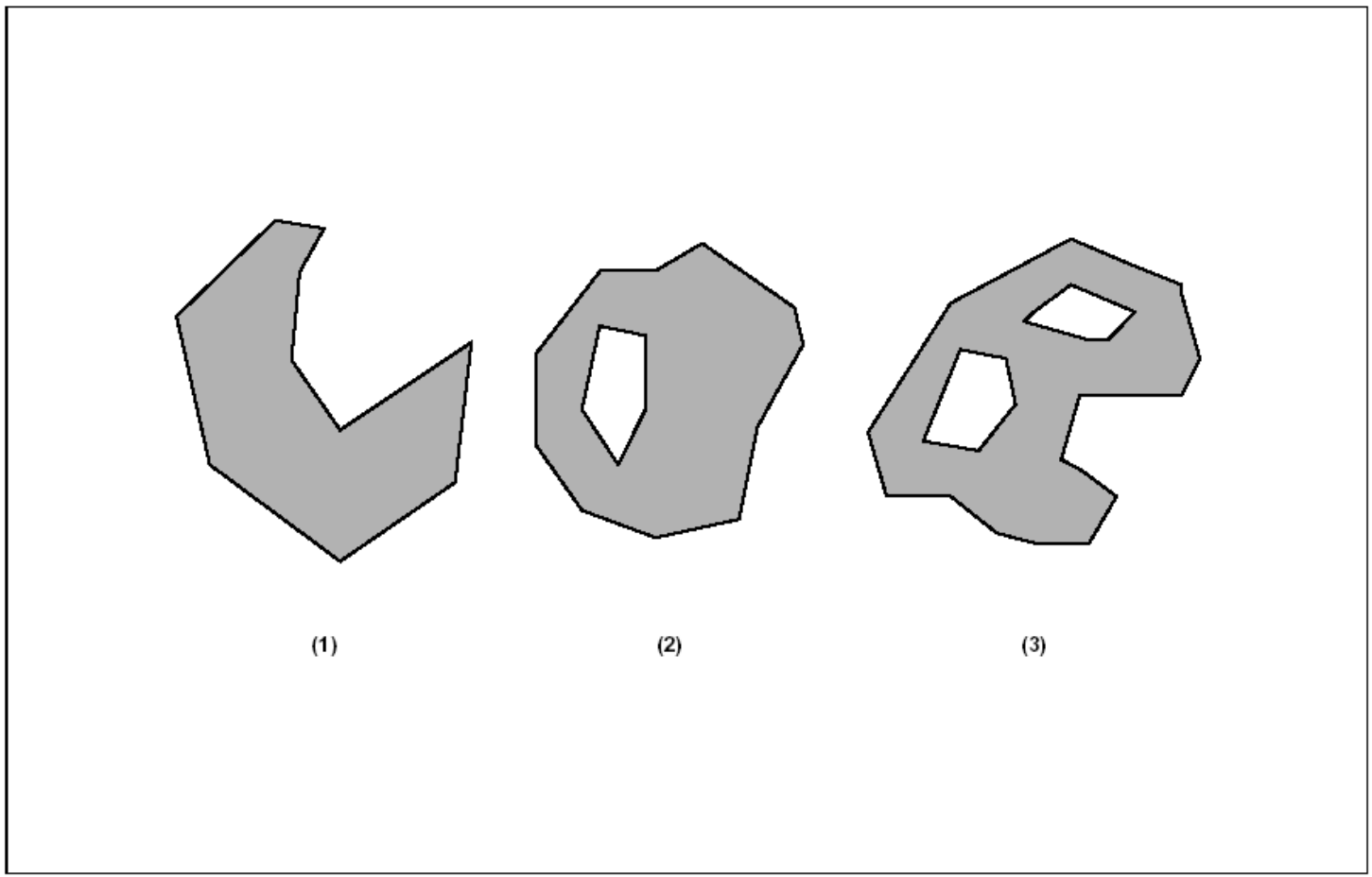



**Figure 2.4—Examples of Polygons with 1, 2 and 3 rings respectively.**

# SFS: Polygon examples

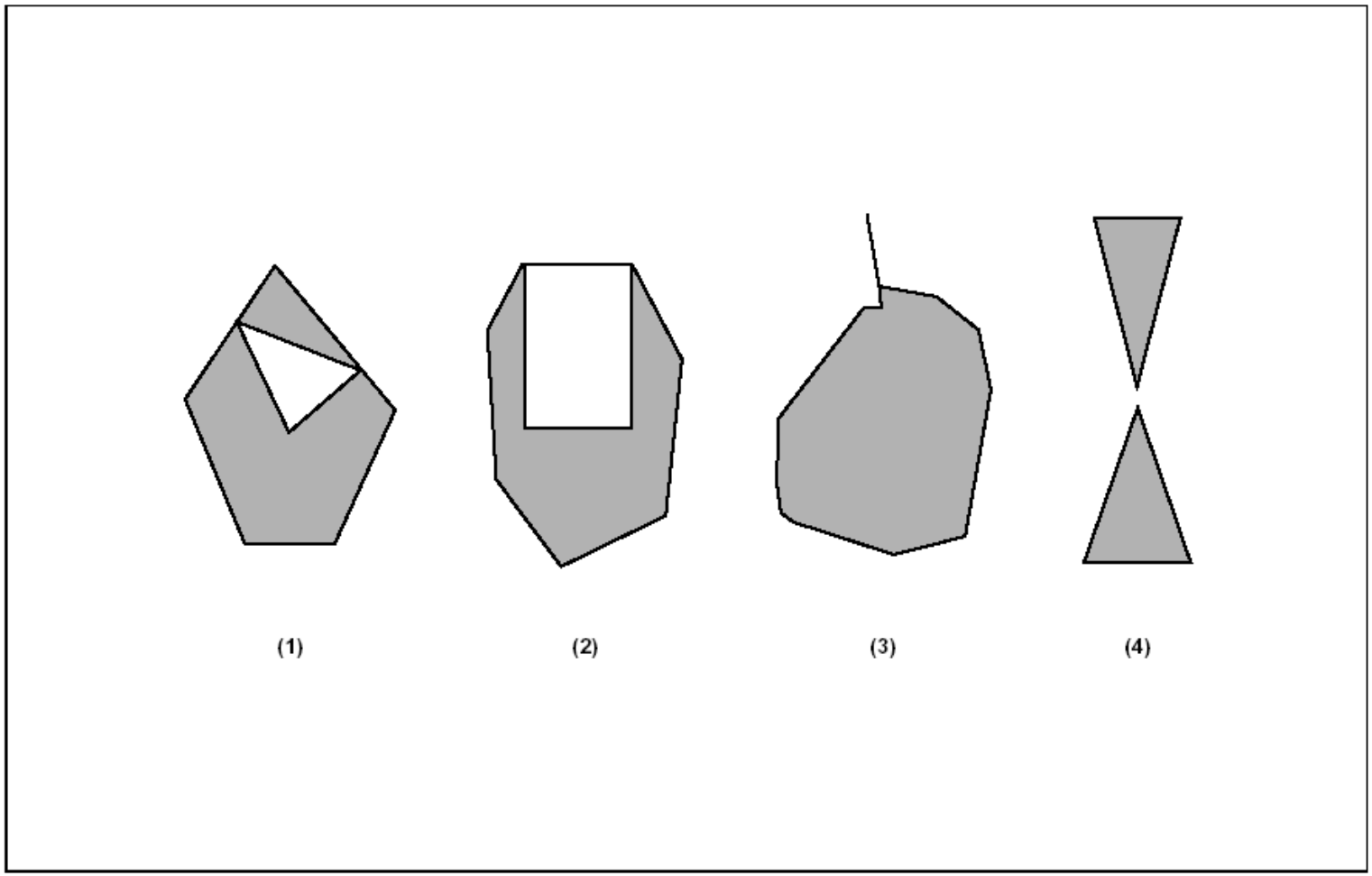



**Figure 2.5—Examples of objects not representable as a single instance of Polygon. (1) and (4) can be represented as 2 separate Polygons.**

# SFS: Polygon methods

**ExteriorRing**( ):LineString—Returns the exterior ring of *this* Polygon.

**NumInteriorRing**( ):Integer—Returns the number of interior rings in *this* Polygon.

**InteriorRingN**(N:Integer):LineString—Returns the Nth interior ring for *this* Polygon as a LineString.

# SFS: MultiSurface

A **MultiSurface** is a two-dimensional geometric collection whose elements are surfaces.

The interiors of any two surfaces in a MultiSurface may not intersect.

The boundaries of any two elements in a MultiSurface may intersect at most at a finite number of points.

MultiSurface is a **non-instantiable** class in this specification.

The instantiable subclass of MultiSurface is **MultiPolygon**, corresponding to a collection of Polygons.

# SFS: MultiSurface methods

**Area**( ):Double—The area of *this* MultiSurface, as measured in the spatial reference system of *this* MultiSurface.

**Centroid**( ):Point—The mathematical centroid for *this* MultiSurface. The result is not guaranteed to be on *this* MultiSurface.

**PointOnSurface**( ):Point—A Point guaranteed to be on *this* MultiSurface.

# SFS: MultiPolygon

A **MultiPolygon** is a MultiSurface whose elements are Polygons..

The boundary of a MultiPolygon is a set of closed curves (LineStrings) corresponding to the boundaries of its element Polygons.

Each Curve in the boundary of the MultiPolygon is in the boundary of exactly 1 element Polygon, and every Curve in the boundary of an element Polygon is in the boundary of the MultiPolygon.

# SFS: MultiPolygon example

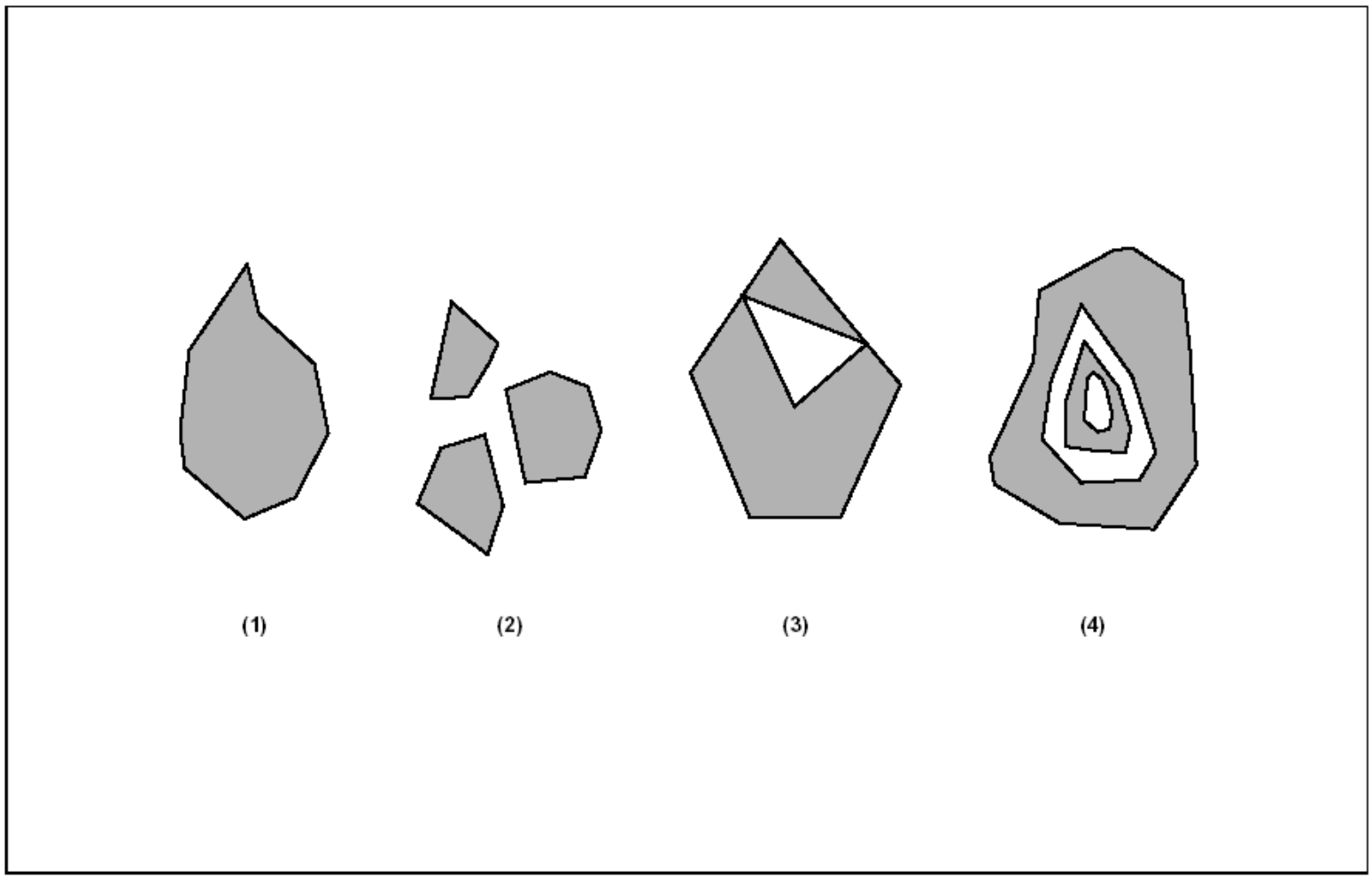



Figure 2.6—Examples of MultiPolygons

# SFS: Relational Operators

**Equals**(anotherGeometry:Geometry):Integer

Returns 1 (TRUE) if *this* Geometry is *'spatially equal'* to anotherGeometry.

**Disjoint**(anotherGeometry:Geometry):Integer

Returns 1 (TRUE) if *this* Geometry is *'spatially disjoint'* from anotherGeometry.

**Touches**(anotherGeometry:Geometry):Integer

Returns 1 (TRUE) if *this* Geometry *'spatially touches'* anotherGeometry.

**Overlaps**(anotherGeometry:Geometry):Integer

Returns 1 (TRUE) if *this* Geometry *'spatially overlaps'* anotherGeometry.

# SFS: Relational Operators

**Within**(anotherGeometry:Geometry):Integer

Returns 1 (TRUE) if *this* Geometry is '*spatially within'* anotherGeometry.

**Contains**(anotherGeometry:Geometry):Integer

Returns 1 (TRUE) if anotherGeometry is *'spatially within' this* Geometry.

**Crosses**(anotherGeometry:Geometry):Integer

Returns 1 (TRUE) if *this* Geometry *'spatially crosses'* anotherGeometry.

**Intersects**(anotherGeometry:Geometry):Integer

Returns 1 (TRUE) if *this* Geometry is *'not spatially disjoint'* from anotherGeometry.

# SFS: Relational Operators

**Relate**(anotherGeometry:Geometry, intersectionPatternMatrix:String):Integer

Returns 1 (TRUE) if *this* Geometry is spatially related to anotherGeometry, by testing for intersections between the Interior, Boundary and Exterior of the two geometries.

The pattern matrix consists of a set of 9 pattern-values, one for each cell *p* in the matrix. The possible pattern values *p* are {T, F, *, 0, 1, 2}.

For example:

```
overlapMatrix = 'T*T***T**';
```

# Components of a GeoRelational-database

A GeoRelational-database is composed of:

- The **Spatial Reference Systems table**, which is named SPATIAL_REF_SYS, stores information on each spatial reference system used in the database.
- The **Geometric Columns Metadata table or view,** which is named GEOMETRY_COLUMNS, provides metadata information on the spatial reference for each geometry column in the database.
- The **SQL Geometry Types** extend the set of available SQL92 types to include Geometry Types.
- A set of **Feature Tables**: each feature table is a table containing one or more geometric attributes that are modeled using a column whose type correspond to a SQL Geometry Type. Feature-to-feature relations are defined as FOREIGN KEY references**.**
- A set of traditional **Tables**.

# Components of a GeoRelational-database

**Spatial Reference Systems table**

```
CREATE TABLE SPATIAL_REF_SYS
(
    SRID INTEGER NOT NULL PRIMARY KEY,
    AUTH_NAME VARCHAR (256),
    AUTH_SRID INTEGER,
    SRTEXT VARCHAR (2048)
)
```

# Components of a GeoRelational-database

**Spatial Reference Systems table**
**Attributes:**

- SRID: an integer value that uniquely identifies each Spatial Reference System within a database.
- AUTH_NAME—the name of the standard or standards body that is being cited for this reference system. EPSG would be a valid AUTH_NAME
- AUTH_SRID—the ID of the Spatial Reference System as defined by the Authority cited in AUTH_NAME.
- SRTEXT—The Well-known Text representation of the Spatial Reference System.

# Components of a GeoRelational-database

**Geometric Columns Metadata table or view**

```
CREATE TABLE GEOMETRY_COLUMNS (
    F_TABLE_CATALOG VARCHAR(256) NOT NULL,
    F_TABLE_SCHEMA VARCHAR(256) NOT NULL,
    F_TABLE_NAME VARCHAR(256) NOT NULL,
    F_GEOMETRY_COLUMN VARCHAR(256) NOT NULL,
    COORD_DIMENSION INTEGER,
    SRID INTEGER REFERENCES SPATIAL_REF_SYS,
    CONSTRAINT GC_PK PRIMARY KEY
      (F_TABLE_CATALOG, F_TABLE_SCHEMA,
      F_TABLE_NAME, F_GEOMETRY_COLUMN)
)
```

# Components of a GeoRelational-database

**Geometric Columns Metadata table or view**

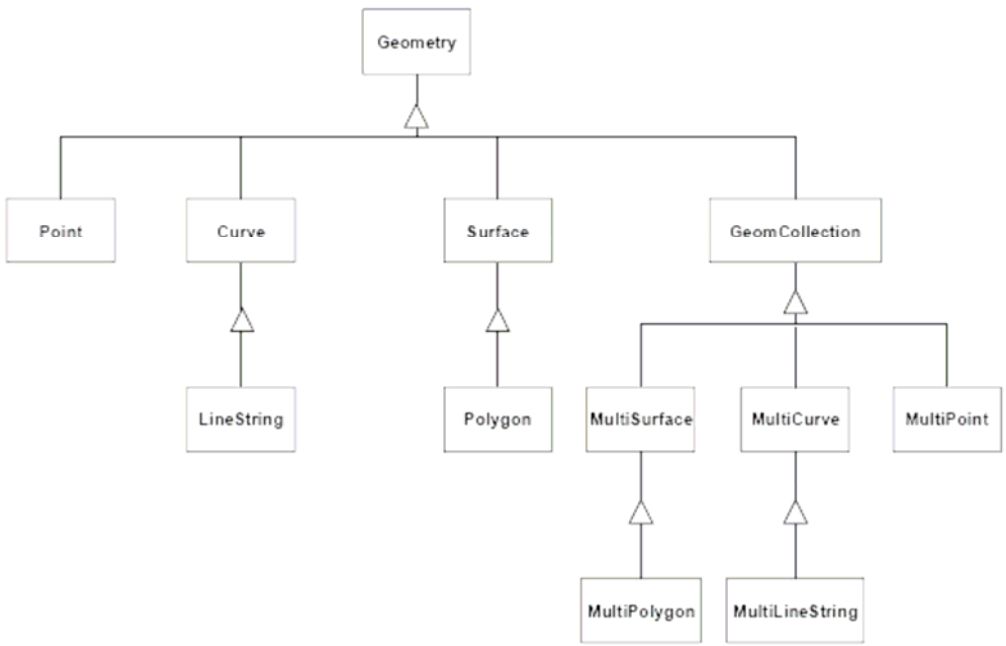
**Attributes:**

- F_TABLE_CATALOG, F_TABLE_SCHEMA, F_TABLE_NAME: the fully qualified name of the feature table containing the geometry column.
- F_GEOMETRY_COLUMN: the name of the geometry column in the feature table.
- COORD_DIMENSION: the coordinate dimension for the geometry values in this column, which will be equal to the number of dimensions in the spatial reference system.
- SRID: the ID of the spatial reference system used for the coordinate geometry in this table. It is a foreign key reference to the SPATIAL_REFERENCES table.

# Components of a GeoRelational-database

**SQL Geometry Types**

*Geometry*, Point, *Curve*, LineString, *Surface*, Polygon, GeomCollection, MultiPoint , *MultiCurve*, MultiLineString, *MultiSurface*, MultiPolygon



# Components of a GeoRelational-database

**Feature Tables**

CREATE TABLE <feature-name>

(

<FID name> <FID type>,

<feature attributes> <other FID type> REFERENCES <other feature view>,

…(other attributes for feature)

<geometry attribute 1> <Geometry type>,

… (other geometric attributes for feature)

PRIMARY KEY <FID name>,

FOREIGN KEY <FID relation name> REFERENCES <FEATURE table> <other FID name>

….

# Components of a GeoRelational-database

**Feature Tables** (continua)

CONSTRAINT SRS_1 CHECK (SRID(<geometry attribute 1>) in (SELECT SRID from GEOMETRY_COLUMNS where F_TABLE_CATALOG = <catalog> and F_TABLE_SCHEMA = <schema> and F_TABLE_NAME = <feature-name> and F_GEOMETRY_COLUMN = <geometry attribute 1>))

... (spatial reference constraints for other geometric attributes)

)

# SQL Textual Representation of Geometry (examples)

| Geometry Type | SQL Text Literal Representation | Comment |
|---|---|---|
| Point | `'POINT (10 10)'` | a `Point` |
| LineString | `'LINESTRING ( 10 10, 20 20, 30 40)'` | a `LineString` with 3 points |
| Polygon | `'POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))'` | a `Polygon` with 1 exterior ring and 0 interior rings |
| Multipoint | `'MULTIPOINT (10 10, 20 20)'` | a `MultiPoint` with 2 point |
| MultiLineString | `'MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))'` | a `MultiLineString` with 2 linestrings |
| MultiPolygon | `'MULTIPOLYGON ( ((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60 ) ))'` | a `MultiPolygon` with 2 polygons |
| GeomCollection | `'GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))'` | a `GeometryCollection` consisting of 2 `Point` values and a `LineString` value |

# Well-known Text Representation of SRS

A spatial reference system, also referred to as a coordinate system, is a geographic (latitude-longitude), a projected (X,Y), or a geocentric (X,Y,Z) coordinate system.

The coordinate system is composed of several objects.

```
<coordinate system> = <projected cs> | <geographic cs> | <geocentric cs>

<projected cs> = PROJCS['<name>', <geographic cs>, <projection>, {<parameter>,}* <linear
unit>]

<projection> = PROJECTION['<name>']

<parameter> = PARAMETER['<name>', <value>]

<value> = <number>
```

# Well-known Text Representation of SRS

```
PROJCS['NAD_1983_UTM_Zone_10N',
    <geographic cs>,
    PROJECTION['Transverse_Mercator'],
    PARAMETER['False_Easting',500000.0],
    PARAMETER['False_Northing',0.0],
    PARAMETER['Central_Meridian',-123.0],
    PARAMETER['Scale_Factor',0.9996],
    PARAMETER['Latitude_of_Origin',0.0],
    UNIT['Meter',1.0]]
```

```
<geographic cs> = GEOGCS['<name>', <datum>, <prime meridian>, <angular unit>]

<datum> = DATUM['<name>', <spheroid>]

<spheroid> = SPHEROID['<name>', <semi-major axis>, <inverse flattening>]

<semi-major axis> = <number>   NOTE: semi-major axis is measured in meters and must be > 0.

<inverse flattening> = <number>

<prime meridian> = PRIMEM['<name>', <longitude>]

<longitude> = <number>
```

The geographic coordinate system string for UTM zone 10 on NAD83 is

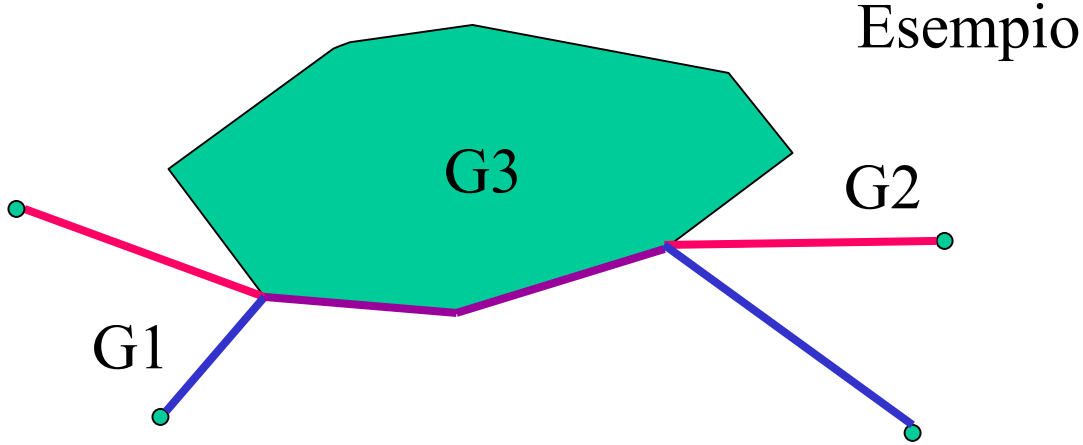
```
GEOGCS['GCS_North_American_1983',
    DATUM['D_North_American_1983',
    SPHEROID['GRS_1980',6378137,298.257222101]],
    PRIMEM['Greenwich',0],
    UNIT['Degree',0.0174532925199433]]
```

# Un modello logico/fisico per Geo-DBMS

**La rappresentazione di insiemi di valori geometrici in un Geo-DBMS**

Metodo 1: insiemi di liste di coordinate

Esempio




G1 = (ln, (<0,0>,<1,2>,<5,1>,<9,2>,<0,17>))

G2 = (ln, (<-2.5,3>,<1,2>,<5,1>,<9,2>,<9.1,17.5>))

G3 = (pg, (<1,2>,<5,1>,<9,2>,<11,3.5>,<10,4.5>,<5.3,5>,<3.5,4.5>,<0,3.2>))

# Un modello logico/fisico per Geo-DBMS

**La rappresentazione di insiemi di valori geometrici in un Geo-DBMS**

Metodo 1: insiemi di liste di coordinate

Vantaggi:

1. Indipendenza tra i valori geometrici $\Rightarrow$ facilità di inserimento di nuovi valori geometrici. Verifica vincoli spaziali a posteriori.
2. Buone prestazioni dell'operazione di visualizzazione grafica della geometria di un sottoinsieme limitato di valori.

Svantaggi:

1. Tutte le selezioni spaziali richiedono l'applicazione di algoritmi di geometria computazionale.
2. Tutte le operazioni di manipolazione (ad esempio il calcolo di intersezioni) richiede algoritmi di geometria computazionale.
3. Ridondanza nella rappresentazione della geometria.

# Un modello logico/fisico per Geo-DBMS

Modelli dei dati che adottano una rappresentazione indipendente degli oggetti geometrici:

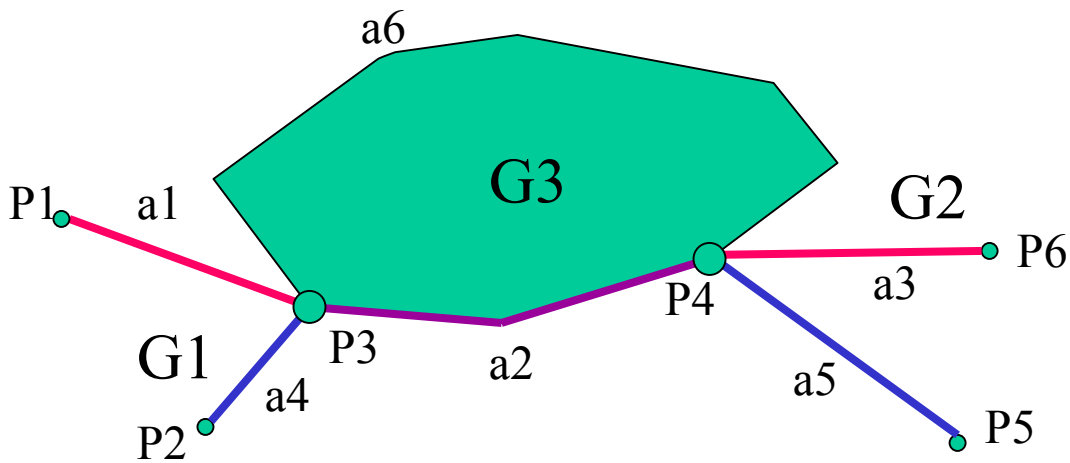
- Modello dei dati base di ORACLE 10g
- Modello dei dati di ArcView/GeoDatabase (ESRI)
- Modello dei dati di PostGIS (estensione di Postgresql per i dati geografici)

# Un modello logico/fisico per Geo-DBMS

**La rappresentazione di insiemi di valori geometrici in un Geo-DBMS**

Metodo 2: struttura basata su strutture topologiche (geometria condivisa)

Esempio




G1 = {L, (a1, a2, a3)}; G2 = {L, (a4, a2, a5)}; G3 = {L, (a2, a6)};

a1 = (P1, (), P3); a2 = (P3, (<5,1>), P4); a3 = (P4, (), P6) ….

P1 = <-2.5,3>; P2 = <0,0>; P3 = <1,2> ...

# Un modello logico/fisico per Geo-DBMS

**La rappresentazione dei valori geometrici in un Geo-DBMS**

Metodo 2: struttura basata su strutture topologiche (geometria condivisa)

Vantaggi:

- Eliminazione della ridondanza nella rappresentazione della geometria.
- Riduzione di alcune operazioni geometriche a selezione di un insieme di identificatori o puntatori a valori geometrici elementari precalcolati.
- Riduzione della ricerca via predicati topologici ad un confronto tra identificatori o puntatori a valori geometrici elementari precalcolati.
- Verifica vincoli spaziali immediata e supportata dalla struttura topologica.

Svantaggi:

- Ripristino della topologia corretta a valle di ogni inserimento di un nuovo valore geometrico.
- La pura visualizzazione di un valore geometrico richiede il ritrovamento della geometria attraverso una catena di puntatori.

# Un modello logico/fisico per Geo-DBMS

Modelli dei dati che consentono di organizzare in strutture topologiche gli attributi geometrici di un insieme di tabelle:

- Modello dei dati avanzato di ORACLE 10g
- Modello dei dati di Arc/Info (ESRI)
- Modello di Radius Topology (Laser-Scan)

# Un modello logico/fisico per Geo-DBMS

**Riepilogo dei costrutti del modello geo-relazionale**

Domini di base

Domini SQL92 + SQL Geometry Types di Simple Feature Specification

Relazione (tabella)

è un costruttore di tipo per tuple, dove gli attributi hanno come dominio un dominio di base.

Strato topologico

è una struttura dati per la rappresentazione integrata di insiemi di valori geometrici di uno o più tipi (geometria condivisa).

# Un modello logico/fisico per Geo-DBMS

**Vincoli tra strati topologici e attributi geometrici in uno schema:**

- ogni strato presente in uno schema contiene solo i valori geometrici di uno o più attributi geometrici appartenenti a relazioni dello schema.
- i valori di un attributo geometrico sono contenuti in uno e un sol strato.

Da ciò deriva che:

- in uno schema che presenti relazioni (tabelle) con attributi geometrici, va indicata dove presente la corrispondenza attributo geometrico/strato topologico.

# Un modello logico/fisico per Geo-DBMS

**Esempio di schema logico/fisico di una base di dati geografica**

***Relazioni***

BOSCO (Tipo: STRING, Pg: MULTIPOLYGON)

PRATO (Pascolo: BOOLEAN, Pg: MULTIPOLYGON)

***Strati topologici***

S_MONTAGNA (MULTIPOLYGON)

***Mapping Attributi geometrici/Strati topologici***

BOSCO.Pg → S_MONTAGNA

PASCOLO.Pg → S_MONTAGNA

# Un modello logico/fisico per Geo-DBMS

**Esempio di istanza di una base di dati geografica**

BOSCO

| Tipo | Pg |
|---|---|
| Ceduo | 23,44 |
| Latifoglia | 8,44 |

PRATO

| Pascolo | Pg |
|---|---|
| True | 14,9 |
| False | 2 |

S_MONTAGNA