Spatial & Geographic Data

Spatial and Geographic Databases

- Spatial databases store information related to spatial locations, and support efficient storage, indexing and querying of spatial data.
- Special purpose index structures are important for accessing spatial data, and for processing spatial join queries.
- **Computer Aided Design (CAD)** databases store design information about how objects are constructed E.g.: designs of buildings, aircraft, layouts of integrated-circuits
- Geographic databases store geographic information (e.g., maps): often called **geographic information systems or GIS**.

Representation of Geometric Information

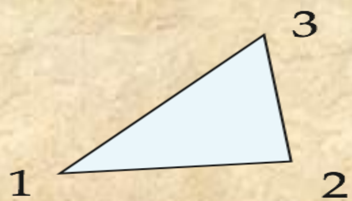
- Various geometric constructs can be represented in a database in a normalized fashion.
- Represent a line segment by the coordinates of its endpoints.
- Approximate a curve by partitioning it into a sequence of segments
  - Create a list of vertices in order, or
  - Represent each segment as a separate tuple that also carries with it the identifier of the curve (2D features such as roads).
- Closed polygons
  - List of vertices in order, starting vertex is the same as the ending vertex, or
  - Represent boundary edges as separate tuples, with each containing identifier of the polygon, or
  - Use **triangulation** — divide polygon into triangles
    - Note the polygon identifier with each of its triangles.

Representation of Geometric Constructs

line segment

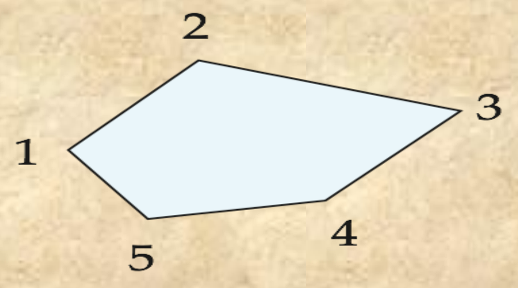
{(x1,y1), (x2,y2)}

triangle



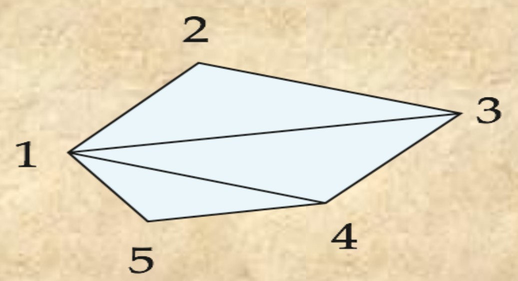
{(x1,y1), (x2,y2), (x3,y3)}

polygon



{(x1,y1), (x2,y2), (x3,y3), (x4,y4), (x5,y5)}

polygon



{(x1,y1), (x2,y2), (x3,y3), ID1}
{(x1,y1), (x3,y3), (x4,y4), ID1}
{(x1,y1), (x4,y4), (x5,y5), ID1}

object **representation**

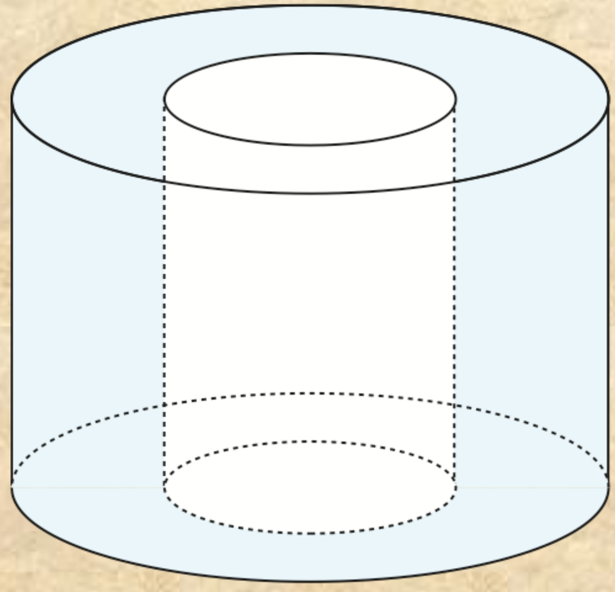
Representation of Geometric Information (Cont.)

- Representation of points and line segment in 3-D similar to 2-D, except that points have an extra z component
- Represent arbitrary polyhedra by dividing them into tetrahedrons, like triangulating polygons.
- Alternative: List their faces, each of which is a polygon, along with an indication of which side of the face is inside the polyhedron.

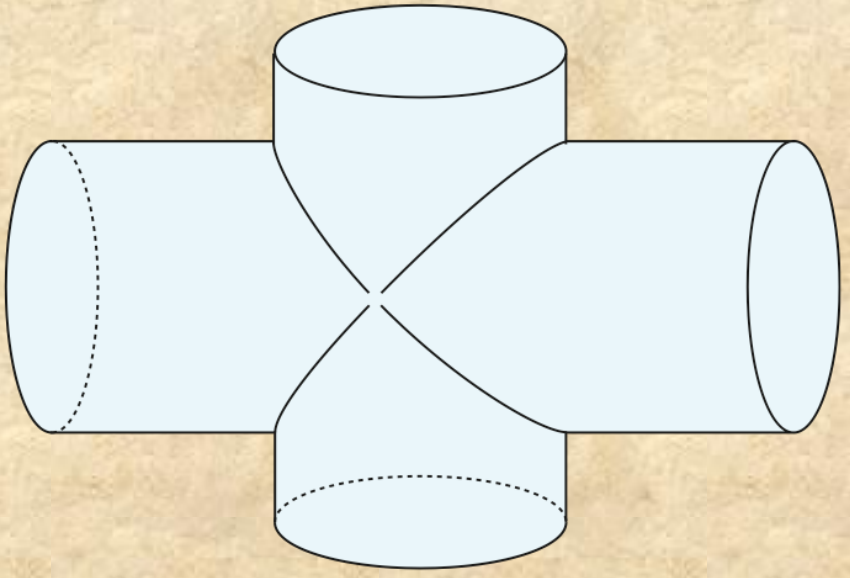
Design Database

- Represent design components as objects (generally geometric objects); the connections between the objects indicate how the design is structured.
- Simple two-dimensional objects: points, lines, triangles, rectangles, polygons.
- Complex two-dimensional objects: formed from simple objects via union, intersection, and difference operations.
- Complex three-dimensional objects: formed from simpler objects such as spheres, cylinders, and cuboids, by union, intersection, and difference operations.
- Wireframe models represent three-dimensional surfaces as a set of simpler objects.

Representation of Geometric Constructs



(a) Difference of cylinders



(b) Union of cylinders

- Design databases also store non-spatial information about objects (e.g., construction material, color, etc.)
- Spatial integrity constraints are important.
  - E.g., pipes should not intersect, wires should not be too close to each other, etc.

Geographic Data

Two types : Raster data & Vector data

- **Raster data** consist of bit maps or pixel maps, in two or more dimensions.
  - Example 2-D raster image: satellite image of cloud cover, where each pixel stores the cloud visibility in a particular area.
  - Additional dimensions might include the temperature at different altitudes at different regions, or measurements taken at different points in time.
- Design databases generally do not store raster data.

Geographic Data (Cont.)

- **Vector data** are constructed from basic geometric objects: points, line segments, triangles, and other polygons in two dimensions, and cylinders, spheres, cuboids, and other polyhedrons in three dimensions.
- Vector format often used to represent map data.
  - Roads can be considered as two-dimensional and represented by lines and curves.
  - Some features, such as rivers, may be represented either as complex curves or as complex polygons, depending on whether their width is relevant.
  - Features such as regions and lakes can be depicted as polygons.

Applications of Geographic Data

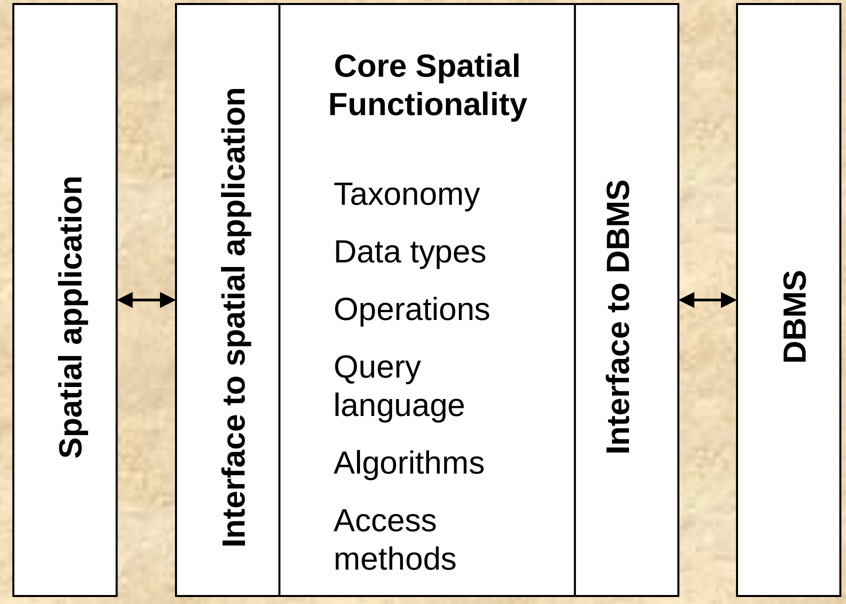
- Examples of geographic data
  - map data for vehicle navigation
  - distribution network information for power, telephones, water supply, and sewage
- Vehicle navigation systems store information about roads and services for the use of drivers:
  - **Spatial data**: e.g., road/restaurant/gas-station coordinates
  - **Non-spatial data**: e.g., one-way streets, speed limits, traffic congestion
- **Global Positioning System (GPS)** unit - utilizes information broadcast from GPS satellites to find the current location of user with an accuracy of tens of meters.
  - increasingly used in vehicle navigation systems as well as utility maintenance applications.

Spatial Database Management System

- Spatial Database Management System (SDBMS) provides the capabilities of a traditional database management system (DBMS) while allowing special storage and handling of spatial data.
- SDBMS:
  - Works with an underlying DBMS
  - Allows spatial data models and types
  - Supports querying language specific to spatial data types
  - Provides handling of spatial data and operations

SDBMS Three-layer Structure

- SDBMS works with a spatial application at the front end and a DBMS at the back end
- SDBMS has three layers:
  - Interface to spatial application
  - Core spatial functionality
  - Interface to DBMS



Spatial Queries

- **Nearness queries** request objects that lie near a specified location.
- **Nearest neighbor queries**, given a point or an object, find the nearest object that satisfies given conditions.
- **Region queries** deal with spatial regions. e.g., ask for objects that lie partially or fully inside a specified region.
- Queries that compute intersections or **unions** of regions.
- **Spatial join** of two spatial relations with the location playing the role of join attribute.

Spatial Queries (Cont.)

- Spatial data is typically queried using a graphical query language; results are also displayed in a graphical manner.
- Graphical interface constitutes the front-end
- Extensions of SQL with abstract data types, such as lines, polygons and bit maps, have been proposed to interface with back-end.
  - allows relational databases to store and retrieve spatial information
  - Queries can use spatial conditions (e.g., contains or overlaps).
  - queries can mix spatial and nonspatial conditions

Spatial Query Language

- Number of specialized adaptations of SQL
  - Spatial query language
  - Temporal query language (TSQL2)
  - Object query language (OQL)
  - Object oriented structured query language (O_2SQL)
- Spatial query language provides tools and structures specifically for working with spatial data
- SQL3 provides 2D geospatial types and functions

Spatial Query Language Operations

- Three types of queries:
  - Basic operations on all data types (e.g. IsEmpty, Envelope, Boundary)
  - Topological/set operators (e.g. Disjoint, Touch, Contains)
  - Spatial analysis (e.g. Distance, Intersection, SymmDiff)

Spatial Data Entity Creation

- Form an entity to hold county names, states, populations, and geographies

```
CREATE TABLE County(
    Name    varchar(30),
    State   varchar(30),
    Pop             Integer,
    Shape   Polygon);
```

- Form an entity to hold river names, sources, lengths, and geographies

```
CREATE TABLE River(
    Name    varchar(30),
    Source  varchar(30),
    Distance        Integer,
    Shape   LineString);
```

Example Spatial Query

- Find all the counties that border on Contra Costa county

```
SELECT  C1.Name
FROM    County C1, County C2
WHERE   Touch(C1.Shape, C2.Shape) = 1 AND C2.Name =
   'Contra Costa';
```

- Find all the counties through which the Merced river runs

```
SELECT  C.Name, R.Name
FROM    County C, River R
WHERE   Intersect(C.Shape, R.Shape) = 1 AND R.Name =
   'Merced';
```

```
CREATE TABLE County(            CREATE TABLE River(
  Name      varchar(30),          Name      varchar(30),
  State     varchar(30),          Source    varchar(30),
  Pop       Integer,              Distance  Integer,
  Shape     Polygon);             Shape     LineString);
```