Chapter 8: Complex Data Types

Database System Concepts, 7th Ed.

Outline

- There are several non-atomic data types (multi-valued, composite..) that are widely used today. 4 categories:
  1) Semi-Structured Data
  2) Object Orientation
  3) Textual Data
  4) Spatial Data

1-) Semi-Structured Data

- Many applications require storage of complex data, whose schema changes often, (especially web applications.)
- The relational model’s requirement of atomic data types may be an overkill
  - E.g., storing set of interests as a set-valued attribute of a user profile may be simpler & efficient than normalizing it. (why?)
- Data exchange can benefit greatly from semi-structured data
  - Exchange can be between applications, or between back-end and front-end of an application
  - Web-services are widely used today, with complex data fetched to the front-end efficiently and displayed using a mobile app or JavaScript
- JSON and XML are widely used semi-structured data models

Features of Semi-Structured Data Models

- **Flexible schema**
  - **Wide column** representation: allow each tuple to have a different set of attributes, can add new attributes at any time
  - **Sparse column** representation: schema has a fixed but large set of attributes, by each tuple may store only a subset
- **Multivalued data types**
  - **Sets**, **multisets**
    - E.g.,: set of interests {‘basketball, ‘La Liga’, ‘cooking’, ‘anime’, ‘jazz’}
  - **Key-value map** (or just **map** for short)
    - Store a set of key-value pairs.
    - The set of specifications may be different for each product.
    - E.g., {(brand, Apple), (ID, MacBook Air), (size, 13), (color, silver)}
    - Operations on maps: *put*(key, value), *get*(key), *delete*(key)
  - **Arrays**
    - Widely used for scientific and monitoring applications

Features of Semi-Structured Data Models

- **Arrays**
  - Widely used for scientific and monitoring applications
  - E.g., readings taken at regular intervals can be represented as array of values instead of (time, value) pairs
    - [5, 8, 9, 11] instead of {(1,5), (2, 8), (3, 9), (4, 11)}
- **Array database**: a database that provides specialized support for arrays
  - E.g., compressed storage, query language extensions etc
  - Oracle GeoRaster, PostGIS, SciDB, etc
- Multi-valued attribute types
  - Modeled using *non first-normal-form* (*NFNF*) data model
  - Supported by most database systems today

Nested Data Types

- Hierarchical data is common in many applications
- JSON: JavaScript Object Notation
  - Widely used today
- XML: Extensible Markup Language
  - Earlier generation notation, still used extensively
- Both provides flexibility in
  - the set of attributes that a record contains
  - the types of these attributes.
  - objects could have sub-objects (tree structure)
  - collect different data related to a particular user into one large object (i.e. a document), allowing data to be retrieved without the need for joins.
- XML is an older representation and is used by many systems mostly for storing configuration

JSON

- Textual representation widely used for data exchange
- Example of JSON data

```
{
        "ID": "22222",
        "name": {
                "firstname: "Albert",
                "lastname: "Einstein"
        },
        "deptname": "Physics",
        "children": [
                {"firstname": "Hans", "lastname": "Einstein" },
                {"firstname": "Eduard", "lastname": "Einstein" }
        ]
}
```

- Types: integer, real, string, and
  - *Objects: are* key-value maps, i.e. sets of (attribute name, value) pairs
  - Arrays are also key-value maps (from offset to value)

JSON

- + JSON is ubiquitous in data exchange today
  - Widely used for web services. For example, an email user interface may invoke web services for each of many tasks
  - Most modern applications are architected around on web services
    - A number of libraries are available to transform data between the JSON representation and the object representation used in languages such as JavaScript, Java, Python, PHP, and other languages.
- + SQL extensions for
  - JSON types for storing JSON data
  - Extracting data from JSON objects using path expressions
    - E.g. *V-> ID*, or *v.ID*
  - Generating JSON from relational data
    - E.g. json.build_object('ID', 12345, 'name', 'Einstein')
  - Creation of JSON collections using aggregation
    - E.g. json_agg aggregate function in PostgreSQL
  - Syntax varies greatly across databases
- - JSON is verbose, i.e. takes up more storage & processing time.
  - Compressed representations such as BSON (Binary JSON) used for efficient **data storage**

XML

- XML uses tags to mark up text
- E.g.

  <course>
  <course id> CS-101 </course id>
  <title> Intro. to Computer Science </title>
  <dept name> Comp. Sci. </dept name>
  <credits> 4 </credits>
  </course>
- Tags make the data self-documenting
- Tags can be hierarchical
- The two organizations must of course agree on what tags appear in the purchase order and what they mean. DTD (document type definition) files are used for this.

Example of Data in XML

```
<purchase order>
   <identifier> P-101 </identifier>
   <purchaser>
        <name> Cray Z. Coyote </name>
        <address> Route 66, Mesa Flats, Arizona 86047, USA
       </address>
   </purchaser>
   <supplier>
       <name> Acme Supplies </name>
       <address> 1 Broadway, New York, NY, USA </address>
   </supplier>
       <itemlist>
             <item>
          <identifier> RS1 </identifier>
          <description> Atom powered rocket sled </description>
          <quantity> 2 </quantity>
           <price> 199.95 </price>
             </item>
             <item>…</item>
       </itemlist>
       <total cost> 429.85 </total cost>
      ….
</purchase order>
```

XML Cont.

- XQuery language developed to query nested XML structures
  - - - Not widely used currently
- SQL extensions to support XML
  - Store XML data
  - Generate XML data from relational data
  - Extract data from XML data types
    - Path expressions
- See Chapter 30 (online) for more information

Knowledge Representation

- Representation of human knowledge is a long-standing goal of AI
  - Various representations of facts and inference rules proposed over time
- **RDF: Resource Description Format. (A representation standart)**
  - Simplified representation for facts, represented as triples (*subject, predicate, object*)
    - E.g., (NBA-2019, *winner*, Raptors)
      (Washington-DC, *capital-of*, USA)
      (Washington-DC, *population*, 6,200,000)
  - Models objects that have attributes, and relationships with other objects (i.e. entities, recourses)
    - Like the ER model, but with a flexible schema
    - (*ID, attribute-name, value*)
    - (*ID1, relationship-name, ID2*)
  - Has a natural graph representation

Triple Syntax of RDF Data

All attribute values are shown in "quotes".

Recourse ID, attr. name and relationship name are shown w/o quotes.

Relationships: instance-of, course_dept, sec_course, classroom, takes, teaches

Entities (resources) 10101,comp_sci..

Attr name: semester, year, sec_id,...

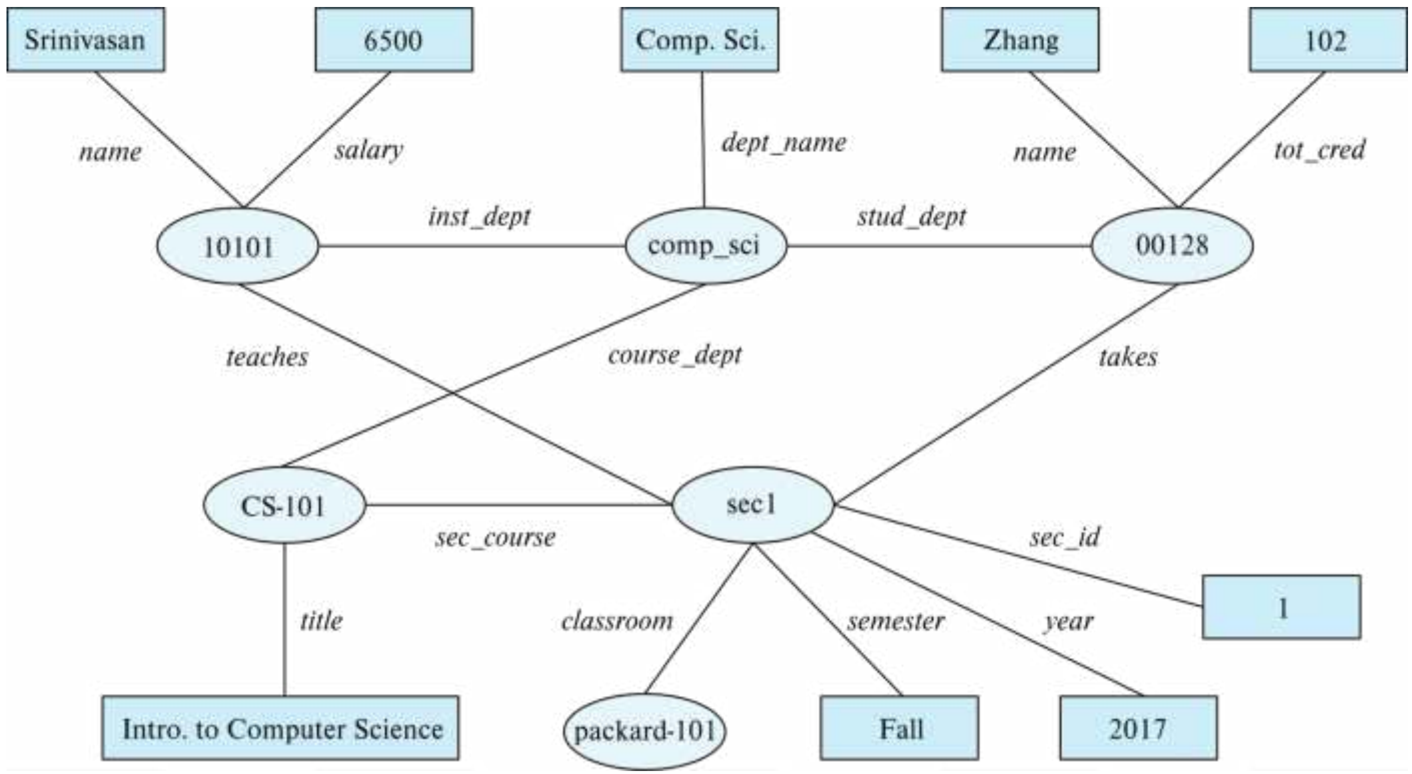
10101	instance-of	instructor .
10101	name	"Srinivasan" .
10101	salary	"6500" .
00128	instance-of	student .
00128	name	"Zhang" .
00128	tot_cred	"102" .
comp_sci	instance-of	department .
comp_sci	dept_name	"Comp. Sci." .
biology	instance-of	department .
CS-101	instance-of	course .
CS-101	title	"Intro. to Computer Science" .
CS-101	course_dept	comp_sci .
sec1	instance-of	section .
sec1	sec_course	CS-101 .
sec1	sec_id	"1" .
sec1	semester	"Fall" .
sec1	year	"2017" .
sec1	classroom	packard-101 .
sec1	time_slot_id	"H" .
10101	inst_dept	comp_sci .
00128	stud_dept	comp_sci .
00128	takes	sec1 .
10101	teaches	sec1 .

.

In contrast to the E-R model and relational schemas, RDF allows new attributes to be easily added to an object and also to create new types of relationships.

Graph View of RDF Data

- **Knowledge graph**



Querying RDF: SPARQL

- Triple patterns
  - ?cid *title* "Intro. to Computer Science"
  - ?cid *title* "Intro. to Computer Science"
    ?sid *course* ?cid
- SPARQL queries
  - **select** ?name
    where {
    ?cid *title* "Intro. to Computer Science" .
    ?sid *course* ?cid .
    ?id *takes* ?sid .
    ?id *name* ?name .
    }
  - Also supports
    - Aggregation, Optional joins (similar to outerjoins), Subqueries, etc.
    - Transitive closure on paths

RDF Representation (Cont.)

- RDF triples represent binary relationships
- How to represent n-ary relationships?
  - Approach 1 (from Section 6.9.4): Create artificial entity, and link to each of the n entities
    - E.g., (Barack Obama, *president-of*, USA, 2008-2016) can be represented as
      (*e1*, *person*, Barack Obama), (*e1*, *country*, USA),
      (*e1*, *president-from*, 2008) (*e1*, *president-till*, 2016)
  - Approach 2: use **quads** instead of triples, with context entity
    - E.g., (Barack Obama, *president-of*, USA, *c1*)
      (*c1*, *president-from*, 2008) (*c1*, *president-till*, 2016)
- RDF widely used as knowledge base representation
  - DBPedia, Yago, Freebase, WikiData, ..
- **Linked open data** project aims to connect different knowledge graphs to allow queries to span databases

Outline

- There are several non-atomic data types (multi-valued, composite..) that are widely used today. 4 categories:
  1) Semi-Structured Data
  2) Object Orientation
  3) Textual Data
  4) Spatial Data

Object Orientation

- **Object-relational data model** provides richer type system
  - with complex data types and object orientation
- Applications are often written in object-oriented programming languages
  - Type system does not match relational type system
  - Switching between imperative language and SQL is troublesome
- Approaches for integrating object-orientation with databases
  - ++ Build an **object-relational database**, adding object-oriented features to a relational database
  - ++ Automatically convert data between programming language model and relational model; data conversion specified by **object-relational mapping**
  - -- Build an **object-oriented database** that natively supports object-oriented data and direct access from programming language

Object-Relational Database Systems

- "structured" User-defined types
  - **create type** *Person*
(*ID* **varchar**(20) **primary key**,
name **varchar**(20),
address **varchar**(20)) **ref from**(*ID*); /* More on this later */
create table *people* **of** *Person*;
- Table types
  - **create type** *interest* **as table** (
topic **varchar**(20),
degree_of_interest **int**);
create table *users* (
ID **varchar**(20),
name **varchar**(20),
interests *interest*);
- Array, multiset data types also supported by many databases
  - In PostgreSQL, create table N (numbers integer[]);
  - In Oracle, create table N (numbers varray(10) of integer);

Type and Table Inheritance

- Type inheritance
  - **create type** *Student* **under** *Person*
    (*degree* **varchar**(20)) ;
    create type *Teacher* **under** *Person*
    (*salary* **integer**);
- Table inheritance syntax in PostgreSQL and oracle
  - **create table** *people* **of** *Person*;

    create table *students*
    (*degree* **varchar**(20))
    inherits *people*;
    create table *teachers*
    (*salary* **integer**)
    inherits *people*;
  - **create table** *people* **of** *Person*;
    create table *students* **of** *Student*
    under *people*;
    create table *teachers* **of** *Teacher*
    under *people*;

Reference Types

- Creating reference types enables to use primary key instead of system-generated ids.
  - **create type** *Person*
    (*ID* **varchar**(20) **primary key**,
    name **varchar**(20),
    address **varchar**(20))
    ref from(*ID*);
    create table *people* **of** *Person*;
    create type *Department* (
    dept_name **varchar(20)**,
    head **ref**(*Person*) **scope** *people*);
    create table *departments* **of** *Department*
    insert into *departments* **values** ('CS', '12345')

Alternative way (harder)::

insert into departments
values ('CS', null);

update departments
set head = (select ref(p)
 from people as p
 where ID = '12345')
where dept name = 'CS';

  - System generated references can be retrieved using subqueries
    - (**select ref**(*p*) **from** *people* **as** *p* **where** *ID* = '12345')
- Using references in **path expressions**
  - **select** *head->name, head->address*
    from *departments*;

Object-Relational Mapping

- Object-relational mapping (ORM) systems allow
  - Specification of mapping between programming language objects and database tuples
  - Automatic creation of database tuples upon creation of objects
  - Automatic update/delete of database tuples when objects are update/deleted
  - Interface to retrieve objects satisfying specified conditions
    - Tuples in database are queried, and object created from the tuples
- ++ Primary goal is to ease the job of programmers who build applications by providing them an object model while retaining the benefits of using a robust relational database underneath.
  - allow programmers to write queries directly on the object model; such queries are translated into SQL queries on the underlying relational database, and result objects are created from the SQL query results.
- ++ object-relational systems can provide significant performance gains over direct access to the underlying database by operating objects cached in primary mem.
- - - suffer from significant performance inefficiencies for bulk database updates, as well as for complex queries that are written directly in the imperative language.
  - Bypass ORM when bulk loading & complex querying
- Ex. Hibernate ORM for Java, Django ORM for Python

Outline

- There are several non-atomic data types (multi-valued, composite..) that are widely used today. 4 categories:
    1) Semi-Structured Data
    2) Object Orientation
    3) Textual Data
    4) Spatial Data

3-) Textual Data

- **Information retrieval**: querying of unstructured data
  - Simple model of keyword queries: given query keywords, retrieve documents containing all the keywords
  - More advanced models rank relevance of documents
  - Today, keyword queries return many types of information as answers
    - E.g., a query “cricket” typically returns information about ongoing cricket matches
- Relevance ranking
  - Essential since there are usually many documents matching keywords
- Different terminology in different contexts.:
  - the field of information retrieval, textual information is "documents"
  - In a database, a text-valued attribute can be considered a "document".
  - In the web, each web page can be considered to be a "document.".
    - Web search engine is a huge information retrieval system.
    - Submit keyword ==> web crawling
    - Submit a question ==> make judgements, reasoning (*muhakeme*)

Ranking using TF-IDF

- Term: keyword occurring in a document/query
- **Term Frequency:** $TF(d, t)$, the relevance of a term t to a document d
  - One definition: $n(d,t)$
  - Another definition: $TF(d, t) = log(1 + n(d,t)/n(d))$
    where
    - $n(d,t)$ = number of occurrences of term t in document d
    - $n(d)$ = number of terms in document d
  - The above notions can be formalized in more detail with place of the term occurrence in the document i.e. title, abstract, beginning, end..
- Suppose a query uses two terms, “database”,“Silberschatz”. ???
- **Inverse document frequency**: IDF(t)
  - One definition: $IDF(t) = 1/n(t)$
  - n(t) denotes the number of documents *(among those indexed by the system)* that contain the term t
- **Relevance** of a document d to a set of terms Q
  - *One definition:* $r(d, Q) = \sum_{t \in Q} TF(d, t) * IDF(t)$
  - Other definitions: user-specified weights, **proximity** of words in the page, **Stop words?**

Ranking Using Hyperlinks

- Hyperlinks provide very important clues to importance
- Google introduced PageRank, a measure of popularity/importance based on hyperlinks to pages
  - Pages hyperlinked from many pages should have higher PageRank
  - Pages hyperlinked from pages with higher PageRank should have higher PageRank
  - Formalized by **random walk** model
- Let $T[i, j]$ be the probability that a random walker who is on page i will click on the link to page j
  - Assuming all links are equal, $T[i, j] = 1/Ni$
  - Ni is the number of links out of page i
- Then PageRank[j] for each page j can be defined as
  - $P[j] = \delta/N + (1 - \delta) * \sum_{i=1}^{N} (T[i, j] * P[i])$
  - Where N = total number of pages, and δ a constant usually set to 0.15

Ranking Using Hyperlinks

- Definition of PageRank is circular, but can be solved as a set of linear equations
  - Simple iterative technique works well
  - Initialize all P[i] = 1/N, N is the number of pages
  - In each iteration use equation $P[j] = \delta/N + (1 - \delta) * \sum_{i=1}^{N} (T[i, j] * P[i])$ to update P
  - Stop iteration when changes are small, or some limit (say 30 iterations) is reached.
- Other measures of relevance are also important. Because, PageRank is a static measure, independent of the keyword query;
  - given a keyword query, it is used in combination with TF–IDF scores of a document to judge its relevance of the document to the keyword query.For example:
    - Keywords in anchor text
    - Number of times who ask a query click on a link if it is returned as an answer

Retrieval Effectiveness

- Measures of effectiveness
  - **Precision**: what percentage of returned results are actually relevant
  - **Recall**: what percentage of relevant results were returned
  - At some number of answers, e.g. precision@10, recall@10
- Keyword querying on structured data and semi-structured data i.e. knowledge bases
  - Useful if users don't know schema, or there is no predefined schema.

    (good news for lay users..)
  - Can represent data as graphs
  - Keywords match tuples
  - Keyword search returns closely connected tuples that contain keywords
    - E.g. on our university database given query "Zhang Katz", Zhang matches a student, Katz an instructor and advisor relationship links them

Outline

- There are several non-atomic data types (multi-valued, composite..) that are widely used today. 4 categories:
  1) Semi-Structured Data
  2) Object Orientation
  3) Textual Data
  4) Spatial Data

Spatial Data

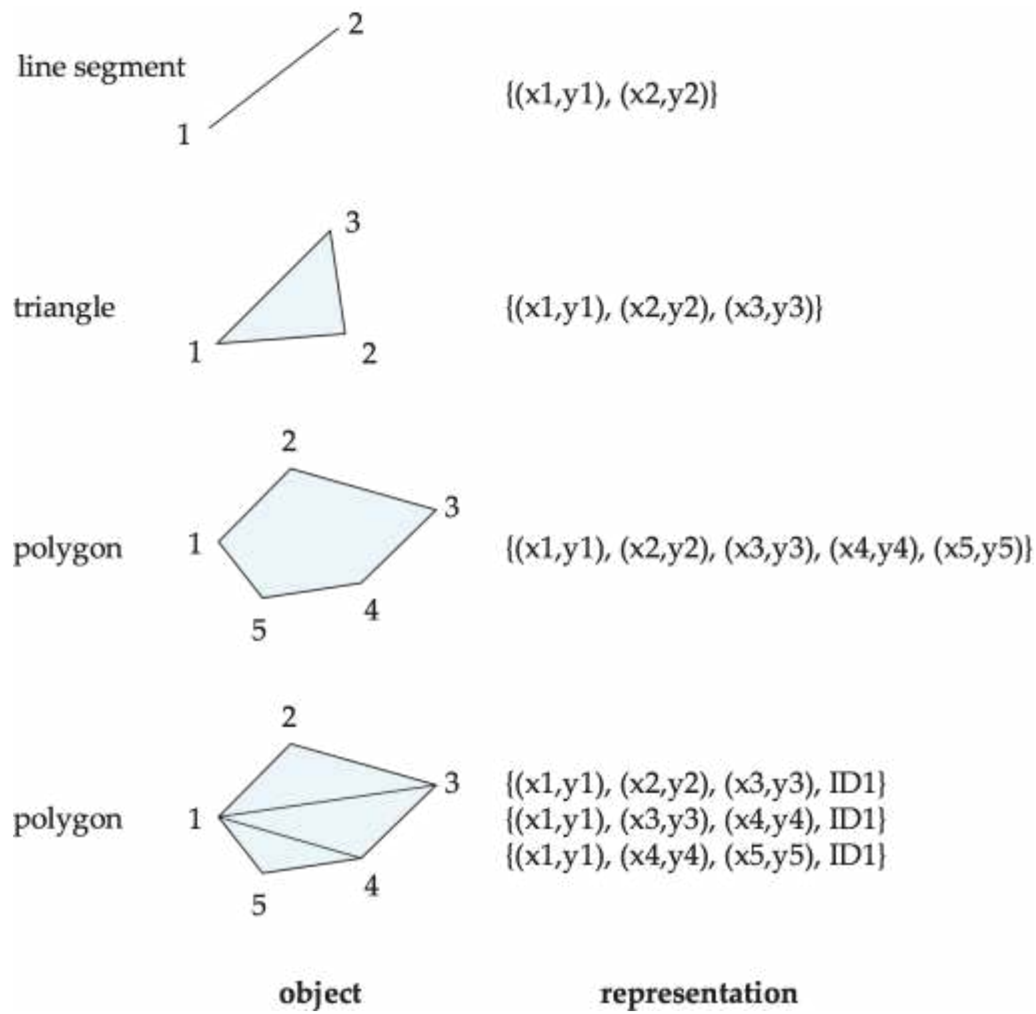
- Spatial databases store information related to spatial locations, and support efficient storage, indexing and querying of spatial data.
- The syntax for representing geographic and geometric data varies by database, although representations based on the Open Geospatial Consortium (OGC) standard
  - **Geographic data** -- road maps, land-usage maps, topographic elevation maps, political maps showing boundaries, land-ownership maps, and so on.
    - **Geographic information systems** are special-purpose databases tailored for storing geographic data.
    - Round-earth coordinate system may be used
      - (Latitude, longitude, elevation)
  - **Geometric data:** design information about how objects are constructed . For example, designs of buildings, aircraft, layouts of integrated-circuits.
    - 2 or 3 dimensional Euclidean space with (X, Y, Z) coordinates

Represented of Geometric Information

Various geometric constructs can be represented in a database in a normalized fashion (see next slide)

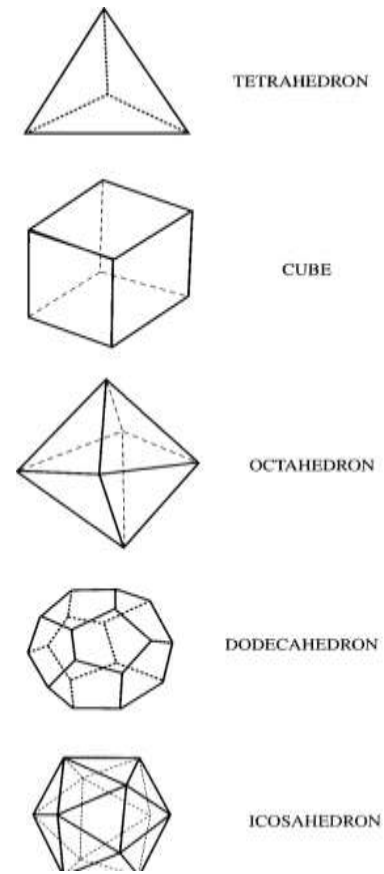
- A **line segment** can be represented by the coordinates of its endpoints.
- A **polyline** or **linestring** consists of a connected sequence of line segments and can be represented by a list containing the coordinates of the endpoints of the segments, in sequence.
  - Approximate a curve by partitioning it into a sequence of segments
    - Useful for two-dimensional features such as roads.
    - Some systems also support *circular arcs* as primitives, allowing curves to be represented as sequences of arc
- **Polygons** is represented by a list of vertices in order.
  - The list of vertices specifies the boundary of a polygonal region.
  - Can also be represented as a set of triangles (**triangulation**)

Representation of Geometric Constructs



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.
- Geometry and geography data types supported by many databases
  - E.g. SQL Server and PostGIS
  - point, linestring, curve, polygons
  - Collections: multipoint, multilinestring, multicurve, multipolygon
  - LINESTRING(1 1, 2 3, 4 4)
  - POLYGON((1 1, 2 3, 4 4, 1 1))
  - Type conversions: *ST GeometryFromText*() and *ST GeographyFromText*()
  - *Operations: ST Union*(), *ST Intersection*(), …



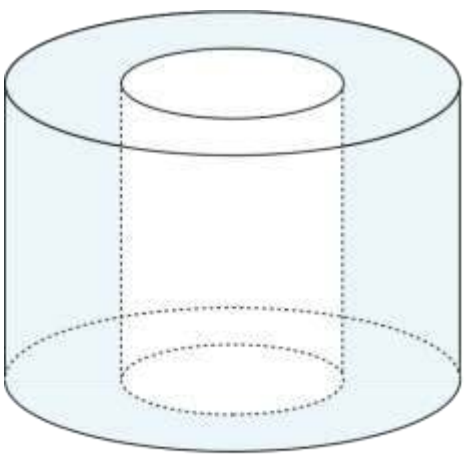
https://www.sciencedirect.com/topics/mathematics/polyhedron

Design Databases

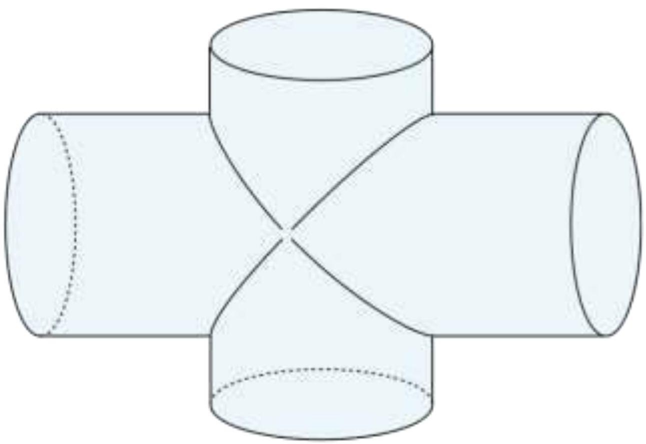
- Represent design components as objects (generally geometric objects); the connections between the objects indicate how the design is structured.
  - Whole project in primary mem.?
  - For large designs, such as the design of a large-scale integrated circuit or the design of an entire airplane, it may be impossible to hold the complete design in memory. ?
  - CAD systems (design DBs) ==> disk-based
- 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

- 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.
  - Importance of Spatial indexes !!



(a) Difference of cylinders



(b) Union of cylinders

Geographic Data

- variety of uses:
  - (online) map: boundaries, rivers, and roads, for example—but also detailed information associated with locations, such as elevation, soil type,land usage, and annual rainfall.
  - navigation services,
  - distribution-network information for public-service utilities (i.e. telephone, electric-power, and water-supply)
- **Raster data** consist of bit maps or pixel maps, in two or more dimensions.
  - Raster data are often represented as tiles, each covering a fixed-size area.
  - 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.

https://carpentries-incubator.github.io/jupyter_maps/02-raster/index.html

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.
- Topographical information, that is information about the elevation (height) of each point on a surface, can be represented in raster or **vector form::.**
  - by dividing the surface into polygons covering regions of (approximately) equal elevation,
- Or TIN (triangulated irregular network): triangles represented by the latitude, longitude, and elevation of each of its corners.

Spatial Queries

- **Region queries** deal with spatial regions. e.g., ask for objects that lie partially or fully inside a specified region
  - E.g., PostGIS *ST_Contains*(), *ST_Overlaps*(), …
- **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.
- **Spatial graph queries** request information based on spatial graphs
  - E.g., shortest path between two points via a road network
- **Spatial join** of two spatial relations with the location playing the role of join attribute.
- Queries that compute intersections or **unions** of regions

End of Chapter 8