Chapter 8: Complex Data Types

Database System Concepts, 7th Ed.

Outline

- Semi-Structured Data
- Object Orientation
- Textual Data
- Spatial Data

Semi-Structured Data

- Many applications require storage of complex data, whose schema changes often
- 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 than normalizing it
- 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 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
    - 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)}
- Multi-valued attribute types
  - Modeled using *non first-normal-form* (*NFNF*) data model
  - Supported by most database systems today
- **Array database**: a database that provides specialized support for arrays
  - E.g., compressed storage, query language extensions etc
  - Oracle GeoRaster, PostGIS, SciDB, etc

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

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
  - Most modern applications are architected around on web services
- 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
  - 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

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.

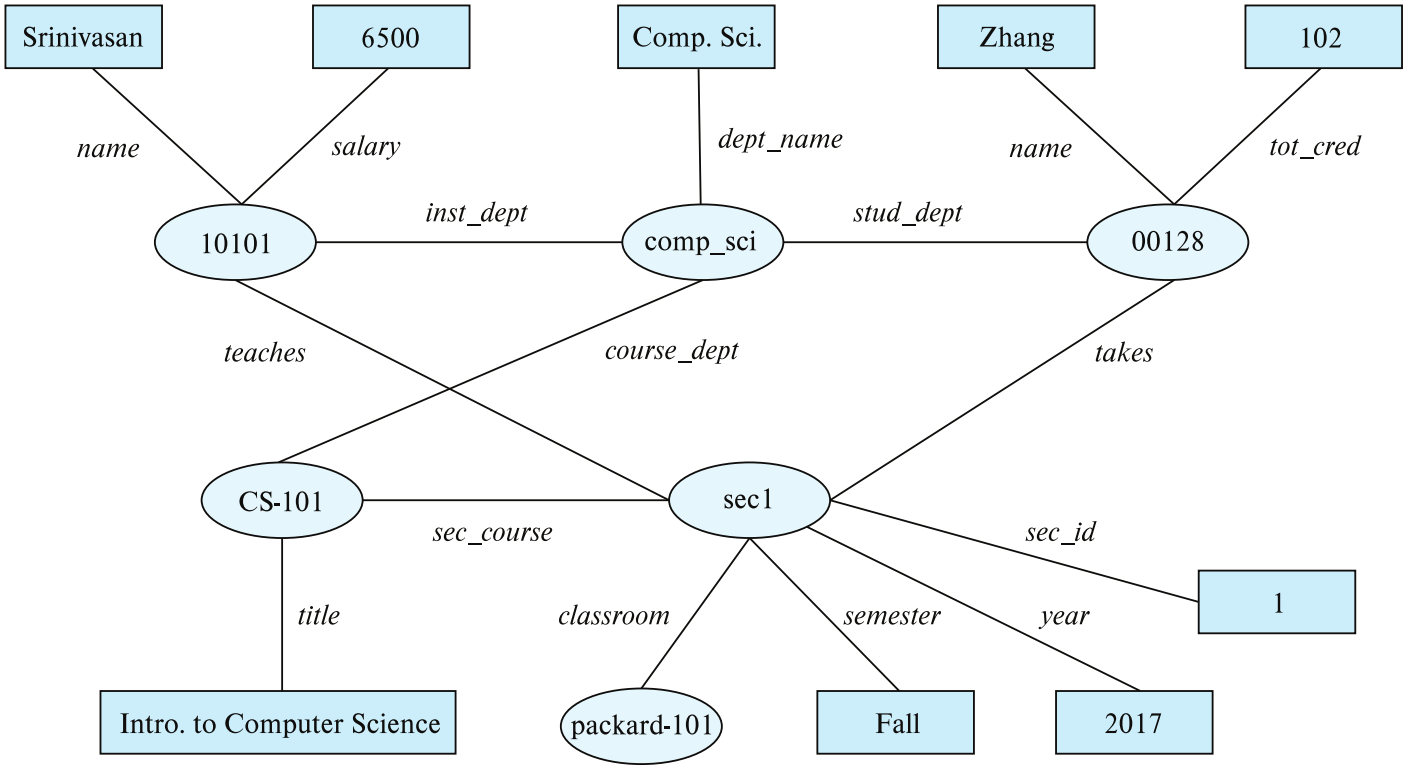
- XQuery language developed to query nested XML structures
  - Not widely used currently
- SQL extensions to support XML
  - Store XML data
  - Generate XML data from relational data
  - Extract data from XML data types
    - Path expressions
- See Chapter 30 (online) for more information

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**
  - 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
    - Like the ER model, but with a flexible schema
    - (*ID, attribute-name, value*)
    - (*ID1, relationship-name, ID2*)
  - Has a natural graph representation

Graph View of RDF Data

- **Knowledge graph**



Triple View of RDF Data

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 .

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

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

- 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
  - Syntax varies by database

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** *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
  - **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')
  - 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
- Details in Section 9.6.2
  - Hibernate ORM for Java
  - Django ORM for Python

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

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*:* *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*
- **Inverse document frequency**: IDF(t)
  - One definition: *IDF*(*t*) = 1/*n*(*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
    - take **proximity** of words into account
    - **Stop words** are often ignored

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, $T[i, j] = 1/Ni$
- 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
  - 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. 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 knowledge bases
  - Useful if users don't know schema, or there is no predefined schema
  - 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

Spatial Data

Spatial Data

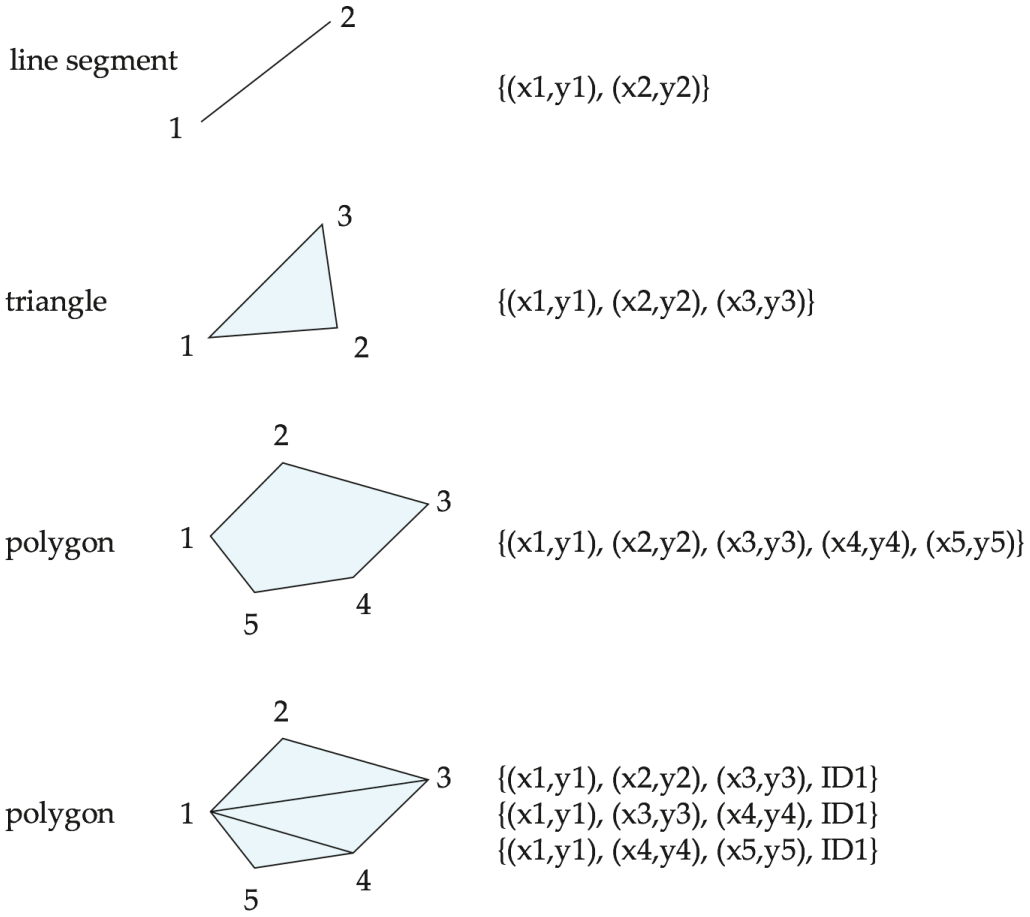
- Spatial databases store information related to spatial locations, and support efficient storage, indexing and querying of spatial data.
  - **Geographic data** -- road maps, land-usage maps, topographic elevation maps, political maps showing boundaries, land-ownership maps, and so on.
    - **Geographic 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.)

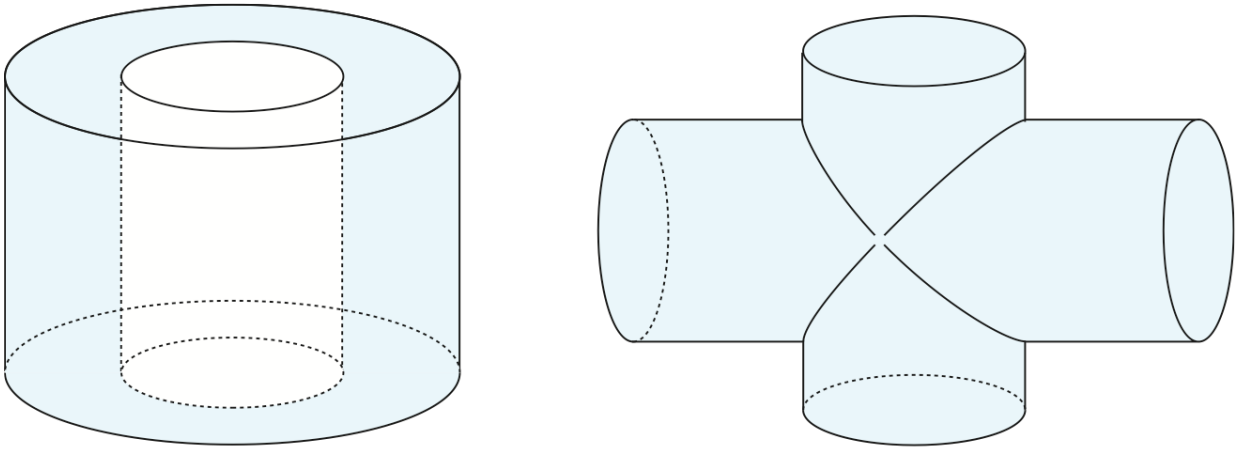
- Representation of points and line segment in 3-D similar to 2-D, except that points have an extra z component
- Represent arbitrary polyhedra by dividing them into tetrahedrons, like triangulating polygons.
- Alternative: List their faces, each of which is a polygon, along with an indication of which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

Design Databases

- 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

- 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.



(a) Difference of cylinders

(b) Union of cylinders

Geographic 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.

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

Chapter 9: Application Development

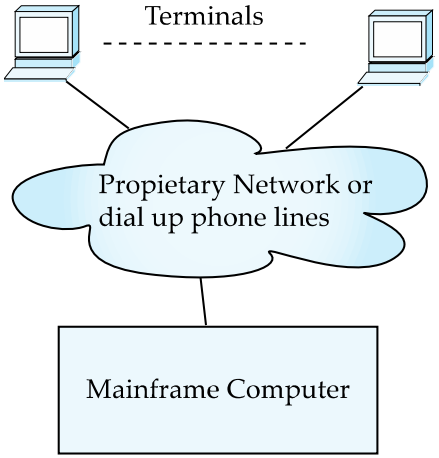
Database System Concepts, 7th Ed.

Application Programs and User Interfaces

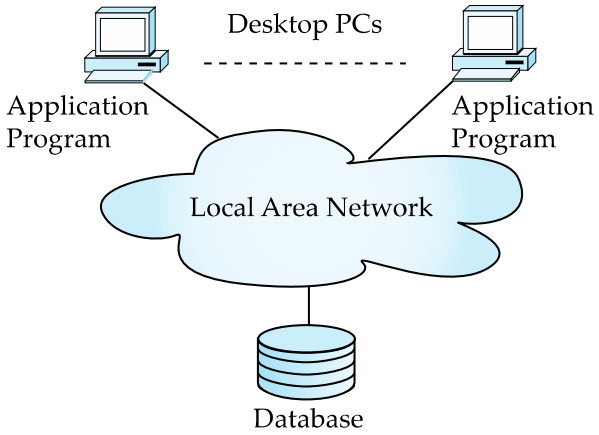
- Most database users do *not* use a query language like SQL
- An application program acts as the intermediary between users and the database
  - Applications split into
    - front-end
    - middle layer
    - backend
- Front-end: user interface
  - Forms
  - Graphical user interfaces
  - Many interfaces are Web-based

Application Architecture Evolution

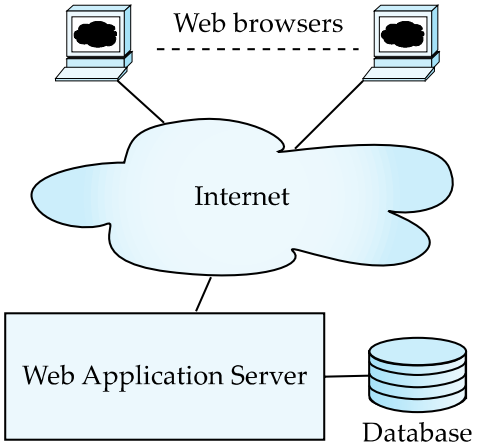
- Three distinct era's of application architecture
  - Mainframe (1960's and 70's)
  - Personal computer era (1980's)
  - Web era (mid 1990's onwards)
  - Web and Smartphone era (2010 onwards)



(a) Mainframe Era



(b) Personal Computer Era



(c) Web era

Web Interface

Web browsers have become the de-facto standard user interface to databases

- Enable large numbers of users to access databases from anywhere
- Avoid the need for downloading/installing specialized code, while providing a good graphical user interface
  - Javascript, Flash and other scripting languages run in browser, but are downloaded transparently
- Examples: banks, airline and rental car reservations, university course registration and grading, an so on.

Sample HTML Source Text

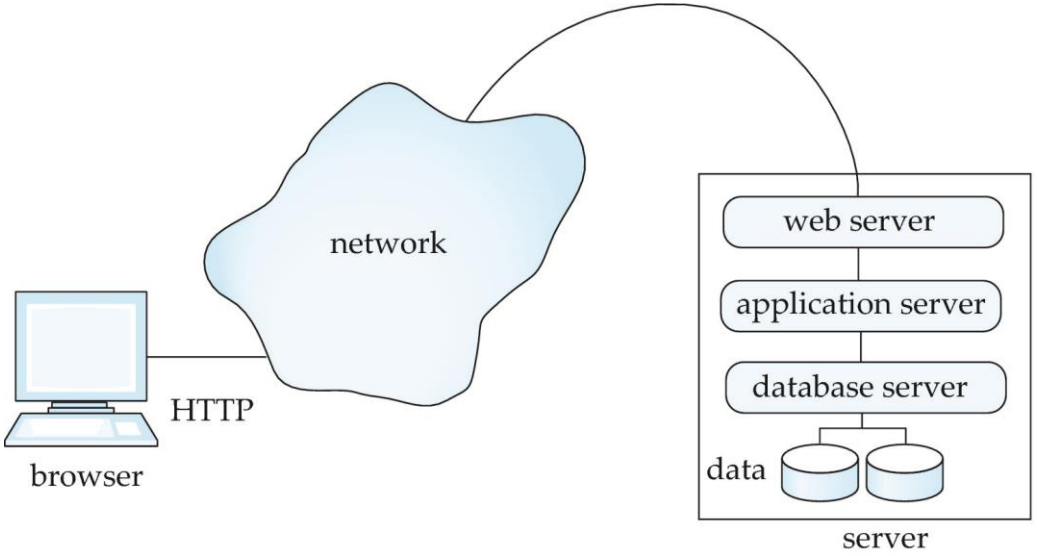
```
<html>
<body>
  <table border>
     <tr> <th>ID</th> <th>Name</th> <th>Department</th> </tr>
     <tr> <td>00128</td> <td>Zhang</td> <td>Comp. Sci.</td> </tr>
     ….
  </table>
   <form action="PersonQuery" method=get>
     Search for:
        <select name="persontype">
           <option value="student" selected>Student </option>
           <option value="instructor"> Instructor </option>
        </select> <br>
     Name: <input type=text size=20 name="name">
     <input type=submit value="submit">
  </form>
</body> </html>
```

Display of Sample HTML Source

ID	Name	Department
00128	Zhang	Comp. Sci.
12345	Shankar	Comp. Sci.
19991	Brandt	History

Search for: Student
Name:
submit

Three-Layer Web Architecture



HTTP and Sessions

- The HTTP protocol is **connectionless**
  - That is, once the server replies to a request, the server closes the connection with the client, and forgets all about the request
  - In contrast, Unix logins, and JDBC/ODBC connections stay connected until the client disconnects
    - retaining user authentication and other information
  - Motivation: reduces load on server
    - operating systems have tight limits on number of open connections on a machine
- Information services need session information
  - E.g., user authentication should be done only once per session
- Solution: use a **cookie**

Sessions and Cookies

- A **cookie** is a small piece of text containing identifying information
  - Sent by server to browser
    - Sent on first interaction, to identify session
  - Sent by browser to the server that created the cookie on further interactions
    - part of the HTTP protocol
  - Server saves information about cookies it issued, and can use it when serving a request
    - E.g., authentication information, and user preferences
- Cookies can be stored permanently or for a limited time

Servlets

- Java Servlet specification defines an API for communication between the Web/application server and application program running in the server
  - E.g., methods to get parameter values from Web forms, and to send HTML text back to client
- Application program (also called a servlet) is loaded into the server
  - Each request spawns a new thread in the server
    - thread is closed once the request is serviced
  - Programmer creates a class that inherits from HttpServlet
    - And overrides methods doGet, doPost, …
  - Mapping from servlet name (accessible via HTTP), to the servlet class is done in a file web.xml
    - Done automatically by most IDEs when you create a Servlet using the IDE

Example Servlet Code

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PersonQueryServlet extends HttpServlet {
   public void doGet (HttpServletRequest request, HttpServletResponse
     response)
                    throws ServletException, IOException
   {
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     out.println("<HEAD><TITLE> Query Result</TITLE></HEAD>");
     out.println("<BODY>");
        ….. BODY OF SERVLET (next slide) …
     out.println("</BODY>");
     out.close();
   }
}
```

Example Servlet Code

```
String persontype = request.getParameter("persontype");
String number = request.getParameter("name");
if(persontype.equals("student")) {
    ... code to find students with the specified name ...
    ... using JDBC to communicate with the database ..
    out.println("<table BORDER COLS=3>");
    out.println(" <tr> <td>ID</td> <td>Name: </td>" + " <td>Department</td> </tr>");
    for(... each result ...){
        ... retrieve ID, name and dept name
        ... into variables ID, name and deptname
        out.println("<tr> <td>" + ID + "</td>" + "<td>" + name + "</td>" + "<td>" + deptname
                + "</td></tr>");
    };
    out.println("</table>");
}
else {
    ... as above, but for instructors ...
}
```

Servlet Sessions

- Servlet API supports handling of sessions
  - Sets a cookie on first interaction with browser, and uses it to identify session on further interactions
- To check if session is already active:
  - if (request.getSession(false) == true)
    - .. then existing session
    - else .. redirect to authentication page
  - authentication page
    - check login/password
    - Create new session
      - HttpSession session = request.getSession(true)
    - Store/retrieve attribute value pairs for a particular session
      - session.setAttribute(“userid”, userid)
  - If existing session:
    HttpSession = request.getSession(false);
    String userid = (String) session.getAttribute(“userid”)

Servlet Support

- Servlets run inside application servers such as
  - Apache Tomcat, Glassfish, JBoss
  - BEA Weblogic, IBM WebSphere and Oracle Application Servers
- Application servers support
  - Deployment and monitoring of servlets
  - Java 2 Enterprise Edition (J2EE) platform supporting objects, parallel processing across multiple application servers, etc

Server-Side Scripting

- Server-side scripting simplifies the task of connecting a database to the Web
  - Define an HTML document with embedded executable code/SQL queries.
  - Input values from HTML forms can be used directly in the embedded code/SQL queries.
  - When the document is requested, the Web server executes the embedded code/SQL queries to generate the actual HTML document.
- Numerous server-side scripting languages
  - JSP, PHP
  - General purpose scripting languages: VBScript, Perl, Python

Java Server Pages (JSP)

- A JSP page with embedded Java code

```
<html>
<head> <title> Hello </title> </head>
<body>
<% if (request.getParameter("name") == null)
{ out.println("Hello World"); }
else { out.println("Hello, " + request.getParameter("name")); }
%>
</body>
</html>
```

- JSP is compiled into Java + Servlets
- JSP allows new tags to be defined, in tag libraries
  - Such tags are like library functions, can are used for example to build rich user interfaces such as paginated display of large datasets

PHP

- PHP is widely used for Web server scripting
- Extensive libaries including for database access using ODBC

```
<html>
  <head> <title> Hello </title> </head>
  <body>
  <?php if (!isset($_REQUEST[ ‘name’ ]))
  { echo “Hello World”; }
  else { echo “Hello, ” + $_REQUEST[ ‘name’ ]; }
  ?>
  </body>
</html>
```

Javascript

- Javascript very widely used
  - Forms basis of new generation of Web applications (called Web 2.0 applications) offering rich user interfaces
- Javascript functions can
  - Check input for validity
  - Modify the displayed Web page, by altering the underling **document object model (DOM)** tree representation of the displayed HTML text
  - Communicate with a Web server to fetch data and modify the current page using fetched data, without needing to reload/refresh the page
    - Forms basis of AJAX technology used widely in Web 2.0 applications
    - E.g. on selecting a country in a drop-down menu, the list of states in that country is automatically populated in a linked drop-down menu

Javascript

- Example of Javascript used to validate form input

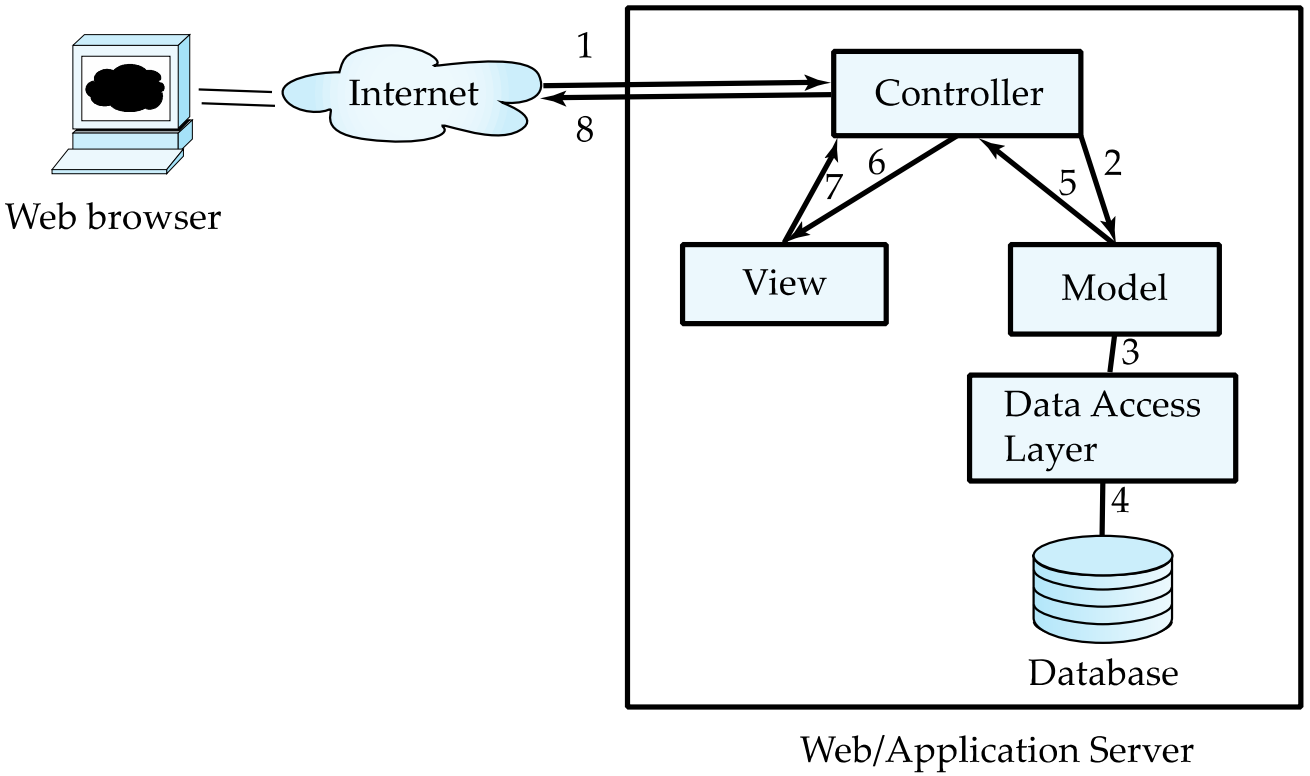
```
<html> <head>
    <script type="text/javascript">
        function validate() {
            var credits=document.getElementById("credits").value;
            if (isNaN(credits)|| credits<=0 || credits>=16) {
                alert("Credits must be a number greater than 0 and less than 16");
                return false
            }
        }
    </script>

</head> <body>
    <form action="createCourse" onsubmit="return validate()">
        Title: <input type="text" id="title" size="20"><br />
        Credits: <input type="text" id="credits" size="2"><br />
        <Input type="submit" value="Submit">
    </form>

</body> </html>
```

Application Architectures

Application Architectures

- Application layers
  - Presentation or user interface
    - **model-view-controller (MVC)** architecture
      - **model**: business logic
      - **view**: presentation of data, depends on display device
      - **controller**: receives events, executes actions, and returns a view to the user
  - **business-logic** layer
    - provides high level view of data and actions on data
      - often using an object data model
    - hides details of data storage schema
  - **data access** layer
    - interfaces between business logic layer and the underlying database
    - provides mapping from object model of business layer to relational model of database

Application Architecture



Business Logic Layer

- Provides abstractions of entities
  - E.g., students, instructors, courses, etc
- Enforces **business rules** for carrying out actions
  - E.g., student can enroll in a class only if she has completed prerequsites, and has paid her tuition fees
- Supports **workflows** which define how a task involving multiple participants is to be carried out
  - E.g., how to process application by a student applying to a university
  - Sequence of steps to carry out task
  - Error handling
    - E.g. what to do if recommendation letters not received on time
  - Workflows discussed in Section 26.2

Object-Relational Mapping

- Allows application code to be written on top of object-oriented data model, while storing data in a traditional relational database
  - Alternative: implement object-oriented or object-relational database to store object model
    - Has not been commercially successful
- Schema designer has to provide a mapping between object data and relational schema
  - E.g., Java class *Student* mapped to relation *student,* with corresponding mapping of attributes
  - An object can map to multiple tuples in multiple relations
- Application opens a session, which connects to the database
- Objects can be created and saved to the database using
  session.save(object)
  - Mapping used to create appropriate tuples in the database
- Query can be run to retrieve objects satisfying specified predicates

Object-Relational Mapping and Hibernate (Cont.)

- The **Hibernate** object-relational mapping system is widely used
  - Public domain system, runs on a variety of database systems
  - Supports a query language that can express complex queries involving joins
    - Translates queries into SQL queries
  - Allows relationships to be mapped to sets associated with objects
    - E.g., courses taken by a student can be a set in Student object
  - See book for Hibernate code example
- The **Entity Data Model** developed by Microsoft
  - Provides an entity-relationship model directly to application
  - Maps data between entity data model and underlying storage, which can be relational
  - Entity SQL language operates directly on Entity Data Model

Web Services

- Allow data on Web to be accessed using remote procedure call mechanism
- Two approaches are widely used
  - **Representation State Transfer (REST)**: allows use of standard HTTP request to a URL to execute a request and return data
    - Returned data is encoded either in XML, or in **JavaScript Object Notation (JSON)**
  - **Big Web Services**:
    - Uses XML representation for sending request data, as well as for returning results
    - Standard protocol layer built on top of HTTP
    - See Section 23.7.3

Disconnected Operations

- Tools for applications to use the Web when connected, but operate locally when disconnected from the Web
  - Make use of HTML5 local storage

Rapid Application Development

- A lot of effort is required to develop Web application interfaces
  - More so, to support rich interaction functionality associated with Web 2.0 applications
- Several approaches to speed up application development
  - Function library to generate user-interface elements
  - Drag-and-drop features in an IDE to create user-interface elements
  - Automatically generate code for user interface from a declarative specification
- Above features have been in used as part of **rapid application development (RAD)** tools even before advent of Web
- Web application development frameworks
  - Java Server Faces (JSF) includes JSP tag library
  - Ruby on Rails
    - Allows easy creation of simple **CRUD** (create, read, update and delete) interfaces by code generation from database schema or object model

Application Performance

Improving Web Server Performance

- Performance is an issue for popular Web sites
  - May be accessed by millions of users every day, thousands of requests per second at peak time
- Caching techniques used to reduce cost of serving pages by exploiting commonalities between requests
  - At the server site:
    - Caching of JDBC connections between servlet requests
      - a.k.a. **connection pooling**
    - Caching results of database queries
      - Cached results must be updated if underlying database changes
    - Caching of generated HTML
  - At the client's network
    - Caching of pages by Web proxy

Application Security

SQL Injection

- Suppose query is constructed using
  - "select * from instructor where name = ’ " + name + "’ "
- Suppose the user, instead of entering a name, enters:
  - X’ or ’ Y’ = ’ Y
- then the resulting statement becomes:
  - "select * from instructor where name = ’ " + "X’ or ’ Y’ = ’ Y" + "’ "
  - which is:
    - select * from instructor where name = ’ X’ or ’ Y’ = ’ Y’
  - User could have even used
    - X’ ; update instructor set salary = salary + 10000; --
- Prepared statement internally uses:
  "select * from instructor where name = ’ X\’ or \’ Y\’ = \’ Y’
- **Always use prepared statements, with user inputs as parameters**
- Is the following prepared statemen secure?
  - conn.prepareStatement("select * from instructor where name = ’ " + name + "’ “)

Cross Site Scripting

- HTML code on one page executes action on another page
  - E.g., <img src = http://mybank.com/transfermoney?amount=1000&toaccount=14523>
  - Risk: if user viewing page with above code is currently logged into mybank, the transfer may succeed
  - Above example simplistic, since GET method is normally not used for updates, but if the code were instead a script, it could execute POST methods
- Above vulnerability called **cross-site scripting (XSS)** or **cross-site request forgery (XSRF or CSRF)**
- **Prevent your web site from being used to launch XSS or XSRF attacks**
  - Disallow HTML tags in text input provided by users, using functions to detect and strip such tags
- **Protect your web site from XSS/XSRF attacks launched from other sites**
  - ..next slide

Cross Site Scripting

Protect your web site from XSS/XSRF attacks launched from other sites

- Use **referer** value (URL of page from where a link was clicked) provided by the HTTP protocol, to check that the link was followed from a valid page served from same site, not another site
- Ensure IP of request is same as IP from where the user was authenticated
  - Prevents hijacking of cookie by malicious user
- Never use a GET method to perform any updates
  - This is actually recommended by HTTP standard

Password Leakage

- Never store passwords, such as database passwords, in clear text in scripts that may be accessible to users
  - E.g., in files in a directory accessible to a web server
    - Normally, web server will execute, but not provide source of script files such as file.jsp or file.php, but source of editor backup files such as file.jsp~, or .file.jsp.swp may be served
- Restrict access to database server from IPs of machines running application servers
  - Most databases allow restriction of access by source IP address

Application Authentication

- Single factor authentication such as passwords too risky for critical applications
  - Guessing of passwords, sniffing of packets if passwords are not encrypted
  - Passwords reused by user across sites
  - Spyware which captures password
- Two-factor authentication
  - E.g., password plus one-time password sent by SMS
  - E.g., password plus one-time password devices
    - Device generates a new pseudo-random number every minute, and displays to user
    - User enters the current number as password
    - Application server generates same sequence of pseudo-random numbers to check that the number is correct.

Application Authentication

- **Man-in-the-middle** attack
  - E.g., web site that pretends to be mybank.com, and passes on requests from user to mybank.com, and passes results back to user
  - Even two-factor authentication cannot prevent such attacks
- Solution: authenticate Web site to user, using digital certificates, along with secure http protocol
- **Central authentication** within an organization
  - Application redirects to central authentication service for authentication
  - Avoids multiplicity of sites having access to user's password
  - LDAP or Active Directory used for authentication

Single Sign-On

- **Single sign-on** allows user to be authenticated once, and applications can communicate with authentication service to verify user's identity without repeatedly entering passwords
- **Security Assertion Markup Language (SAML)** standard for exchanging authentication and authorization information across security domains
  - E.g., user from Yale signs on to external application such as acm.org using userid joe@yale.edu
  - Application communicates with Web-based authentication service at Yale to authenticate user, and find what the user is authorized to do by Yale (e.g., access certain journals)
- **OpenID** standard allows sharing of authentication across organizations
  - E.g., application allows user to choose Yahoo! as OpenID authentication provider, and redirects user to Yahoo! for authentication

Application-Level Authorization

- Current SQL standard does not allow fine-grained authorization such as "students can see their own grades, but not other's grades"
  - Problem 1: Database has no idea who are application users
  - Problem 2: SQL authorization is at the level of tables, or columns of tables, but not to specific rows of a table
- One workaround: use views such as

  create view *studentTakes* **as**
  select *
  from *takes*
  where *takes.ID = syscontext.user_id*()

  - where syscontext.user_id() provides end user identity
    - End user identity must be provided to the database by the application
  - Having multiple such views is cumbersome

Application-Level Authorization (Cont.)

- Currently, authorization is done entirely in application
- Entire application code has access to entire database
  - Large surface area, making protection harder
- Alternative: **fine-grained (row-level) authorization** schemes
  - Extensions to SQL authorization proposed but not currently implemented
  - Oracle Virtual Private Database (VPD) allows predicates to be added transparently to all SQL queries, to enforce fine-grained authorization
    - E.g., add *ID= sys_context.user_id()* to all queries on student relation if user is a student

Audit Trails

- Applications must log actions to an audit trail, to detect who carried out an update, or accessed some sensitive data
- Audit trails used after-the-fact to
  - Detect security breaches
  - Repair damage caused by security breach
  - Trace who carried out the breach
- Audit trails needed at
  - Database level, and at
  - Application level

End of Chapter 9