Chapter 7: Complex Data Types

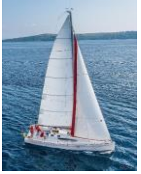
Presented by

Prof. Hosam El-Sofany

Professor of Computer Science
Department of Informatics and Computer Systems
College of Computer Science

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
  - 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

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*);
  - **create table** *people* **of** *Person*;
  - **insert into** *people* (*ID, name, address*) **values** ('12345', 'Srinivasan', '23 Coyote Run');
- Type inheritance
  - **create type** *Student* **under** *Person*
    (*degree* **varchar**(20)) ;
    create type *Teacher* **under** *Person*
    (*salary* **integer**);

Type and Table Inheritance

- 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
- Table inheritance syntax in PostgreSQL and oracle
  - **create table** *students*
    (*degree* **varchar**(20))
    inherits *people*;
    create table *teachers*
    (*salary* **integer**)
    inherits *people*;

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

Textual Data → Cancel

- **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

Spatial Data

Spatial Data → Cancel

- 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

Spatial Queries → Cancel

- **Region queries** deal with spatial regions. e.g., ask for objects that lie partially or fully inside a specified region
- **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

End of Chapter 7