CSC309 *Programming on the Web*

week 5: database

Amir H. Chinaei, Spring 2017

Office Hours: M 3:45-5:45 BA4222

ahchinaei@cs.toronto.edu
http://www.cs.toronto.edu/~ahchinaei/

review

- **so far:**
  - **front-end**
    - **structure & semantic**, **appearance**, **behavior**
    - many design tips
- **next:**
  - **back-end**
    - we start with **databases**
      - structured & semi-structured data

what is a database?

- it is a **collection of data**, typically describing the activities of one (or more related) application(s)
- the goal is to organize data in a way that facilitates **efficient *retrieval* and *modification***
- **note:** the data maintained by a system are much more important/valuable than the system itself
- A **database management system** (DBMS) is a software program to assist in maintaining and utilizing large databases

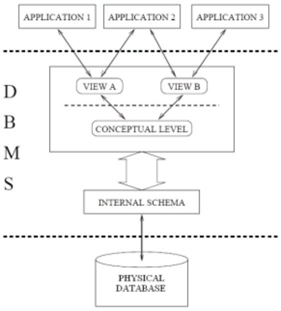
advantages of using a dbms

- data independence
- efficient data access
- data integrity and security
- data administration
- concurrent access and crash recovery
- reduced application development time

history

- 1962 IDS, first general purpose dbms by Charles Bachmann @ GE; Late 1960s IMS DBMS @ IBM
- 1971 Relational Data Model by Edgar Codd @ IBM
- 1973 Bachmann wins Turing award
- 1976 E-R Model by Peter Chen
- Late 1970s IBM's System R
- 1980s DB2 (SQL), Oracle, Informix, Sybase
- 1981 Codd wins Turing award
- Late 1980s O-O DBMSs
- 1990s SQL expansion, Internet development, XML
- Late 1990s, Relational DBMSs incorporate objects
- 1998 Jim Gray wins Turing award

3-level schema architecture



more on data independence

- **Idea:** application programs are isolated from changes in the way the data is structured & stored.
  - Indirect access supports:
    - advanced data structures
    - data restructuring
    - distribution and load balancing,
    - …
    - all without changes to applications
  - **Note:** A very important advantage of using a DBMS!

more on data independence

- **Logical**: applications immune from changes in the logical structure of the data.
  - Example:
    - Student (*name: string, major: string, DOB: integer*)
    - …
    - …
- **Physical**: applications immune from physical storage details.
  - Such as

    the file structure and the choice of indexes

more on relational model

Idea. All information is organized in flat relations.

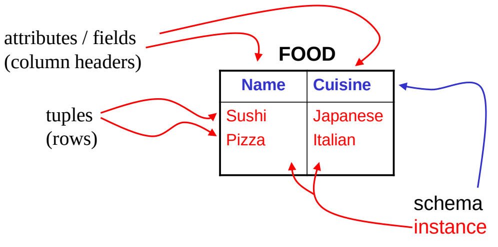
- Features:
  - very simple and clean data model
  - *often* matches how we think about data
  - abstract model that underlies SQL, the most popular database language
  - powerful and *declarative* query/update languages
  - semantic integrity constraints

transaction

A **transaction** is any ***one execution*** of a process in a DBMS, which is seen as a series of **actions—**such as *reads* and *writes*, followed by a *commit* or an *abort.*

- Properties of transactions: **(ACID)**
  - **Atomic:** either all actions or nothing are carried out.
  - **Consistency:** must preserve the DB constraints.
  - **Isolation:** understandable without considering other transactions.
  - **Durability:** once committed, the changes made are permanent.

a relation is a table



more tabular form

FOOD

Name	Cuisine
Pizza	Italian
Stroganoff	Russian
Poutine	Canadian

STUDENT

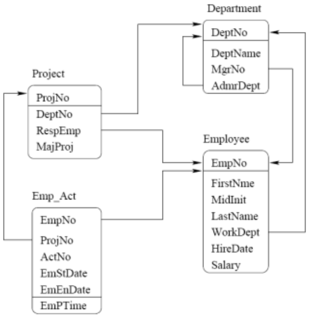
ID	Name	Major
1022083920	Adam	Math
901183280	Saniya	CS

LIKES

Student	Food
1022083920	Pizza
1022083920	Poutine
901183280	Pizza

that's why relations are often called "tables".

another example



SQL examples

```
❖ INSERT INTO food VALUES ( “Pizza”, “Canadian” );
❖ UPDATE food SET cuisine = “Italian”
     WHERE name = “Pizza”;
❖ SELECT name FROM food
     WHERE cuisine = “Russian”;
❖ SELECT cuisine, COUNT(*) AS “count”
     FROM food
     GROUP BY cuisine;
❖ SELECT DISTINCT cuisine
     FROM food,
              (SELECT food as name FROM likes,student
               WHERE  major=“CS”) csLikes
     WHERE food.name=csLikes.name;
```

summary

- Using a database to manage data helps:
  - to remove common code from applications
  - to provide uniform access to data
  - to guarantee data integrity
  - to manage concurrent access
  - to protect against system failure
  - to set access policies to data
  - . . .

XML

- eXtensible Markup Language
  - uses tags to specify semantics of data
    - for example: “food name”
- in a **well-formed XML**
  - elements have to nest properly
  - there must be one unique root element
  - attribute values must always be within quotes
- DTD (document type definition)
  - limits the elements and gives a grammar for their use
- **a valid XML**
  - has a DTD and conforms to it

example: well-formed XML

```
<? xml version = “1.0” standalone = “yes” ?>
<foodservices>
    <foodservice><name>Pizza Hut</name>
          <food><name>Pasta</name>
                    <cuisine>Italian</cuisine>
          </food>
          <food><name>Pizza</name>
                    <cuisine>Italian</cuisine>
          </food>
          …
    </foodservice>
     …
</foodservices>
```

example: DTD structure

```
< !DOCTYPE foodservices [
    <!ELEMENT foodservices (foodservice*)>
    <!ELEMENT foodservice (name, food+)>
    <!ELEMENT food (name, cuisine)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT cuisine (#PCDATA)>
]>
```

- **A DTD is essentially a CFG for the documents**.
  - The order of elements is important
    - The first sub-element of a food is its name, the second is its cuisine
  - Recursive structures are allowed.
    - <!ELEMENT node (leaf | (node, node)) >
    - <!ELEMENT leaf (#PCDATA)>

example: use of DTDs

```
<? xml version = “1.0” standalone = “no” ?>


< !DOCTYPE foodservices SYSTEM “foodservices.dtd”>


<foodservices>
    <foodservice><name>Pizza Hut</name>
        <food><name>Pasta</name><cuisine>Italian</cuisine></food>
        <food><name>Pizza</name><cuisine>Italian</cuisine></food>
    </foodservice>
</foodservices>
```

example: schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

example: use of schema

```
<?xml version="1.0"?>
<note
   xmlns="http://www.w3schools.com"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.w3schools.com note.xsd">

   <to>Tove</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Don't forget me this weekend!</body>
</note>
```

XSLT

- XML stylesheet transformations
- XSLT is an XML-based programming language that is used for transforming XML into other document formats

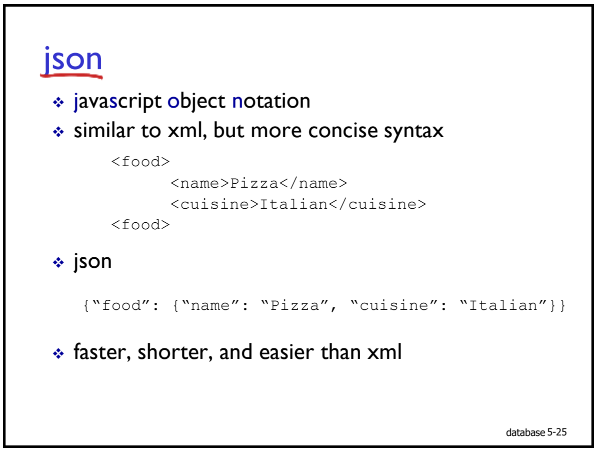
example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th style="text-align:left">Title</th>
      <th style="text-align:left">Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

xml dom processing in js

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var xhttp;
xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);}};
xhttp.open("GET", "books.xml", true);
xhttp.send();

function myFunction(xml) {
    var x, i, txt, xmlDoc;
    xmlDoc = xml.responseXML;
    txt = "";
    x = xmlDoc.getElementsByTagName("title");
    for (i = 0; i < x.length; i++) {
        txt += x[i].childNodes[0].nodeValue + "<br>";}
    document.getElementById("demo").innerHTML = txt;}
</script>
</body>
</html>
```



json

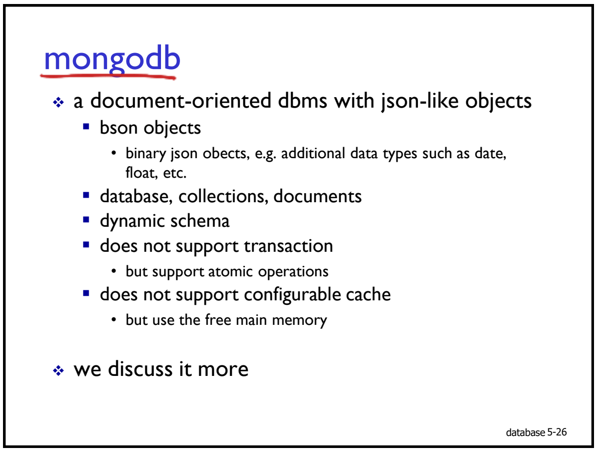
- javascript object notation
- similar to xml, but more concise syntax

```
<food>
      <name>Pizza</name>
      <cuisine>Italian</cuisine>
<food>
```

- json

```
{“food”: {“name”: “Pizza”, “cuisine”: “Italian”}}
```

- faster, shorter, and easier than xml



mongodb

- a document-oriented dbms with json-like objects
  - bson objects
    - binary json obects, e.g. additional data types such as date, float, etc.
  - database, collections, documents
  - dynamic schema
  - does not support transaction
    - but support atomic operations
  - does not support configurable cache
    - but use the free main memory
- we discuss it more