CS 133: Databases

Fall 2019
Lec 23 – 12/3
Database Design: OO and XML
Prof. Beth Trushkowsky

Warm-up Exercise

(See exercise sheet. You can start before class.)

To avoid anomalies caused by data redundancy.

Goals for Today

- Understand the motivation behind object-oriented (OODBMS), object-relational (ORDBS), and object-relational mapping (ORM)
- Reason about non-relational DBMSs
- Explore XML: semi-structured data model; querying capability

Reflections on the Relational Model

- Relations are the key concept
  - Clean and simple, efficient implementation
  - **Primitive data types**, e.g., strings, integer, (and **BLOB**)
  - Great: normalization, query optimization, and theory
- Some issues
  - No **complex data types** or objects
  - No inheritance or encapsulation

Stonebraker's Classification of DBMS Applications

- Classification of the applications that require DBMS technology
  - One size doesn't fit all!
  - RDBMS, object-relational DBMS, object-oriented DBMS

	Simple Data	Complex Data
Query		
No Query		

http://db.cs.berkeley.edu/papers/Informix/www.informix.com/informix/corpinfo/zines/whitpprs/illuswp/wave.htm

Stonebraker's Classification of DBMS Applications

- Text editor application
  - Open file, make changes, write file back to disk
  - "Queries"
    - getFile()
    - writeFile()
  - Data Model
    - Arbitrary sequence of characters

	Simple Data	Complex Data
Query		
No Query	File System	

http://db.cs.berkeley.edu/papers/Informix/www.informix.com/informix/corpinfo/zines/whitpprs/illuswp/wave.htm

Stonebraker's Classification of DBMS Applications

- Business data processing
  - Store a collection of structured records, each of which has attributes
  - Data are simple integers, floats and character strings
  - Relations with SQL queries

	Simple Data	Complex Data
Query	RDBMS	
No Query	File System	

```
create table emp (
  name    varchar(30),
  age     int,
  salary  float,
  dept    varchar(20));
```

```
select name
from emp
where age < 40 and salary > 40000;
```

http://db.cs.berkeley.edu/papers/Informix/www.informix.com/informix/corpinfo/zines/whitpprs/illuswp/wave.htm

Stonebraker's Classification of DBMS Applications

- Facilities planner for a company that has an open floor plan
  - Occasionally rearrange floor plan to reclaim space

	Simple Data	Complex Data
Query	RDBMS	
No Query	File System	OODBMS

```
create table employee (
  name        varchar(30),
  space       polygon,
  adjacency   set-of (employee));

create table floors (
  number      int,
  asf         swiss-cheese-polygon);
```

Complex data types!

```
spaceReclamation()
{
  read all employees;
  read all floors;
  compact();
  write all employees;
}
```

With persistent variables, only need `compact()`

Not unlike CAD applications that motivated these systems

http://db.cs.berkeley.edu/papers/Informix/www.informix.com/informix/corpinfo/zines/whitpprs/illuswp/wave.htm

Stonebraker's Classification of DBMS Applications

	Simple Data	Complex Data
Query	RDBMS	Object-relational DBMS (ORDBMS)!
No Query	File System	OODBMS

http://db.cs.berkeley.edu/papers/Informix/www.informix.com/informix/corpinfo/zines/whitpprs/illuswp/wave.htm

Exercise 2: Design relational schema

- Possible relations:
  - Books(booktitle, year, pub_name, pub_branch)
  - Authored(booktitle, author_name, position)
  - HasKeyword(booktitle, keyword)
  - Publisher(name, branch, address)
- Might also have:
  - Authors(name)
  - Keywords(word)

- Wouldn't it be nice if we could do this:

Title	Author_array	Publisher_info	Year	Keyword_set
Compilers	[Smith, Jones]	(McGraw-Hill, New York, 55 Park Ave)	2019	{parsing, analysis}
Networks	[Jones, Frick]	(Oxford, London, 12 Oxford St)	2020	{Internet, Web}

Example adapted from: Database System Concepts - 6th Edition

Running Example: Dinky's Entertainment Company

- Hollywood conglomerate
  - Collection of cartoon characters (e.g., Herbert the Worm)
  - Films featuring Herbert
  - Licensing for images, voice, action figures, etc.

- Database need
  - Manage sales and leasing records for Herbert-related products, as well as films

Disclaimer: the following schema examples use features proposed in the SQL:1999 standard.

Specific DBMSs may not comply with syntax/features!

Complex Types: Abstract Data Types

- **Motivation**: data types that represent image, voice, video footage
  - Richer structure
  - Special functions to manipulate objects of these types

```
CREATE ABSTRACT DATA TYPE jpeg_image();
```

```
CREATE TABLE Frames(
    frameno integer,
    image jpeg_image,
    category integer);
```

```
CREATE FUNCTION is_sunrise(jpeg_image)
RETURNS boolean AS EXTERNAL NAME
'file.class' LANGUAGE java;
```

```
SELECT F.frameno, thumbnail(F.image)
FROM Frames F
WHERE is_sunrise(F.image) AND is_herbert(F.image);
```

Complex Types: Structured Data Types

- **Motivation**: types with *internal structure* help with data abstraction
  - No longer only have atomic data types

```
CREATE TABLE Films(
   filmno integer,
   title text,
   stars varchar(25) array[10]);
```

```
SELECT F.title
FROM Films F
WHERE F.stars[1] =
      'Herbert the Worm';
```

Array index starts at 1 ☹

```
CREATE TYPE theater_t AS ROW(
   tno integer,
   name text,
   address text);
```

Can access an item *i* with type `theater_t` using dot notation, e.g., *i.name*

Added in SQL:2003 is an unordered collection called *multiset*

Complex Types: Object Identifiers and References

- **Motivation**: new data types might be quite large, so want to store *references* to them, not copies

```
CREATE TABLE Theaters OF theater_t REF is tid SYSTEM GENERATED;
```

```
CREATE TABLE NowShowing(
   film integer,
   theater REF(theater_t) SCOPE Theaters,
   start date,
   end date);
```

Dereferencing the pointer using ->

```
SELECT N.theater->name, N.theater->address, F.title
FROM Nowshowing N, Films F
WHERE N.film = F.filmno AND F.stars[1] = 'Herbert the Worm';
```

Exercise 3

- For each *lead* star (first star in the array), show the count of theaters currently showing a movie with them as the lead (you can assume today's date is "*today*")

```
CREATE TABLE NowShowing(
   film integer,
   theater REF(theater_t) SCOPE Theaters,
   start date,
   end date);
```

```
CREATE TABLE Films(
   filmno integer,
   title text,
   stars varchar(25) array[10]);
```

```
SELECT F.stars[1] AS leadStar, COUNT(DISTINCT theater->tno)
FROM Films F, Nowshowing N
WHERE  F.filmno = N.film AND start <= today AND end > today
GROUP BY F.stars[1];
```

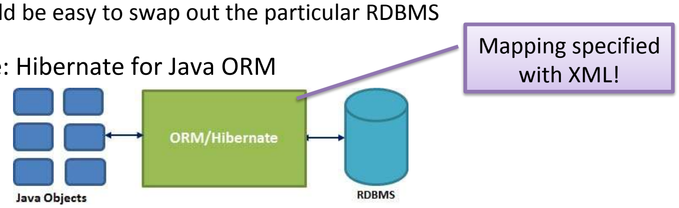
Extensibility

- Support indexes over new data types
  - E.g., GiST in PostgreSQL
    - Template tree-based index, you write functions to split nodes, etc.
- New Aggregations!
  - E.g., CREATE AGGREGATE in PostgreSQL
    - State transition function
    - Final function

Just like in SimpleDb!

Object-Relational Mapping (ORM)

- ***Motivation***: Solve the *object-relational impedance mismatch* problem
  - Give programmers the object model, with robust RDBMS underneath
  - Programmer defines mapping between objects and tuples in relations
  - Should be easy to swap out the particular RDBMS
- Example: Hibernate for Java ORM



- Common with web frameworks and Model-View-Control (MVC)
  - Django (python)
  - Play (java or scala)
  - Ruby on Rails (ruby)
  - …

http://www.tutorialspoint.com/hibernate/hibernate_overview.htm

Object-Relational Mapping (ORM)

Django

```
from django.db import models

class Musician(models.Model):
    name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)

class Album(models.Model):
    artist = models.ForeignKey(Musician)
    name = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars = models.IntegerField()
```

```
# Create a Musician
>>> m = Musician(name="Billy Joel", "Piano")
>>> m.save()
# Find 5-star albums
>>> Album.objects.filter(num_stars = 5)
```

Ruby on Rails

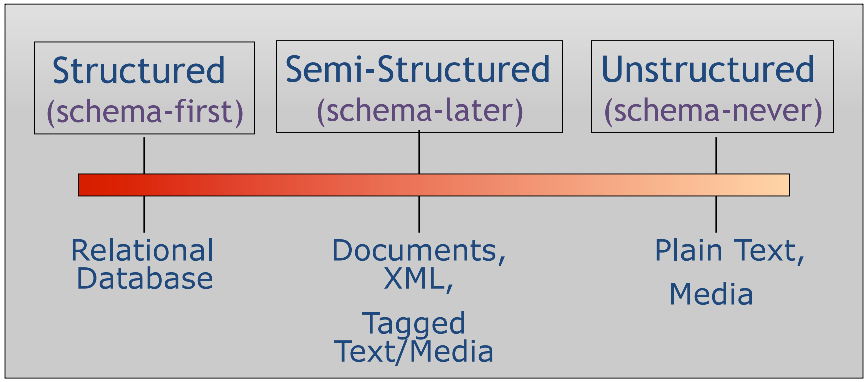
```
class Manager < ActiveRecord::Base
  has_many :employees
end

class Employee < ActiveRecord::Base
  belongs_to :manager   # foreign key - manager_id
end
```

One-to-many relationship

http://blog.hasmanythrough.com/2007/1/15/basic-rails-association-cardinality

The Structure Spectrum



Thanks Mike Franklin

XML: eXtensible Markup Language

- A document's ***markup* is metadata** not intended as part of output
  - *Markup language*: formal description of which parts of document are **content vs. markup**

HTML:
set of markup *tags* pre-defined

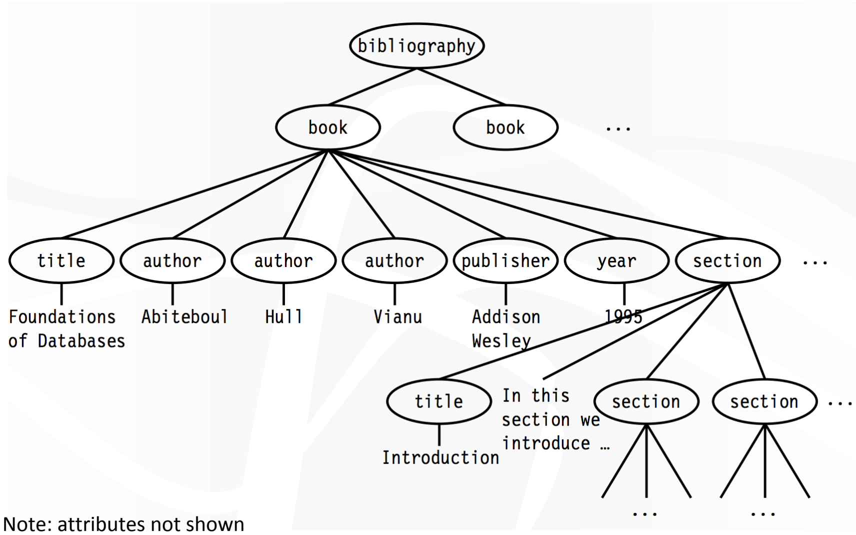
```
<html>
    <head>
        <title>CS 133 – Databases </title>
    </head>
    <body>
    …
```

```
<bibliography>
    <book ISBN="ISBN-10" price="160.00">
        <title>Database Management Systems</
title>
        <author>ramakrishnan</author>
        <author>Gehrke</author>
        <publisher>McGraw-Hill</publisher>
        <year>2003</year>
        <is_textbook/>
    </book>
</bibliography>
```

XML:
set of markup *tags* defined/modified by application as needed

Thanks to Jun Yang for some XML content and examples

XML Tree Representation



Note: attributes not shown

XML Terminology

- **Tag** names: book, title, …
  - Start tags: <book>, <title>, …
  - End tags: </book>, </title>, …
  - An ***element*** is enclosed by a pair of start and end tags: <book>…</book>
- Elements can be nested: <book>…<title>…</title>…</book>
- Empty elements can be abbreviated: <is_textbook/>
- Elements can also have ***attributes***: <book ISBN="…"price="160.00">

```
<bibliography>
  <book ISBN="ISBN-10" price="160.00">
    <title>Database Management
Systems</title>
    <author>Ramakrishnan</author>
    <author>Gehrke</author>
    <publisher>McGraw-Hill</publisher>
    <year>2003</year>
    <is_textbook/>
  </book>
  <book>…</book>
</bibliography>
```

Well-formed XML documents have a **single root** element and **properly nested** elements

XPath Expressions

- XPath specifies **path expressions** that **match XML data** by navigating down (and occasionally up and across) the tree
- Result is a sequence of items (nodes in the original document)

Example XPath query:
`/bibliography/book/author`

All author elements reachable along this path from the root

XPath Expressions (cntd)

- [*condition*] filters a ***sequence***
  - An item in the sequence is retained if *condition* evaluates to true on that item
  - Evaluates to true as long as it evaluates true for *at least one* node in the sequence

`/bibliography/book[@price<50]`

Book elements with price attribute less than 50

Defining an XML "Schema"

- A valid XML document conforms to a Document Type Definition (**DTD**)
  - Grammar for the XML document
  - Constraints on structures and values of elements, attributes, …

The *bibliography* element can have one or more *book* child elements

```
<!DOCTYPE bibliography [
   <!ELEMENT bibliography (book+)>
   <!ELEMENT book (title, author*, publisher?, year?, section*)>
   <!ATTLIST book ISBN CDATA #REQUIRED>
   <!ATTLIST book price CDATA #IMPLIED>
   <!ELEMENT title (#PCDATA)>
   <!ELEMENT author (#PCDATA)>
   <!ELEMENT publisher (#PCDATA)>
   <!ELEMENT year (#PCDATA)>
   <!ELEMENT section (title, (#PCDATA)?, section*)>
]>
```

The *book* element has an optional *price* attribute that is character data

Basic XPath constructs

/	separator between steps in a path
name	matches any child element with this tag name
*	matches any child element
@name	matches the attribute with this name
@*	matches any attribute
//	matches any descendent element or the current element itself
.	matches the current element
..	matches the parent element

Exercise 4

What will the following Xpath expressions yield?

a) `/bibliography/book[@price > 100]/author`

Authors of books that cost more than 100

b) `/bibliography/book[year < 2000]/@price`

Prices of books published before 2000

c) `/bibliography/book[author='Gehrke']`

Context item is a sequence of authors… will return true for a book if at *least one* of the authors matches.

XQuery

- XPath + full-fledged SQL-like query language
- An XQuery expression in general can return a new resulting XML document!
- Example: Find all books with price lower than $50

doc( ) specifies the document to query

```
<result>
{
   doc("bib.xml")/bibliography/book[@price<50]
}
</result>
```

Things outside {}'s are copied to output verbatim

Things inside {}'s are evaluated and replaced by the results

Matching book elements (and their descendants) copied to output!