Review

Final Exam

• WOODW 135 Thr 12/16/10 8:00 AM 10:30 AM
**SQL**
**Relational Algebra**
**ER model and ER-Relational mapping**
*Database programming techniques*
*XML*
**Normalization**
*Disk storage*
*Index*
*Query execution and optimization*

* Small questions
** “Big” questions

---

**SQL Programming Language**

- Database Programming: Techniques and Issues
- Three approaches:
  - Embedded SQL, Dynamic SQL, and SQLJ
  - Database Programming with Function Calls: SQL/CLI and JDBC
  - Database Stored Procedures and SQL/PSM
Approaches to Database Programming

• **Embedding** database commands in a general-purpose programming language
  – Database statements identified by a special prefix
  – **Precompiler** or **preprocessor** scans the source program code
    • Identify database statements and extract them for processing by the DBMS
  – Called **embedded SQL**

Approaches to Database Programming (cont’d.)

• Using a library of database functions
  – **Library of functions** available to the host programming language
  – **Application programming interface (API)**
• Designing a brand-new language
  – **Database programming language** designed from scratch
    – Example: Oracle’ PL/SQL
• First two approaches are more common
Impedance Mismatch

• Differences between database model and programming language model
  – Database model: columns and their data types
  – Database model: rows
• **Binding** for each host programming language
  – Specifies for each attribute type the compatible programming language types
• Cursor or iterator variable
  – Loop over the tuples in a query result

Typical Sequence of Interaction in Database Programming

• Open a connection to database server
• Interact with database by submitting queries, updates, and other database commands
• Terminate or close connection to database
Database Programming with Function Calls: SQL/CLI & JDBC

- Use of function calls
  - Dynamic approach for database programming
- Library of functions
  - Also known as application programming interface (API)
  - Used to access database

JDBC: SQL Function Calls for Java Programming
Database Stored Procedures and SQL/PSM

• **Stored procedures**
  – Program modules stored by the DBMS at the database server
  – Can be functions or procedures

• **SQL/PSM (SQL/Persistent Stored Modules)**
  – Extensions to SQL
  – Include general-purpose programming constructs in SQL

Database Stored Procedures and Functions

• **Persistent stored modules**
  – Stored persistently by the DBMS

• Useful:
  – When database program is needed by several applications
  – To reduce data transfer and communication cost between client and server in certain situations
  – To enhance modeling power provided by views
Comparing the Three Approaches

• Embedded SQL Approach
  – Query text checked for syntax errors and validated against database schema at compile time
  – For complex applications where queries have to be generated at runtime
    • Function call approach more suitable

Comparing the Three Approaches (cont’d.)

• Library of Function Calls Approach
  – More flexibility
  – More complex programming
  – No checking of syntax done at compile time

• Database Programming Language Approach
  – Does not suffer from the impedance mismatch problem
  – Programmers must learn a new language
Structured, Semistructured, and Unstructured Data

- **Structured data**
  - Represented in a strict format
  - Example: information stored in databases
- **Semistructured data**
  - Has a certain structure
  - Not all information collected will have identical structure
  - Schema information mixed in with data values
  - **Self-describing data**
- **Unstructured data**
  - Example: HTML

**Semistructured data as a tree or a graph**

*Figure 12.1*
Representing semistructured data as a graph.
What is XML?

• Stands for EXtensible Markup Language
• Transports and stores data over the web
  – To carry data, not to display data
• Is a markup language much like HTML
• Is self-descriptive

“XML is a software- and hardware-independent tool for carrying information.”

Difference Between XML and HTML

• XML is not a replacement for HTML.
• XML and HTML have different goals:
  – XML to transport and store data
    • Focus: “What data is”
  – HTML to display data
    • Focus: “Howe data looks”
• XML tags are not predefined. You must define your own tags
Example XML Document

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Tree Structure of XML Documents
XML Syntax Rules

- All XML Elements Must Have a Closing Tag
  - XML declaration does not have a closing tag
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
  Example: `<b><i>This text is bold and italic</i></b>`
- XML Documents Must Have a Root Element
- XML Attribute Values Must be Quoted

- XML can have empty elements
- Comments: `<!-- This is a comment -->`

XML Validation

- XML with correct syntax is "Well Formed" XML.
- Well-formed XML documents can contain errors. They will stop your XML applications.
- XML validated against a Document Type Definition (DTD) is "Valid" XML.
- XML schema is the successor of DTDs
XML Document Type Definition (DTD)

• DTD defines the document structure with a list of legal elements and attributes.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The external DTD file:

```xml
<!DOCTYPE note [
  <!ELEMENT note (to, from, heading, body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]
```

Why Use a DTD?

• With a DTD, each of your XML files can carry a description of its own format.
• With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
• Your application can use a standard DTD to verify that the data you receive from the outside world is valid.
• You can also use a DTD to verify your own data.
**XML Schema**

- XML-based alternative to DTD
- The XML Schema language is also referred to as XML Schema Definition (XSD).
- Successors of DTDs
  - Are extensible to future additions
  - Are richer and more powerful than DTDs
  - Are written in XML
  - Support data types
  - Support namespaces

---

**Document Object Model (DOM)**

- defines a standard way for accessing and manipulating documents.
- a XML parser could convert an XML document into an XML DOM object
- DOM views an XML document as a tree-structure.
- All elements can be accessed through the DOM tree. Their content (text and attributes) can be modified or deleted, and new elements can be created. The elements, their text, and their attributes are all known as nodes.
XML Query Languages

• Two query language standards
  – XPath
    • is used to navigate through elements and attributes in an XML document.
    • Specifies path expressions to identify certain nodes (elements) or attributes within an XML document that match specific patterns
  – XQuery
    • Uses XPath expressions but has additional constructs
    • XQuery is to XML what SQL is to database tables

Informal Design Guidelines for Relation Schemas

• Measures of quality
  – Making sure attribute semantics are clear
  – Reducing redundant information in tuples
  – Reducing NULL values in tuples
  – Disallowing possibility of generating spurious tuples
Guideline 1

• Design relation schema so that it is easy to explain its meaning
• Do not combine attributes from multiple entity types and relationship types into a single relation

<table>
<thead>
<tr>
<th>name</th>
<th>id</th>
<th>level</th>
<th>advisor_id</th>
<th>advisor_office</th>
<th>favorite_advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark</td>
<td>1</td>
<td>Senior</td>
<td>349</td>
<td>AN151</td>
<td>350</td>
</tr>
<tr>
<td>??</td>
<td>1</td>
<td>???</td>
<td>350</td>
<td>MH134</td>
<td>??</td>
</tr>
<tr>
<td>Kathy</td>
<td>2</td>
<td>Senior</td>
<td>146</td>
<td>AN240</td>
<td>146</td>
</tr>
<tr>
<td>??</td>
<td>1</td>
<td>???</td>
<td>351</td>
<td>AN130</td>
<td>??</td>
</tr>
<tr>
<td>??</td>
<td>1</td>
<td>???</td>
<td>352</td>
<td>AN131</td>
<td>??</td>
</tr>
<tr>
<td>??</td>
<td>2</td>
<td>???</td>
<td>351</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>David</td>
<td>3</td>
<td>Junior</td>
<td>349</td>
<td>???</td>
<td>349</td>
</tr>
</tbody>
</table>

Guideline 2

• Design base relation schemas so that no update anomalies are present in the relations
• If any anomalies are present:
  – Note them clearly
  – Make sure that the programs that update the database will operate correctly
Guideline 3

• Avoid placing attributes in a base relation whose values may frequently be NULL
• If NULLs are unavoidable:
  – Make sure that they apply in exceptional cases only, not to a majority of tuples

Guideline 4

• Design relation schemas to be joined with equality conditions on attributes that are appropriately related
  – Guarantees that no spurious tuples are generated
• Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations
Normalization

• ideas → E/R → relations → better (normalized) relations
• **Normalization**: process of making relations better by decomposing them into smaller relations to
  – reduce redundancy
  – eliminate update anomalies
• final goal: all relations in Boyce-Codd Normal Form (BCNF)
• Analytic tool: **Functional Dependency**

---

Functional Dependency (cont’d)

• R is a relation, A and B are its attributes.
  R.A → R.B if and only if for each value of A no more than one value of B is associated.
  Namely
  if \( t_1 \) and \( t_2 \) are two tuples in the relation \( R \) and \( t_1(A) = t_2(A) \) then we must have \( t_1(B) = t_2(B) \)
Functional Dependency (cont’d)

• R is a relation, R.A -> R.B
• A and B can be sets of attributes

<table>
<thead>
<tr>
<th>Essn</th>
<th>Dependent_name</th>
<th>Sex</th>
<th>Bdate</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>333445555</td>
<td>Alice</td>
<td>F</td>
<td>1986-04-05</td>
<td>Daughter</td>
</tr>
<tr>
<td>333445555</td>
<td>Theodore</td>
<td>M</td>
<td>1983-10-25</td>
<td>Son</td>
</tr>
<tr>
<td>333450500</td>
<td>Joy</td>
<td>F</td>
<td>1908-03-03</td>
<td>Spouse</td>
</tr>
<tr>
<td>987654321</td>
<td>Abner</td>
<td>M</td>
<td>1942-02-28</td>
<td>Spouse</td>
</tr>
<tr>
<td>123456789</td>
<td>Michael</td>
<td>M</td>
<td>1988-01-04</td>
<td>Son</td>
</tr>
<tr>
<td>123456789</td>
<td>Alice</td>
<td>F</td>
<td>1988-12-30</td>
<td>Daughter</td>
</tr>
<tr>
<td>123456789</td>
<td>Elizabeth</td>
<td>F</td>
<td>1967-05-05</td>
<td>Spouse</td>
</tr>
</tbody>
</table>

Normalization of Relations

• Assumption: a set of FD is given to each relation; each relation has a designated primary key
• Takes a relation schema through a series of tests
  – Certify whether it satisfies a certain normal form
  – Proceeds in a top-down fashion
• Normal form tests
  Definition. The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.
Definitions of Keys and Attributes Participating in Keys

- Definition of **superkey** and **key** (key has to be minimal)
- **Candidate key**
  - If more than one key in a relation schema
    - One is **primary key**

**Definition.** An attribute of relation schema $R$ is called a prime attribute of $R$ if it is a member of some candidate key of $R$. An attribute is called nonprime if it is not a prime attribute—that is, if it is not a member of any candidate key.
First Normal Form

• Part of the formal definition of a relation in the basic (flat) relational model
• Only attribute values permitted are single atomic (or indivisible) values

First Normal Form (cont’d.)

• Techniques to achieve first normal form
  – Remove attribute and place in separate relation
  – Expand the key
  – Use several atomic attributes
Second Normal Form

• Based on concept of **full functional dependency**
  – Versus **partial dependency**

<table>
<thead>
<tr>
<th>Dependent_id</th>
<th>Dependent_name</th>
<th>Sex</th>
<th>Bdate</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>333445555</td>
<td>Alice</td>
<td>F</td>
<td>1986-04-05</td>
<td>Daughter</td>
</tr>
<tr>
<td>333445555</td>
<td>Theodore</td>
<td>M</td>
<td>1983-10-25</td>
<td>Son</td>
</tr>
<tr>
<td>333445555</td>
<td>Joy</td>
<td>F</td>
<td>1958-05-03</td>
<td>Spouse</td>
</tr>
<tr>
<td>987654321</td>
<td>Abner</td>
<td>M</td>
<td>1942-02-28</td>
<td>Spouse</td>
</tr>
<tr>
<td>123458789</td>
<td>Michael</td>
<td>M</td>
<td>1988-01-04</td>
<td>Son</td>
</tr>
<tr>
<td>123458789</td>
<td>Alice</td>
<td>F</td>
<td>1988-12-30</td>
<td>Daughter</td>
</tr>
<tr>
<td>123458789</td>
<td>Elizabeth</td>
<td>F</td>
<td>1967-05-05</td>
<td>Spouse</td>
</tr>
</tbody>
</table>

Consider a new column called “Employ_Name” is added into this table.

**Second Normal Form**

**Definition.** A relation schema $R$ is in 2NF if every nonprime attribute $A$ in $R$ is fully functionally dependent on the primary key of $R$.

• Practice: second normalize the previous table with “Employ_name” attribute
• Second normalize into a number of 2NF relations
  – Nonprime attributes are associated only with part of primary key on which they are fully functionally dependent
Practice

Third Normal Form

- Based on concept of transitive dependency
  - X -> Y, Y->Z and Y is neither a candidate key or a subset of any key
- Example:
Definition. According to Codd’s original definition, a relation schema $R$ is in 3NF if it satisfies 2NF and no nonprime attribute of $R$ is transitively dependent on the primary key.

- Problematic FD
  - Left-hand side is part of primary key
  - Left-hand side is a nonkey attribute

### Practice

<table>
<thead>
<tr>
<th>Tournament</th>
<th>Year</th>
<th>Winner</th>
<th>Winner Date of Birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indiana Invitational</td>
<td>1998</td>
<td>Al Fredrickson</td>
<td>21 July 1975</td>
</tr>
<tr>
<td>Cleveland Open</td>
<td>1999</td>
<td>Bob Albertson</td>
<td>28 September 1968</td>
</tr>
<tr>
<td>Des Moines Masters</td>
<td>1999</td>
<td>Al Fredrickson</td>
<td>21 July 1975</td>
</tr>
<tr>
<td>Indiana Invitational</td>
<td>1999</td>
<td>Chip Masterson</td>
<td>14 March 1977</td>
</tr>
</tbody>
</table>
Practice

- CAR_SALE (CarID, Option_type, Option_Listprice, Sale_date, Discounted_price)
- CarID → Sale_date
- Option_type → Option_Listprice
- CarID, Option_type → Discounted_price

Practice

- Student (Sid, Name, Age, Did, Dname)
Queries:
1. Find the title and year of all books published by “Addison Wesley” (a publisher)
2. Find all authors who have published any books by “Addison Wesley”
3. How many books are published by “Addision Wesley” in year 2009?
4. Print all the name and the number of books published by each author.
5. Which publisher published the largest number of books?
### Disk Storage

<table>
<thead>
<tr>
<th></th>
<th>Heap File</th>
<th>Sorted File</th>
<th>Hashed File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan all recs</td>
<td>BD</td>
<td>BD</td>
<td>1.25 BD</td>
</tr>
<tr>
<td>Equality Search</td>
<td>0.5 BD</td>
<td>(D \log_2 B)</td>
<td>D</td>
</tr>
<tr>
<td>Range Search</td>
<td>BD</td>
<td>(D (\log_2 B + # \text{ of pages with matches}))</td>
<td>1.25 BD</td>
</tr>
<tr>
<td>Insert</td>
<td>2D</td>
<td>Search + BD</td>
<td>2D</td>
</tr>
<tr>
<td>Delete</td>
<td>Search + D</td>
<td>Search + BD</td>
<td>2D</td>
</tr>
</tbody>
</table>

### RAID

### Single Level Indexes

**Table 18.2** Properties of index types.

<table>
<thead>
<tr>
<th>Type of Index</th>
<th>Number of (First-level) Index Entries</th>
<th>Dense or Non-dense (Sparse)</th>
<th>Block Anchoring on the Data File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary</td>
<td>Number of blocks in data file</td>
<td>Non-dense</td>
<td>Yes</td>
</tr>
<tr>
<td>Clustering</td>
<td>Number of distinct index field values</td>
<td>Non-dense</td>
<td>Yes/no(^4)</td>
</tr>
<tr>
<td>Secondary (key)</td>
<td>Number of records in data file</td>
<td>Dense</td>
<td>No</td>
</tr>
<tr>
<td>Secondary (nonkey)</td>
<td>Number of records or number of distinct index field values</td>
<td>Dense or Non-dense</td>
<td>No</td>
</tr>
</tbody>
</table>
Multi-Level Indexes

- Because a single-level index is an ordered file, we can create a primary index to the index itself;
  - In this case, the original index file is called the first-level index and the index to the index is called the second-level index.
- We can repeat the process, creating a third, fourth, ..., top
- level until all entries of the top level fit in one disk block
- A multi-level index can be created for any type of first-level index (primary, secondary, clustering) as long as the first-level index consists of more than one disk block

Dynamic Multilevel Indexes Using B+-Trees

- Most multi-level indexes use B-tree or B+-tree data structures because of the insertion and deletion problem
  - This leaves space in each tree node (disk block) to allow for new index entries
- These data structures are variations of search trees that allow efficient insertion and deletion of new search values.
- In B-Tree and B+-Tree data structures, each node corresponds to a disk block
- Each node is kept between half-full and completely full
• Index on multiple keys
• Bitmap index

Query execution and optimization
• External sort: Sort-Merge strategy
• Implementation of selection
• Implementation of join: sort-merge join
Projection Submission

• If you have GUI
• If you don’t have GUI