Database Issues

The following databases are created with password as 'changeit'
- anticyber
- cyber
- cedar
- dogwood
- elm
- clan

Goals

- Separate the application code from the database
  - Encourages you to “think objects” – not relational
  - Requires you to defer the design of the DB
  - Allows you to modify the design of the DB without changing all your application code
- Consistent application-level view of data
- Portability of DBMS (change DBMS vendor without rewriting a substantial part of application)
- Easy access to test data and GUI-related data
- Good DB performance
References

- Object-Relational Mapping SW

- TopLink - Reference implementation of JPA
  [htten.wikipedia.org/wiki/TopLink](htten.wikipedia.org/wiki/TopLink)

- JDBC
  [docs.oracle.com/cd/E11882_01/java.112/e16548/toc.htm](docs.oracle.com/cd/E11882_01/java.112/e16548/toc.htm)
  [docs.oracle.com/cd/B28359_01/java.111/b31224.pdf](docs.oracle.com/cd/B28359_01/java.111/b31224.pdf)
  [en.wikipedia.org/wiki/Java_Database_Connectivity](en.wikipedia.org/wiki/Java_Database_Connectivity)

- API
  - java.sql and javax.sql

Approaches

- Create data classes corresponding to domain classes
- Create a robust persistence layer
- Use a persistence layer framework (e.g., Hibernate, Java Persistence API, etc)

Don’t write DB access code directly in your JSPs (including tags)

We deal with JPA in a different session

Good to get experience with a persistence framework
Your Own Persistence Layer

- You decide which of your classes and which attributes of those classes need to persist (i.e., stored in DB)
- Ideally, you should just need to send the following messages to an object to have the desired result
  - save
  - retrieve
  - delete
- Allows you to build your application before you complete your DB
- Allows you to insert simple test data in domain objects

Approach to Testing

- Approach – incremental builds
  - DB access does not need to be in early builds
  - Entity class getter methods (e.g., getBookTitle) can be initially implemented with simple test data
Database Preliminaries

- Recap important topics
- Recap terminology
- Use a good DB modelling tool (e.g., Workbench)
- You will implement the DB on a shared CS server (e.g., MySQL)
- You might do some early testing on a DB on your system (e.g., Derby)

An Entity

- Usually corresponds to something concrete in the domain of the application
- Represented by a rectangle
- An instance is a particular occurrence of an entity (corresponds to a row in a DB table)
Attributes

- Also referred to as properties
- An attribute is a discrete data element that describes an entity
- Attribute names should be meaningful

Identifiers (Primary Keys)

- Every instance of an entity (think row of a table) must be uniquely identified
- An identifier (primary key) can be one or more attributes
- Better to use an identifier that does not relate to a domain attribute (guaranteed uniqueness)
- A leading asterisk denotes an identifier (sometimes, another notation is used, e.g., PK)
DB Naming Conventions

- No universal standard
- Good to be consistent within a project
- Camel case used frequently

CSE308 DB Naming Conventions

- Camel case for table names (upper cc) and column names (lower cc)
- Table names – plural (unlike OO convention)
- Column names – singular
- Primary Key field – ID
- Avoid acronyms and abbreviations except where well known (e.g., PI for Principal Investigator)
Data Modeling

- A technique for modeling data
- We assume
  - RDM Model (Relational Data Model)
- The goal is to identify the structure of data to be stored in the database

ER Model is applicable to non-relational DB, but we assume a relational implementation

The Building Blocks

- Entity
- Attribute
- Relationship
- Identifier

Remember our notation is plural entity name in upper camel case and singular attribute names in lower camel case
A Well-Formed Data Model

- Follow organization (e.g., your company) convention
- No ambiguity
  - All entities, attributes, relationships, and identifiers are defined
  - Names are meaningful to the client

Relationships

- ERD and RDM show relationships between entities
  - 1-1
  - 1-many
  - Recursive
- ERD shows
  - Many-many
  - No foreign keys
- RDM usually shows
  - Associative entity (in-between table)
  - Foreign keys

Workbench uses more of a DB model style (not an ERD)
RDM Iteration

- You will usually iterate on your RDM model, as you learn the entities and relationships better
  - Is the level of detail correct?
  - Do the names represent the real-world entities and attributes?
  - Are the model components situated so that the design is clear (e.g., no hidden or crossing lines)
  - Are all exceptions handled?
  - Is the model accurate?

Normalization

- A theoretical foundation for the relational model
- Application of a series of rules that gradually improve the design
  - Minimize redundancy
  - Minimize dependency
- Objectives*
  - Free the collection of relations from undesirable insertion, update and deletion dependencies
  - Isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database

* Wikipedia
Normal Forms

- Based on rules about relationships among the columns of a table
- Removes data redundancies that can cause update anomalies
- A classification of relations
  - 1NF
  - 2NF
  - 3NF
  - BCNF
  - 4NF
  - 5NF

Usually, only the first 3 normal forms are applied to a DB

Data Redundancy

- Major aim of relational database design is to group columns into tables to:
  1. minimize data redundancy and
  2. reduce file storage space required by implemented base tables

Problems associated with data redundancy are illustrated in the example on the following slides
Session 12 – Database Issues

StaffDistributionCenters Table

- Note the details of a distribution center are repeated for every employee (not normal form)

<table>
<thead>
<tr>
<th>ID</th>
<th>StaffNo</th>
<th>staffName</th>
<th>position</th>
<th>salary</th>
<th>dCenterNo</th>
<th>dAddress</th>
<th>dTelNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1500</td>
<td>Tom Daniels</td>
<td>Manager</td>
<td>$48,000.00</td>
<td>D0001</td>
<td>8 Jefferson Way, Portland, OR 97201</td>
<td>503-555-3618</td>
</tr>
<tr>
<td>2</td>
<td>S0003</td>
<td>Sally Adams</td>
<td>Assistant</td>
<td>$30,000.00</td>
<td>D0001</td>
<td>8 Jefferson Way, Portland, OR 97201</td>
<td>503-555-3618</td>
</tr>
<tr>
<td>3</td>
<td>S0010</td>
<td>Mary Martinez</td>
<td>Manager</td>
<td>$51,000.00</td>
<td>D0002</td>
<td>City Center Plaza, Seattle, WA 98122</td>
<td>206-5556756</td>
</tr>
<tr>
<td>4</td>
<td>S3250</td>
<td>Robert Chin</td>
<td>Assistant</td>
<td>$33,000.00</td>
<td>D0002</td>
<td>City Center Plaza, Seattle, WA 98122</td>
<td>206-5556756</td>
</tr>
<tr>
<td>5</td>
<td>S2250</td>
<td>Sally Stern</td>
<td>Manager</td>
<td>$48,000.00</td>
<td>D0004</td>
<td>2 W. El Camino, San Francisco, CA 94087</td>
<td>822-555-3131</td>
</tr>
<tr>
<td>6</td>
<td>S0415</td>
<td>Art Peters</td>
<td>Manager</td>
<td>$42,000.00</td>
<td>D0003</td>
<td>14 - 8th Avenue, New York, NY 10012</td>
<td>212-371-3000</td>
</tr>
</tbody>
</table>

Update Anomalies

- Tables that contain redundant information may potentially suffer from update anomalies
- Types of update anomalies include:
  - Insertion – how do you insert details of a new distribution center that has no employees?
  - Deletion – when we delete the last employee in a distribution center, we lose the information about the distribution center
  - Modification – changes to a distribution center must be made for all records containing that distribution center
Better Design

First Normal Form (1NF)

- All rows must have the same number of columns
- Single valued attributes only

No universal agreement as to what would disqualify a table from being in 1NF

Typical violation of 1NF

Resist the temptation to include repeated fields as CSV text
Example - Table not 1NF

Repeated field

Converting to 1NF

Replace a repeating group with a foreign key relationship
Second Normal Form (2NF)

- Violated when a non-key column is a fact about part of the primary key
- A column is not fully functionally dependent on the primary key

<table>
<thead>
<tr>
<th>itemno</th>
<th>customerid</th>
<th>quantity</th>
<th>customer-credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>57</td>
<td>25</td>
<td>OK</td>
</tr>
<tr>
<td>34</td>
<td>679</td>
<td>3</td>
<td>POOR</td>
</tr>
</tbody>
</table>

Mainly applies to tables with multiple natural keys

---

Third Normal Form (3NF)

- Violated when a non-key column is a fact about another non-key column, restated as
- A column is not fully functionally dependent on the primary key

<table>
<thead>
<tr>
<th>stock code</th>
<th>nation</th>
<th>exchange rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>MG</td>
<td>USA</td>
<td>0.67</td>
</tr>
<tr>
<td>IR</td>
<td>AUS</td>
<td>0.46</td>
</tr>
</tbody>
</table>

Exchange rate is a fact about a nation
Example - not 3NF

Values in dAddress and dTelNo can be determined from dCenterNo

Values in staffNo, staffName, position and salary are determined from ID

<table>
<thead>
<tr>
<th>ID</th>
<th>StaffNo</th>
<th>StaffName</th>
<th>dTelNo</th>
<th>salary</th>
<th>dCenterNo</th>
<th>dAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51500</td>
<td>Tom Daniels</td>
<td>20001</td>
<td>$40,000.00</td>
<td>8 Jefferson Way, Portland, OR 97201</td>
<td>503-555-3618</td>
</tr>
<tr>
<td>2</td>
<td>50005</td>
<td>Sally Adams</td>
<td>30001</td>
<td>$30,000.00</td>
<td>8 Jefferson Way, Portland, OR 97201</td>
<td>503-555-3038</td>
</tr>
<tr>
<td>3</td>
<td>90010</td>
<td>Mary Martinez</td>
<td>40002</td>
<td>$53,000.00</td>
<td>City Center Plaza, Seattle, WA 98122</td>
<td>206-555-6756</td>
</tr>
<tr>
<td>4</td>
<td>53750</td>
<td>Robert Chin</td>
<td>50002</td>
<td>$33,000.00</td>
<td>City Center Plaza, Seattle, WA 98122</td>
<td>206-555-6756</td>
</tr>
<tr>
<td>5</td>
<td>52250</td>
<td>Sally Stern</td>
<td>60004</td>
<td>$40,000.00</td>
<td>2 W. El Camino, San Francisco, CA 94087</td>
<td>822-555-3131</td>
</tr>
<tr>
<td>6</td>
<td>50415</td>
<td>Art Peters</td>
<td>70003</td>
<td>$42,000.00</td>
<td>14 - 8th Avenue, New York, NY 10012</td>
<td>212-371-3000</td>
</tr>
</tbody>
</table>

Interface Issues

- The application deals with objects, language specific data types, and higher level concepts
- The DB deals with relational tables and SQL
- Translation is usually performed to allow the two components to work together

Translation required DB code and application libraries
**ODBC**

- Open Database Connectivity (ODBC) provides a standard software API (C) to a DBMS.
- An ODBC implementation contains a library, one or more drivers, and DB support.
  - Library provides a general abstraction to the DB.
  - Drivers contain DB specific detail.
- Adaptive programming techniques (e.g., introspection) are often required.

**JDBC**

- Java Database Connectivity (JDBC) provides a standard software API (Java) to a DBMS.
- A JDBC implementation contains a library, one or more drivers, and DB support.
- J2SE contains JDBC libraries (java.sql, javax.sql) and a bridge to ODBC.
- JDBC
  - JDBC 4.2 is the most recent.
Session 12 – Database Issues

Drivers

- Old approach (still supported)
  - Drivers are loaded dynamically
    ```java
    Class.forName( "com.somejdbcvendor.TheirJdbcDriver" );
    ```
  - When the driver is loaded it registers with DeviceManager

- New approach
  - DataSource
  - ConnectionPoolDataSource

Driver Types

- Type 1 - JDBC-ODBC Bridge (used when no pure-Java driver is available)
- Type 2 - Native API (converts JDBC calls into native DBMS calls, usually using some non-Java code)
- Type 3 - Network Protocol driver (uses a middle tier application server to convert data)
- Type 4 - Native-protocol driver (pure-Java)
**JDBC Connections**

- You get a connection to the DB through the DataSource or DriverManager (remember, your driver is registered with the DriverManager)
  ```java
  Connection conn = DriverManager.getConnection("jdbc:somejdbcvendor:other data needed by jdbc vendor", "myLogin", "myPassword");
  ```
- Connection pooling supported since JDBC 3.0

**JDBC Abstractions**

- Connection (and connection pooling)
- Statement
  - Statement
  - PreparedStatement
  - CallableStatement
- Transaction
- Result set
- Others
## Typical JDBC Operation Sequence

- Open a connection to the DB
- Prepare a statement object
- Execute the statement query (e.g., `executeQuery` or `executeUpdate` methods of Statement object)
- Query returns a ResultSet object
- Use the next method of the ResultSet object to iterate through the results
- Close the ResultSet and Statement objects

Similar approach when making updates to DB

---

### JDBC Connection

- Means for a server to communicate with a DB server
- In many DBMS, a connection is a “heavy” action
- Connection pooling is often used to reduce overhead of connections
Connection Pooling

Technique used to mitigate the overhead in repeatedly opening and closing DB connections

Approach
- Pool of connections is created
- Requests for connections are made to the object holding the pool of connections by the application
- Application returns (releases) the connection to the pool when DB access is completed

Implementation
- JDBC support
- Do it yourself

Transactions

Unit of work within a DBMS

Goals
- Keep a database consistent, even in the event of a system failure
- Synchronize access in the case of multiple simultaneous access to the DB

Typical steps
- Begin the transaction
- Execute a set of data manipulations and/or queries
- If no errors occur then commit the transaction and end it
- If errors occur then roll back the transaction and end it
ResultSet

- Essentially a table resulting from a query
- Supports methods to allow iteration through the table
- Maintains a cursor that points to the current row in the result set
- Supports various navigation methods

System Issues

- Which DBMS?
- Which DBMS server (yours or CS shared)?
- What is your approach to connection pooling?
- What is your approach to object/relational mapping?
- What is your data migration plan?
- What is your DB population plan?
- Can you switch between a test DB and a live DB?