Objectives

- Review: Version Control, JSPs
- Quality Attributes of Web Software
- Introduction to Relational Databases, SQL
- JDBC

Version Control Review

- Why do we need version control?
- What can we do with version control?
  - What doesn't it do?
- What version control software are we using?
- How do you get a working copy of code that is stored in version control?
- How do you publish your changes to the public copy of the code?
  - What should you do before publishing your changes?

JSPs and Organization Review

- What motivated the development of JSPs (in addition to servlets)?
- What is in a JSP file?
- How do JSPs execute?
- What are your goals when organizing your code in a JSP (versus what goes into a servlet)?
- Where can we put JSPs so that users can't directly access them?
  - Why would you want to do that?

Most important points?

DISCUSSION OF "QUALITY ATTRIBUTES"

Discussion of "Quality Attributes"

- More to usability than navigation
  - How easy to do the functionality
- What are some of the differences between traditional applications and web applications?
  - Leads to differences in quality attributes

Comparison of Applications

Attribute	Traditional	Web Applications
Location	On clients	Client, Server (& more)
Languages	Java, C, C++, etc.	Traditional languages and Scripting languages, HTML, Other languages
Technologies		Network, DB
Development Team	Programmers	Programmers, graphics designers, usability engineers, Network, DB
Economics	Time to market	Returning customers; later but better
Releases	Infrequent (~monthly), expensive	Frequent (~days), inexpensive

Quality Attributes

Attribute	Web Applications
Reliability	Must work, or go to another site
Usability	Must be usable, or go to another site
Security	Protect user data, information
Availability	24/7/365
Scalability	Thousands of requests per second, more?
Maintainability	Short maintenance cycle, frequent updates
Time-to-market	Later but better is okay

Discussion

- What are examples of sites that you used to use but you switched because something better came along?

JIRA: SOFTWARE AND PROJECT MANAGEMENT

Jira

http://csjira.wlu.edu

- Collaboration tool
- Issue Tracking Tool
  - Alternative to Bugzilla
  - One of few proprietary tools we'll use

Project Organization

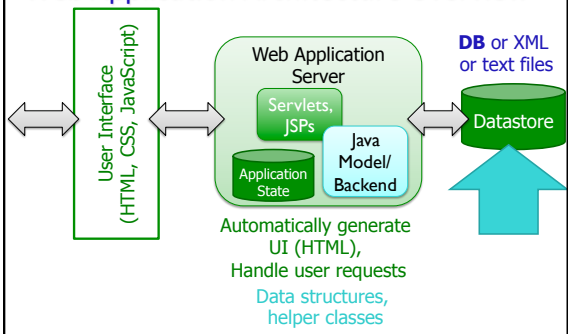
- Code base organization

Looking Ahead

- Now: Project
  - Revise requirements
    - Add todos into JIRA
  - Revise Static Mockups -- close to "real"
    - Guide for next steps
    - Emailed to Clients for approval, discussion
- This afternoon
  - Databases, SQL, JDBC

DATABASES AND SQL

Web Application Architecture Overview



Database Overview

- Store data in such a way to allow *efficient* storage, search, and update
- **Relational Data Model** - currently most popular type of database
  - Different vendors: PostgreSQL, Oracle, MySQL, DB2, MSSQL
  - Data is stored in **tables**
  - ***Attributes***: column names (one word)
  - ***Entities***: rows in table
  - Often contain ***primary key***: a set of columns that uniquely identify a row

Example Students Table

- id is the primary key
- What are the attributes?
- What are the entities?

id	lastName	firstName	gradYear	major
10011	Aaronson	Aaron	2013	CSCI
43123	Brown	Allison	2014	ENGL

Example Students Table

- id is the primary key
- What are the attributes?
- What are the entities?

Attributes

Entities

id	lastName	firstName	gradYear	major
10011	Aaronson	Aaron	2013	CSCI
43123	Brown	Allison	2014	ENGL

Courses Table

- Primary key is (Department, Number)
  - As a group, these uniquely identify a row

department	number	name	description
CSCI	101	Survey of Computer Science	A survey of …
CSCI	111	Fundamentals of Programming I	An introduction to …

SQL: STRUCTURED QUERY LANGUAGE

SQL: Structured Query Language

- Standardized language for manipulating and querying relational databases
  - May be slightly different depending on DB vendor
- Pronounced "S-Q-L" or "Sequel"

SQL: Structured Query Language

- Reserved words are not case-sensitive
  - I will tend to write them in all-caps and bold
  - Tables, column names - may be case sensitive
- Commands end in **;**
  - Can have extra white space, new lines in commands
  - End when see **;**
- Represent string literals with single quotes **' '**

SELECT Command

- Queries the database
- Returns a result—a ***virtual table***
- Syntax:

Optional

```
SELECT column_names
FROM table_names [WHERE condition];
```

  - Columns, tables separated by commas
  - Can select all columns with *
  - Where clause specifies constraints on what to select from the table

SELECT Examples

- `SELECT * FROM Students;`

id	lastName	firstName	gradYear	major
10011	Aaronson	Aaron	2013	CSCI
43123	Brown	Allison	2014	ENGL

- `SELECT lastName, major FROM Students;`

Virtual Tables

lastName	major
Aaronson	CSCI
Brown	ENGL

WHERE Conditions

- Limits which rows you get back
- Comparison operators: `>, >=, <, <=, <>`
- Can contain **AND** for compound conditions
- **LIKE** matches a string against a pattern
  - Wildcard: **%**, matches any sequence of 0 or more characters
- **IN** : match any
- **BETWEEN**: Like comparison using **AND**, inclusive

SELECT Examples

- What do these select statements mean?
  - `SELECT * FROM Students WHERE major='CSCI';`
  - `SELECT firstName, lastName FROM Students WHERE Major='CSCI' AND gradYear=2016;`
  - `SELECT lastName FROM Students WHERE firstName LIKE 'Eli%';`

SELECT Examples

- What do these select statements mean?
  - `SELECT lastName FROM Students WHERE Major IN ('CSCI', 'PHYS', 'MATH');`
  - `SELECT lastName FROM Students WHERE Major NOT IN ('CSCI', 'PHYS', 'MATH');`
  - `SELECT firstName FROM Students WHERE gradYear BETWEEN 2016 AND 2018;`

Set vs Bag Semantics

Set vs Bag Semantics

- Bag
  - Duplicates allowed
  - Number of duplicates is significant
  - Used by SQL by default
- Set
  - No duplicates
  - Use keyword **DISTINCT**

Set vs Bag

```
SELECT lastName
FROM Students;
```

lastName
Smith
…
Smith
Jones
Jones

```
SELECT DISTINCT lastName
FROM Students;
```

lastName
Smith
Jones

Aggregates

- Standard SQL aggregate functions: **COUNT, SUM, AVG, MIN, MAX**
- Can only used in the **SELECT** part of query
- Example
  - `SELECT COUNT(*), AVG(GPA) FROM Students WHERE gradYear=2013`

ORDER BY

- Last operation performed, last in query
- Orders:
  - **ASC** = ascending
  - **DESC** = descending
- Example

```
SELECT firstName, lastName
FROM Students WHERE gradYear=2016
ORDER BY GPA DESC;
```


Majors Table

- Another table to keep track of majors
- Primary Key: id

id	name	department
1	ART-BA	ART
2	ARTH-BA	ART

Changes Students Table

- Use an id to identify major (primary key)

Majors:

id	name	department
1	ART-BA	ART
2	ARTH-BA	ART

Foreign Key

Students:

id	last Name	first Name	gradYear	majorID
10011	Aaronson	Aaron	2013	123
43123	Brown	Allison	2014	157

JOIN Queries

- Join two tables on an attribute

Majors:

id	name	department
1	ART-BA	ART
2	ARTH-BA	ART

Students:

id	last Name	first Name	gradYear	majorID
10011	Aaronson	Aaron	2013	123
43123	Brown	Allison	2014	157

```
SELECT lastName, name
FROM Students, Majors
WHERE Students.majorID=Majors.id;
```


JOIN Queries

- Join two tables on an attribute

```
SELECT lastName, name
FROM Students, Majors
WHERE Students.majorID=Majors.id;
```

From Students From Majors

lastName	name
Aaronson	CSCI
Brown	ENGL

JOIN Queries

- What if two tables have the same column name?
  - Add the table name and a . to the beginning of the column, i.e., **TableName.columnName**

```
SELECT Students.lastName, Majors.name
FROM Students, Majors
WHERE Students.majorID=Majors.id;
```

What if Students Have Multiple Majors?

- We don't necessarily want to add another column to Students table
  - What if student has 3 majors?
- Example of Many to Many Relationship
- Solution: Create **StudentsToMajors** table:

studentID	majorID
435	243
435	232

Primary Key: (StudentID, MajorID)
Foreign Keys from Students, Majors Tables

INSERT Statements

- You can add rows to a table

```
INSERT INTO Majors VALUES
( 354, 'BioInformatics-BS', 'CSCI');
```

Assumes filling in all values, in column order

- Preferred Method: include column names
  - Don't depend on order

```
INSERT INTO Majors (id, name, department)
VALUES ( 354, 'BioInformatics-BS', 'CSCI');
```


INSERT Statements

- Automatically create ids

```
INSERT INTO Majors (id, name, department)
VALUES ( nextval('majors_sequence'),
'Bio-Informatics-BS', 'CSCI' );
```

- If table is set up appropriately, let the DB handle creating unique ids:

```
INSERT INTO Majors (name, department)
VALUES ( 'Bio-Informatics-BS', 'CSCI' );
```


UPDATE Statement

- You can modify rows of a table
- Use **WHERE** condition to specify which rows to update
- Example: Update a student's married name

```
UPDATE Students SET
LastName='Smith-Jones' WHERE id=12;
```

- Example: Update all first years to undeclared

```
UPDATE Students SET majorID=345
WHERE gradYear=2016;
```


DELETE Statement

- You can delete rows from a table

```
DELETE FROM table [ WHERE condition ];
```

- Example

```
DELETE FROM EnrolledStudents WHERE
hasPrerequisites=False AND course_id=456;
```


Using a Database

- DBMS: Database management system
- Using PostgreSQL in this class
  - Free, open source
- Slight differences in syntax between DBMSs
- DBMS can contain multiple databases
  - Need to say which DB you want to use

Designing a DB

- Design tables to hold your data
  - Data's name and types
- Similar to OO design
  - No duplication of data
  - Have pointers to info in other tables
- Main difference: no lists
  - If you think "list", think of a OneToMany or a ManyToMany table that contains the relationships between the data

Standard Data Types

- Standard to SQL
  - CHAR - fixed-length character
  - VARCHAR - variable-length character
    - Requires more processing than CHAR
  - INTEGER - whole numbers
  - NUMERIC
  - Names for types in specific DB may vary
- More data types available in each DB

PostgreSQL Data Types

- Names for standard data types
  - Numeric: `int, smallint, real, double precision`
  - Strings
    - `char(N)` - fixed length (padded)
    - `varchar(N)` - variable length, with a max
    - `text` - variable unlimited length
- Additional useful data types
  - `date, time, timestamp, and interval`
  - `Timestamp` includes both date and time

Constraints

- **`PRIMARY KEY`** may not have null values
- **`UNIQUE`** may have null values
  - Example: username when have a separate id
- **`FOREIGN KEY`**
  - Use key from another ("foreign") table
  - Example: shopping cart has its own id; references the user's id as owner
- **`CHECK`**
  - value in a certain column must satisfy a Boolean (truth-value) expression
  - Example: GPA >= 0

Creating a Table

- Example:

```
CREATE TABLE weather (
    city            varchar(80),
    temp_lo         int,       -- low temperature
    temp_hi         int,       -- high temperature
    prcp            real,      -- precipitation
    date            date
);
```


Project Databases

- What tables should you need?
- What data?
- What constraints?

JDBC

JDBC: Java Database Connectivity

- Database-independent connectivity
  - JDBC converts generalized JDBC calls into vendor-specific SQL calls
- Classes in `java.sql.*` and `javax.sql.*` packages

Using JDBC in a Java Program

1. Load the **database driver**
2. Obtain a **connection**
3. Create and execute **statements** (SQL queries)
4. Use **result sets** (tables) to navigate through the results
5. **Close** the connection

Elaborate in following slides…

java.sql.DriverManager

- Provides a **common access layer** for different database drivers
- Requires that each driver used by the application be registered before use
- Load the database driver by its name using ClassLoader:

```
Class.forName("org.postgresql.Driver");
```


Creating a Connection

- After loading the DB driver, create the **connection** (see API for all ways)

Type of DB — Location of DB, port optional — DB name

```
String url = "jdbc:postgresql://hopper:5432/cs335";
Connection con = DriverManager.getConnection(url,
                        username, password);
```

- Close connection when done
  - Release resources

```
con.close();
```

Where should these code fragments go in a servlet?

Statements

```
Statement stmt = con.createStatement();
```

- `executeQuery(String sql)`
  - Returns a **`ResultSet`**, which is like a virtual table of results
  - Iterate through `ResultSet`, row by row

```
rs = stmt.executeQuery("SELECT * FROM table");
```

- `executeUpdate(String sql)` to update table
  - Returns an integer representing the number of affected rows

Iterating Through `ResultSets`

- Example:

```
ResultSet rs = stmt.executeQuery("SELECT * " +
          "FROM majors");

while( rs.next()) {
   String name= rs.getString("name");
   String dept = rs.getString(2); // column 2
   System.out.println(name + "\t" + dept);
}
```

- Can access column values by *name* or which column (count starts at 1, left to right)

Useful `ResultSet` Methods

- Number of rows in the result:

```
rs.last();
int numberOfRows = rs.getRow();
```

- Information about the table, such as number, types, and properties of columns:
  - `ResultSetMetaData getMetaData()`

Prepared Statements

Preferred approach to make SQL statements

- `con.prepareStatement(String template)`
  - Compile SQL statement "templates"
- Reuse statement, passing in parameters
  - Java handles formatting of Strings, etc. as parameters
  - More secure (more later)

```
updateSales = con.prepareStatement("INSERT"
+ "INTO Sales (quantity, name) VALUES"+
"(?, ?)");                    ? = Parameter
```

- Set parameters
  - `updateSales.setInt(1, 100);`
  - `updateSales.setString(2, "French Roast");`
  - Columns start at 1

JDBC

- API Documentation: `java.sql.*`
  - `Statements`, `Connections`, `ResultSets`, etc. are all **Interfaces**
    - Driver/Library implements interfaces for its database
- Limitations
  - Java doesn't compile the SQL statements
    - Exact syntax depends on DB
    - Compile, run, verify queries outside of Java for your database
    - Then copy and use in Java code

Using PostgreSQL on Command-Line

- In a terminal, **ssh** into **hopper**
  - ssh hopper
- Run the PostgreSQL client: **psql** , connecting to the appropriate database
  - psql cs335
- At the prompt, type in SQL statements, ending in ;

Examples Using JDBC

Transactions in JDBC

- By default, a connection is in **auto-commit** mode
  - Each statement is a transaction
  - Automatically committed as soon as executed

Transactions in JDBC

- You can turn off auto-commit and execute multiple statements as a transaction
  - Database can keep handling others' reads
  - Others won't see updates until you commit

```
con.setAutoCommit(false);
// execute SQL statements …
con.commit(); // commit those statements
con.setAutoCommit(true);
```

- Can call `rollback` to abort updates

Storing Passwords

- Use `md5` function on passwords
  - `md5('password')`
- Compare user's input password md5'd with password in database
  - `SELECT COUNT(id) FROM Users WHERE username=? AND password=md5(?);`
- Example: username and password = 'test'

Connection Pool

- Want to reuse DB connections
  - Reduce overhead of creating and closing connections to database
- Could write our own connection pool class
  - Many examples online
- Apache wrote the one that we'll use
  - `http://commons.apache.org/dbcp/`

Using the Connection Pool

- Create a `DBManager` that contains a `DataSource` object in the `ServletContext`
  - All the servlets can see the `ServletContext`
  - Shared resource, given name, value
- When implementing a servlet that requires a DB connection
  - `init` method gets the `DBManager` object from the `ServletContext`
  - When need a connection, call `getConnection` on `DBManager` object

Servlets and JDBC

- In general, we want to minimize the use of JDBC in the servlets
- Same queries in multiple servlets
  - Don't want to duplicate code
  - If DB tables or queries change, only change in one place
- Instead, we want to have Java classes (model) that communicate with the DB
  - Convert `ResultSets` to objects that servlets/JSPs can use
- Suggestion: add methods to `DBManager` that execute queries and return Java objects, as appropriate

TODO

- Lab 6 – by tonight midnight
  - Must be done on Linux machines
  - Restrictions on DB access
- Revisions of requirements, static mockup, etc.
  - Tonight at midnight