Objectives

• Review: Testing
• Introduction to Relational Databases, SQL

Testing Review

• Describe the general testing process
  ➢ What validates/verifies the output from a program?
• How does the process change for Web applications?
• What are some of the testing frameworks?
  ➢ How do they work?
• What is the tool that we will use for testing?
  ➢ What is its advantages?
• What is the difference between testing and debugging?

Web Application Testing Overview

Test case

Web Application

HTML Responses, Email, DB, Files

Actual Output

Expected Output

Comparator

pass or fail

Oracle

Web Stuff

• Spike in usage: WLU’s new Google web search crawler
• Warning in Google Search results

Project Overview

HTML

Automatically generate UI (HTML), Handle user requests

Backends, JSPs, ...

Servlets, JSPs, ...

Backend (Java)

Web Application

State

Data structures, helper classes

DB or XML or text files

Databases

• 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
  ➢ 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?

<table>
<thead>
<tr>
<th>id</th>
<th>lastName</th>
<th>firstName</th>
<th>gradYear</th>
<th>major</th>
</tr>
</thead>
<tbody>
<tr>
<td>10011</td>
<td>Aaronson</td>
<td>Aaron</td>
<td>2008</td>
<td>CSCI</td>
</tr>
<tr>
<td>43123</td>
<td>Brown</td>
<td>Allison</td>
<td>2009</td>
<td>ENGL</td>
</tr>
</tbody>
</table>

Courses Table
• Primary key is (Department, Number)
  ➢ As a group, these uniquely identify a row

<table>
<thead>
<tr>
<th>department</th>
<th>number</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSCI</td>
<td>101</td>
<td>Survey of Computer Science</td>
<td>A survey of …</td>
</tr>
<tr>
<td>CSCI</td>
<td>111</td>
<td>Fundamentals of Programming I</td>
<td>An introduction to …</td>
</tr>
</tbody>
</table>

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”

SELECT Command
• Queries the database
• Returns a result -- a virtual table
• Syntax:
  ```sql
  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

Example Students Table
• ID is the primary key
• What are the attributes?
• What are the entities?

<table>
<thead>
<tr>
<th>id</th>
<th>lastName</th>
<th>firstName</th>
<th>gradYear</th>
<th>major</th>
</tr>
</thead>
<tbody>
<tr>
<td>10011</td>
<td>Aaron</td>
<td>Aaronson</td>
<td>2008</td>
<td>CSCI</td>
</tr>
<tr>
<td>43123</td>
<td>Brown</td>
<td>Allison</td>
<td>2009</td>
<td>ENGL</td>
</tr>
</tbody>
</table>
**SELECT Examples**

- **SELECT * FROM Students;**
  
<table>
<thead>
<tr>
<th>id</th>
<th>lastName</th>
<th>firstName</th>
<th>gradYear</th>
<th>major</th>
</tr>
</thead>
<tbody>
<tr>
<td>10011</td>
<td>Aaronson</td>
<td>Aaron</td>
<td>2008</td>
<td>CSCI</td>
</tr>
<tr>
<td>43123</td>
<td>Brown</td>
<td>Allison</td>
<td>2009</td>
<td>ENGL</td>
</tr>
</tbody>
</table>

- **SELECT lastName, major FROM Students;**
  
<table>
<thead>
<tr>
<th>lastName</th>
<th>major</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aaronson</td>
<td>CSCI</td>
</tr>
<tr>
<td>Brown</td>
<td>ENGL</td>
</tr>
</tbody>
</table>

**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=2008;**
  - **SELECT lastName FROM Students WHERE firstName LIKE 'Sara%';**

**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;**
  
<table>
<thead>
<tr>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Jones</td>
</tr>
<tr>
<td>Jones</td>
</tr>
</tbody>
</table>

- **SELECT DISTINCT lastName FROM Students;**
  
<table>
<thead>
<tr>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
</tr>
<tr>
<td>Jones</td>
</tr>
</tbody>
</table>
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=2008

ORDER BY
• Last thing done, last in query
• Orders:
  ➢ ASC = ascending
  ➢ DESC = descending
• Example
  SELECT firstName, lastName FROM Students WHERE gradYear=2008 ORDER BY GPA DESC;

Majors Table
• Another table to keep track of majors
• Primary Key: id

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>department</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ART-BA</td>
<td>ART</td>
</tr>
<tr>
<td>2</td>
<td>ARTH-BA</td>
<td>ART</td>
</tr>
</tbody>
</table>

Changes Students Table
• Use an id to identify major (primary key)

<table>
<thead>
<tr>
<th>id</th>
<th>lastName</th>
<th>firstName</th>
<th>gradYear</th>
<th>majorID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10011</td>
<td>Aaronson</td>
<td>Aaron</td>
<td>2008</td>
<td>123</td>
</tr>
<tr>
<td>43123</td>
<td>Brown</td>
<td>Allison</td>
<td>2009</td>
<td>157</td>
</tr>
</tbody>
</table>

JOIN Queries
• Join two tables on an attribute

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

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.major=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 **Students2Majors** table:

<table>
<thead>
<tr>
<th>studentID</th>
<th>majorID</th>
</tr>
</thead>
<tbody>
<tr>
<td>435</td>
<td>243</td>
</tr>
<tr>
<td>435</td>
<td>232</td>
</tr>
</tbody>
</table>

**Primary Key:** (StudentID, MajorID)

**Foreign Keys from Students, Majors Tables**

---

**INSERT Statements**

- You can add rows to a table
  - Example of Many to Many Relationship
  - Solution: Create **Students2Majors** table:

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

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

```sql
INSERT INTO Majors (id, name, department) VALUES (354, 'BioInformatics-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

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

- Example: Update all first years to undeclared

```sql
UPDATE Students SET major=345 WHERE gradYear=2011;
```

---

**DELETE Statement**

- You can delete rows from a table

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

---

**Creating a Database**

- 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 Many2Many 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
- FOREIGN KEY
  - Use key from another (“foreign”) table
- CHECK
  - value in a certain column must satisfy a Boolean (truth-value) expression

Creating a Table

- Example:

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

Project Database

- What tables should we have?
  - What data?
  - What constraints?

Midterm Info

- Midterm - next Wednesday
- Covers
  - HTML, CSS
  - WWW in general - distributed ideas
  - Usability
  - Servlets, JSPs, Application organization
  - Synchronization, Version Control
  - JDBC
TODO

- Static HTML Mockup
  - Demo on Monday at 2.45 p.m.
- Study for midterm