Objectives

- Review: JavaScript
- Quality Attributes of Web Software
- Introduction to Relational Databases, SQL
- JDBC

JavaScript review

- True or False: JavaScript is just like Java
- What is the role of JavaScript in web applications?
- How do you declare a variable? (2 ways)
- How do you write text to the web page?
- What is the syntax for functions?
- What are some examples of events?
- How do you access a particular element in a document?
  - What are some ways to change that element?
- True or False: Validating a form on the client side is just as good as validating it on the server side

Form Validation

Client-side

Pros

- Catches errors earlier
- Reduces network traffic
  - no need to go to the server if

Cons

- User can bypass client-side validation (e.g., turn off JavaScript or not use the browser)

Server-Side

- The buck stops here

Why You Need Server-Side Validation

My partner -- the server-side web developer -- was using a web application, as a client/user. The application wouldn't let him do something because he hasn't paid the bill for this month. (It's June 1, and the automatic withdrawal happens later in the month.) The buttons were disabled. He edited the HTML, removing the disabled attribute from the button, and completed the transaction.

Quality Attributes

- What are ***quality attributes***?
- How are web applications different from "traditional"/desktop applications?
- React to "For most application types, commercial developers have traditionally had little motivation to produce high-quality software."
- What are differences between 2002 (when article was originally published) and now?
  - N.B.: still a highly cited paper
- Let's add another point in the comparison: video games, mobile apps
  - Compare mobile apps with web and desktop

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, Cloud
Development Team	Programmers, graphics designers, usability engineers	Programmers, graphics designers, usability engineers, Network, DB, Server/Cloud experts
Economics	Time to market	Returning customers; later but better
Releases	Infrequent (~monthly), expensive	Frequent (~days), inexpensive

Quality Attributes

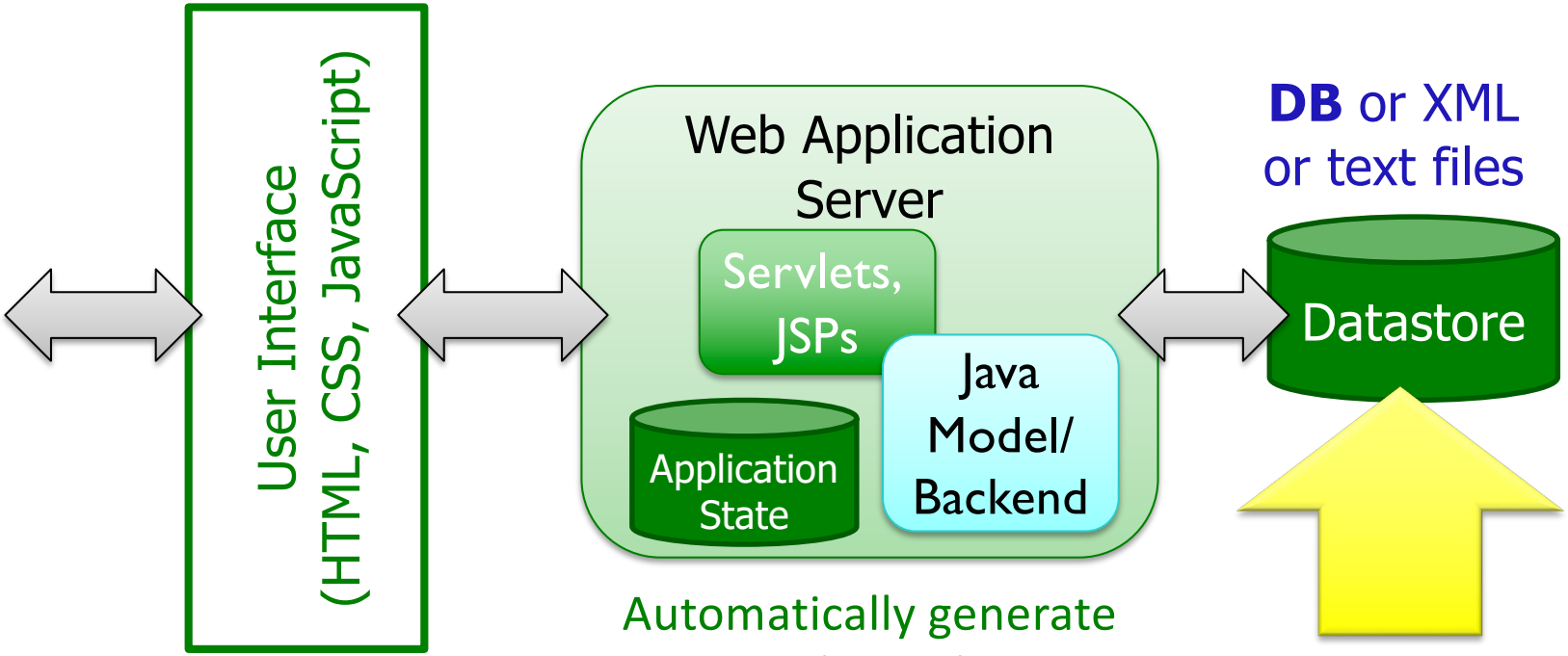
Attribute	Web Applications
Reliability	Must work, or go to another site
Usability	Must be usable, or go to another site
Security	Protect user data, information
Availability	24/7/365
Scalability	Thousands of requests per second, more?
Maintainability	Short maintenance cycle, frequent updates
Time-to-market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

Discussion

- What are examples of sites that you used to use but you switched because something better came along?
  - How easy is it to switch now?

DATABASES AND SQL

Web Application Architecture Overview



Automatically generate UI (HTML), Handle user requests

Data structures, helper classes

Database Overview

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

DBMS Popularity

Rank May 2024	Rank Apr 2024	Rank May 2023	DBMS	Database Model	Score May 2024	Score Apr 2024	Score May 2023
1.	1.	1.	Oracle	Relational, Multi-model	1236.29	+2.02	+3.66
2.	2.	2.	MySQL	Relational, Multi-model	1083.74	-3.99	-88.72
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	824.29	-5.50	-95.80
4.	4.	4.	PostgreSQL	Relational, Multi-model	645.54	+0.49	+27.64
5.	5.	5.	MongoDB	Document, Multi-model	421.65	-2.31	-14.96
6.	6.	6.	Redis	Key-value, Multi-model	157.80	+1.36	-10.33
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	135.35	+0.57	-6.28
8.	8.	↓ 7.	IBM Db2	Relational, Multi-model	128.46	+0.97	-14.56
9.	9.	↑ 11.	Snowflake	Relational	121.33	-1.87	+9.61
10.	10.	↓ 9.	SQLite	Relational	114.32	-1.69	-19.54

Ranking based on web site mentions, searches, questions, job offers, professional profiles, social network mentions

https://db-engines.com/en/ranking

- Free, open source
- Evolved from UC Berkeley Database Ingres
- Has more advanced features than MySQL

- The DBMS that we'll use!

Terminology: Database vs Database Management System

- When I say “database”, I could be referring to either the running DBMS (which can hold more than one database) or a specific database

Example Students Table

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

id	lastName	firstName	gradYear	major
10011	Aaronson	Aaron	2025	CSCI
43123	Brown	Allison	2024	ENGL

Example Students Table

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

id	lastName	firstName	gradYear	major
10011	Aaronson	Aaron	2025	CSCI
43123	Brown	Allison	2024	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 tend to write them in all-caps and bold to distinguish them in the slides
  - 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:


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

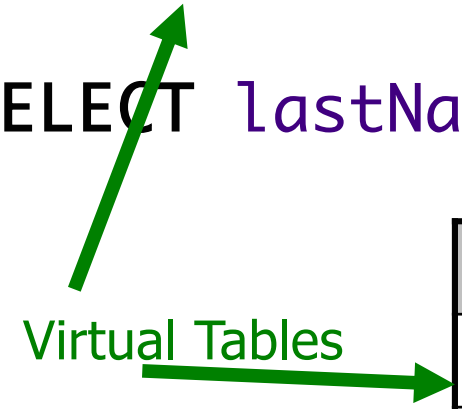
  - Columns, tables separated by commas
  - Can select all columns with *
  - Where clause specifies constraints on what to select from the table

SELECT Examples

- `SELECT * FROM Students;`

id	lastName	firstName	gradYear	major
10011	Aaronson	Aaron	2025	CSCI
43123	Brown	Allison	2024	ENGL

- `SELECT lastName, major FROM Students;`



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? (What data question would they answer?)
  - `SELECT * FROM students WHERE major='CSCI';`
  - ```
    SELECT firstName, lastName
    FROM students WHERE major='CSCI'
    AND gradYear=2024;
    ```
  - ```
    SELECT firstName, lastName
    FROM students WHERE major='CSCI'
    AND gradYear<>2024;
    ```
  - ```
    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 2024 AND 2026;`

Set vs Bag Semantics

- Data structures review

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 use in the **SELECT** part of query

- Example
  - ```
    SELECT COUNT(*), AVG(GPA)
    FROM students WHERE gradYear=2024;
    ```
    ```

ORDER BY

- Last operation performed, last in query
- Orders:
  - **ASC** = ascending
  - **DESC** = descending
- Example
  - ```
    SELECT firstName, lastName
    FROM Students WHERE gradYear=2024
    ORDER BY GPA DESC;
    ```
    ```

Majors Table

- Let's introduce a new table to keep track of majors
- Primary Key: id

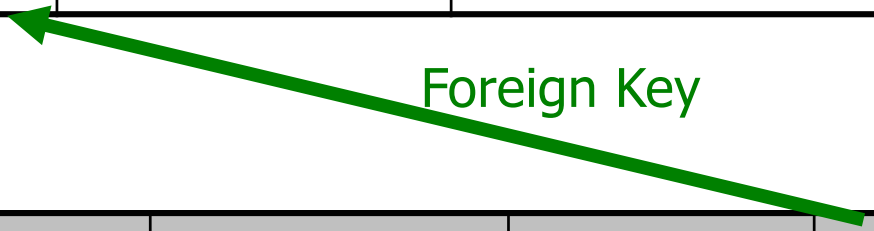
id	name	department
1	ART-BA	ART
2	ARTH-BA	ART

Changes `Students` Table

- Use an id to identify major (primary key)

`Majors:`

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



`Students:`

id	last Name	first Name	gradYear	majorID
10011	Aaronson	Aaron	2021	123
43123	Brown	Allison	2022	157

Join Queries

Does a cross product of the joined tables

- Example:
  - Performing a select on 3 tables, each with two rows
  - `SELECT * FROM A, B, C;`

A1	B1	C1
A2	B2	C2
A	**B**	**C**

- Results in this virtual table:

A1	B1	C1
A1	B1	C2
A1	B2	C1
A1	B2	C2
A2	B1	C1
A2	B1	C2
A2	B2	C1
…	…	…

JOIN Queries

- Join two tables on an attribute

Majors:

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

Students:

id	lastName	firstName	gradYear	majorID
10011	Aaronson	Aaron	2025	123
43123	Brown	Allison	2024	157
…	…	…	…	…

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

Join Queries: Breaking it down

Does a cross product of the joined tables

```
SELECT lastName, name
FROM Students, Majors;
```

Id	Name	Dept	Id	Lname	Fname	…
M1			S1			
M1			S2			
M1			…			
M1			Sn			
M2			S1			
M2			S2			
M2			…			
M2			Sn			
…			…			

JOIN Queries: Breaking it down

2) Keep only the rows that satisfy the **WHERE** clause

3) Keep only the requested columns

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

From **Students** → | lastName | name | ← From **Majors**

lastName	name
Aaronson	CSCI
Brown	ENGL

JOIN Queries

- What if two joined 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? Handling 2 minors?
- Solution: Create **`StudentsToMajors`** table:

studentID	majorID
435	243
435	232

Primary Key: `(studentID, majorID)`

Foreign Keys
from `Students`, `Majors` Tables

- Example of ***Many to Many Relationship***

JOIN Queries

- To find the students' majors with this new StudentsToMajors table, we would query

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

- Creates cross product of all 3 tables, then keep only the rows that satisfy the WHERE clause, and only include the specified columns

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=2027;
```

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 specify 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 of N (padded)
    - `varchar(N)` - variable length, with a max of N
    - `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
);
```

Storing Passwords (toy example)

- Passwords should not be stored in plaintext
- Use the hashing function **`md5`** to store/compare passwords
  - `md5('password')`
- Compare user's input password md5'd with password in database
  - `SELECT COUNT(id) FROM Users WHERE username='test' AND password=md5('password');`
  - What are the possible outputs from this query?

There are stronger ways to encrypt passwords, but for this practice exercise, this is fine.

Using PostgreSQL on Command-Line

- **ssh** into **bartik**
- Run the PostgreSQL client: **`psql`** , connecting to the appropriate database
- At the prompt, type in SQL statements, ending with ;
- Use \q to quit out of the PostgreSQL client
- Use space bar to page through results (rows)and q to stop paging

PostgreSQL Practice

- Display all the tables: \dt
- Display the schema for the students table: \d students
- View all information about all the students
- View just the last names of the students
- View just the last names of the students who are seniors
- View all the information about the majors
- Do a join on the students and majors tables (retrieving all the columns)
  1. Now, add studentstomajors to the join
  2. Add a WHERE clause that requires that the student's id needs to match the studentstomajor's student id
  3. And, finally, add a WHERE clause that requires that the major's id needs to match the studenttomajor's major id

JDBC

JDBC: **J**ava **D**ata**b**ase **C**onnectivity

- 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...

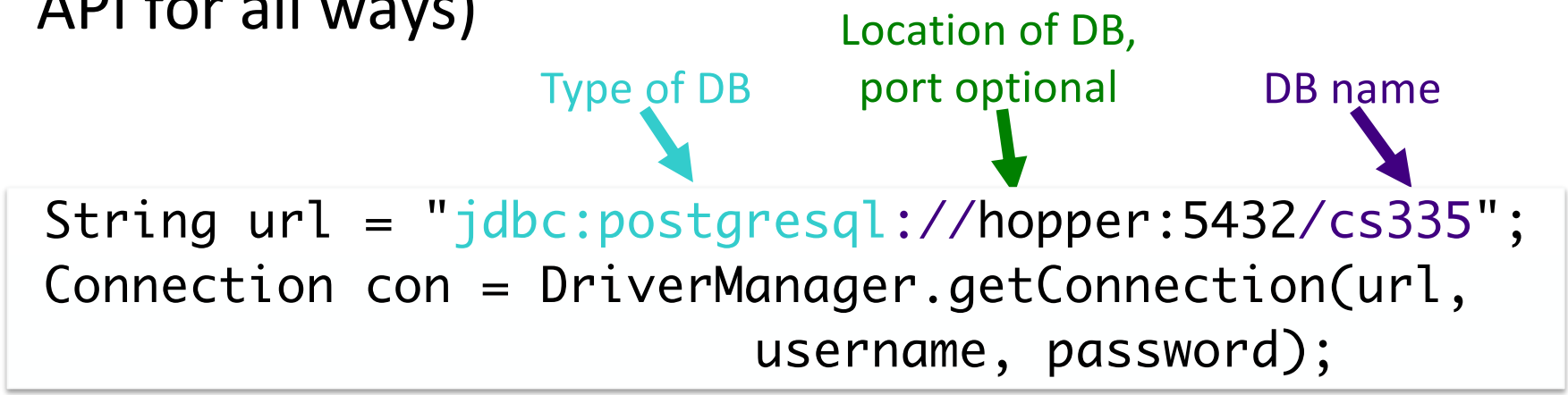
java.sql.DriverManager

- 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)



- 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

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

  - Then, iterate through ResultSet, row by row
- 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

- `rs.next()` – moves cursor one row forward
  - **Returns** true if the new current row is valid; false if there are no more rows
- To get the number of rows in the result:

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

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

Prepared Statements

Preferred approach to make SQL statements

- `con.prepareStatement(String template)`
  - Compile SQL statement "templates"
- Allows statements to be reused with ***parameters***
  - Java handles formatting of Strings, etc. as parameters
  - More secure (more later)
- Example:

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

Template statement ? = Parameter

Must set the parameters before executing

Prepared Statements

Preferred approach to make SQL statements

- `con.prepareStatement(String template)`
  - Compile SQL statement "templates"
- Allows statements to be reused with ***parameters***

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

? = Parameter

- Set parameters (starting at 1). Example:
  - `updateSales.setInt(1, 100);`
  - `updateSales.setString(2, "French Roast");`
    - Java handles formatting the String appropriately for the query
- Then, execute query, similar to (regular) `Statements`

Typical Process for Using PreparedStatements

- Create a connection
- Create prepared statement
- Set the parameters
- Execute the query/update
  - If it's an update, confirm that returned number of updates is what you expected
  - If it's a query, process the returned ResultSet

Note: you won't always have to do all of the above process.
Example: Multiple prepared statements may be created from one connection.

JDBC

- API Documentation: `java.sql.*`
  - `Statements`, `Connections`, `ResultSets`, etc. are all **Interfaces**
    - Driver/Library *implements* interfaces for its DBMS
- Limitations
  - Java doesn't compile the SQL statements
    - Exact syntax depends on DB

Best Practice

1. Test/run/verify concrete queries using DBMS command line

2. Turn concrete queries into template queries for prepared statements in JDBC

Example 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

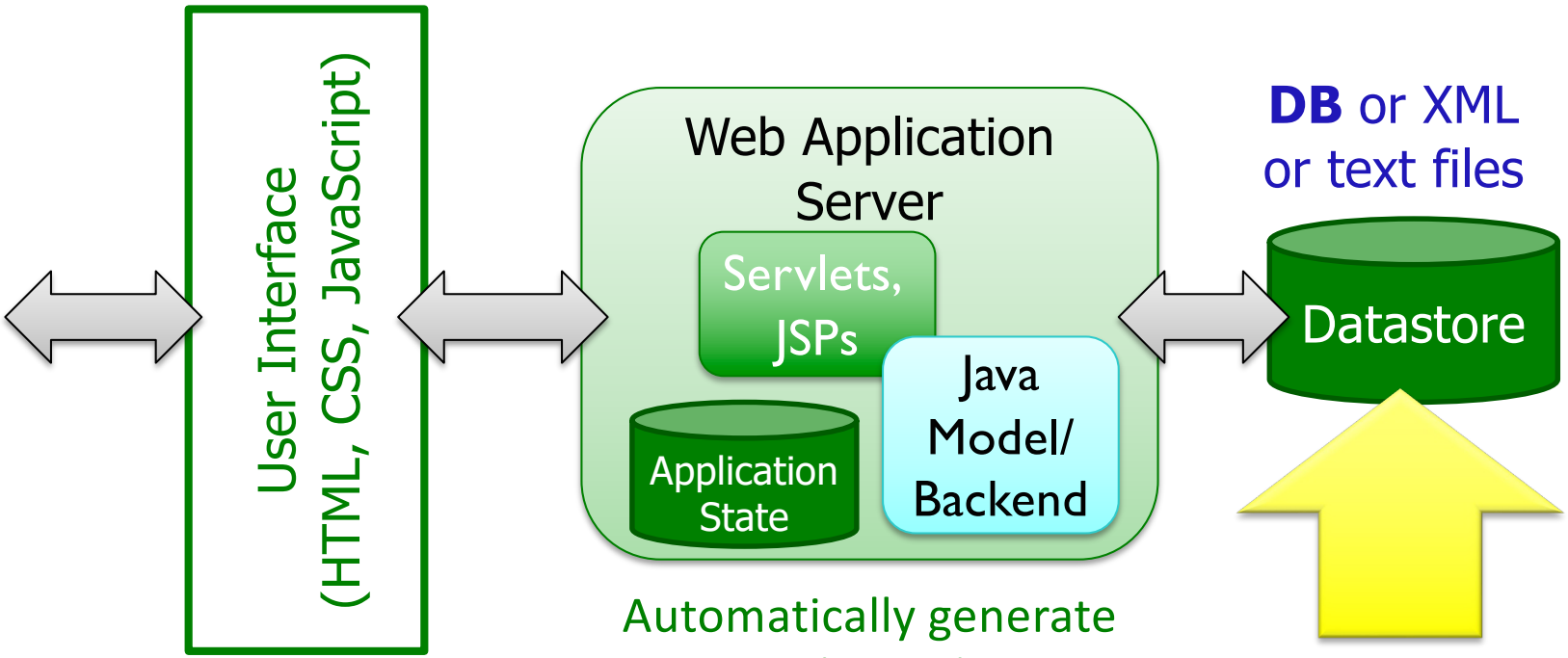
```
con.setAutoCommit(false);
// execute SQL statements …
con.commit(); // commit those statements
con.setAutoCommit(true);
```

- Can call `rollback` to abort updates

Servlets and JDBC

- In general, we want to minimize the use of JDBC in the servlets
  - Separation of concerns
- DB-related concerns
  - Same queries in multiple servlets
    - Don't want to duplicate code
    - If DB tables or queries change, only change in one place
  - Managing of limited number of connections of database
- Instead, have Java classes (model) that communicate with the DB
  - Convert `ResultSets` to objects that servlets/JSPs can use
- We'll use frameworks that help with this

Web Application Architecture Overview



Automatically generate UI (HTML), Handle user requests

Data structures, helper classes

"The Hack"

> Second Washington University hacked data base! Washington and Lee University full unedited database!
> gist.github.com/anonymous/4971…
> <https://t.co/3fqGJwXC>#SweetInfoOp
> <http://twitter.com/search?q=%23SweetInfoOp>

- Notified by W&L News Director
- President's Day
- Actual link: https://gist.github.com/anonymous/4971936
  - Target : http://www.cs.wlu.edu/ (not the DB server)
  - Only had some of the data, not all in actual database
- Figured out they just found my posted SQL file on the assignment page
  - Purposedly public; no security breach

TODO

- Lab 8 – by tonight at 11:59 p.m.