

# Databases, Web Applications (CGI and Servlets) Workshop

Software Engineering  
CS577B 2001

# Databases: what and how

- A database is just a persistent (ie, on disk) repository for your data. Types include:
  - “flat” or plain text files.
  - Indexed files (eg dbm, gdbm).
  - Serializable objects (objects that can write themselves to a file automatically).
  - ODBMS (Object Database Management System).
  - RDBMS (Relational Database Management System).
  - Weird stuff like Microsoft Access
- Usually, when someone says ‘database’, they mean an RDBMS.

# RDBMS terminology

- SQL (Standard Query Language): the lingua franca of databases.
- Schema: SQL code that sets up the tables for a database.
- Query: a request for data, usually written in SQL.
- Record: a fixed set of data fields.
- Table: a collection of records (all of the same type).
- Row: a record from a table.

# Features of an RDBMS

- Relational data model: all data is stored in tables, where a table is a list of records, where each record in a table has a fixed set of fields (type, number, order and size are fixed).
  - Supports efficient computation of queries.
- Provides various tools:
  - A backend server handles requests across the network, processing queries and maintaining an efficient representation of the data on disk.
  - Client programs that interface with the server.
  - Client APIs for various languages, so that programs can access the database. Also, support for product-independent APIs, including ODBC and JDBC.
- Usually supports some amount of standard SQL.

# RDBMS in the real world

- Commercial databases:
  - Oracle
  - Sybase
  - IBM DB2
  - MS SQL Server
- Free databases:
  - PostgreSQL (great for the price)
  - MySQL (fast but lacking some features)
- General caveat: although all of these support some subset of standard SQL, they include proprietary, non-portable extensions. Unfortunately, these extensions are often quite useful, or necessary. Thus database code isn't usually too portable.

# Benefits of an RDBMS

- The benefits of using a real database are nearly infinite (for some tasks):
  - File storage is performed automatically.
  - SQL provides a very easy way to specify the selection and arrangement of data (compared to parsing files yourself, at least).
  - High efficiency: RDBMS are highly optimized...they'll be faster than anything you might write yourself, except for very small data.
  - Scalability to very large data sets is automatic.
- Trade-off considerations
  - Complexity of application (multi-tier, heterogeneous)
  - Cost (initial, license fees, maintenance, etc.)
  - Scope, “more power than needed”, designing to DB
  - Other usual COTS considerations

# SQL: 2 slide primer

- A great SQL book: “A Visual Introduction to SQL”, by Trimble and Chappell.
- Most common SQL commands:
  - CREATE: create a new table.
  - INSERT: insert a row in a table.
  - UPDATE: modify an existing table row.
  - SELECT: query some data from some tables.

(SQL commands are traditionally written in all CAPS, but it's not necessary.)

# SQL example, making the DB

```
mydb=> create table people (name varchar(20), food  
    varchar(40));
```

```
CREATE
```

```
mydb=> insert into people (name, food) values  
    ('fred', 'fish');
```

```
INSERT 22570 1
```

```
mydb=> insert into people (name, food) values  
    ('annabelle', 'cheerios');
```

```
INSERT 22571 1
```

```
mydb=> insert into people (name, food) values ('joe',  
    'pizza');
```

```
INSERT 22572 1
```

```
mydb=> update people set food = 'fish' where name =  
    'joe';
```

```
UPDATE 1
```

# SQL example, querying the DB

```
mydb=> select * from people;
```

```
name      |food
-----+-----
fred      |fish
annabelle|cheerios
joe       |fish
(3 rows)
```

```
mydb=> select name from people where food = 'fish';
```

```
name
----
fred
joe
(2 rows)
```

# What is a web application?

- A web app operates in your browser, interacting with a server via HTTP; the interface is a series of “pages”, encoded in HTML (unlike a “normal” application, like Netscape).
- A web app has state (unlike static web pages). (ie, the application remembers things about the user from one page to another).
  - Web pages are stateless

# Examples of web applications

- Web apps are:
  - shopping carts (amazon.com)
  - text-based clients (hotmail.com)
  - portals (home.netscape.com)
- Web apps are not:
  - PowerPoint, Photoshop, Netscape, Eudora.
  - Java applets (depending on your point of view, and the applet in question).
  - Simple static web pages

# Web application considerations

- A web app can't:
  - Directly manipulate the local system (files or otherwise).
  - Do complex, interactive graphical interfaces (limited to the simpler parts of HTML, usually, maybe with some javascript or JSP goodies).
  - Do things quickly (dependent on network).
- Web app advantages:
  - Ultimate portability (runs in browser).
  - Run in public terminals (users need not own a computer).
  - Centralized processing and storage.

# One way to do a web app: CGI

- CGI (Common Gateway Interface) provides a standard mapping from an HTTP request to program.
  - Programs may be written in any language...thus all the power of conventional programming and existing libraries can be brought to bear on the web.
  - Programs may be passed parameters through environment variables (QUERY\_STRING) or on the standard input (HTTP POST or PUT).
- CGI is widely supported; most web hosting companies are set up to run CGI scripts for you.
- UCS does not like custom CGI scripts and usually will not let you deploy them.

# An example CGI script

```
#!/usr/bin/perl

print "Content-type: text/html\n\n";
print "<body bgcolor=white>\n<p>Hi!\n";
print "<p>Time is: ";
print `date`;
```

# How CGI works



(When the web server receives a request for a CGI script, it invokes the script with parameters from the request, redirecting the script's standard output back to the client.)

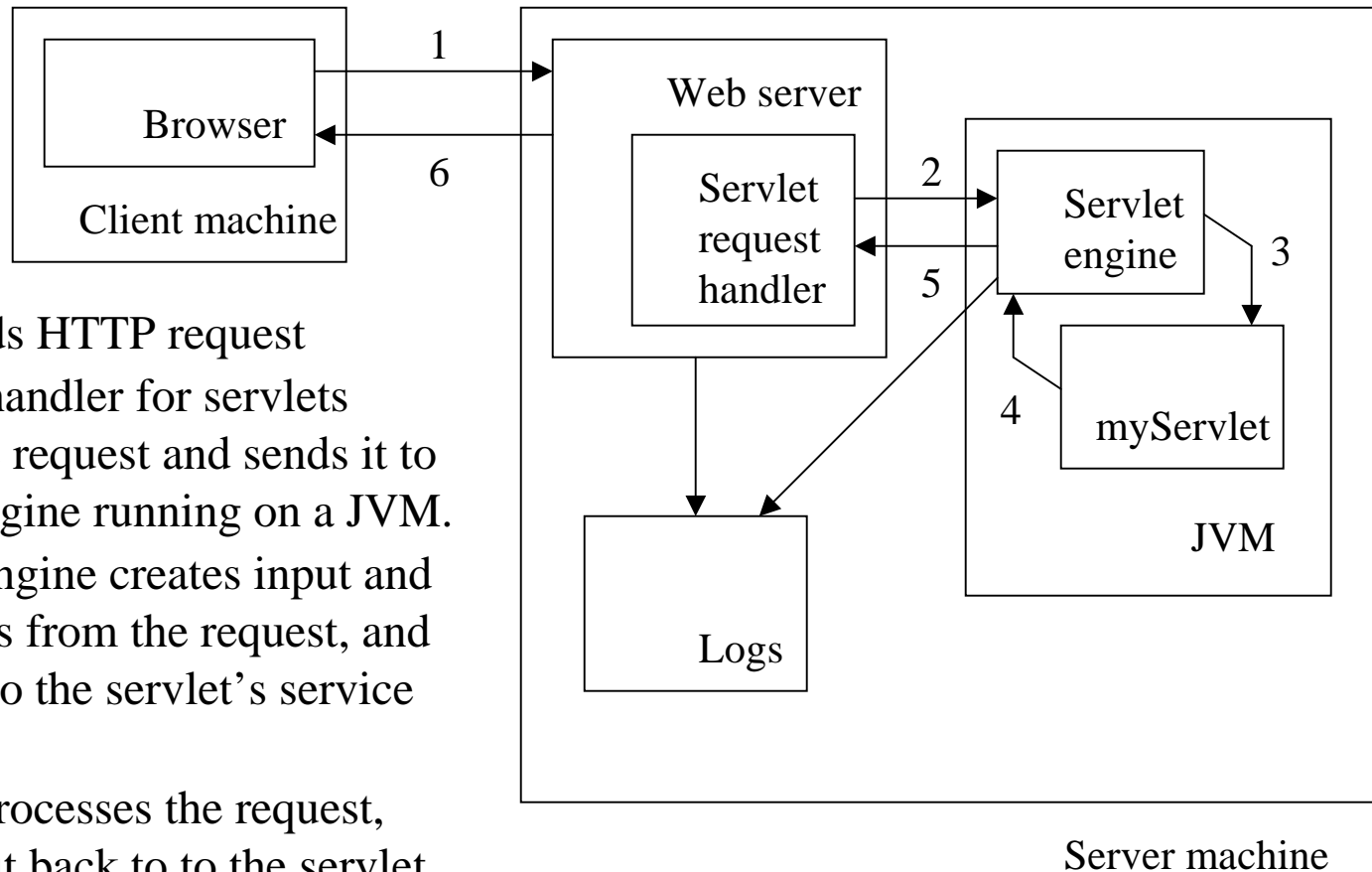
# Pros and Cons of CGI

- CGI good:
  - Maps http onto an arbitrary program.
  - Simple and rugged.
  - Widely available.
  - Supported by libraries in several languages, including Perl.
  - Easy.
- CGI bad:
  - Insecure, if not done carefully.
  - Per-request program invocations are slow (particularly interpreted scripts, eg Perl).

# Servlets

- Servlets are the Java alternative to CGI.
- All web servlets extend the `javax.servlet.http.HttpServlet` class.
- The webserver maps requests onto an instance object for the requested servlet (as in CGI). Multithreading is built into servlets, so one servlet object can handle multiple simultaneous connections.
- An `HttpServlet` contains methods for answering HTTP requests: `doGet` and `doPost` are the main ones. Each of these methods is passed two objects implementing the `HttpServletRequest` and `HttpServletResponse` interfaces; these objects contain the request parameters and open streams for reading (PUT or POST data) and writing (the output HTML or other data).

# Servlet Architecture



1. Browser sends HTTP request
2. The request handler for servlets intercepts the request and sends it to the servlet engine running on a JVM.
3. The servlet engine creates input and output objects from the request, and passes them to the servlet's service method.
4. The servlet processes the request, writing output back to to the servlet engine.
5. The servlet engine passes the output back to the web server.
6. The web server returns the ouput to the client browser.

(Based on the Apache Jserv architecture.  
See [java.apache.org](http://java.apache.org).)

# Servlet example

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException
    {
        // set header field first
        res.setContentType("text/html");

        // then get the writer and write the response data
        PrintWriter out = res.getWriter();
        out.println("<body bgcolor=white>\n<h1>Hi!\n");
        out.close();
    }
}
```

# Servlet resources

- The servlet API specifications and the JSDK (Java Servlet Development Kit) are available from Sun:  
<http://java.sun.com/products/servlet/index.html>
- Servlet-aware web servers include:
  - Various Netscape servers
  - MS IIS
  - Apache (free!, see [java.apache.org](http://java.apache.org), runs on Unix or NT)
  - The JSDK contains “servletrunner”, a program that will run servlets well enough for testing.
- A decent programming guide is part of the Java Tutorial:  
<http://java.sun.com/docs/books/tutorial/servlets/index.html>

# Pros and Cons of Servlets

- Servlets good:
  - In Java, so generally better written, more scalable and more reusable (vs CGI in Perl).
  - In Java, so may harness all the myriad Java APIs (JDBC, beans, enterprise beans, serialization, etc.).
  - Multithreaded architecture is very quick (compared to CGI process startup).
- Servlets bad:
  - In Java, so slower to develop, compared to CGI in a scripting language (lower level string API, requires compilation) (this is alleviated somewhat by use of a visual IDE).
  - Still new: few hosting services are set up to handle servlets, at least not as cheaply as they can CGI. Also, the API specification continues to be revised.

# Java Server Pages (JSP)

- Puts Java (not Javascript) inside HTML pages via special “<%” and “%>” tags
  - Uses “.jsp” extension, but basically an HTML file
- Requires an application server such as Tomcat, JRun
  - Slow on first run as it compiles Java code
  - Java is run on application server machine
- Makes heavy use of “beans” and EJB framework
  - Can manage sessions and lots of other good stuff
- Good reference materials at:  
<http://java.sun.com/products/jsp/>
- Quick reference card at:  
<http://java.sun.com/products/jsp/pdf/card11.pdf>

# Simple JSP example

```
<%@ page import="java.util.*" %>
<HTML>
<BODY>
<%
// This scriptlet declares and initializes
   "date"
System.out.println( "Evaluating date now" );
java.util.Date date = new java.util.Date();
%>
Hello!  The time is now <%  out.println( date );
out.println( "<BR>Your machine's address is " );
out.println( request.getRemoteHost()); %>
</BODY>
</HTML>
```

# JSP or no JSP

- JSP simpler than Servlets
  - JSP eventually become servlets which may be more appropriate in some cases
  - Has similar pros and cons
- JSP is difficult to set up and debug
  - Cryptic error screens when something's wrong
    - Error output from Java (exceptions etc.) go to server log
    - Often have to output debug data to screen, slow and painful
  - Confusing structure
    - Java code mixed with HTML and Javascript and ???
    - Code spread in multiple segments, obscure namespaces
  - Lots of “behind the scenes” configuration and client server nonsense