Applications

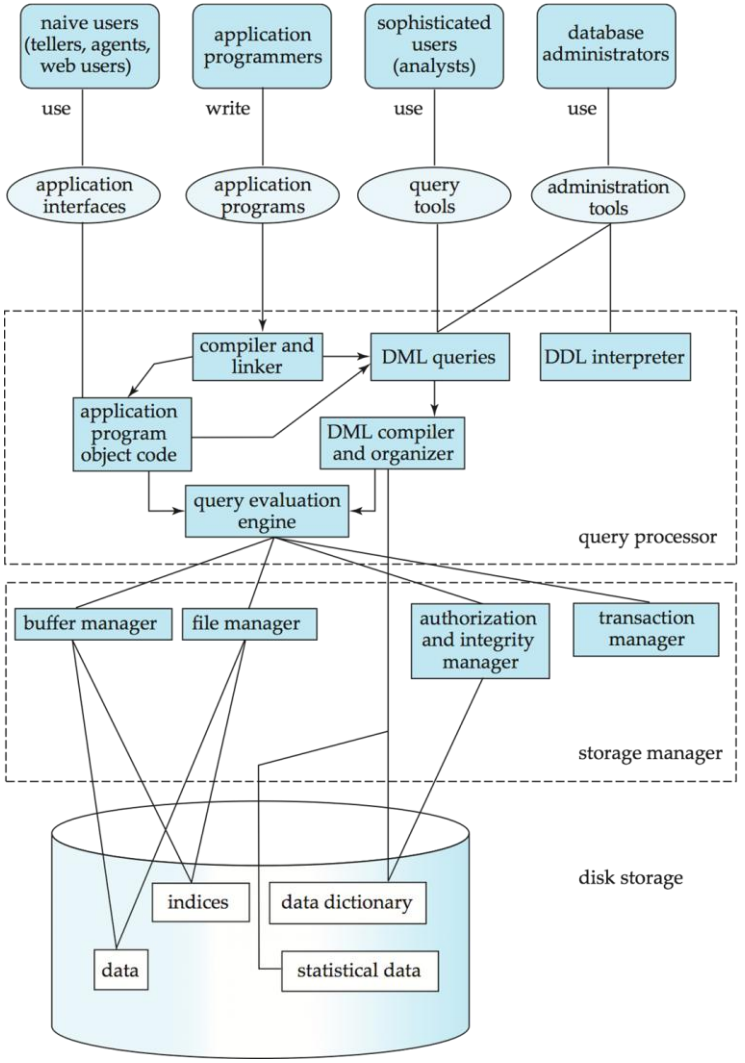
CS460 Databases for Data Scientists

UNIVERSITY OF MANNHEIM
Data and Web Science Group

Recap: The Big Picture

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Users interact with databases *indirectly*
  - i.e., via *applications*
  - no direct usage of SQL
- Most applications today have a database under the hood, e.g.,
  - shopping portals
  - news web sites
  - games



Today's Lecture

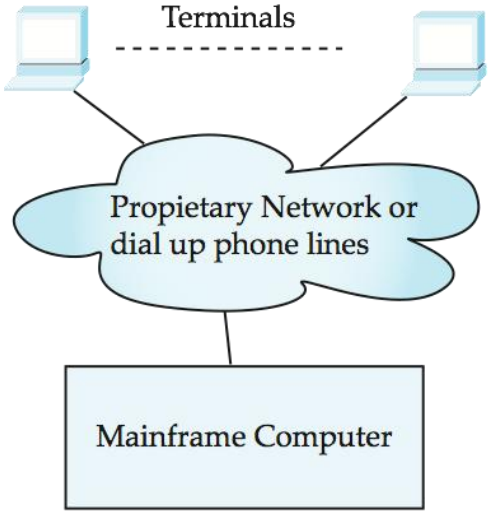
UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Architectures for database centric applications
  - HTTP/HTML/Session/Cookies
  - Server/Client Side Scripting
- Legacy Systems
- Performance Tuning
  - Bottlenecks
  - Database Design
- Security Issues
  - SQL Injection
  - Cross Site Scripting
  - Password Leakage
  - Application Authentication/Authorization

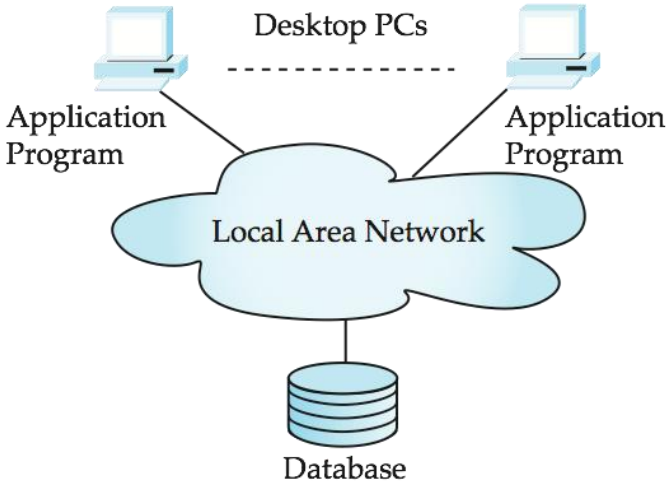
Application Architecture Evolution

UNIVERSITY OF MANNHEIM
Data and Web Science Group

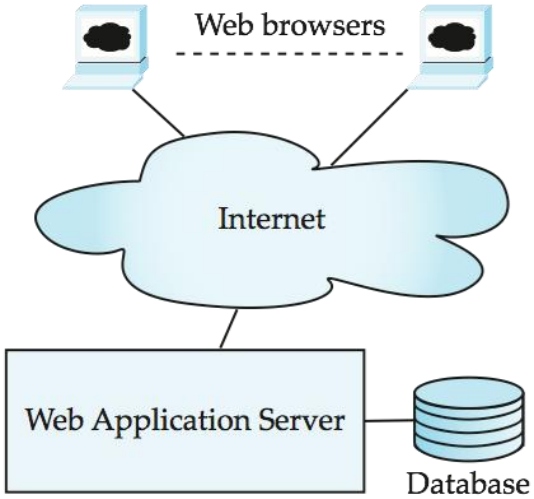
- Three eras of application architecture
  - mainframe (1960's and 70's)
  - personal computer era (1980's)
  - Web era (since 1990's, nowadays mostly mobile Web)



(a) Mainframe Era



(b) Personal Computer Era



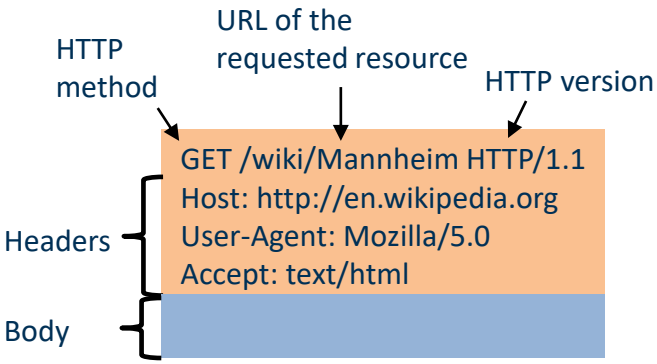
(c) Web era

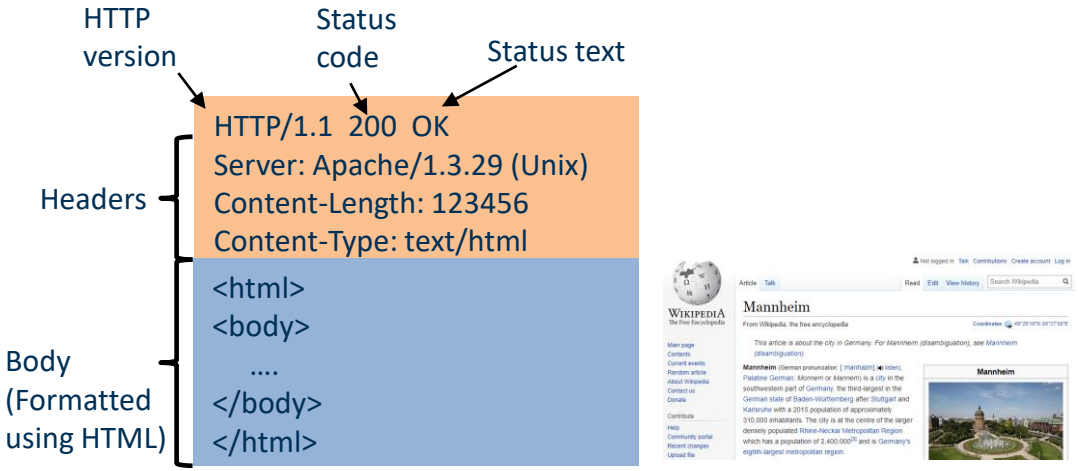
Web Interface

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Web browsers
  - de-facto standard user interface to databases
  - multi-user, location agnostic interface
  - no need for downloading/installing specialized code, while providing a good graphical user interface
    - JavaScript and other scripting languages run in browser, but are downloaded transparently
  - Examples: banks, airline and rental car reservations, university course registration and grading, ...

Web based Applications in a Nutshell





Web based Applications in a Nutshell

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- HyperText Transfer Protocol (**HTTP**)
  used for communication with the Web server
  - URL may identify a document or an executable program
    - executed by HTTP server
    - creates HTML documents, which is sent back to client
    - Web client can pass extra arguments with the name of the document
- Web documents are *hypertext* documents
  formatted using HyperText Markup Language (**HTML**)
  - HTML documents contain
    - text along with font specifications, and other formatting instructions
    - hypertext links to other documents
    - forms, enabling users to enter data which can then be sent back to the Web server

Sample HTML Source Text

UNIVERSITY OF MANNHEIM
Data and Web Science Group

```
<html>
<body>
  <table border>
    <tr> <th>ID </th> <th>Name</th> <th>Department</th>
    </tr><tr> <td>00128</td> <td>Zhang</td> <td>Comp. Sci.</td></tr>
    ....
  </table>
  <form action="PersonQuery" method= "get">
    Search for:
    <select name="persontype">
        <option value="student" selected>Student </option>
        <option value="instructor"> Instructor </option>
    </select> <br>
    Name: <input type="text" size=20 name="name">
    <input type ="submit " value="submit">
  </form>
</body>
</html>
```

ID	Name	Department
00128	Zhang	Comp. Sci.
12345	Shankar	Comp. Sci.
19991	Brandt	History

Search for: Student
Name:
submit

HTTP and Sessions

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- The HTTP protocol is **connectionless**
  - Once the server replies to a request, the server closes the connection with the client, and forgets all about the request
  - Motivation: reduce load on server
    - operating systems have tight limits on number of open connections on a machine
- Information services need session information
  - E.g., user authentication should be done only once per session
- Solution: **cookies**

Sessions and Cookies

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- A **cookie** is a small piece of text containing identifying information
  - Sent by server to browser
    - Sent on first interaction, to identify session
  - Sent by browser with each request
    - part of the HTTP protocol
  - Server saves information about cookies it issued, and can use it when serving a request
    - E.g., authentication information, and user preferences
- Cookies can be stored permanently or for a limited time

Server Side HTML Generation

UNIVERSITY OF MANNHEIM
Data and Web Science Group

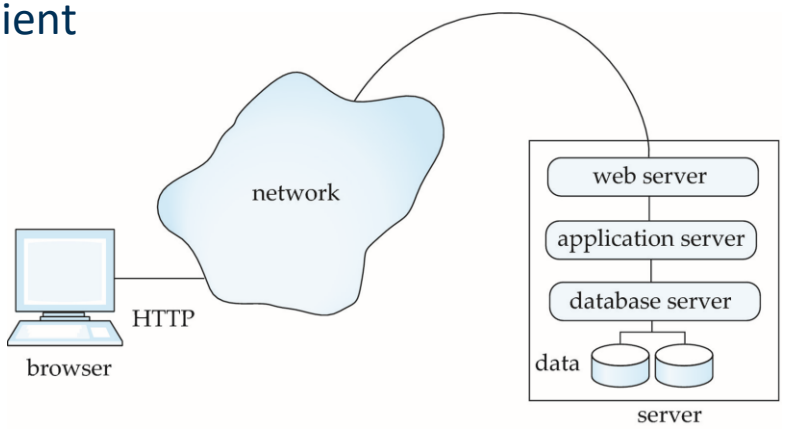
- Paradigms
  - **Programming**
    - HTML is generated by Java/Python program
    - Example: **Servlets**
  - **Scripting**
    - Programming language is embedded in HTML
    - Example: **Java Server Pages (JSP), PHP**

Server Side - Programming – Servelets

```
public class PersonQueryServlet extends HttpServlet {
  public void doGet (HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException
  {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HEAD><TITLE> Query Result</TITLE></HEAD>");
    out.println("<BODY>");
    String name = request.getParameter("name");
    out.println(“Your name is: “ + name);
    //... code to find students with the specified name ...
    //... using JDBC to communicate with the database ..
    out.println("</BODY>");
    out.close();
  }
}
```

Server Side - Programming – Servelets

- Application program (also called a servlet) run inside application servers such as
  - Apache Tomcat, Glassfish, Jboss, BEA Weblogic, IBM WebSpher, …
- Java Servlet specification
  - defines an API between the application server and application program
    - methods to get parameter values from Web forms
    - methods to send HTML text back to client
- This is a three layer architecture
  - Web, application, database server
  - Web and application server can also be combined (two layer architecture)



Server Side – Scripting – Java Server Pages (JSP)

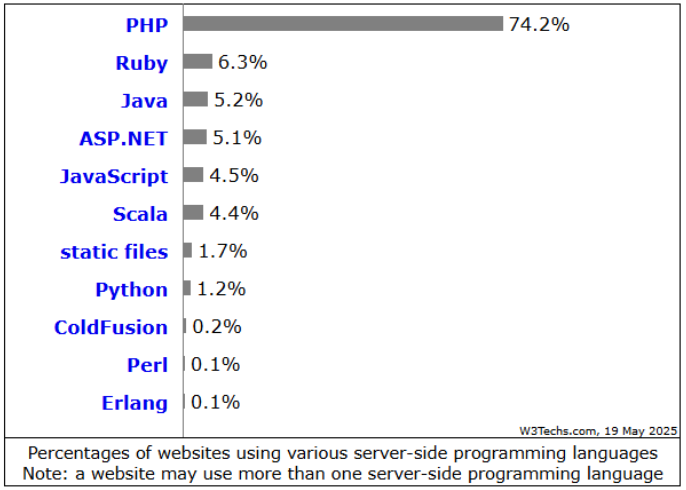
UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Server-side scripting with JSP
  - HTML document with embedded executable code and/or SQL queries
  - When the document is requested, the Web server executes the embedded code/SQL queries to generate the actual HTML document
- JSP is compiled into Java + Servlets

```
<html>
<head> <title> Hello </title> </head>
<body>
<%
    if (request.getParameter("name") != null){
        out.println("Hello, " + request.getParameter("name"));
    }
%>
</body>
</html>
```

Server Side – Scripting – PHP

- PHP is widely used for Web server scripting



Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language

```
<html>
<head> <title> Hello </title> </head>
<body>
<?php
    if (isset($_REQUEST['name'])){
      echo "Hello, " + $_REQUEST['name'];
    }
?>
</body>
</html>
```


Source: https://w3techs.com/technologies/overview/programming_language

Client Side - Scripting

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Browsers can fetch certain scripts (client-side scripts) or programs along with documents, and execute them in "safe mode" at the client site
  - Today: mostly Javascript
  - Historic: Macromedia Flash/Shockwave for animation/games, VRML, Java Applets
- Client-side scripts/programs allow documents to be active
  - E.g., animation by executing programs at the local site
  - E.g., ensure that values entered by users satisfy some correctness checks
  - Permit flexible interaction with the user
    - Executing programs at the client site speeds up interaction by avoiding many round trips to server

Source: https://de.wikipedia.org/wiki/Virtual_Reality_Modeling_Language

Javascript

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Javascript very widely used
  - forms basis of new generation of Web applications (called Web 2.0 applications) offering rich user interfaces
- Javascript functions can
  - check input for validity
  - modify the displayed Web page
    - by altering the underlying **document object model (DOM)** tree
  - communicate with a Web server to fetch data and modify the current page using fetched data, without needing to reload/refresh the page
    - forms basis of AJAX technology used widely in Web 2.0 applications
    - e.g., loading further content upon scrolling down a Web page
    - e.g. on selecting a country in a drop-down menu, the list of states in that country is automatically populated in a linked drop-down menu

Legacy Systems

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Older-generation systems that are incompatible with current generation standards and systems but still in production use
  - Today's hot new system is tomorrows legacy system!

- Porting legacy system applications to a more modern environment can be very problematic
  - Legacy system may involve millions of LoC, written over decades
    - Original programmers usually no longer available
    - **Reverse engineering:** process of going over legacy code to
      - Come up with schema designs in ER or OO model /get high level view
    - **Re-engineering:** reverse engineering followed by design of new system

Legacy Systems

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Switching over from old to new system is a major problem
  - Production systems are in every day, generating new data
  - Stopping the system may bring all of a company's activities to a halt, causing enormous losses
- **Big-bang approach**:
  - Implement complete new system
  - Populate it with data from old system
    - No transactions while this step is executed
    - Scripts are created to do this quickly
  - Shut down old system and start using new system
  - Danger with this approach: what if new code has bugs or performance problems, or missing features
    - Company may be brought to a halt

Legacy Systems

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **Chicken-little approach:**
  - Replace legacy system one piece at a time
  - Use wrappers to interoperate between legacy and new code
    - E.g., replace front end first, with wrappers on legacy backend
      - Old front end can continue working in this phase in case of problems with new front end
    - Replace back end, one functional unit at a time
      - All parts that share a database may have to be replaced together, or wrapper is needed on database as well
  - Drawback: significant extra development effort to build wrappers and ensure smooth interoperation
    - Still worth it if company's life depends on system

Performance Tuning

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Adjusting various parameters and design choices
  - to improve system performance for a specific application
  - notion: continuous improvement rather than waterfall model
- Tuning is best done by
  1) identifying bottlenecks, and
  2) eliminating them
- Three levels of tuning
  - **Hardware**,
    e.g., add disks, memory, use faster processor
  - **Database system parameters**,
    e.g., buffer size, checkpointing intervals
  - **Higher level database design**,
    e.g., schema, indices, and transactions

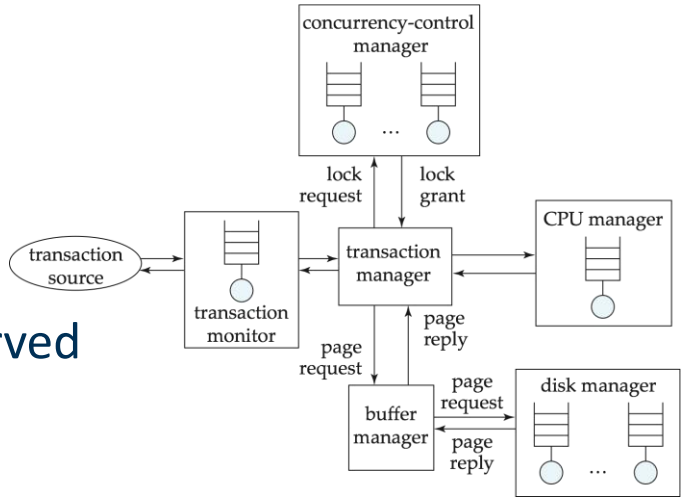
Bottlenecks

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Performance of most systems usually limited by performance of one or a few components
  - these are called **bottlenecks**
  - 80/20 rule: 20% of code consume 80% of execution time
    - spend more time on optimizing those 20%
- Bottlenecks may be in hardware
  (e.g., disks are very busy, CPU is idle), or in software

- Removing one bottleneck often exposes another
- **De-bottlenecking** consists of repeatedly finding bottlenecks, and removing them

Identifying Bottlenecks



- Transactions request a sequence of services
  - E.g., CPU, Disk I/O, locks
- Concurrent transactions wait for a requested service while others are being served
- Notion: database as a **queueing system** with a queue for each service
  - Transactions repeatedly do the following
    - request a service, wait in queue for the service, and get serviced
- Bottlenecks in a database system typically show up as very high utilizations (very long queues) of a particular service
  - e.g., disk vs. CPU utilization
  - 100% utilization leads to very long waiting time:
    - Rule of thumb: design system for about 70% utilization at peak load
    - utilization over 90% should be avoided

Tuning of Hardware

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Even well-tuned transactions typically require a few I/O operations
  - Typical disk supports about 100 random I/O operations per second
  - Suppose each transaction requires just 2 random I/O operations
    - to support *n* transactions per second,
      we need to distribute data across *n*/50 disks (ignoring skew)
- Number of I/O operations per transaction can be reduced by keeping more data in memory
  - If all data is in memory, I/O needed only for writes
  - Keeping frequently used data in memory reduces disk accesses, reducing number of disks required, but has a memory cost
- Five minute rule:
  - if a page that is randomly accessed is used **more frequently than once in five minutes**, it should be kept in memory

Tuning the Database Design

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **Schema tuning**
  - Vertically partition relations to isolate the data that is accessed most often
    - e.g., split *account* into two, (*account-number*, *branch-name*) and (*account-number*, *balance*).
      - branch name need not be fetched unless required
      - More rows per block → less block transfers
- Improve performance by storing a **denormalized relation**
  - E.g., store join of *account* and *depositor*; branch-name and balance information is repeated for each holder of an account
    - join need not be computed repeatedly
    - trade-off: more space and more work for programmer to keep relation consistent on updates
    - Better to use materialized views (see later)

Tuning the Database Design (Cont.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Incidental violations of normal forms
  - e.g., storing join tables that would be split by normalization
- Incidental violations of domain model
  - Example: each person can have many phone numbers (1:n)
  - Theoretically sound solution: two tables (person, phone)
  - Practical observation: not more than four in 1M persons
    - rather introduce attributes phone1, phone2, phone3, phone4
    - avoids joins with long tables

Tuning the Database Design (Cont.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

Materialized Views

- Materialized views can help speed up certain queries
  - Particularly aggregate queries
- Overheads
  - Space
  - Time for view maintenance
    - Immediate view maintenance: done as part of update transaction
      - time overhead paid by update transaction
    - Deferred view maintenance: done only when required
      - update transaction is not affected, but system time is spent on view maintenance
      - until updated, the view may be out-of-date
- Materialized Views should be preferred over denormalized schema since view maintenance is system's responsibility, not programmer's
  - Avoids inconsistencies caused by errors in update programs

Tuning the Database Design (Cont.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- How to choose set of materialized views
  - Helping one transaction type by introducing a materialized view may hurt others
    - selections including aggregates will be speed up
    - updates are slowed down
  - Choice of materialized views depends on costs
    - Users often have no idea of actual cost of operations
  - Overall, manual selection of materialized views is tedious
- Some database systems provide tools to help to choose views to materialize
  - “Materialized view selection wizards”

Tuning the Database Design (Cont.)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **Index tuning**
  - Create appropriate indices to speed up slow queries/updates
  - Speed up slow updates by removing excess indices (tradeoff between queries and updates)
  - Choose type of index (B-tree/hash) appropriate for most frequent types of queries
  - Choose which index to make clustered
- Index tuning wizards look at past history of queries and updates (the **workload**) and recommend which indices would be best for the workload

Application Security

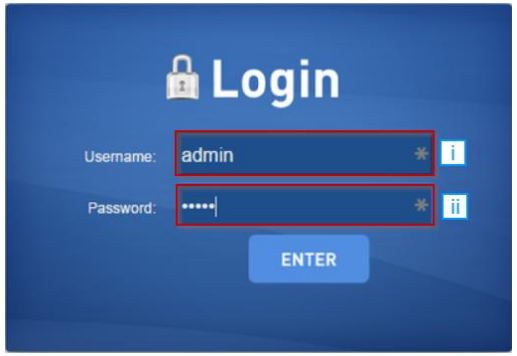
UNIVERSITY OF MANNHEIM
Data and Web Science Group

SQL Injection

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- In an application, users enter data
  - this is a possible entry point for hackers!
- Consider the following code (in a Servelet):

```
String user = request.getParameter("username");
String password = request.getParameter("password");
String query = "SELECT * FROM users
                    WHERE username = '" + user + "'
                    AND password = '" + password + "';";
// execute: if there is a result, the login worked
```



- Good user:
  - username "John", password "test123"
- Bad user:
  - username "Jack", password "test123' OR 1=1;--"

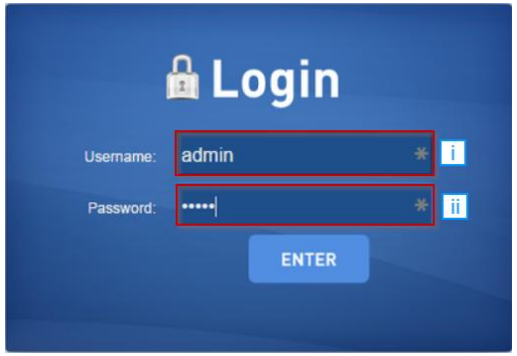
SQL Injection

" ends the java string

```
String query = "SELECT * FROM users
               WHERE username = '" + user + "'
               AND password = '" + password + "';";
```

- Good user:
  - username "John", password "test123"

```
String query = "SELECT * FROM users
               WHERE username = 'John'
               AND password = 'test123';
```



- Bad user:
  - username "Jack", password "test123' OR 1=1;-- "

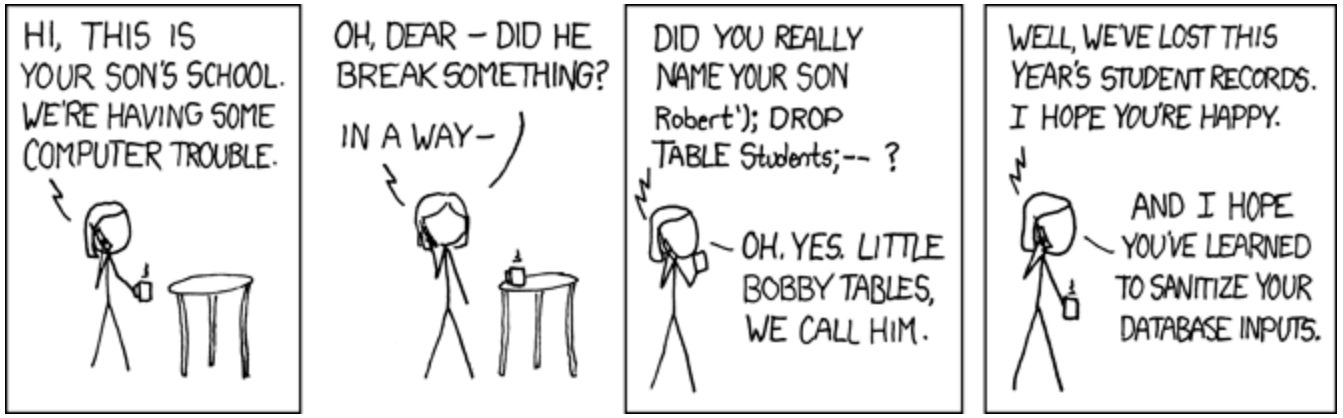
```
String query = "SELECT * FROM users
               WHERE username = 'John'
               AND password = 'test123' OR 1=1;-- ';
```

-- Is a line comment in SQL

SQL Injection

- Variant 1: Manual
  - Check input for and mask/replace/remove special characters
- Variant 2: Using prepared statements in Java

```
PreparedStatement stmt = connection.prepareStatement
  ("SELECT * FROM users WHERE username=? AND password=?");
stmt.setString(1, user);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
```



Cross Site Scripting

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- HTML code on one page executes action on another page

```
<img src = http://mybank.com/transfermoney?amount=1000&toaccount=14523>
```

  - Risk: if user viewing page with above code is currently logged into mybank, the transfer may succeed
  - Above example simplistic, since GET method is normally not used for updates, but if the code were instead a script, it could execute POST methods

- Above vulnerability called **cross-site scripting (XSS)** or **cross-site request forgery (XSRF or CSRF)**

Cross Site Scripting

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Prevent your web site
  from being used to launch XSS or XSRF attacks
  - Disallow HTML tags in text input provided by users, using functions to detect and strip such tags
- Protect your web site
  from XSS/XSRF attacks launched from other sites
  - Use **referer** value (URL of page from where a link was clicked) provided by the HTTP protocol, to check that the link was followed from a valid page served from same site, not another site
  - Ensure IP of request is same as IP from where the user was authenticated
    - prevents hijacking of cookie by malicious user
  - **Never** use a GET method to perform any updates

Password Leakage

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **Never** store passwords, such as database passwords, **in clear text in scripts (program code)** that may be accessible to users
  - E.g. in files in a directory accessible to a web server
    - `connect_db("root","password123")`
    - Normally, web server will execute, but not provide source of script files such as file.jsp or file.php, but…
    - source of editor backup files such as file.jsp~, or .file.jsp.swp may be served
- Restrict access to database server from IPs of machines running application servers
  - Most databases allow restriction of access by source IP address

Password Leakage

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **Never** store user passwords as plain text in a database!
- Hackers may get access to the database and read them
  - e.g., username “Jack”, password “test123; SELECT * FROM users”

- Typical best practice: store password hashes, e.g., md5
  - hashing is fast in one direction, hard in the other
  - Query:
    - SELECT * FROM users WHERE user=? and password=md5(?)
  - Changing passwords
    - UPDATE users SET password=md5(?) WHERE user=?
  - This way, passwords are never stored in plain text anywhere

Password Leakage

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Attacks for hashed passwords: dictionary and brute force attacks

Dictionary Attack

```
Trying apple          : failed
Trying blueberry      : failed
Trying justinbieber   : failed
...
Trying letmein         : failed
Trying s3cr3t         : success!
```

Brute Force Attack

```
Trying aaaa           : failed
Trying aaab           : failed
Trying aaac           : failed
...
Trying acdb           : failed
Trying acdc           : success!
```

- Lookup Tables

- Adding Salt to the password (appending a random string)
  - Lookup tables won't work
  - Store the salt (random string) and the hash
- Do not implement your own crypto algorithm (use e.g. phpass)

Application Authentication

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Single factor authentication such as passwords
  too risky for critical applications
  - guessing of passwords, sniffing of packets if passwords are not encrypted
  - passwords reused by user across sites
  - spyware which captures password
- Two-factor authentication
  - e.g. password plus one-time password sent by SMS
  - e.g. password plus one-time password devices
    - device generates a new pseudo-random number every minute, and displays to user
    - user enters the current number as password
    - application server generates same sequence of pseudo-random numbers to check that the number is correct

Application Authentication

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **Man-in-the-middle** attack
  - E.g. web site that pretends to be mybank.com, and passes on requests from user to mybank.com, and passes results back to user
  - Even two-factor authentication cannot prevent such attacks
- Solution: authenticate Web site to user, using digital certificates, along with secure http protocol
- **Central authentication** within an organization
  - application redirects to central authentication service for authentication
  - avoids multiplicity of sites having access to user's password
  - LDAP or Active Directory used for authentication

Single Sign-On

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- **Single sign-on** allows user to be authenticated once, and applications can communicate with authentication service to verify user's identity without repeatedly entering passwords
- **Security Assertion Markup Language (SAML)** standard for exchanging authentication and authorization information across security domains
  - e.g. user from Yale signs on to external application such as acm.org using userid joe@yale.edu
  - application communicates with Web-based authentication service at Yale to authenticate user, and find what the user is authorized to do by Yale (e.g. access certain journals)
- **OpenID** standard allows sharing of authentication across organizations
  - e.g. application allows user to choose Yahoo! as OpenID authentication provider, and redirects user to Yahoo! for authentication

Application-Level Authorization

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Current SQL standard does not allow fine-grained authorization such as "students can see their own grades, but not other's grades"
  - Problem 1: Database has no idea who are application users
  - Problem 2: SQL authorization is at the level of tables, or columns of tables, but not to specific rows of a table
- One workaround: use views such as

```
CREATE VIEW studentTakes AS
SELECT *
FROM takes
WHERE takes.ID = USER()
```

no SQL standard; varies from implementation to implementation

  - where USER() provides end user identity
    - end user identity must be provided to the database by the application
  - Having multiple such views is cumbersome

Audit Trails

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Applications must log actions to an audit trail, to detect who carried out an update, or accessed some sensitive data
- Audit trails used after-the-fact to
  - detect security breaches
  - repair damage caused by security breach
  - trace who carried out the breach
- Audit trails needed at
  - Database level, and at
  - Application level

Summary

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Databases do not run by themselves, but in context
  - applications work on top
- A good database design is essential, but there's also
  - security
  - performance,
  - …
- There's quite a few trade offs
  - storage vs. velocity
  - update vs. read time
  - …

  → there's no once size fits all solution!

Questions?