Dr Greg Wadley
David Eccles

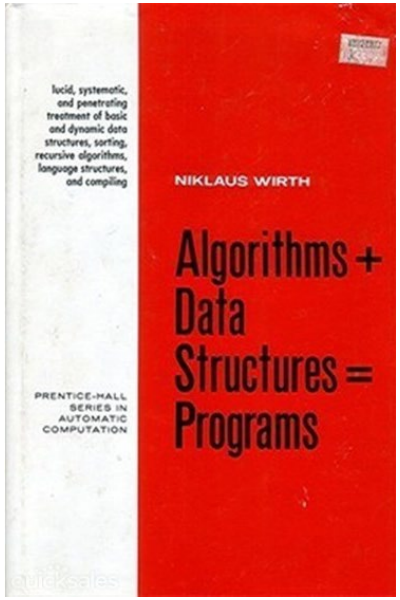
INFO90002 Database Systems & Information Modelling

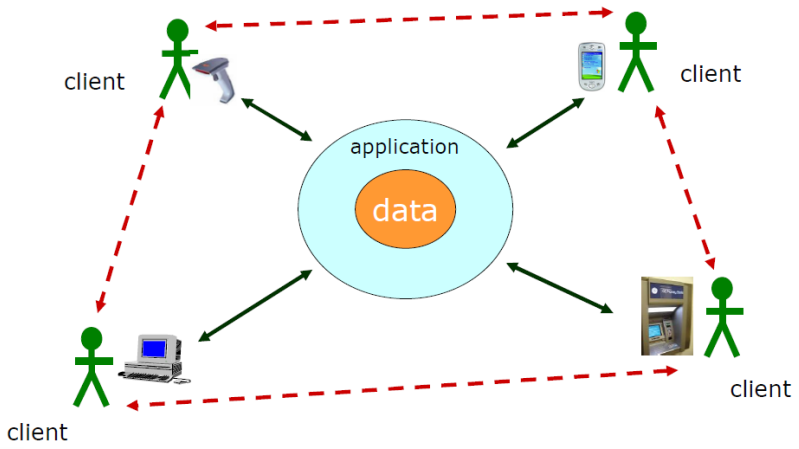
Lecture 11

Applications
Web Applications
and Databases

Today's Session…

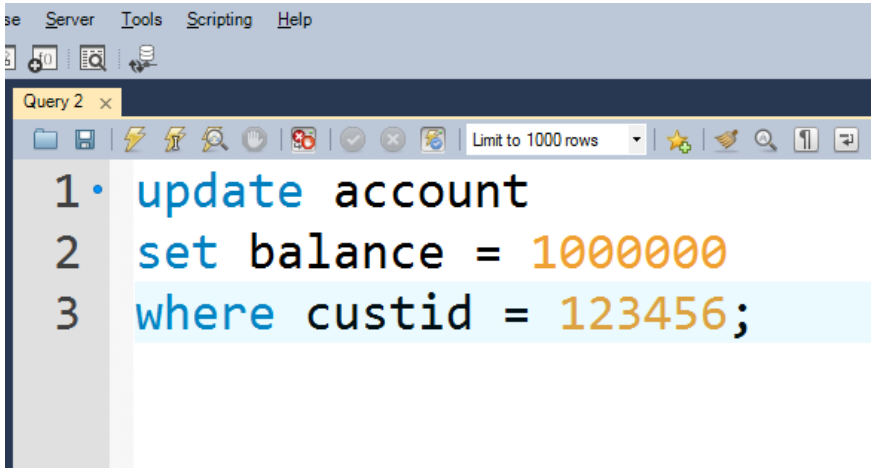
- How end-users access the database
- Business logic
- Stored procedures and triggers
- Embedding databases inside applications
- Application architectures
- Web applications
- How web apps work
- Making an HTML document
- Connecting to the DB
- Web services

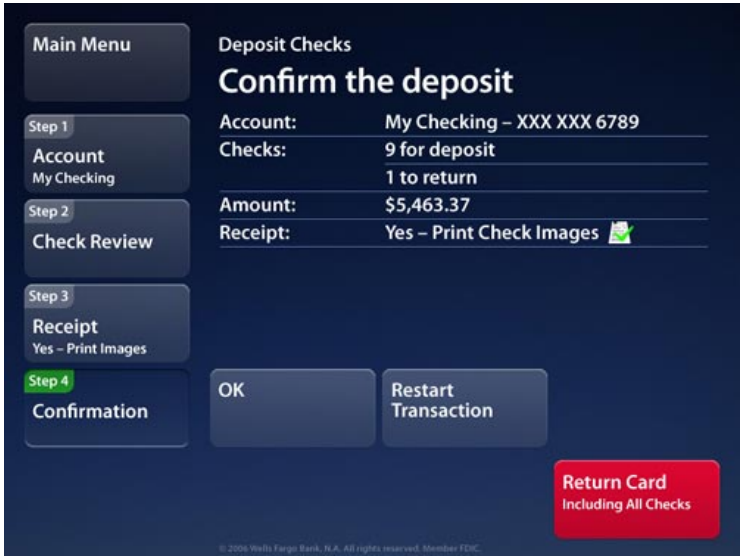




Limitations of SQL

- SQL is declarative, intuitive, versatile, but …
  - cannot express all possible queries in SQL
  - need to enforce business rules beyond domain/ref. integrity
  - need procedural constructs such as loops and decisions
  - would you give end-users a query browser? Why not?
  - need a user interface that is both friendly and constraining





How to handle business logic?

- Examples of business logic:
  - Check name and password. If good, login, if bad, error message
  - Insert one row in *Order* table, then several in *OrderItem* table
  - Check amount < balance. If so, subtract amount from one row in bank account table, then add amount to another row
  - For all rows in Customer table, send out monthly statements
- Procedural programming languages can do:
  - Sequence (several steps performed in order)
  - Iteration (loops)
  - Control flow (conditionals, decisions)
  - User interface (accept input and present output for users)
- SQL is specialized for low-level data access

Example business logic

- Customer places an order
  - Accept inputs from user (e.g. via web form)
  - Insert row into Order table
  - Repeat for each product ordered:
    - Check Product table shows sufficient quantity in stock. If so:
      - Insert one row into OrderItem table
      - Change Product table in-stock, Customer table amount-owing
  - If no errors encountered, end successfully
- Customer moves money from savings to credit card account
  - Accept inputs from user (via ATM, internet banking or mobile app)
  - Select balance from savings account
  - Is there enough money to withdraw? If so:
    - Update savings account balance = balance – withdrawal
    - Update credit card balance = balance + withdrawal
  - If no errors encountered, end successfully

Procedural Language + SQL (PL SQL)

- Need to combine data manipulation with the ability to handle sequence, iteration, decision. Different approaches:
  - "Embedded SQL"
    - "host language" = C, Fortran, Cobol, Java, etc.
    - SQL statements are embedded in code and replaced with library calls during compilation
  - "Dynamic SQL"
    - host language sends SQL to DBMS via middleware e.g. ODBC/JDBC
    - data is passed back to program as record-set
    - host language can handle business and presentation logic
  - Stored Procedures, Triggers
    - procedural code is stored and executed in the DBMS
    - enforce business logic within the database
    - in SQL-92 standard, but implemented differently in different DBMS
    - largely obsolete

Stored Procedures and Triggers

- Advantages
  - Compiled SQL statements
  - Faster code execution
  - Reduced network traffic
  - Improved security and data integrity
  - Business logic under control of DBA
  - Thinner clients
- Disadvantages
  - Code is not under the control of the application programmer
  - Proprietary language
    - e.g. MySQL Stored Proc's can't be used in Oracle or SQL Server

Example stored procedure

1. accept person details as inputs
2. check whether the person is already in the database
3. if yes, return error
4. if no, add to database

(source: Hoffer chapter 8)

```
CREATE OR REPLACE PROCEDURE p_registerstudent
(
 p_first_name   IN VARCHAR2
 p_last_name    IN VARCHAR2
 p_email        IN VARCHAR2
 p_username     IN VARCHAR2
 p_password     IN VARCHAR2
 p_error        OUT VARCHAR2
)
IS
l_user_exists NUMBER := 0;
l_error      VARCHAR2(2000);

BEGIN

BEGIN
    SELECT COUNT(*)
    INTO   l_user_exists
    FROM   users
    WHERE  username = p_username;

  EXCEPTION
  WHEN OTHERS THEN
    l_error := 'Error: Could not verify username';
  END;

IF l_user_exists = 1 THEN
  l_error := 'Error: Username already exists !';
ELSE

  BEGIN
    INSERT INTO users VALUES(p_first_name,p_last_name,p_email,p_username,p_password,SYSDATE);

  EXCEPTION
    WHEN OTHERS THEN
      l_error := 'Error: Could not insert user';
  END;
END IF;

p_error = l_error;
END p_registerstudent;
```

Procedure p_registerstudent accepts first and last name, email, username, and password as inputs and returns the error message(if any).

This query checks whether the username entered already exists in the database.

If the username already exists, an error message is created for the user.

If the username does not exist in the database, the data entered are inserted into the database.

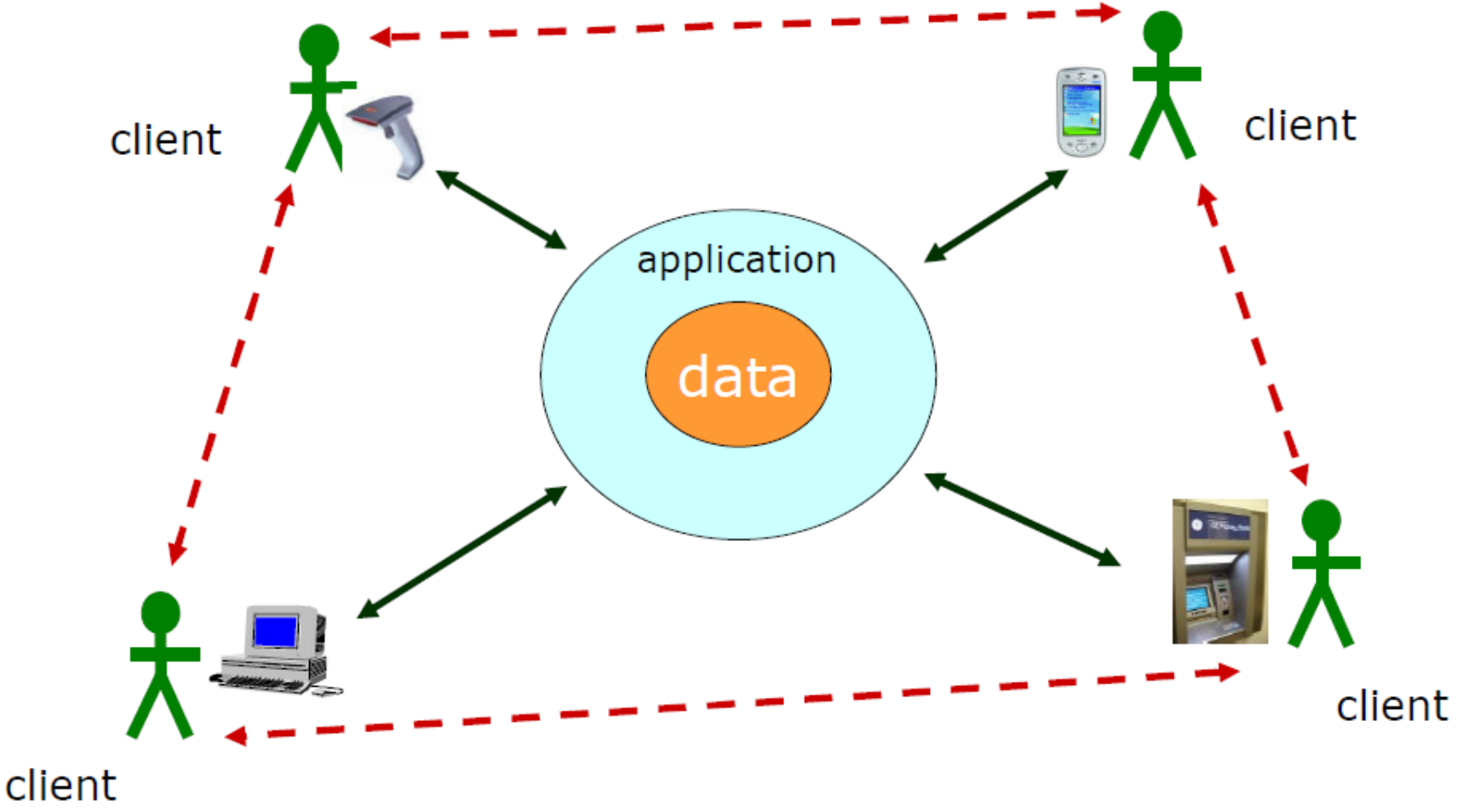
THE UNIVERSITY OF MELBOURNE

Application Architectures

system architecture = “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution“

- ISO/IEC/IEEE 42010:2011
Systems and software engineering — Architecture description

System architecture



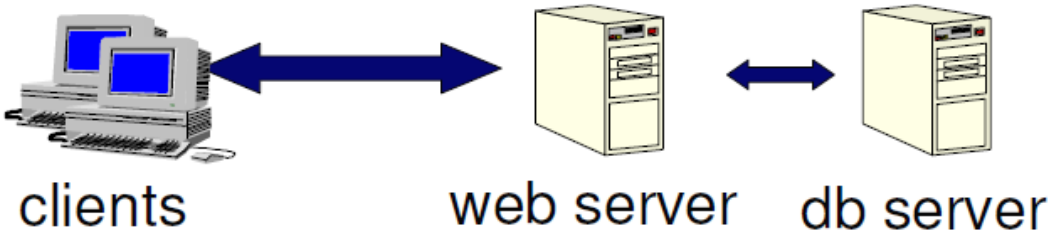
System architecture

- An information system must provide
  - Presentation logic
    - input (keyboard, touchscreen, voice, sensor etc.)
    - output (large screen, printer, phone, ATM etc.)
  - Business logic
    - input and command handling
    - enforcement of business rules
  - Storage logic
    - persistent storage of data
    - enforcement of data integrity

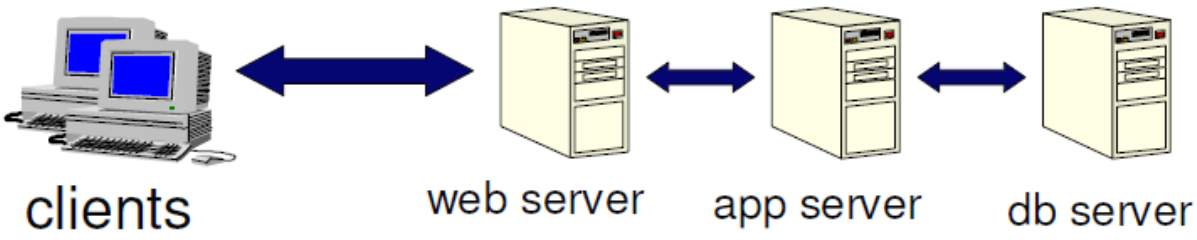
Multi-tiered architectures

- 2 tiers

- 3 tiers

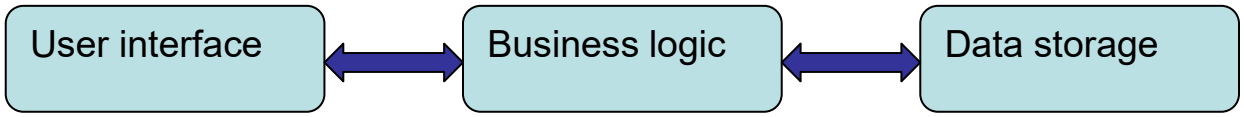


- 4 tiers



Evolution of application architectures

- Mainframe / dumb terminal
  - One large computer handles all logic
  - Problems: doesn't scale with number of users
- Client-Server architecture
  - 2-tier: e.g. file server, database, web
  - 3-tier: separation of Presentation, Processing and Storage logic
- Web architecture
  - a particular form of 3 or 4 tier architecture



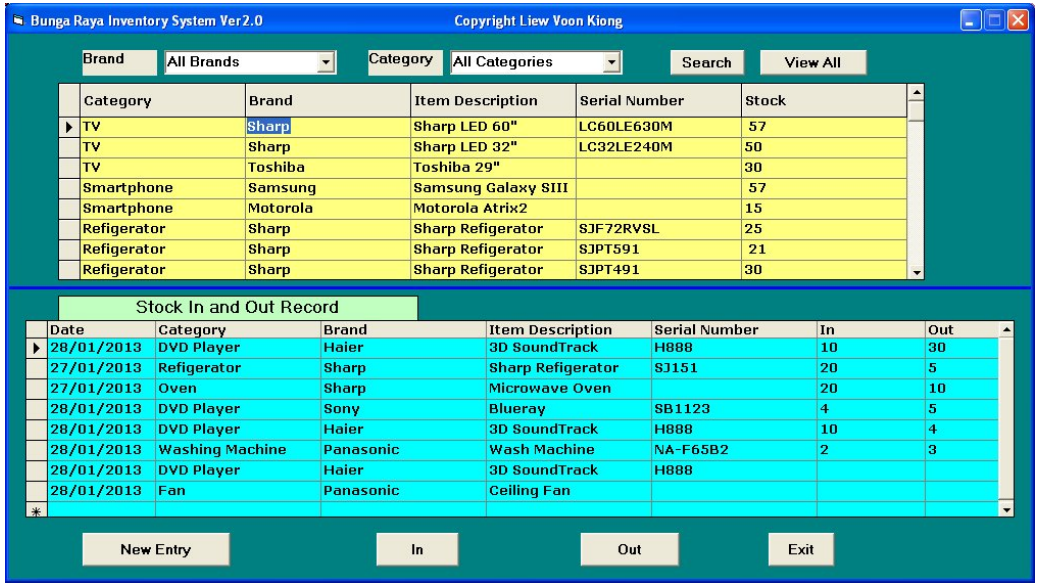
Mainframe ("1Tier")

- Mainframes and mini-computers
- Dumb terminals (no processing at client end)
- Entire application ran on the same computer
  - Database
  - Business logic
  - User interface

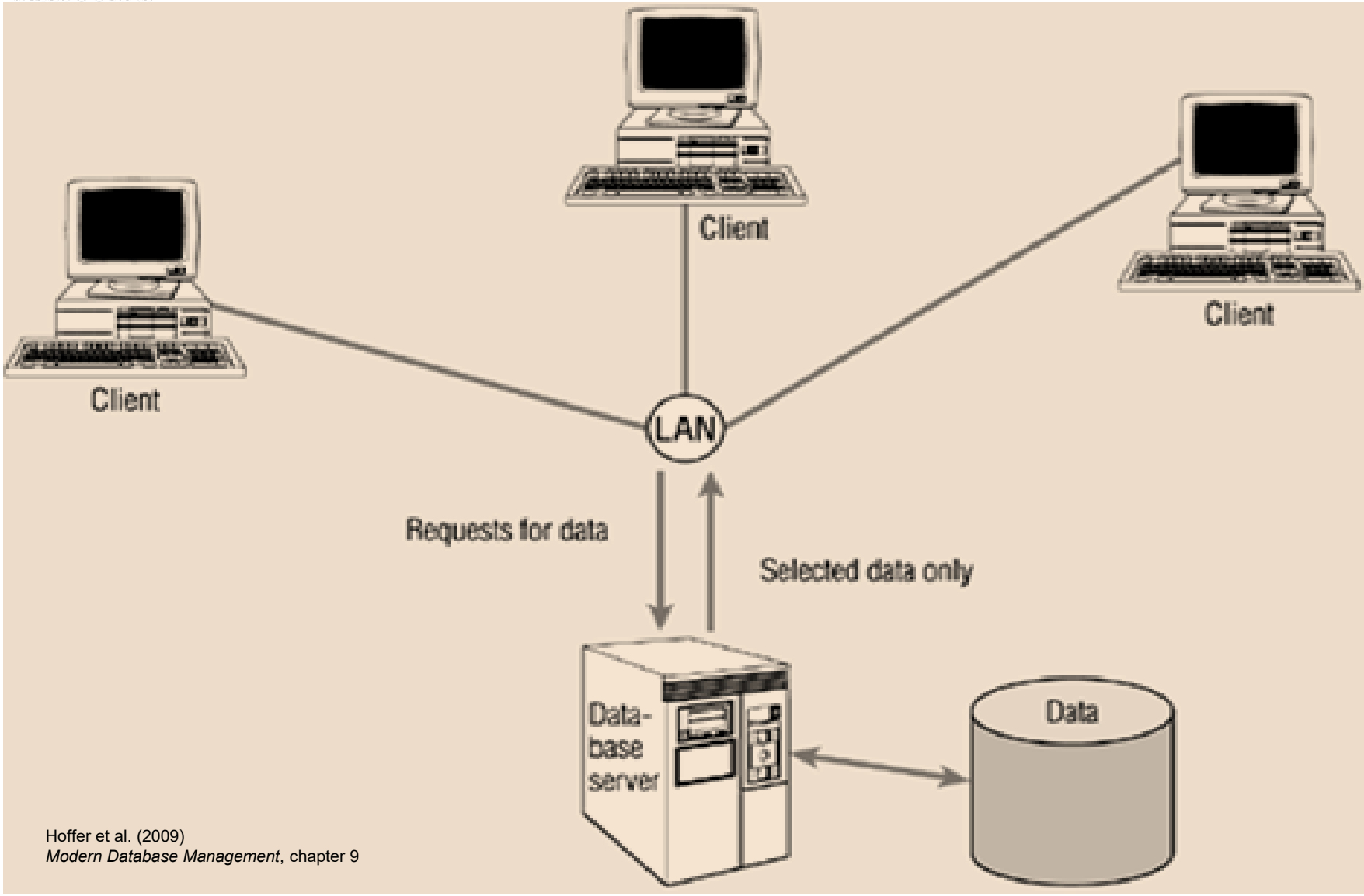
- Enabling technologies included:
  - Embedded SQL
  - Report generators

Client Server - 2 Tier

- Server is a relational DBMS
  - data storage and access is done at the DBMS
- SQL queries sent to DB server, which returns raw data
- Presentation, business logic is handled in client application
- Platforms like Visual Basic (1990s into 2000s)



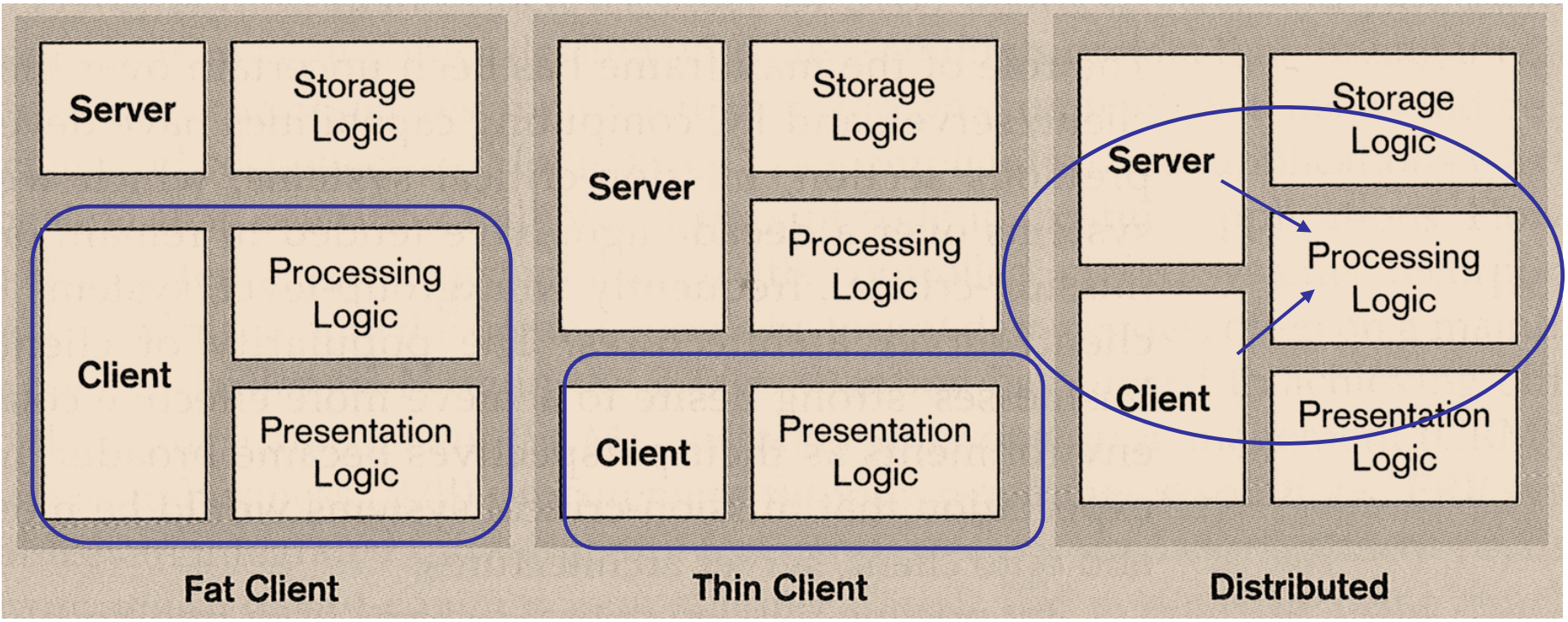
2 Tier Example



Hoffer et al. (2009)
Modern Database Management, chapter 9

Distribution of Processing Logic

- 2-tier distributions
  - Processing logic could be at client, server, or both



Hoffer et al. (2009)
Modern Database Management, chapter 9

2-Tier advantages and disadvantages

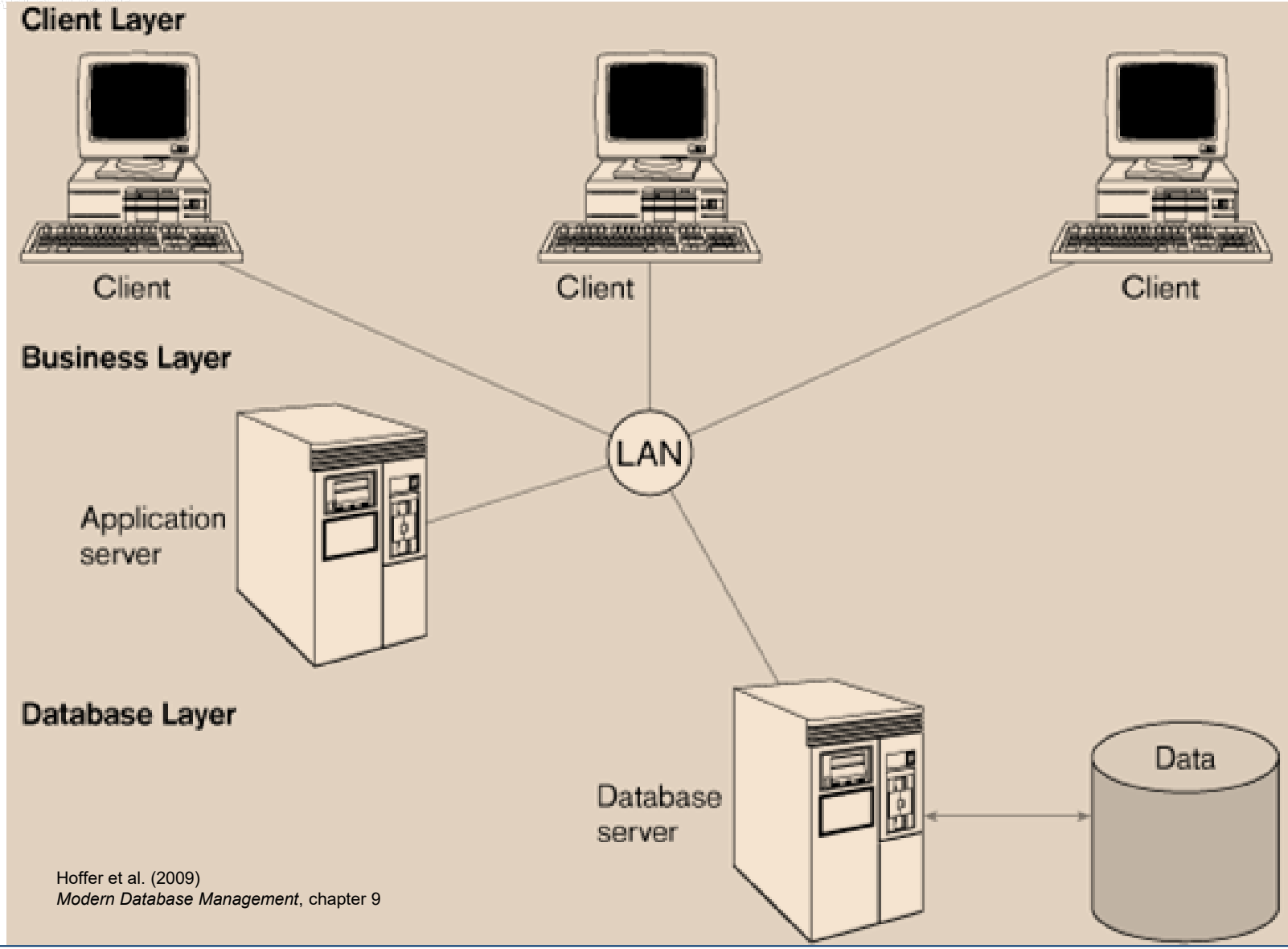
- Advantages
  - Clients and server share processing load
  - Good data integrity since data is all processed centrally
  - Stored procedures allow some business rules to be implemented on the database server
- Disadvantages
  - Presentation, data model, business logic are intertwined at client
  - If DB schema changes, all clients break
  - Updates need to be deployed to all clients
  - DB connection for every client, thus difficult to scale
  - Difficult to implement beyond the organization (to customers)
  - Interoperability issues

3-Tier architecture

- Client program <-> Application server <-> Database server

- Presentation logic
  - Client handles interface
    - Thinner clients
      - Limited or no data storage (possibly no hard disk)
- Business logic
  - Application Server deals with business logic
- Storage logic
  - Database server deals with data persistence and access

A Three-tier architecture - Example



Hoffer et al. (2009)
Modern Database Management, chapter 9

3-Tier advantages and disadvantages

- Advantages
  - Scalability
  - Technological flexibility (can change business logic easily)
  - Can swap out any single component fairly easily
  - Long-term cost reduction
  - Improved security – customer machine does presentation only
- Disadvantages
  - High short-term costs
  - Tools and training
  - Complex to design
  - Variable standards

3-Tier (web based –)

- Browser handles presentation logic
- Browser talks to web server via simple, standard protocol
- Business logic and data storage handled on server(s)
- Pros
  - Everyone has a browser
  - No need for install and maintain client software
  - HTML and HTTP are simple standards, widely supported
  - Opens up the possibility of global access to database
- Cons
  - Even more complexity in the middle-tier
  - Simple standards = hard to make complex application
  - Global access = potential security nightmare (next page)

Security in multi-tier applications

- Network environment creates complex security issues
- Security can be enforced at different tiers:
  - application password security
    - for allowing access to the application software
  - database-level password security
    - for determining access privileges to tables
  - secure client/server communication
    - via encryption

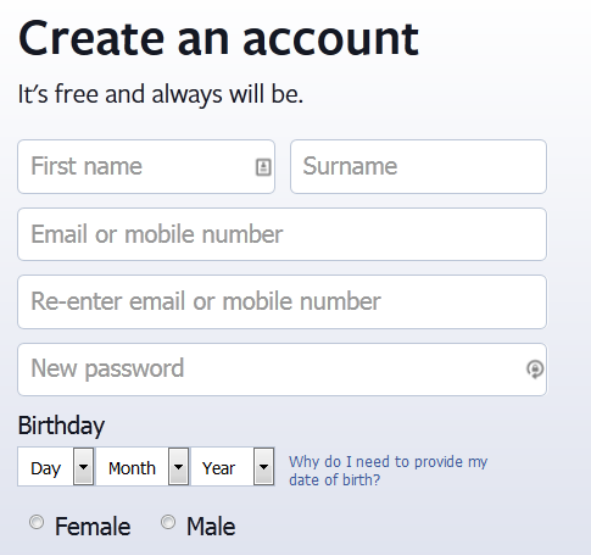
THE UNIVERSITY OF
MELBOURNE

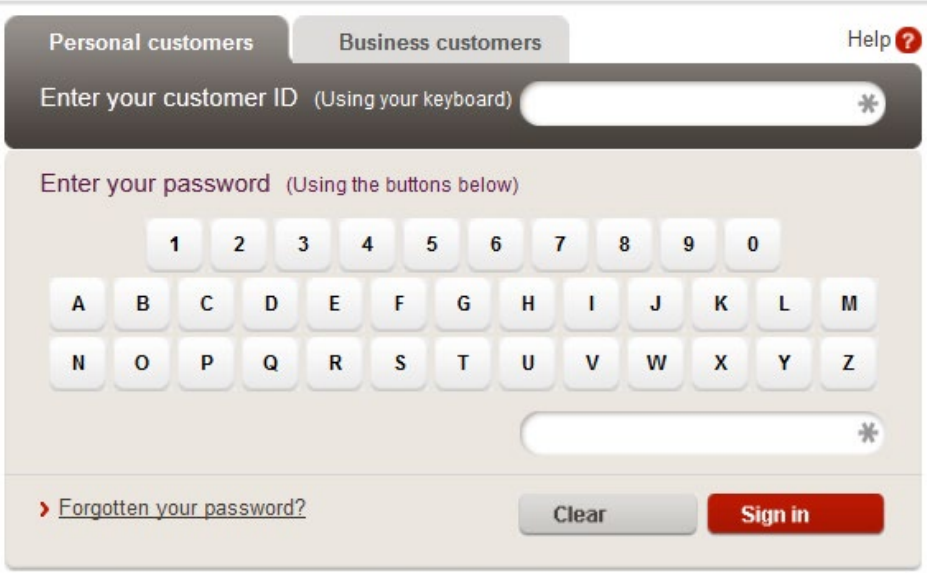
Web Applications

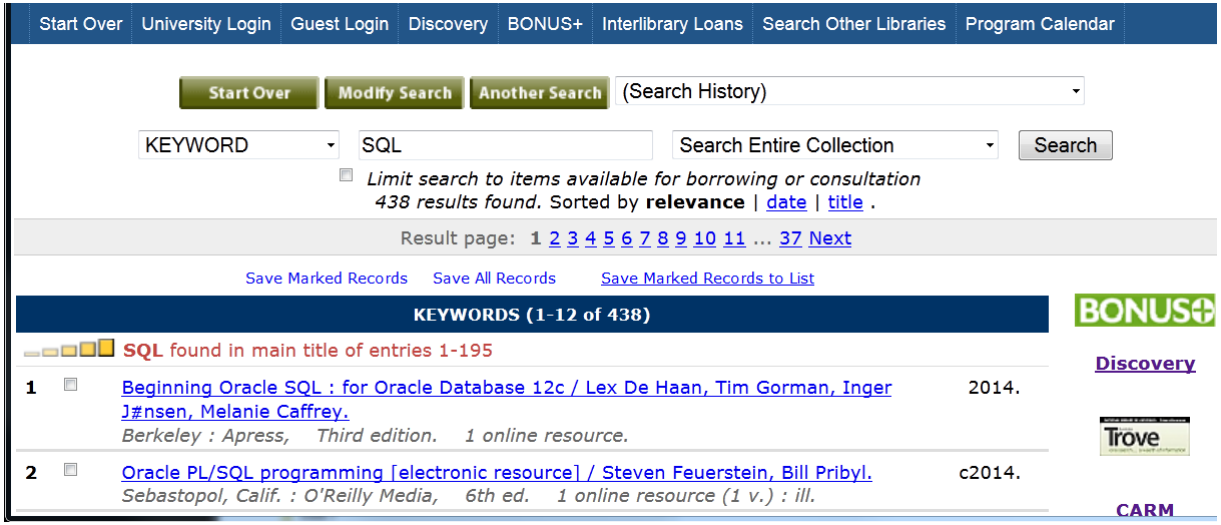
Overview of Web Apps

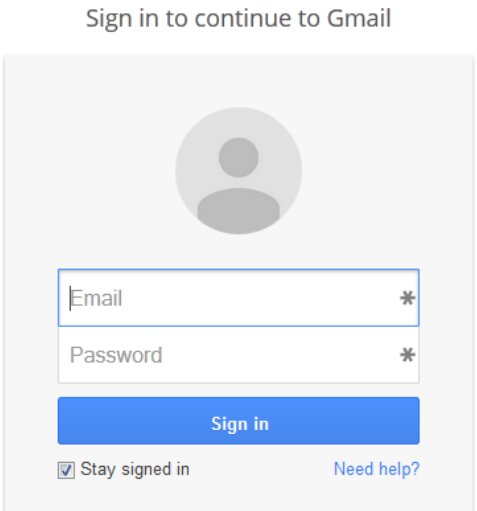
- Why web apps?
- How web apps work
- Making an HTML document
- Connecting to the DB
- Demo web app
- Web services

Example web applications

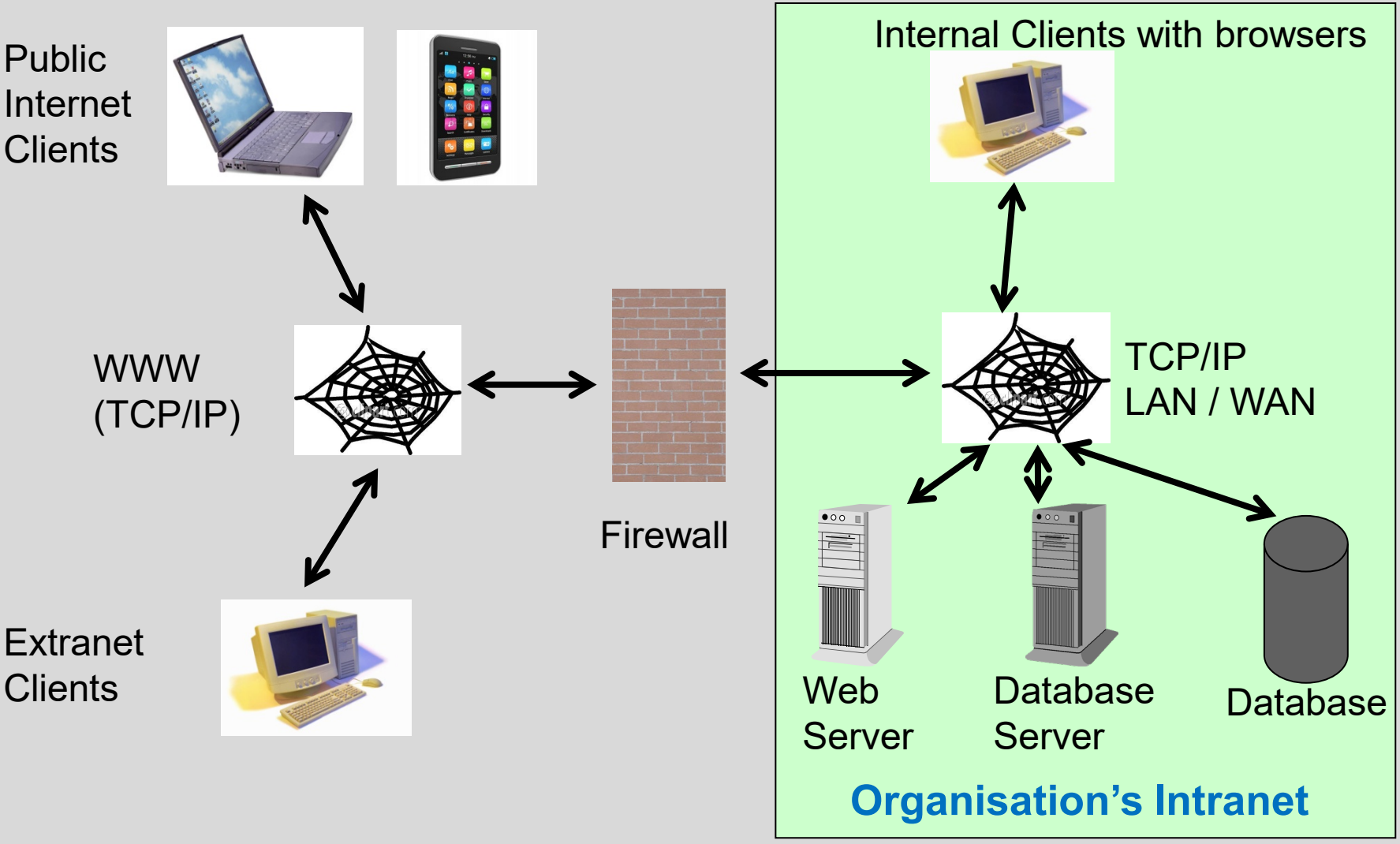






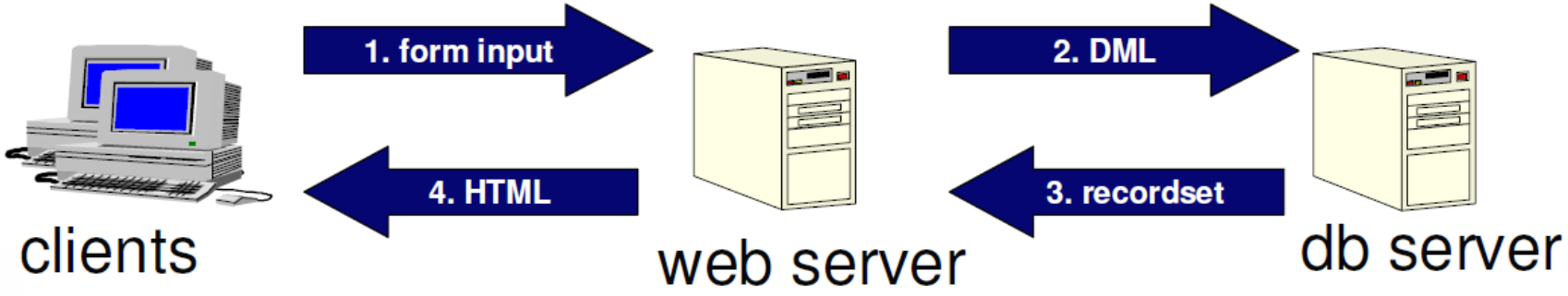


Architecture of a web app



Why create web applications?

- Web browsers are ubiquitous
- No need to install client software for external customers
- Simple communication protocols
- Platform and Operating System independent ("interoperable")
- Reduction in development time and cost
- Has enabled eGov, eBusiness, eCommerce, B2B, B2C



Web infrastructure

- Browser
  - Software that retrieves and displays HTML documents
- Web Server
  - Software that responds to requests from browsers by transmitting HTML and other documents to browsers
- Web pages (HTML documents)
  - Static web pages
    - content established at development time
  - Dynamic web pages
    - content dynamically generated using data from database
- World Wide Web (WWW)
  - The total set of interlinked hypertext documents residing on Web servers worldwide
- Internet
  - Global network infrastructure that hosts the WWW

Web-related languages

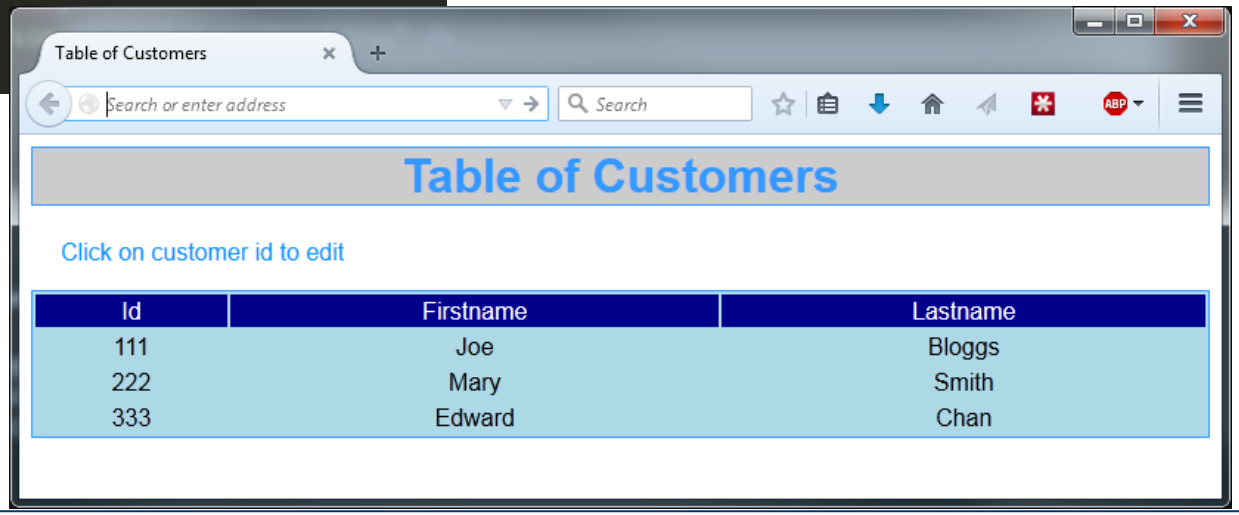
- Hypertext Markup Language (HTML)
  - Markup language used to define a web page
- Cascading Style Sheets (CSS)
  - Control appearance of an HTML document
- JavaScript (JS)
  - Scripting language that enable interactivity in HTML documents
- Extensible Markup Language (XML)
  - Markup language used to transport data between web services

For more info www.w3schools.com (but after you've finished EXAMS and NOT before!)

Web page = HTML document

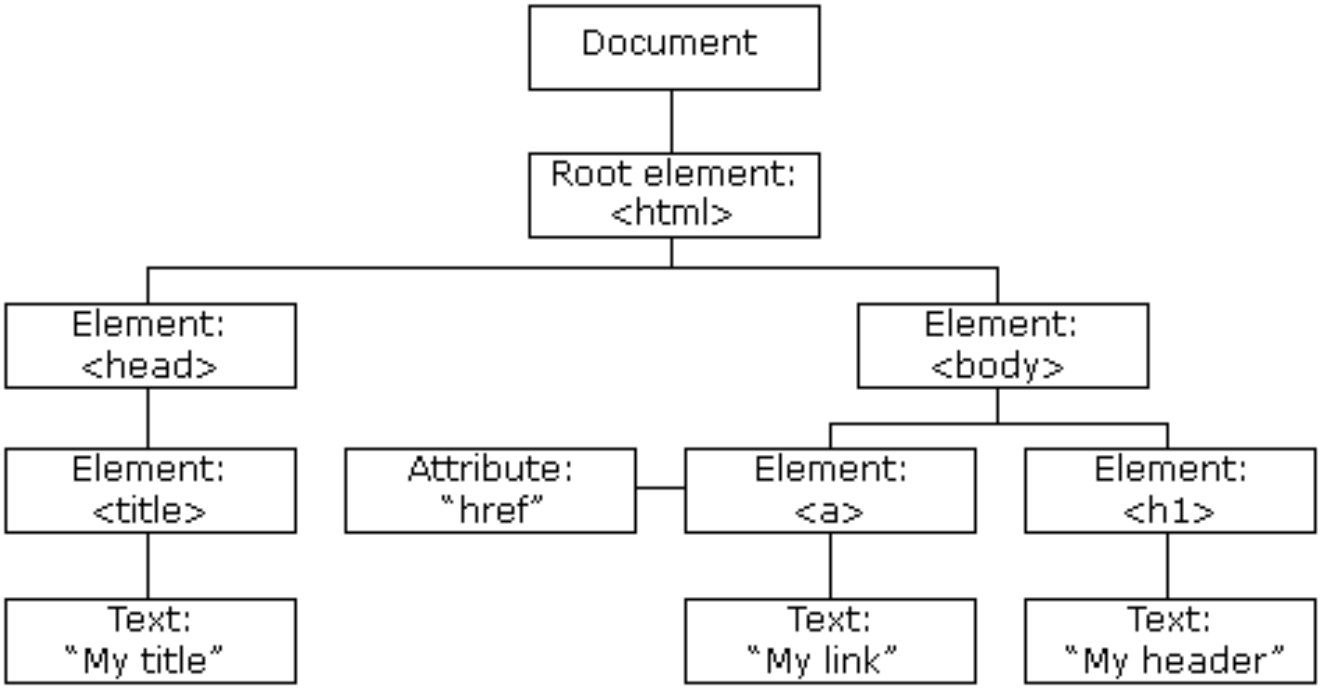
- a structured file of elements defined by HTML tags
- interpreted by web browser for display

```
<head>
    <title>Table of Customers</title>
    <link rel="stylesheet" href="simple.css" type="text/css" />
</head>

<body>
    <h1>Table of Customers</h1>
    <p>Click on customer id to edit</p>
    <table>
        <thead>
            <tr><td>Id<td>Firstname<td>Lastname</tr>
        </thead>
        <tr><td>111<td>Joe<td>Bloggs</tr>
        <tr><td>222<td>Mary<td>Smith</tr>
        <tr><td>333<td>Edward<td>Chan</tr>
    </table>
</body>

```



Structure of an HTML document

- elements are structured as a tree (one web page = one tree)
- divided into a HEAD and a BODY
- the BODY is what you see displayed in the browser
- BODY is divided into elements such as headings, paragraphs, tables, lists …



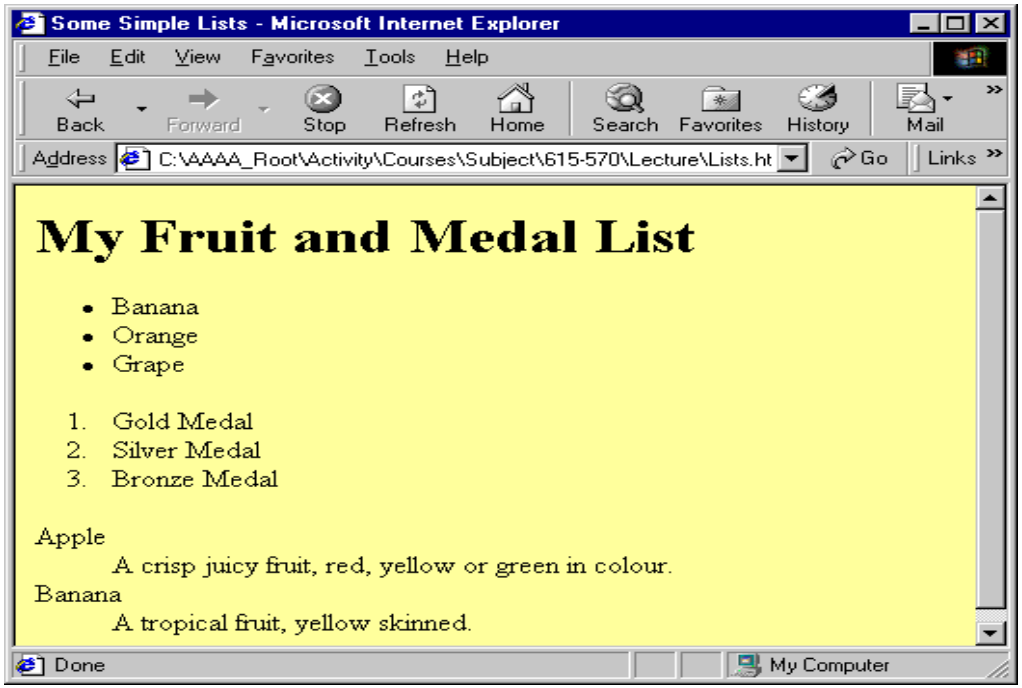
picture source: W3 Schools

Important HTML elements

<HEAD> … </HEAD>	• document header.
<BODY> … </BODY>	• document body
<H1> … </H1>	• Heading type 1
<H6> … </H6>	• … to Heading type 6.
<P> … </P>	• paragraph.
<TABLE>	• table
<TR>	• table row
<TD>	• table data
<UL>	• list
<LI>	• list item

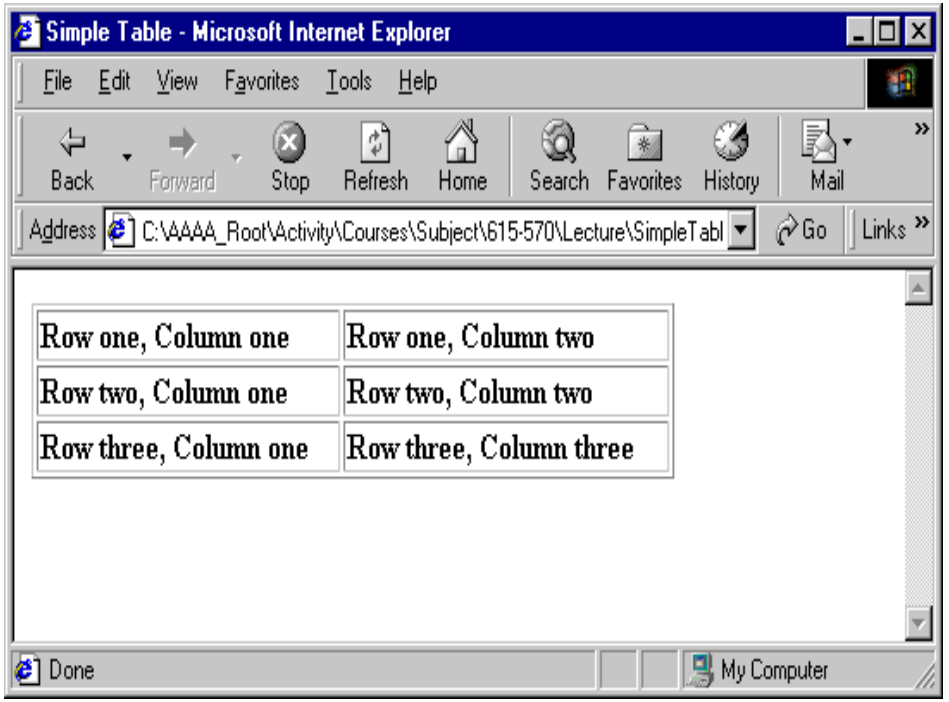
HTML list

```
<HTML> <HEAD> <title>Some Simple Lists</title> </HEAD>
<BODY bgcolor="#FFFF99">
<H1>My Fruit and Medal List </H1>
<UL>
  <LI>Banana</LI>
  <LI>Orange</LI>
  <LI>Grape</LI>
</UL>
<OL>
  <LI>Gold Medal</LI>
  <LI>Silver Medal</LI>
  <LI>Bronze Medal</LI>
</OL>
<DL>
  <DT>Apple
    <DD>A crisp juicy fruit, red, yellow or green in colour.
  <DT>Banana
    <DD>A tropical fruit, yellow skinned.
</DL>
</BODY> <HTML>
```



HTML table

```
<HTML>  <HEAD> <TITLE>Simple Table</TITLE>
<html>
<head>
<style>
table, th, td
{ border: 1px solid black;}
</style></HEAD>
<BODY>
<TABLE>
  <TR>
    <TD>Row one, Column one</TD>
    <TD>Row one, Column two</TD>
  </TR>
  <TR>
    <TD>Row two, Column one</TD>
    <TD>Row two, Column two</TD>
  </TR>
  <TR>
    <TD>Row three, Column one</TD>
    <TD>Row three, Column three</TD>
  </TR>
</TABLE>  </BODY> </HTML>
```



HTML form

- Forms allow users to input data to a web page
- The web server process the user's input using the file named in the 'action' attribute.

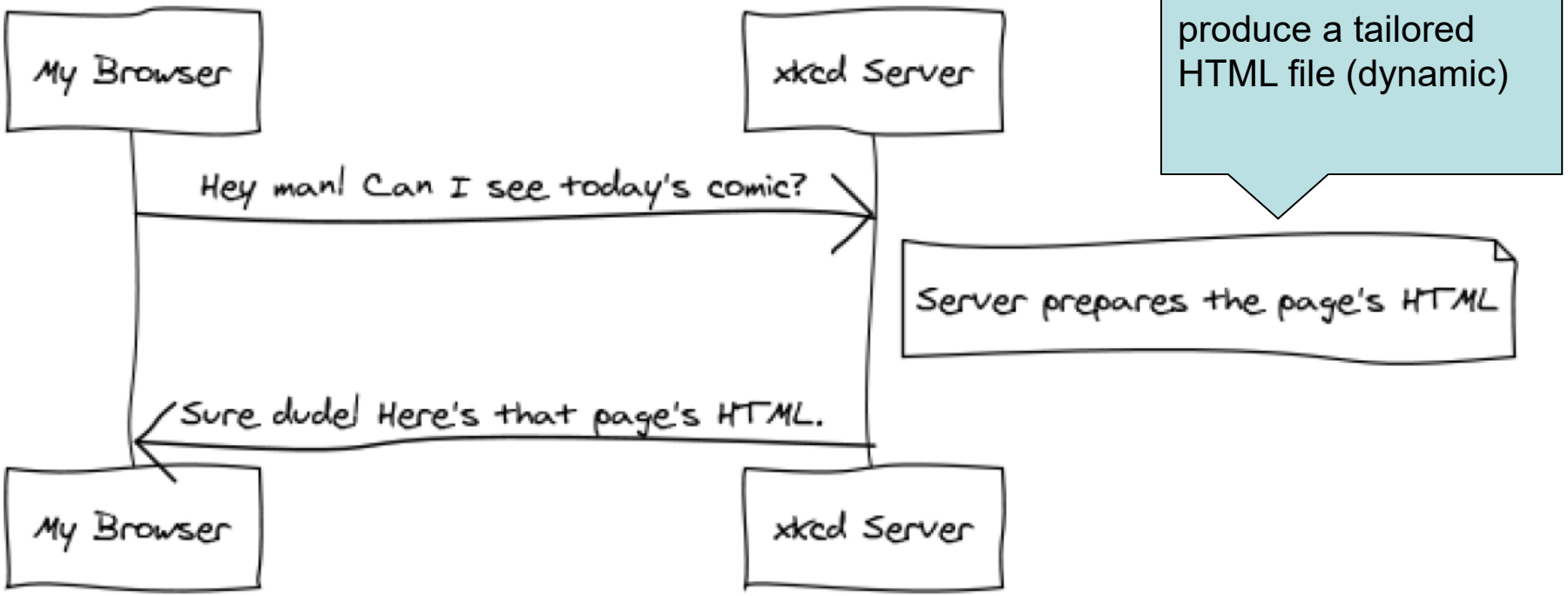
```
<form action = "buy.pl" method="post">

<p> What would you like to buy? <br>
<input type="text" name="product">

<p> How many? <br>
<input type="text" name="quantity"> <br>

<input type="submit" value="Do e-Commerce">
</form>
```

Example HTML form

browser displays form, sends input data to a script called 'buy.pl'

HTTP: how HTML documents move

- User wants to see a web page
- Types URL into browser
- Browser fetches page from server and displays it



could simply load a pre-prepared HTML file (static), or query the database to produce a tailored HTML file (dynamic)

picture source: Symfony Book

THE UNIVERSITY OF
MELBOURNE

Web App Examples

Static vs Dynamic web pages

- STATIC web page
  - the URL identifies a file on the server's file system
  - server fetches the file and sends it to the browser
  - the file contains HTML
  - browser interprets the HTML for display on screen
- DYNAMIC web page
  - URL identifies a program to be run
  - web app runs the program
  - program typically retrieves data from database
  - elements such as TABLE, LIST are populated with data
    - web app uses LOOPS to fill the contents of TABLEs and LISTs.
    - e.g. SELECT * FROM Product; (returns a set of product entities)
    - FOR p IN ProductList, print a row in HTML table

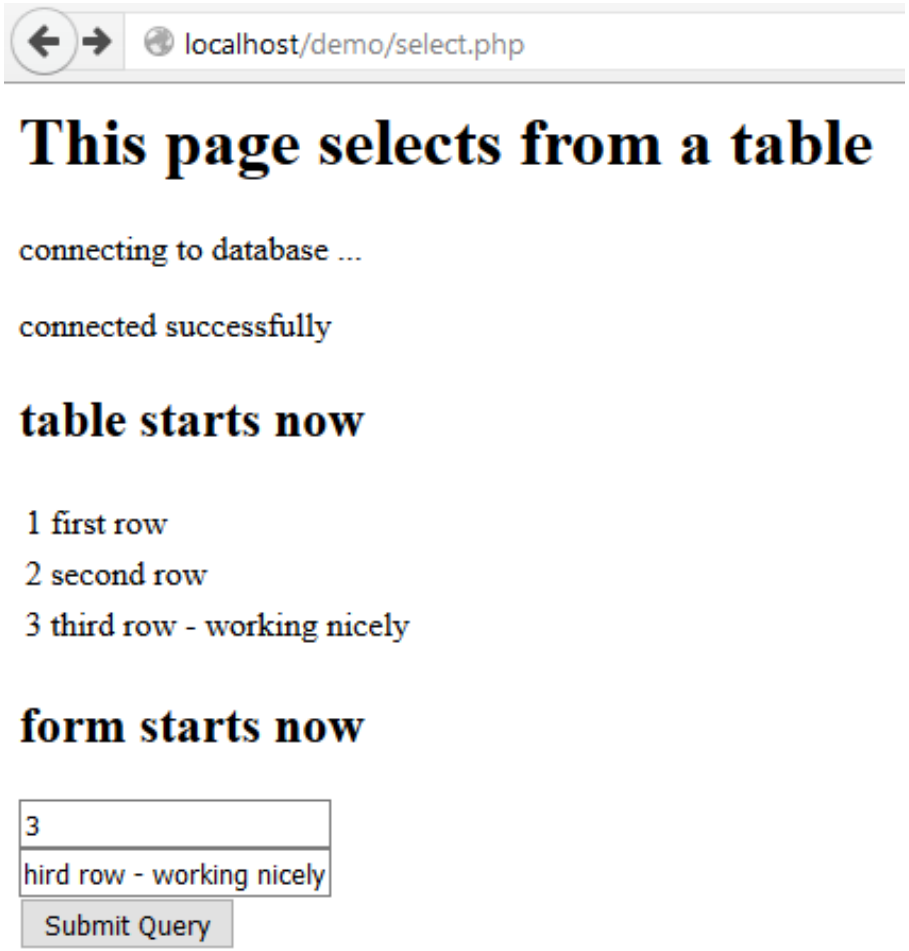
Simple web app using PHP and SQL

- program logs into db
- selects all rows from database table
- displays them inside an HTML table

```php
<?php

print '<h1> This page selects from a table </h1>';

print '<p> connecting to database ... </p>';
// connect to server, select database
$link = mysql_connect('localhost', 'root', '')
    or die('Could not connect: ' . mysql_error());
print '<p> connected successfully </p>';
mysql_select_db('webappdemo') or die('could not select database');

// perform SQL query
$query = 'SELECT * FROM mytable';
$result = mysql_query($query) or die('Query failed: ' . mysql_error());

print '<h2> table starts now </h2>';

// print results in an HTML table
print "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    print "\t<tr>\n";
    foreach ($line as $col_value) {
        print "\t\t<td>$col_value</td>\n";
    }
    print "\t</tr>\n";
}
print "</table>\n";

```

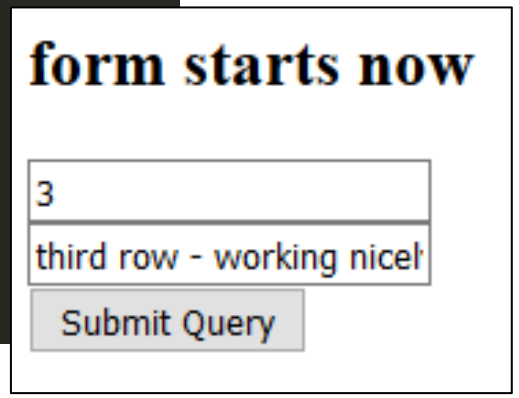
Simple web app: *select*

```php
<?php

print '<h1> This page selects from a table </h1';

print '<p> connecting to database ... </p>';
// connect to server, select database
$link = mysql_connect('localhost', 'root', '')
    or die('Could not connect: ' . mysql_error(
print '<p> connected successfully </p>';
mysql_select_db('webappdemo') or die('could not

// perform SQL query
$query = 'SELECT * FROM mytable';
$result = mysql_query($query) or die('Query fai

print '<h2> table starts now </h2>';

// print results in an HTML table
print "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL
    print "\t<tr>\n";
    foreach ($line as $col_value) {
        print "\t\t<td>$col_value</td>\n";
    }
    print "\t</tr>\n";
}
print "</table>\n";
```

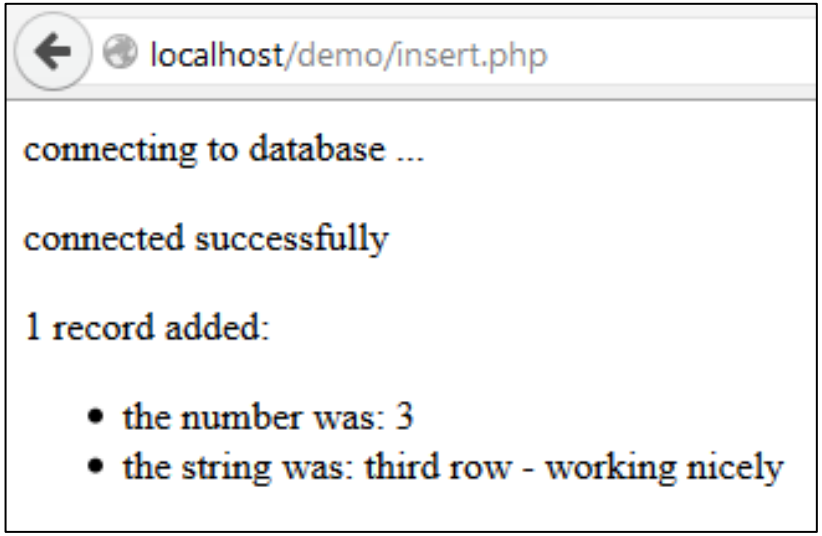


Simple web app: *insert*

```
37  print '<h2> form starts now </h2>';
38
39  // display a form for entering data
40  print '<form action="insert.php" method="post">';
41  print '<input type="text" name="number" value="type a number" />
42  print '<input type="text" name="string" value="type a string" />
43  print '<input type="submit" value="send to database" />';
44  print '</form>';
45  ?>
```




```
1   <?php
2
3   print '<p> connecting to database ... </p>';
4   // connect to server, select database
5   $link = mysql_connect('localhost', 'root', '')
6       or die('Could not connect: ' . mysql_error());
7   print '<p> connected successfully </p>';
8   mysql_select_db('webappdemo') or die('could not select database');
9
10  // form the INSERT statement from the user's input
11  $sql="insert into mytable values ('$_POST[number]','$_POST[string]')";
12
13  // run the INSERT statement
14  if (!mysql_query($sql,$link))
15    die('Error: ' . mysql_error());
16
17  // print friendly message
18  print "<p> 1 record added:  </p>";
19  print "<ul>";
20  print "<li>the number was: " . $_POST['number'];
21  print "<li>the string was: " . $_POST['string'];
22  print "</ul>";
23
24  // close connection to database
25  mysql_close($link);
26
27  ?>
```



Problems with old-style web apps

```
30  $sql = "insert into EVENT values (null, null, 'L', '" . $_SESSION["thisClient"] . "', 'logged in')";
31  mysql_query($sql);
```

- Placing “raw” SQL inside PHP/HTML files
  - Mixes presentation, business, database logic
  - Hard to maintain when things change
  - Want separation of concerns e.g. MVC
- Lots of reinvention of wheels
  - each dev writes their own solution to common features
  - e.g. login security, presentation templates, database access
- Increasing variety of clients e.g. phones and tablets
  - Manually program for different platforms
- => web application frameworks
  - examples: Ruby on Rails, .Net, Symfony, AngularJS, Django

Web Services

- The WWW allows humans to access remote databases
- Web Services allow *computers* to access remote databases
- 2 major approaches: SOAP and REST
  - Simple Object Access Protocol
  - Representational State Transfer
- structured data usually returned in XML or JSON format
- REST nouns are resources, addressed via URIs
- REST verbs correspond to DML statements
- GET (select), POST (insert), PUT (update), DELETE (delete)
- Try this example web service
  https://www.googleapis.com/books/v1/volumes?q=quilting

XML and JSON data formats

- used by web services for data exchange

- XML
  eXensible Markup Language
- JSON
  JavaScript Object Notation

Source: www.w3school.org

The following JSON example defines an employees object, with an array of 3 employee records:

JSON Example

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

The following XML example also defines an employees object with 3 employee records:

XML Example

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

What is examinable

- Identify the limitations of SQL
- Advantages and Disadvantages of Stored Procedures
- Distribution of Processing Logic
- Database Architectures
- Web languages
- Web architecture
- HTML elements
- How static and dynamic web pages work (high level)

Next Time

- Data Warehousing

THE UNIVERSITY OF MELBOURNE

11. Applications & Web Applications

Dr Greg Wadley, David Eccles