Chapter 2: Interactive Web Applications

2.1 Interactivity and Multimedia in the WWW architecture

2.2 Client-Side Multimedia in the Web (Example HTML5)

2.3 Interactive Server-Side Scripting (Example PHP)

2.4 Data Storage in Web Applications (Example Database Access in PHP)

2.5 Integrated Server/Client-Side Scripting (Example jQuery/AJAX)
Database Management Systems: A Quick Reminder

• Database:
  – Structured collection of data items
  – Stored persistently
  – Provides access to a common data pool for multiple users

• Database Management System (DBMS):
  – Collection of programs for administration and usage of a database
  – Various base models for DBMS:
    » Old: network model, hierarchical model
    » Dominant: relational model
    » Alternative: object-oriented model

• Relational databases:
  – Good methodological support for design of data schema
  – Standardized language interface SQL (Structured Query Language)
Prerequisites and Basic Architecture

Database server

Database 1
- Table 1.1
- Table 1.2

Database 2
- Table 2.1
- Table 2.2

Administration software

User programs

Other Servers (e.g. Web Server)
MySQL

- Open source software system
  - Frequently used also in commercial context
  - [www.mysql.com](http://www.mysql.com)
- Software package providing:
  - Database server (mysqld)
  - Administration program (mysqladmin)
  - Command line interface (mysql)
  - Various utility programs
- Communication between programs on local host: *socket* interface
  - Bidirectional data stream exchange between programs
  - Similar to files
Before Creating Anything in the Database...

- Using a database requires careful *information design*.
- Which are the data to be stored?
- Are there existing data to connect to?
- What is the *schema* of the data to be stored?
  - E.g. Entity-Relationship diagrams as a tool
  - Transformation into relational database schema (table design)
- Once a database is filled with data and in use, it is difficult to modify!
  - Database schema design has to be carried out with great care!
- Most important rule: Avoid redundant storage of information
  - But keep performance in mind...
Creating Database Tables (1)

• Prerequisites:
  – Database server running
  – Socket connection between programs intact
  – User accounts with adequate privileges known

• First step: Create `database`
  – Container for many tables
  – Requires special privileges
  – Example SQL:
    ```
    create database music;
    ```

• Second step: Choose used `database`
  – Sets the context for further interactions
  – Example SQL:
    ```
    use music
    ```
Creating Database Tables (2)

• Third step: Create **tables**
  – According to earlier design
  – Each table should provide a unique identifier (**primary key**)
  – SQL Example:
    ```sql
    create table mysongs (code VARCHAR(5), title VARCHAR(20), artist VARCHAR(20), album VARCHAR(20), runtime INT);
    ```
  – Further steps: Defining keys, indices etc.

• Fourth step: Fill tables with **data**
  – Simplest case: Individual SQL commands
  – Better: Import from structured data file
  – Frequent: Special programs for importing and creating data
  – SQL Example:
    ```sql
    insert into mysongs
    values ('1','One','U2','The Complete U2',272);
    ```
**SQL Monitor Output**

```sql
code | varchar(5) | YES | NULL | NULL |
title | varchar(20) | YES | NULL | NULL |
artist | varchar(20) | YES | NULL | NULL |
album | varchar(20) | YES | NULL | NULL |
runtime | int(11) | YES | NULL | NULL |
```

5 rows in set (0.00 sec)
## Queries with SQL

```
mysql> select * from mysongs;
+----------------+-----------------+-----------------+-----------+
| code | title         | artist          | album           | runtime |
+----------------+-----------------+-----------------+-----------+
| 1     | One            | U2              | The Complete U2 | 272      |
| 2     | In the End     | Linkin Park     | Hybrid Theory   | 216      |
| 3     | Wheel in the Sky | Journey    | Infinity        | 252      |
| 4     | Lady in Black  | Uriah Heep      | Lady in Black   | 281      |
| 5     | Smoke on the Water | Deep Purple | Machine Head   | 378      |
| 6     | Analog Man     | Joe Walsh       | Analog Man      | 243      |
+----------------+-----------------+-----------------+-----------+
6 rows in set (0.00 sec)
```

```
mysql> select title from mysongs where runtime>250;
+----------------+
| title          |
+----------------+
| One            |
| Wheel in the Sky |
| Lady in Black  |
| Smoke on the Water |
+----------------+
4 rows in set (0.00 sec)
```
Server-Side Databases, PHP and MySQL

- Special libraries for database access:
  - "Database extensions"
  - Generic for all database systems
- For specific database systems:
  - "Vendor specific database extensions"
- For MySQL:
  - MySQL-specific database extensions to PHP
Connecting to a Database from PHP

• First step: *Connect* to server
  – Establish a connection for data exchange between Web Server/PHP plugin and database server
  – Often local (sockets), if both programs on same machine
  – Requires hostname, (database) username, password
  – PHP function: `mysql_connect()`
    » Returns a link (resource) which can be used for `mysql_close()`

• Second step: *Select* a database
  – Corresponds to the SQL command `use`
  – Requires database name (and possibly link to server)
  – PHP function: `mysql_select_db()`
    » Returns Boolean result (success)
Example: Connecting to Database

```php
<?php

$link = mysql_connect('localhost','root','demopw')
    or die ('Could not connect: '.mysql_error());
echo 'Connected.<br/>';

mysql_select_db('music')
    or die ('Could not select db.');
echo 'DB selected.<br/>';

...

?>
```
Sending Database Queries from PHP

- Basic idea (as in all programming language/database integrations):
  - SQL queries are given as strings to library functions
- Most important function in MySQL extensions to PHP:
  `mysql_query()`
  - Requires SQL query as parameter (optionally link to server as 2nd param.)
  - "Query" includes also INSERT, UPDATE, DELETE, DROP (SQL)!
- Return value in case of SELECT, SHOW, DESCRIBE and similar:
  - Result set represented by resource value
  - Special functions to retrieve result data as PHP data structures
    - `mysql_num_rows()`
      » Number of rows returned
    - `mysql_fetch_array()`
      » Reads one row of data and transforms it into an array
      » Makes the next row available
Example: Reading Data From a Query in PHP

```php
<?php
...

$query = 'SELECT * FROM mysongs';
$result = mysql_query($query);

while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
    foreach ($row as $element) {
        echo $element;
        echo ', ';
    }
    echo('<br/>');
}
...
?>
```
Creating HTML Output From SQL Query (1)

<!DOCTYPE html>

<html>
<head>
  <title>Database table in HTML</title>
</head>

<?php
$link = mysql_connect('localhost','root','demopw')
or die ('Could not connect: '.mysql_error());
mysql_select_db('music') or die ('Could not select db.');?

$dbaccess_html.php
Creating HTML Output From SQL Query (2)

...<body>
<h1>The following table is retrieved from MySQL:</h1>
<table>
  <?php
  $query = 'SELECT * FROM mysongs';
  $result = mysql_query($query)
    or die ('Query failed'.mysql_error());
  while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
    echo '\t<tr><n';
    foreach ($row as $element) {
      echo '\t\t<td>'; echo $element; echo '</td><n';
    }
    echo '</tr><n';
  }
  echo '\t</tr><n';
  ?>
</table>
Creating HTML Output From SQL Query (3)

...  
```php
    mysql_free_result($result);
    mysql_close($link);
?>

</body>
</html>
Chapter 2: Interactive Web Applications

2.1 Interactivity and Multimedia in the WWW architecture

2.2 Client-Side Multimedia in the Web (Example HTML5)

2.3 Interactive Server-Side Scripting (Example PHP)

2.4 Data Storage in Web Applications (Example Database Access in PHP)

2.5 Integrated Server/Client-Side Scripting (Example jQuery/AJAX)

Literature:

http://jquery.com
jQuery

- See jquery.com
  - John Resig 2006
- JavaScript Library to assist with
  - traversal and manipulation of HTML through DOM
  - event handling
  - animations
  - Simple AJAX applications (see later)
- Current versions: 1.10.2 and 2.0.3
  - Examples use 2.0.3
- jQuery is currently the most used JavaScript library
  - 25 Oct 2013: 56.7% of all Websites, 92.4% market share in JS libraries
    (see http://w3techs.com/technologies/overview/javascript_library/all)
- Further libraries build on jQuery (e.g. jQueryUI)
- jQuery is essentially on large JavaScript file
  - included locally or through a delivery network of servers
Using jQuery

• Include the library into any file where jQuery is to be used
  – Locally: `<script type="text/javascript">jquery.js</script>`
  – From jQuery Web site or through various Content Delivery Networks
• jQuery is accessible as a global function and as an object instance
  – Function “jQuery”, abbreviated as “$”
• jQuery includes “Sizzle” engine to traverse and manipulate DOM trees
  – Frequent pattern: `$ (selector expression)`
• jQuery provides additional utility functions
  – Frequent pattern: `$ .fname(parameters)`
• jQuery supports event handlers
  – Frequent pattern: `DOMObject .eventname(function)`
  – Convenient pattern: Using local anonymous functions
• jQuery should be executed after DOM tree is ready
  (not necessarily after loading all content)
  – Event handler for `ready` event
Event Handler for jQuery $ready$ Event

- Standard place to put jQuery code, in a script block at the end of page

```javascript
$(document).ready(
  function() {
    ... jQuery Code ...
  }
);
```
Example: Interactive Highlighting in Table

- Assuming HTML and CSS code for table (similar to above):

```html
<table>
  <thead>
    <tr>
      <th>#</th>
      <td>Title</td> ...
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>1</td>
      <td>One</td> ...
    </tr>
  </tbody>
</table>
```

```css
<style>
  table {...}
  th, td {...}
  thead {
    background-color: black;
    color: white;
  }
  tr.hilite {
    background-color: grey;
    color: white;
  }
</style>
```
jQuery DOM Selection

- Typical selector arguments for $( selector )
  - document
  - HTML element names
  - Identifiers (referring to HTML id attribute): #ident
  - CSS classes: .classname
  - Special filters: :filtername

- Path constraints: Space-separated list of selectors
  - Have to appear as (possibly indirect) successors in tree

- Selecting all table rows:
  - $( 'tr' )

- Adding event handler for hover event:
  - $( 'tr' ).hover(function() { ...hilite... });
  - hover: Same handler called on mouseenter and mouseleave event

- Does this select the appropriate parts of the page?
Detour: Anonymous Functions in JavaScript

- Named function, e.g. for event handler:
  ```javascript
  function handleHover() {
    ...hilite...
  }
  $( 'tr' ).hover(handleHover);
  ```

- Omitting the name of the function, giving function body at calling position:
  ```javascript
  $( 'tr' ).hover(function(){...hilite...});
  ```
jQuery DOM Manipulation

- jQuery provides functions to
  - modify attributes of HTML elements
  - modify CSS classes attached to HTML elements
  - add or remove parts of the DOM tree
  - retrieve HTML text from DOM tree
  - create DOM tree from HTML strings

- Good practice: Use CSS, assign styles dynamically with jQuery
  - Add or remove class:
    ```javascript
    object.addClass(class), object.removeClass(class)
    ```
  - Add class if not present, and remove class if present (toggle):
    ```javascript
    object.toggleClass(class)
    ```

- Example:
  ```javascript
  $('#mysongs tbody tr').hover(function() {
      $(this).toggleClass("hilite");
  });
  ```
  `jq_table2.html`
Example: Extending HTML Table Using jQuery

- Make rows of the table selectable by adding a checkbox:
  - Add a new column to the table
  - Add an appropriate head entry in table head row
  - Add checkbox data cells to all table body rows
- jQuery code for table head:
  ```javascript
  $('#mysongs thead tr').
  append(''
      <th>Select</th>
  ');
```
- jQuery code for table body:
  ```javascript
  $('#mysongs tbody tr').
  append(''
      <td style="text-align: center">
        <input/ type="checkbox">
      </td>
  ');
```
Restructuring jQuery Code

- Good practice: Store result set of complex selection and re-use it
- Concepts from functional programming for code optimization
  - Functions as objects, passed as parameters
  - E.g. `collection.each(fn)`: applies function `fn` to all objects contained in `collection`
- Example (using tree navigation and tests for HTML elements in DOM):

```javascript
$('#mysongs tr').each(function(){
    if ($(this).parent().is('thead'))
        $(this).append('<th>Select</th>');</p>
    if ($(this).parent().is('tbody'))
        $(this).append(''
            <td style="text-align: center">
                <input/ type="checkbox">
            </td>
        ');

    $(this).hover(function(){
        $(this).toggleClass('hilite');
    });
});
```
Method Chaining

• In analogy to composition functional for functions:
  – using adequate result types, functions are easy to be used in *chains*

• jQuery: Most functions return an object compatible to the object on which
  the function was called

• Simple generic example:
  ```javascript
  $(...).addClass('classname').css(css_prop, css_value);
  ```

• Executing another jQuery query on result set:
  ```javascript
  collection.find('selector');
  ```

• Running example:
  ```javascript
  $(this)
    .append(''
      <td style="text-align: center">
        <input/ type="checkbox"></td>'
    .find(':checkbox')
    .change(event handler for change event);
  ```
Example: Highlighting Selected Rows in Table

```javascript
.find(':checkbox').change(function() {
    if ($(this).prop('checked')) {
        $(this).parents('tr').addClass('checked');
        numCheckedRows++;
    } else {
        $(this).parents('tr').removeClass('checked');
        numCheckedRows--;
    }
});
```

`parents(element_type)`: moves upwards in the tree and selects all elements of given `element_type`
Animations in jQuery

• jQuery enables time-dependent transitions
  – between CSS property values
  – adjustable in duration and linearity (using non-linear “easing” in/out)
• Animations fit into method chains easily
• Generic animation method: `animate()`
• Animations fit into method chains easily
• Shortcut methods for frequent animations:
  – `show(speed), hide(speed)` for DOM elements
  – simple parameter `speed` with values `slow, normal, fast`
• Example:
  
  ```javascript
  if (numCheckedRows==0) $('#btn').show("slow");
  if (numCheckedRows==1) $('#btn').hide("slow");
  ```
Combining PHP, Database Access, jQuery

- jQuery code as part of server page in PHP/MySQL setting
  - Mostly fixed jQuery/JavaScript within HTML page
  - Theoretically possible: Computation of jQuery/JavaScript code within PHP

```html
<body>
  <h1>The following table is retrieved from MySQL:</h1>
  <div style="width: 600px">
    <table id="mysongs" style="width: 600px">
      <thead>...
      <tbody>
        <?php
          $query = 'SELECT * FROM mysongs';
          $result = mysql_query($query) ...;
        ...
        ?></tbody>
      </table>
    </div>
    <input id='btn' type='button' value='...' />
  </body>
  <script src="jquery.js"></script>
  <script>
    $( document ).ready(function() { ... })
  </script>
```
Selecting Information Using jQuery/DOM

- Example: Get the IDs of all checked table rows
  - For instance to put them into a shopping cart

```javascript
$('#btn').click(function() {
    var selIds = $('#mysongs input:checked').map(function() {
        return $(this).parents('tr').children().first()
    })
});
```

*map* functional
(also from functional programming):
Applying a function pointwise to a collection
Sending Selected Data to Server

• HTTP traditional way:
  – Filling a form
  – Sending form data (URL-encoded or multipart-encoded)
  – Data has to be key-value pairs
• New forms may evolve (see next lecture)
• Seen from a jQuery perspective:
  – Sending a request is an option in event handling of input elements (here: buttons)
  – “AJAX” can be used for sending data without thinking about HTML forms
Asynchronous JavaScript + XML (AJAX)

- James Garrett 2005:
- Catchy name for an idea which was in use already at the time:
  - Google Suggest
  - Google Maps
- Basic idea:
  - Loading data from server is decoupled from changes in the presentation
- Advantages:
  - User can interact fluidly with the application
  - Information from server is fetched at regular intervals - display can always stay up-to-date
- AJAX is not a technology, it is a combination of known technologies
  - XHTML, CSS, DOM, XML, XSLT, JavaScript, XMLHttpRequest
- There are AJAX-like applications which use neither JavaScript nor XML
  - E.g. using Flash and querying servers in the background
Classical Synchronous Web Application Model
Asynchronous Web Application Model
AJAX and Client-Side Scripting

- AJAX applications are programs executed in the Web browser
  - Require a runtime environment
  - Usually programmed in JavaScript
- AJAX applications need to modify or construct HTML to be displayed in the browser
  - Requires access to loaded/displayed HTML
  - *Domain Object Model* (DOM) is used for accessing and manipulating page content
- jQuery integrates AJAX support
  - Sending requests
  - Evaluating results
    » Manipulating DOM
Sending Request Using jQuery

```javascript
$('#btn').click(function()

    var selIdsText = $('#mysongs input: checked').
            map(function(){
               return $(this).parents('tr').children().first()
            }).
        text();

    $.ajax({
      type: 'POST',
      url: 'serverDummy.php',
      data: {selection: selIdsText}
    });
})
```

`text()`:
Creates text from DOM tree
(much more flexible variants)

`dbajax_jquery.php`
serverDummy.php

```php
<?php
    $value = $_REQUEST['selection'];
    $file = fopen("dummyData.txt", "w");
    if ($file) {
        fputs($file, "selection: " . $value . "\n");
        fclose($file);
    }
?>
```

- Of course, in a realistic setting, data received by the server is processed by operating background systems
  - Here, may want to create a table in MySQL referring to `mysongs` table
See the Overall Picture?

Browser → User

Open URL (php) → Server

HTML+JS/jQuery

User ← Browser

AJAX request (jQuery) → Server

Server → DB

write

DB → Server

read
Asynchronous Requests Returning a Result

User → Browser

Open URL (php) → Server

HTML+JS/jQuery → Browser

AJAX request (jQuery) → Server

return data (e.g. JSON) → Browser

Server → DB

read → Server

read → DB

User ← Browser
jQuery AJAX Requests with Result

• jQuery `ajax` method
  – (and shorthands `get` and `post`)
  – creates a request to server
• Standard arguments, like:
  – `url`: URL address to send request to
  – `settings`: Key-value pairs
• Example settings:
  – `dataType`: Kind of data expected for answer (e.g. xml, json, html)
  – `success(data, status)`:  
    JavaScript function to be called in case of successful server response
  – `error(requestObj, message, errorObject)`:  
    JavaScript function to be called in case of server response indicating errors
Paradigm Shift for Servers

• Traditional Web server:
  – Retrieves, computes and sends HTML data mainly
• AJAX-oriented server:
  – Constructs plain string, XML or JSON data as response to client
• Methods, toolkits, libraries remain the same (essentially):
• E.g.:
  – Client requests HTML5/jQuery/PHP page and interacts with it
  – Server sends JSON response for jQuery-initiated request
  – Client (JavaScript/jQuery) has to deal with response format
    » e.g. parseJSON method