# Chapter 2: Interactive Web Applications

1. Interactivity and Multimedia in the WWW architecture
2. Client-Side Multimedia in the Web (Example HTML5)
3. Interactive Server-Side Scripting (Example PHP)
4. Data Storage in Web Applications (Example Database Access in PHP)
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)

• Document-oriented databases:
  – Based on document trees, APIs for queries (“NoSQL”)
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
- 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) primary key, 
                        title VARCHAR(20), artist VARCHAR(20), 
                        album VARCHAR(20), runtime INT);
    ```

- 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
mysql> describe mysongs;
+-------------------+--------+------+-----+--------------------------+-----------+
| Field             | Type   | Null | Key | Default | Extra     |
+-------------------+--------+------+-----+--------------------------+-----------+
| code              | varchar(5) | NO   | PRI | 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 (0.01 sec)
```
Queries with SQL

```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)
```

```sql
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

• Libraries for database access:
  – "Database extensions" for server-side scripts
  – Depend on type of database
  – May require additional installations

• For PHP and MySQL:
  – MySQL database extensions usually pre-installed
  – Three different APIs for PHP
    » Original MySQL API (deprecated since PHP 5.5)
    » MySQL Improved Extension (mysqli) — used here
    » PHP Data Objects (PDO) interface
Connecting to a Database from PHP

• Steps:
  – Original SQL: First connect to server, then select (use) a database
  – Improved PHP API: Combined into one step
• **Connect** to server and **select** a database
  – Establish a connection for data exchange between Web Server/PHP plugin and database server
  – Local communication (through socket), if both programs on same machine
  – TCP/IP connection to remote server is possible
  – Requires hostname, (MySQL) username, password, database name
  – PHP: Create a new mysqli object
    » Returns an object which can be used for further operations
• Performance optimization:
  – Persistent connections and connection pools
Example: Connecting to Database

```php
$db = new mysqli('localhost','root','demopw','music');

if ($db->connect_error) {
    die('Failed to connect: '.$db->connect_error);
}

echo 'Connected to server and DB selected.<br/>';
...
?>
```
Sending Database Queries from PHP

• Basic idea (in all programming language/database integrations):
  – SQL queries given as strings to library functions

• MySQL/PHP:

  `query()` method of `mysqli` object
  – 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 as `mysqli_result` object
  – Special functions and variables to process result data (examples):
    – `$num_rows()`
      » Number of rows
    – `fetch_assoc()`
      » Reads one row of result data and returns it as associative array
      » Makes the next row available
Example: Reading Data From a Query in PHP

```php
<?php    ... $db = ... connecting, selecting ...
$query  = 'SELECT * FROM mysongs';
$result = $db->query($query);
if (!$result) {
    die('Query failed: '.$db->error);
}
while ($row = $result->fetch_assoc()){
    foreach ($row as $element) {
        echo $element;
        echo ', ';
    }
    echo("<br/>");
}
...
?>
```

---

`dbaccess.php`
Creating HTML Output From SQL Query (1)

<!DOCTYPE html>

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

<?php
    $db = new mysqli('localhost','root','demopw','music');
    if ($db->connect_error) {
        die('Failed to connect: '.$db->connect_error);
    }
?>

---

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 = $db->query($query)
        or die ('Query failed'.$db->error);
    while ($row = $result->fetch_assoc()) {
        echo "\t<tr><tr>
            foreach ($row as $element) {
                echo "\t\t<td>";
                echo $element;
                echo "</td><tr>\n";
            }
        echo "\t</tr><tr>\n";
    }
?>

</table>
Creating HTML Output From SQL Query (3)

...  
<?php  
    $result->free();  
    $db->close();  
?>  

</body>  
</html>
Outlook: Using MongoDB (Document-Oriented)

Heinrichs-MacBook-Pro: hussmann$ mongo
MongoDB shell version: 2.6.5
> use music
switched to db music
> db.mysongs.insert({code:'1', title:'One', artist:'U2',album:'The Complete U2',runtime:272})
WriteResult({ "nInserted" : 1 })

...  
> db.mysongs.find({runtime: {$gt: 250}}, {title: true})
{ "_id" : ObjectId("5448042878b2c1f62e542dc4"),
  "title" : "One" }
{ "_id" : ObjectId("544804cf78b2c1f62e542dc5"),
  "title" : "Wheel in the Sky" }
{ "_id" : ObjectId("5448054978b2c1f62e542dc6"),
  "title" : "Lady in Black" }
{ "_id" : ObjectId("5448054e78b2c1f62e542dc7"),
  "title" : "Smoke on the Water" }
> quit()

JavaScript takes the role of SQL!
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:

O’Reilly 2014

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: 1.11.1 and 2.1.1
  - Examples use 2.1.1
- jQuery is currently the most used JavaScript library
  - 22 Oct 2014: 60.6% of all Websites, 94.4% 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 one 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: `DOMNode.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
  - executed when DOM tree has been loaded (*event handler*)

```html
<script src="jquery.js"></script>
<script>
    function runJQuery() {
        alert("run some jQuery code now");
    }

    $( document ).ready(runJQuery);
</script>
```

*jq_init0.html*
Using Anonymous Functions in JavaScript

```html
<script>
    function runJQuery() {
        alert("run some jQuery code now");
    }

    $( document ).ready(runJQuery);
</script>

Rewritten with anonymous event handler:

```html
<script>
    $( document ).ready(function() {
        alert("run some jQuery code now");
    });
</script>
```
Example: Interactive Highlighting in Table

• Assuming HTML and CSS code for table:

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

```
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

- Example: Handler hover event on table rows:
  - $( 'tr' ).hover(function() { ...hilite... });
  - hover: Same handler called on mouseenter and mouseleave event

- Does this select the appropriate parts of the page?
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:
    `object.addClass(class), object.removeClass(class)`
  - Toggle (add/remove) class:
    `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 column

• 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: Re-use selection results (optimization)
• Apply concepts from functional programming:
  – E.g. `collection.each(fn)`: applies function `fn` to all objects contained in `collection`

• Example:

```javascript
$('#mysongs tbody tr').each(function() {
    $(this).append(''
        <td style="text-align: center">
            <input type="checkbox">
        </td>
    ');  
    $(this).hover(function() {
        $(this).toggleClass('hilite');
    });
});
```
Method Chaining

- jQuery: Most functions return an object compatible to the object on which the function was called
- Create *method chains by function composition*
- Simple generic example:
  
  ```
  $(...).addClass('classname').
  css(css_prop, css_value);
  ```
- Executing another jQuery query on result set:
  
  ```
  collection.find('selector');
  ```
- Running example:
  
  ```
  $(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 (“easing” in/out)
- Generic animation method: `animate()`
- 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
  - jQuery/JavaScript sent from (PHP-enabled) Web server

```html
<body>
  <h1>The following table is retrieved from MySQL:</h1>
  <div style="width: 600px">
    <table id="mysongs" style="width: 600px">
      <thead>...<thead>
      <tbody>
        <php>
          $query = 'SELECT * FROM mysongs';
          $result = mysql_query($query) ...;
        </php>
      </tbody>
    </table>
    <input id='btn' type='button' value='...'></input>
  </div>
</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 selIdsTextArray = $('#mysongs input:checked').
        map(function(){
            return $(this).parents('tr').children().first().text()
        }).
    toArray();
    ...
})
```

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

• HTTP traditional *synchronous* way:
  – Filling a form, sending a request (GET or POST)
  – Request data: key-value pairs with simple value types
  – Response data: HTML
  – Waiting for response before updating page

• Modern *asynchronous* way ("AJAX"):
  – Sending a request from JavaScript
  – Request and response data:
    String encoding of data structures (e.g. JSON)
  – *Continue script in parallel to waiting for response*

• AJAX is easy with jQuery!
Sending Request Using jQuery

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

    var selIdsTextArray = $('#mysongs input:checked').map(...).toArray();
    var selIdsJson = JSON.stringify(selIdsTextArray);

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

});
```

`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
Asynchronous Requests Returning a Result

User ↔ Browser → Server

Open URL (php)

HTML+JS/jQuery

AJAX request (jQuery)

Server → DB

read

DB

return data (e.g. JSON)
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 (may contain JSON data)
- 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