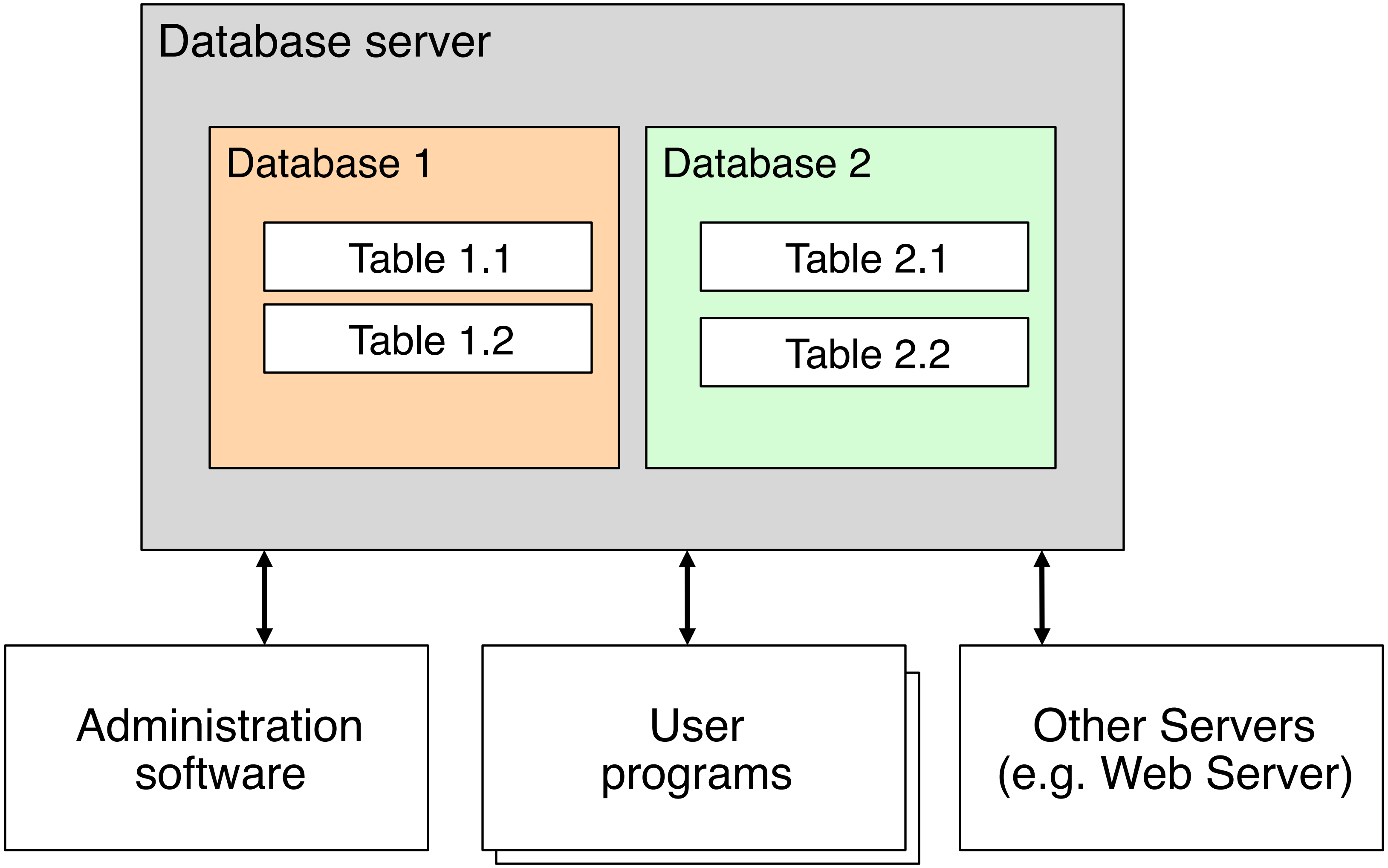
Chapter 2: Interactive Web Applications

Database Management Systems: A Quick Reminder

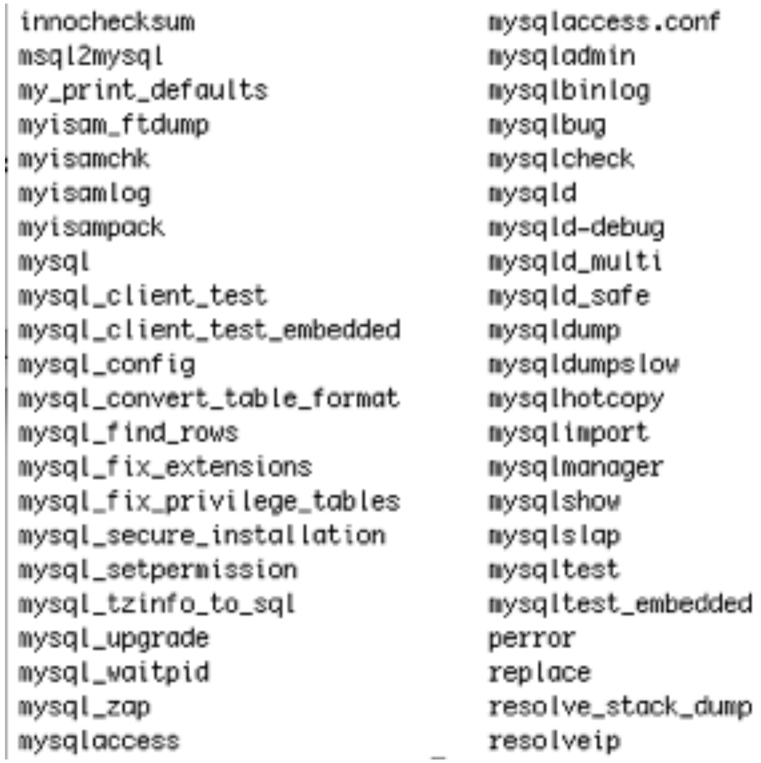
- Database:
  - Structured collection of data items
  - Stored persistently
  - Provides access to a common data pool for multiple users
- Database Management System (DBMS):
  - Collection of programs for administration and usage of a database
  - Various base models for DBMS:
    - Old: network model, 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



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



```
innochecksum                    mysqlaccess.conf
msql2mysql                      mysqladmin
my_print_defaults               mysqlbinlog
myisam_ftdump                   mysqlbug
myisamchk                       mysqlcheck
myisamlog                       mysqld
myisampack                      mysqld-debug
mysql                           mysqld_multi
mysql_client_test               mysqld_safe
mysql_client_test_embedded      mysqldump
mysql_config                    mysqldumpslow
mysql_convert_table_format      mysqlhotcopy
mysql_find_rows                 mysqlimport
mysql_fix_extensions            mysqlmanager
mysql_fix_privilege_tables      mysqlshow
mysql_secure_installation       mysqlslap
mysql_setpermission             mysqltest
mysql_tzinfo_to_sql             mysqltest_embedded
mysql_upgrade                   perror
mysql_waitpid                   replace
mysql_zap                       resolve_stack_dump
mysqlaccess                     resolveip
```

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)

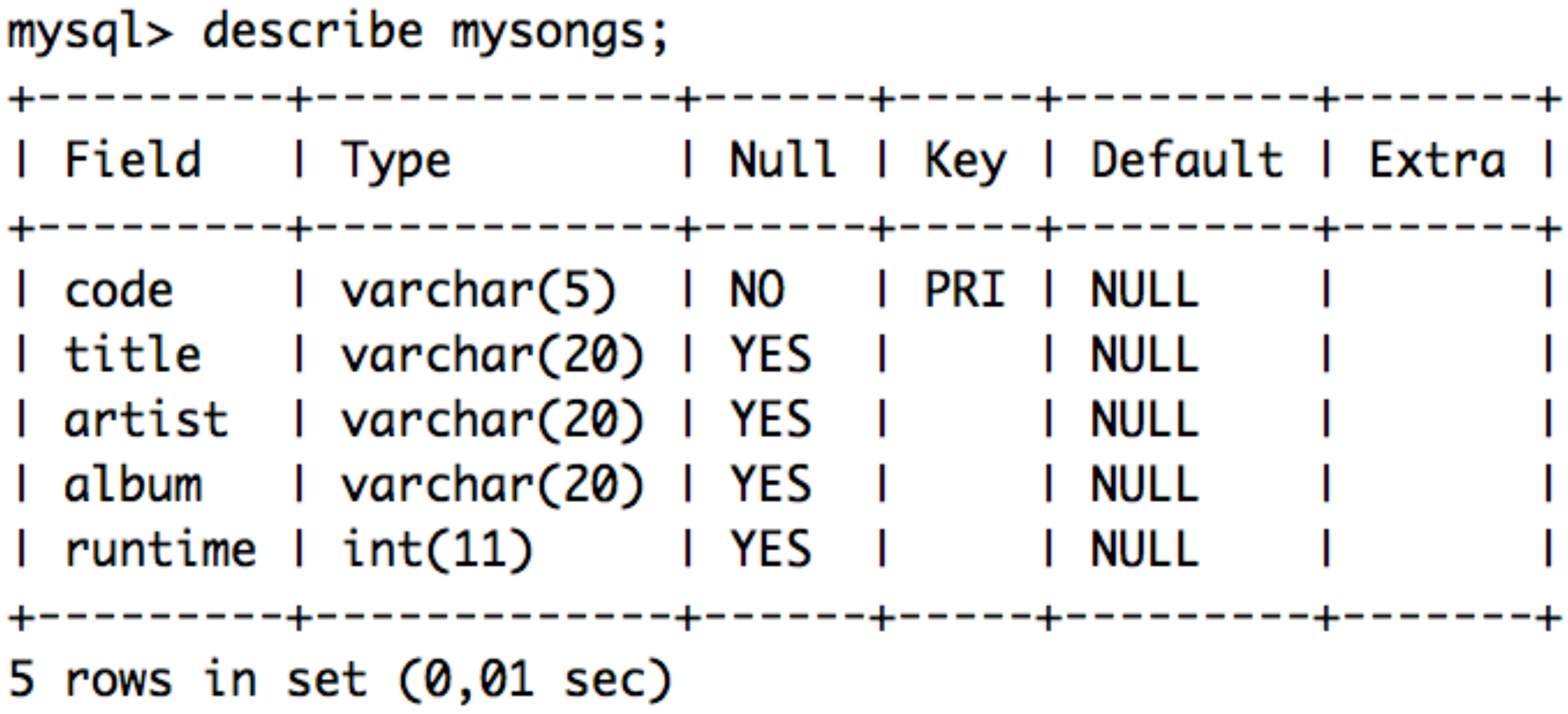
- Third step: Create ***tables***
  - According to earlier design
  - Each table should provide a unique identifier ***(primary key)***
  - SQL Example:

```
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:

```
insert into mysongs
  values ('1','One','U2','The Complete U2',272);
```

SQL Monitor Output



```
mysql> describe mysongs;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| code    | varchar(5)  | NO   | PRI | NULL    |       |
| title   | varchar(20) | YES  |     | NULL    |       |
| artist  | varchar(20) | YES  |     | NULL    |       |
| album   | varchar(20) | YES  |     | NULL    |       |
| runtime | int(11)     | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
5 rows in set (0,01 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

- 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

Excursus (1): Object-Orientation in PHP

- mysqli allows object-oriented programming
- Object-orientation:
  - everything is an object
  - objects encapsulate data --> logical grouping of variables and functions
  - objects have **attributes** and **methods**

- in PHP (file lecture.php):

```
class Lecture{
    var $title = "Online Multimedia";
    var $semester = "Winter 2015/2016";
    var $professor = "Prof. Dr. Heinrich Hussmann";
    var $guestLecturers = array();

    function addLecturer($name){
        $this->guestLecturers[] = $name;
    }
}
```

Excursus (2): Using Classes in PHP

- To use the class in another script, it needs to be imported:
  - `require("lecture.php");`
  - `require_once("lecture.php");`

- Afterwards, classes can be **instantiated** and become objects that we can assign to variables:

```
$mmn = new Lecture();
```

- Accessing member variables and methods is done using the **arrow notation**:

```
echo 'Semester' . $mmn->semester;

$mmn->addLecturer("Seitz");
foreach($mmn->guestLecturers as $lecturer){
    echo $lecturer;
}
```

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 (property)`**
    - » Number of rows in the result set
  - **`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   … $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 = $result->fetch_assoc()) {
                  echo "\t<tr>\n";
                  foreach ($row as $element) {
                        echo "\t\t<td>";
                        echo $element;
                        echo "</td>\n";
                  }
                  echo "\t</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

Literature:

D.S. McFarland: JavaScript and jQuery: The Missing Manual, 3rd ed., 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: 1.11.3 and 2.1.4
  - Examples use 2.1.1
- jQuery is currently the most used JavaScript library
  - 04 Nov 2015: 66.8% of all Websites, 95.6% 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: ***`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 places to put jQuery code:
  - in a script block at the end of page
  - executed when DOM tree has been loaded *(event handler)*

```
<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

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

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

Rewritten with anonymous event handler:

```
<script>
    $( document ).ready(
        function() {
            alert("run some jQuery code now");
        };
    );
</script>
```

jq_init1.html

Example: Interactive Highlighting in Table

- Assuming HTML and CSS code for table:

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

```
<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 **`$( `*`selector`*` )`**
  - **`document`**
  - HTML element names
  - Identifiers (referring to HTML **`id`** attribute): **`#`*`ident`***
  - CSS classes: **`:.`*`classname`***
  - Special filters: **`:`*`filtername`***
- Path constraints: Space-separated list of selectors
  - Have to appear as (possibly indirect) successors in tree
- Example: Handler **`hover`** event on table rows:
  - **`$( 'tr' ).hover(function() { ...`*`hilite`*`... });`**
  - **`hover:`** Same handler called on **`mouseenter`** and **`mouseleave`** event
- Does this select the appropriate parts of the page?

jq_table1.html

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:

```
$("#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:

```
$('#mysongs thead tr').
   append('
    <th>Select</th>');
```

- jQuery code for table body:

```
$('#mysongs tbody tr').
   append('
    <td style="text-align: center">
      <input/ type="checkbox">
    </td>');
```

jq_table3.html

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:

```
$('#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

```
.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*

jq_table4.html

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:

```
if (numCheckedRows==0) $('#btn').show("slow");
if (numCheckedRows==1) $('#btn').hide("slow");
```

- More animations: https://api.jquery.com/category/effects/

jq_table5.html

Combining PHP, Database Access, jQuery

- jQuery code as part of server page in PHP/MySQL setting
  - jQuery/JavaScript sent from (PHP-enabled) Web server

```
<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) ...;
...      ?>
        </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

```
$('#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

dbaccess_jquery.php

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

```
$('#btn').click(function(){
  var selIdsTextArray = $('#mysongs input:checked').
    map(...).toArray();
  var selIdsJson = JSON.stringify(selIdsTextArray);

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

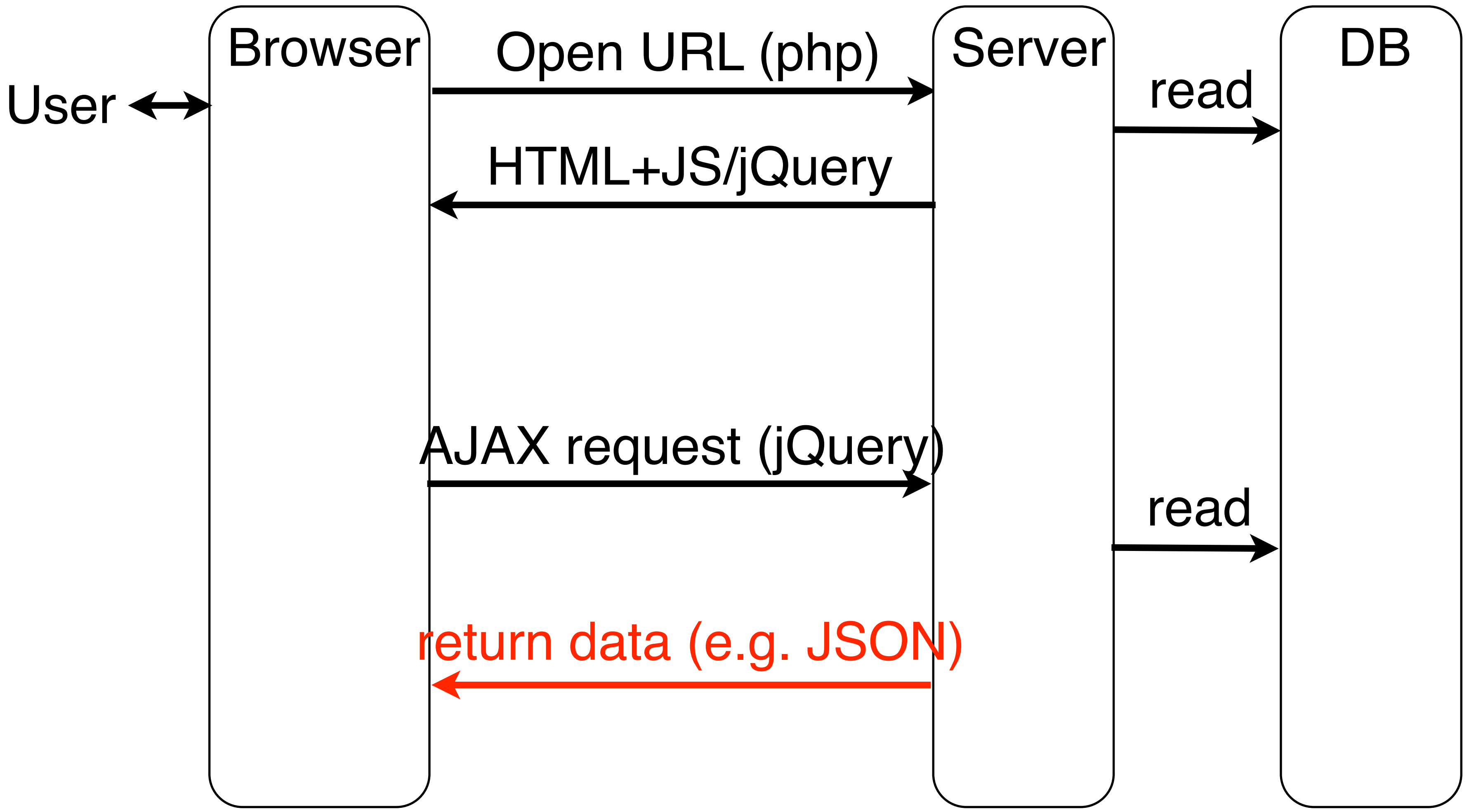
dbajax_jquery.php

serverDummy.php

```
<?php
   $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



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
  - **`contentType`**: Kind of data sent to the server
    often 'application/json'