

Datenbankprogrammierung mit PHP

PHP UND MYSQL

MySQL

- Relationales Datenbanksystem
 - Ursprünglich entwickelt von MySQL AB, jetzt Oracle
- Open-Source
 - Zusätzlich auch kommerzielle Enterprise-Version
- Sehr populär im Web-Umfeld
 - Nativer Support in PHP mit Apache
 - Konnektoren für viele weitere Plattformen
- Seit Version 5.x wird voller SQL3-Standard unterstützt
 - Sichten, Fremdschlüssel, Indizes, Trigger, ...
 - Allerdings abhängig von verwendeter Speicher-Engine

MySQL (2)

- MySQL-Server kann entweder als Anwendung oder als Dienst gestartet werden
 - Verbindung zu Server wird über TCP/IP hergestellt
 - Anmeldung (Benutzername, Passwort) ist erforderlich
 - Standardbenutzer für Administration: root
- Eine Server-Instanz kann mehrere Datenbanken verwalten
 - Nach erfolgreicher Verbindung zu Server muss Datenbank ausgewählt werden
 - Jede Datenbank kann beliebig viele Tabellen enthalten

Speicher-Engines

- Daten einer Tabelle werden jeweils von einer Speicher-Engine verwaltet
- Speicher-Engine wird pro Tabelle festgelegt
 - Ist für physikalische Speicherung der Daten einer Tabelle verantwortlich
- MySQL bietet Reihe von vordefinierten Engines
 - MyISAM
 - Standard bis Version 5.5.5
 - Optimiert für schnellen Datenzugriff und Volltextsuche
 - Keine Unterstützung für Fremdschlüssel und Transaktionen
 - InnoDB
 - Standard seit Version 5.5.5
 - Ausgerichtet auf Datenintegrität
 - Unterstützt Fremdschlüssel und Transaktionen
 - MEMORY
 - Speicherung von Tabellendaten im Hauptspeicher
 - ...
- Erweiterung des Servers durch eigens programmierte Engines möglich
 - Werden bei Bedarf dynamisch geladen

Sprachstruktur

- Abfragesprache entspricht weitgehend dem SQL-Standard
- Groß- und Kleinschreibung
 - Kein Unterschied bei Schlüsselwörtern
 - ACHTUNG: Bei Tabellennamen abhängig von Betriebssystem, da jede Tabelle in einer eigenen Datei gespeichert wird
- Benutzerdefinierte Bezeichner dürfen Sonderzeichen enthalten oder auch reservierte Wörter sein
 - Müssen in diesem Fall zwischen Rückwärtsakzentzeichen („backticks“) gestellt werden („identifier quoting“)
 - Beispiel: `SELECT id, `FROM` FROM `table #1 +asdf!`;`

Sprachstruktur (2)

- Zeichenketten unter einfachen oder doppelten Anführungszeichen
 - Reservierte Zeichen werden mit „\“ eingebaut
 - Anführungszeichen innerhalb einer Zeichenkette können auch verdoppelt werden
- Kommentare
 - Mit „#“ oder „--“ bis Zeilenende
 - Mit „/*“ und „*/“ innerhalb einer Anweisung

Datentypen

- Numerische Datentypen
 - BIT, TINYINT = BOOL, SMALLINT, INT, MEDIUMINT, BIGINT
 - FLOAT(n, d), DOUBLE(n, d), DECIMAL(n, d)
- Datum und Uhrzeit
 - DATE, DATETIME, TIME, YEAR(2 | 4)
 - TIMESTAMP
- Zeichenketten
 - CHAR(n), VARCHAR(n)
 - TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT
- Binärdaten
 - BINARY(n), VARBINARY(n)
 - TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB
- Spezielle Datentypen
 - ENUM('wert1','wert2','wert3',...)
 - SET('wert1','wert2','wert3',...)

Administrationswerkzeuge

- Kommandozeilenwerkzeug „mysql“
 - Direkter Zugriff auf Datenbanken über Kommandozeile mittels SQL
- MySQL Workbench
 - Rich-Client zur grafischen Verwaltung
- PHPMyAdmin
 - Web-basierter Client
 - Implementiert in PHP
 - Weit verbreitet
 - Standard-Werkzeug zur Administration von bei Provider gehosteten Datenbanken

Datenbanken verwalten

- Datenbank anlegen und verwenden

```
CREATE DATABASE levis;  
USE levis;
```

- Datenbank löschen

```
DROP DATABASE levis;
```

- Alle Datenbanken am Server anzeigen

```
SHOW DATABASES;
```

- Aktuell verwendete Datenbank abfragen

```
SELECT DATABASE();
```

Tabellen erstellen und löschen

- Anlegen mit CREATE TABLE
 - Für einzelne Felder werden Name und Datentyp angegeben
 - Primärschlüssel kann über PRIMARY KEY definiert werden
 - Zu verwendende Speicher-Engine kann angegeben werden
- Möglichkeit zum Abfragen von Metadaten
 - Alle Tabellen in der aktuellen Datenbank
 - Struktur einer bestimmten Tabelle
- Löschen mit DROP TABLE

```
-- create students table
CREATE TABLE students (
    id INT(11),
    name VARCHAR(255) NOT NULL,
    gender ENUM('M','F') NOT NULL,
    address VARCHAR(255),
    dateOfBirth DATE,
    PRIMARY KEY (id)
) ENGINE=InnoDB;

-- show all tables in database
SHOW TABLES;

-- display information about students
DESCRIBE students;

-- drop students table
DROP TABLE students;
```

Daten einfügen

- Einfügen einzelner Zeilen mit INSERT
- Einfügen von mehreren Datensätzen aus Textdatei mit LOAD DATA
 - Ein Datensatz pro Zeile
 - Tabulator als Trennzeichen zwischen Feldern
 - „\N“ steht für NULL

```
-- insert single students
INSERT INTO students VALUES (1,
    'Max Mustermann', 'M', NULL,
    '1985-03-30');
INSERT INTO students VALUES (2,
    'Susi Super', 'F', NULL, NULL),
(3, 'Marianne Mustermann', 'F',
    'Strebergasse 3',
    '1992-11-17');

-- load students from file
LOAD DATA LOCAL INFILE 'C:\list.txt'
    INTO TABLE students;
```

Fremdschlüssel und referenzielle Integrität

- Ab Version 5.x auch Beziehungen und referenzielle Integrität
 - Lösch- und Aktualisierungsweitergabe werden ebenfalls unterstützt
- Tabelle muss mit InnoDB-Engine verwaltet werden
- Für Fremdschlüsselfeld muss Index angelegt werden
- Beziehung wird über entsprechende Fremdschlüssel-Einschränkung hergestellt

```
CREATE TABLE lectures (  
    id INT(11) NOT NULL,  
    name VARCHAR(255) NOT NULL,  
    PRIMARY KEY (id)  
) ENGINE=InnoDB;
```

```
CREATE TABLE results (  
    lectureId INT(11) NOT NULL,  
    studentId INT(11) NOT NULL,  
    grade INT(11) NOT NULL,  
    PRIMARY KEY (lectureId, studentId),  
    KEY lectureId (lectureId),  
    KEY studentId (studentId),  
) ENGINE=InnoDB;
```

```
ALTER TABLE results  
ADD CONSTRAINT results_ibfk_1 FOREIGN KEY  
    (lectureId) REFERENCES lectures (id),  
ADD CONSTRAINT results_ibfk_2 FOREIGN KEY  
    (studentId) REFERENCES students (id)  
ON UPDATE CASCADE ON DELETE CASCADE;
```

Daten abfragen

- Abfrage von Daten mit SELECT-Anweisung
 - Spezifizieren von Abfrage-Kriterien mittels WHERE
 - Mustervergleiche in Zeichenketten mit LIKE-Operator
 - Sortieren des Abfrageergebnisses mit ORDER BY [ASC, DESC]

```
-- basic select statement  
SELECT * FROM students;
```

```
-- select with criteria  
SELECT * FROM students  
WHERE name LIKE '%Mustermann'  
      AND dateOfBirth IS NOT NULL;
```

```
-- select with columns  
SELECT name, dateOfBirth FROM students  
WHERE id = 1;
```

```
-- select with sorting  
SELECT * FROM students  
ORDER BY dateOfBirth DESC, name;
```

Daten abfragen (2)

- Verknüpfung von Tabellen über JOIN
- Gruppierung mit Aggregat-Funktionen mit GROUP BY und HAVING
- Verschachtelte Abfragen

```
-- query all results for a student
SELECT s.name, l.name, grade FROM results r
INNER JOIN students s ON s.id = r.studentId
INNER JOIN lectures l ON l.id = r.lectureId
WHERE s.id = 3
ORDER BY grade, l.name
```

```
-- query students and their average
-- only considering students with more
-- than three stored results
SELECT s.id, s.name, AVG(r.grade) average
FROM students s
INNER JOIN results r ON r.studentId = s.id
GROUP BY s.id, s.name
HAVING COUNT(*) > 3
```

```
-- query average grade for all lectures
SELECT l.*, (SELECT AVG(grade)
FROM results r
WHERE r.lectureId = l.id) averageGrade
FROM lectures l
```

Daten abfragen (3)

- Beschränkung der Anzahl der zurückgelieferten Datensätze mit LIMIT
 - Optional kann auch Anzahl von zu überspringenden Datensätzen angegeben werden
 - Z. B. nützlich für das seitenweise Laden und Anzeigen von Listen

```
-- query first five lectures in database  
SELECT * FROM lectures LIMIT 5
```

```
-- query the three best students  
SELECT s.id, s.name, AVG(r.grade) average  
FROM students s  
INNER JOIN results r ON r.studentId = s.id  
GROUP BY s.id, s.name LIMIT 3
```

```
-- query first ten students starting with "M"  
SELECT * FROM students WHERE name LIKE 'M%'  
ORDER BY name LIMIT 10  
-- ...and the next ten (= second page)...  
SELECT * FROM students WHERE name LIKE 'M%'  
ORDER BY name LIMIT 10, 10  
-- ...and the next ten (= third page)...  
SELECT * FROM students WHERE name LIKE 'M%'  
ORDER BY name LIMIT 20, 10  
-- or alternatively:  
SELECT * FROM students WHERE name LIKE 'M%'  
ORDER BY name LIMIT 10 OFFSET 20
```

Daten aktualisieren und löschen

- Aktualisieren von Daten mit UPDATE-Anweisung
- Löschen von Daten mit DELETE-Anweisung
- Referenzielle Integrität wird von Datenbanksystem entsprechend durchgesetzt

```
-- update a student
UPDATE students
SET dateOfBirth = '1983-12-15'
WHERE id = 2;
-- (referential integrity will also
-- update all connected results here due
-- to CASCADE setting for foreign key)
UPDATE students SET id = 9 WHERE id = 2;
```

```
-- delete a student
-- (referential integrity will also
-- delete all connected results here due
-- to CASCADE setting for foreign key)
DELETE FROM students WHERE id = 1;
```

```
-- delete all lectures
-- (this will not work until all
-- results have been deleted due to
-- referential integrity)
DELETE FROM lectures;
```


Automatisch generierte Werte

- MySQL kann automatisch eindeutige Werte für eine Tabellenspalte generieren
 - Spalte muss mit `AUTO_INCREMENT` deklariert werden
 - Startwert kann auf Tabellenebene explizit festgelegt werden
- Beim Einfügen in Tabelle kann Spalte mit Autowert ausgelassen werden
 - Alternativ kann `NULL` als Wert angegeben werden
- Zuletzt generierter Wert kann über Funktion `LAST_INSERT_ID()` ausgelesen werden

```
-- teachers table with automatically
-- incremented id starting at 100
CREATE TABLE teachers (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    PRIMARY KEY (id),
    UNIQUE KEY name (name)
) ENGINE=InnoDB AUTO_INCREMENT=100;

-- creating new teachers
INSERT INTO teachers (name) VALUES
    ('Teacher A'), -- id = 100
    ('Teacher B'), -- id = 101
    ('Teacher C'); -- id = 102
INSERT INTO teacher VALUES
    (NULL, 'Teacher D'); -- id = 103

SELECT LAST_INSERT_ID();
```

Transaktionen

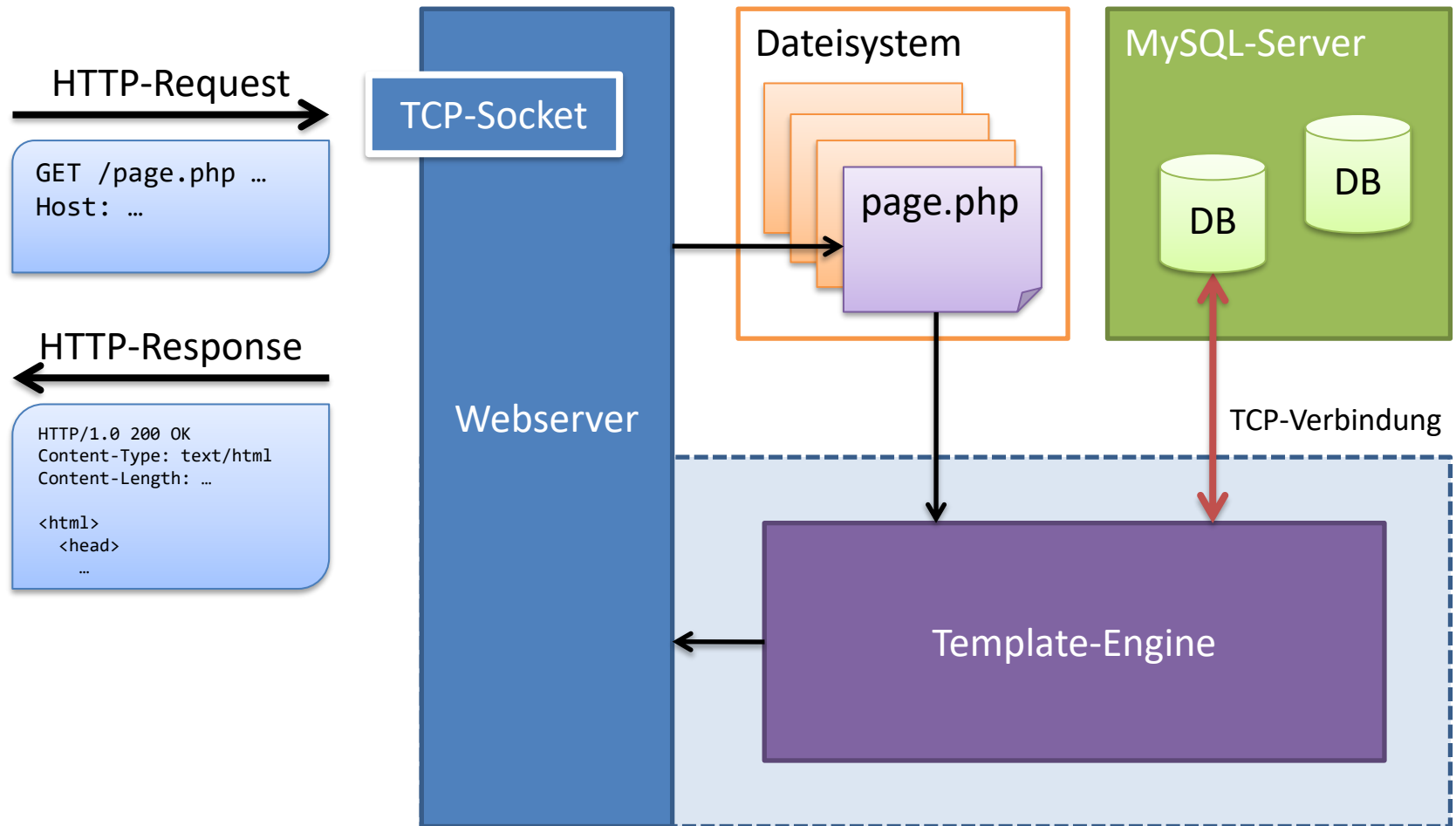
- MySQL unterstützt Transaktionen auf Datenbankebene
 - Betroffene Tabellen müssen mit Speicher-Engine InnoDB verwaltet werden
- BEGIN startet neue Transaktion
- COMMIT oder ROLLBACK beenden laufende Transaktion

```
-- start new transaction and delete some data
BEGIN;
DELETE FROM teachers;
-- rollback transaction (changes are dropped)
ROLLBACK;

-- store new student with some results and
-- only keep student if all results can be
-- successfully stored
-- (auto increment is assumed for student id)
BEGIN;
INSERT INTO students VALUES (NULL,
    'Max Mustermann', 'M', NULL,
    '1985-03-30');
SET @studentId = LAST_INSERT_ID();
INSERT INTO results VALUES
    (1, @studentId, 3),
    (3, @studentId, 1),
    (6, @studentId, 4);
COMMIT;
```

MYSQL AM WEBSERVER

MySQL am Webserver



MySQL und PHP

- PHP bietet Unterstützung für MySQL über fix integrierte Bibliotheken
 - Bibliothek „ext/mysql“
 - Sammlung von Funktionen für Zugriff auf MySQL-Datenbank
 - Mittlerweile veraltet
 - Bibliothek „ext/mysqli“
 - MySQL „improved“
 - Objektorientiert
 - Verbesserungen bei Geschwindigkeit und Sicherheit

Datenbankzugriff mit „ext/mysql“

- Verbindung zu Datenbank wird über Connection-Objekt hergestellt
- Absetzen einer Abfrage erzeugt Cursor-Objekt
- Auslesen der Daten durch Iterieren über den Cursor
- Connection und Cursor müssen immer sauber freigegeben werden

```
//connect to server and database
$con = new mysqli($server, $userName, $password,
                  $database);

if ($con->connect_error) {
    die('Unable to connect. Error: ' . $con->connect_errno);
}

//create and execute query
$query = 'SELECT id, name, address FROM students';
if ($result = $con->query($query)) {
    //read result and create student objects
    $students = array();
    while ($row = $result->fetch_object())
        $students[] = new Student($row->id, $row->name,
                                   $row->address);

    //free result
    $result->close();
} else {
    die("Error in $query: " . $con->error);
}

//close connection
$con->close();
```

Klasse „mysqli_result“

- Ergebnis einer Abfrage ist Objekt der Klasse „mysqli_result“
 - Repräsentiert einen Cursor
- Bietet verschiedene Varianten zum Auslesen eines Eintrags aus dem Cursor
 - Indiziertes Feld
 - Assoziatives Feld
 - Als Objekt

```
//create and execute query
$query = 'SELECT id, name, address FROM students';
if ($result = $con->query($query)) {

    ...

    //example: indexed array
    $students = array();
    while ($row = $result->fetch_row())
        $students[] = new Student($row[0], $row[1],
                                   $row[2]);

    //example: associative array
    $students = array();
    while ($row = $result->fetch_assoc())
        $students[] = new Student($row['id'], $row['name'],
                                   $row['address']);

    //example: object
    $students = array();
    while ($row = $result->fetch_object())
        $students[] = new Student($row->id, $row->name,
                                   $row->address);

    ...
}
```

Transaktionen

- Direkte Unterstützung in „ext/mysqli“
 - mysqli::commit()
 - mysqli::rollback()
- „Auto-Commit“
 - Standardmäßig aktiv
 - Muss für zusammenhängende Folge von Statements deaktiviert werden

```
//connect to server and database
$con = ...;

//turn off auto commit
$con->autocommit(false);

//execute a query
$query1 = 'INSERT INTO students VALUES (NULL, "Max",
                                           "Mustermann")';

$con->query($query1);
...

//execute another query in same transaction
$query2 = 'INSERT INTO students VALUES (NULL, "Susi",
                                           "Superfrau")';

$con->query($query2);
...


//commit transaction
$con->commit();

//close connection
$con->close();
```


SQL-Injection

- Direkter Einbau von Variablenwerten in SQL-Abfragen führt zu schwerwiegendem Sicherheitsproblem
 - Durch geschickt gewählte Werte Umgehen von Bedingungen oder sogar Abändern der Datenbank möglich

```
//query articles of first category with custom filter
$result = array();
$filter = $_REQUEST['filter'];
$qry = $con->query("SELECT * FROM articles WHERE
                  categoryId = 1 AND
                  description LIKE '%$filter%'");
if (!$qry) die('Database error.');
```



```
while ($article = $qry->fetch_object())
    $result[] = new Article($article->id, ...);
$qry->close();
```

```
//suitable filter string, no problem
FILTER: bike
SQL: SELECT * FROM articles WHERE categoryId = 1 AND
      description LIKE '%bike%';
```

```
//BAD: filter ignored
FILTER: ' OR 1=1; --
SQL: SELECT * FROM articles WHERE categoryId = 1 AND
      description LIKE '%" OR 1=1; --%';
```

```
//WORSE: data is deleted
FILTER: '; DELETE FROM articles; --
SQL: SELECT * FROM articles WHERE categoryId = 1 AND
      description LIKE '%'; DELETE FROM articles;
      --%';
```

SQL-Injection (2)

- Prüfung und Absicherung von Variablenwerten erforderlich
 - mysqli::real_escape_string(...)

```
//query articles of first category with custom filter
$result = array();
$filter = $con->real_escape_string($_REQUEST['filter']);
$query = $con->query("SELECT * FROM articles WHERE
                    categoryId = 1 AND
                    description LIKE '%$filter%'");
if (!$query) die('Database error.');
```

```
while ($article = $query->fetch_object())
    $result[] = new Article($article->id, ...);
$query->close();
```

```
//was always okay
FILTER: bike
SQL: SELECT * FROM articles WHERE categoryId = 1 AND
     description LIKE '%bike%';
```

```
//""" is escaped now --> not a problem anymore
FILTER: ' OR 1==1; --
SQL: SELECT * FROM articles WHERE categoryId = 1 AND
     description LIKE '%\' OR 1==1; --%';
```

```
//""" is escaped now --> not a problem anymore
FILTER: '; DELETE FROM articles; --
SQL: SELECT * FROM articles WHERE categoryId = 1 AND
     description LIKE '%\'; DELETE FROM articles;
     --%';
```

Parametrisierte Abfragen

- SQL-Statements werden mit Platzhaltern für Parameter definiert
- Variablen werden an Platzhalter gebunden
- Datenbank prüft Parameter vor Ausführung der Abfrage entsprechend
- Vorteile
 - SQL-Injection wird vermieden
 - Zusätzlich auch Geschwindigkeitsvorteile möglich, wenn eine Abfrage mehrfach verwendet wird

Parametrisierte Abfragen mit „ext/mysql“

- Abfrage vorbereiten
 - Platzhalter „?“ für Parameter
 - Ergebnis ist Objekt der Klasse „mysql_stmt“
- Parameter binden
 - Datentypen für Parameter müssen angegeben werden
 - „i“ (Integer), „s“ (String), „d“ (Double), „b“ (BLOB), ...
- Abfrage ausführen
- Rückgabewerte binden
- Datensätze abrufen

```
//connect to server and database
$con = ...

//create query and prepare statement
$query = 'SELECT id, address FROM students WHERE
        firstName = ? AND lastName = ?';
$stmt = $con->prepare($query);

//bind parameters
$stmt->bind_param('ss', $firstName, $lastName);

//execute statement
if ($stmt->execute()) {
    //bind result
    $stmt->bind_result($id, $address);
    //read result and create student objects
    $students = array();
    while ($result->fetch())
        $students[] = new Student($id, $address);
} else {
    die('Error: ' . $con->error);
}

//free statement
$stmt->close();

//close connection
$con->close();
```

Datenbank-Abstraktion

- PHP Data Objects (PDO)
 - Objektorientierte Datenbankzugriffsschicht
 - Abstraktion unterschiedlicher Datenbanksysteme über verschiedene Datenbanktreiber
 - Unterstützung für MySQL, PostgreSQL, SQLite, MSSQL, ...
 - SQL-Anweisungen nach wie vor aber spezifisch (z. B. „LIMIT“ vs. „TOP“)
- Objektrelationale Mapper
 - Völlige Abstraktion des Datenbankzugriffs
 - Relationale Datenbankstruktur wird auf Klassen abgebildet
 - „Data-Mapper“ verwandelt Datenbankeinträge in Objekte und umgekehrt
 - Diverse Frameworks und Bibliotheken
 - Doctrine, Propel, ...

Speicherung von Passwörtern

- Passwörter nicht im Klartext ablegen
 - Wird Zugriff auf Datenbank erlangt, sind alle Passwörter automatisch bekannt
- ➔ Passwörter als Hash speichern
 - Ursprüngliches Passwort ist nicht mehr rekonstruierbar
 - Verifizierung erfolgt über Vergleich der Hashwerte
- Typische Hashfunktionen sind nicht für Hashen von Passwörtern geeignet
 - MD5, SHA1, SHA256 etc. sind auf Geschwindigkeit und Effizienz optimiert
 - Brute-Force-Angriffe somit heutzutage leicht möglich
- ➔ Spezielle „rechenintensive“ Hashfunktionen
 - Z. B. „bcrypt“ (basierend auf Krypto-Algorithmus „Blowfish“)

Speicherung von Passwörtern (2)

- Aufwändig zu berechnender Hash alleine ist noch nicht sicher genug
 - Gleiche Passwörter liefern gleiche Hashwerte
 - Regenbogentabellen = Datenstrukturen zur schnellen Auffindung von ursprünglichen Werten für Hashwerte
- ➔ Salzen von Hashwerten
 - Generierung eines zufälligen Werts (= Salz) bei Speicherung des Passworts
 - Salz fließt in Berechnung des Hashes mit ein (und wird danach gemeinsam mit Passwort-Hash gespeichert)
 - Dadurch auch bei gleichen Passwörtern immer unterschiedliche Hashwerte

Speicherung von Passwörtern (3)

- PHP bietet natives API zum Hashen und Verifizieren von Passwörtern
- Funktion **password_hash()**
 - Erzeugt Salz, berechnet Hash und liefert beides zusammen mit Informationen über verwendeten Algorithmus zurück

\$2y\$10\$6z7GKa9kpDN7KC3ICW1Hi.f0/to7Y/x36WUKNP0IndHdkdR9Ae3K

- Algorithm
- Algorithm options (eg cost)
- Salt
- Hashed password

- Funktion **password_verify()**
 - Verifiziert ein Passwort gegen einen gegebenen Hash mit Salz

```
//USER CREATION
$newUser = "user";
$newPwd = "p@ssw0rd";
//create password hash
$hashedPwd = password_hash($newPwd, PASSWORD_DEFAULT);
//create new user
$newUser = $con->real_escape_string($newUser);
$hashedPwd = $con->real_escape_string($hashedPwd);
$con->query("INSERT INTO users (userName, passwordHash)
VALUES ('$newUser', '$hashedPwd');");

//AUTHENTICATION
$enteredUser = $_POST['userName'];
$enteredPwd = $_POST['password'];
//try to retrieve user
$enteredUser = $con->escape_string($enteredUser);
$qry = $con->query("SELECT passwordHash FROM users
WHERE userName = '$enteredUser'");
if (!$qry) die('Database error.');
```

```
$res = $qry->fetch_result();
//check password hash
$loginOk = $res && password_verify($enteredPwd,
$res->passwordHash));
```