

## Vorlesung



University of Applied Sciences

# DBSP

## Unit

Datenbank-gestützte Webapplikationen

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

1



University of Applied Sciences



## Prof. Dr. rer. nat. Nane Kratzke

*Praktische Informatik und  
betriebliche Informationssysteme*

- Raum: 17-0-10
- Tel.: 0451 300 5549
- Email: [kratzke@fh-luebeck.de](mailto:kratzke@fh-luebeck.de)



@NaneKratzke

Updates der Handouts auch über Twitter #dbsp

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

2

**Handout zur Vorlesung**  
**DBSP – DB-gestützte Webapplikationen – Unit 14**

## Übergreifende Ziele der Lehrveranstaltung



University of Applied Sciences

Client- und Serverseitige Entwicklung

PHP (Serverseitig)

JavaScript (Clientseitig)

„Hosten“ von Apps

Framework Erfahrungen

CMS (Drupal)

WebServices (Google-Maps)

jQuery

Datenbank-Integration

Berücksichtigung von Sicherheitsaspekten

HTML-Injections

SQL-Injections

Session Hijacking

Login-Systeme

Um sich weitere Web-Technologien autodidaktisch erarbeiten zu können.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

3

## Units



University of Applied Sciences

Unit 1  
Cloud Computing  
IaaS

Unit 2  
CMS Drupal

Unit 3  
HTML und CSS

Unit 4 - 7  
PHP I - IV

Unit 8  
Sessions, Cookies,  
Formulare und  
Login-System

Unit 9  
JavaScript

Unit 10  
Drupal Module  
Development

Unit 11  
Datenmodellierung

Unit 12 - 13  
Datenbanken und SQL  
Vom Datenmodell zur  
Datenbank

Unit 14  
Datenbank-gestützte  
Web-Applikationen

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

4

## Datenbank – Server – Client

Wo waren wir nochmal?

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

Wir sind hier!

Datenbank      Server      Client

SQL      HTML/CSS

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme    5

## Zum Nachlesen ...

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

**Datenbanken**  
Grundlagen und Design

Konzepte, Entwurf, Design, Implementierung  
Konkrete Erklärungen am Praxisbeispiel  
Zahlreiche Aufgaben mit Musterlösungen

**Kapitel 2**  
Datenbanksysteme, Datenbankanwendungen und Middleware

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme    6

## Zum Nachlesen ...



University of Applied Sciences



### MySQLi Online Dokumentation

<http://php.net/manual/de/book.mysqli.php>

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

7

## Zum Nachlesen ...



University of Applied Sciences



### Bereitgestellte Skripte:

#### Datenbanken und PHP

- **Kapitel 2:** Datenbank Architekturen
- **Kapitel 3:** Datenbanken bei dynamischen Webseiten

#### PHP Programmierung

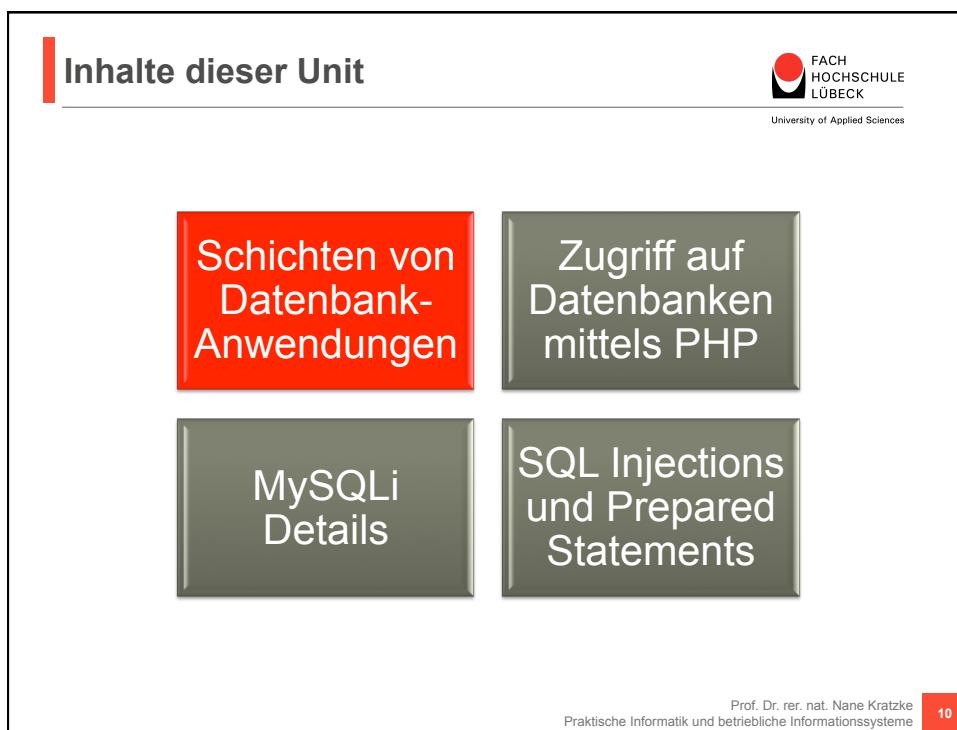
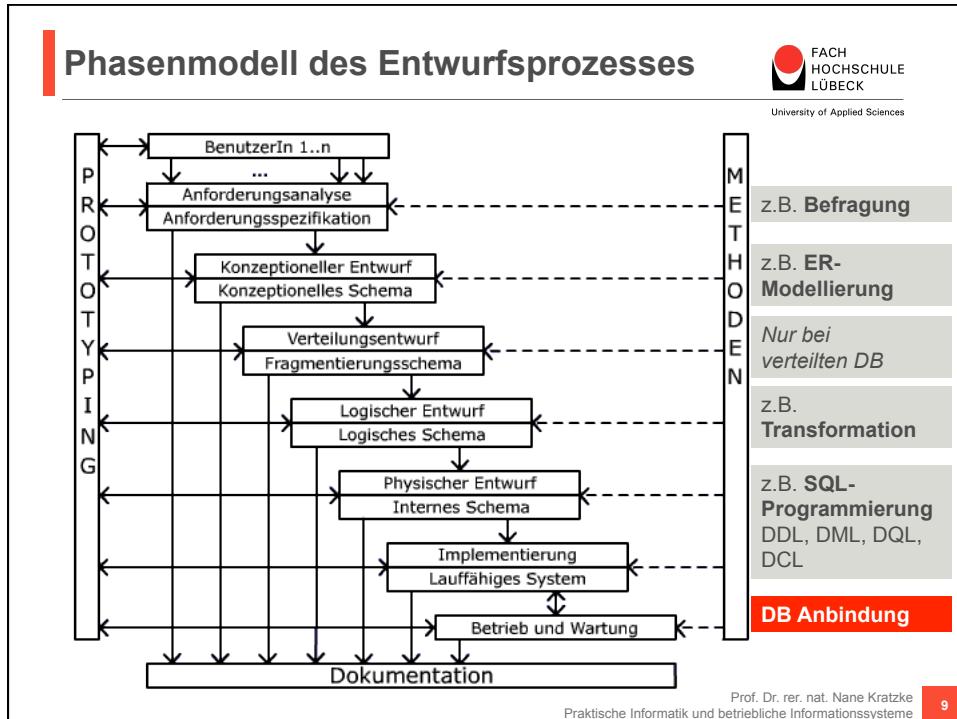
- **Kapitel 18:** Einleitung in Datenbank-gestützte Websysteme
- **Kapitel 19:** Architekturen datenbankgestützter Websysteme
- **Kapitel 20:** Datenbanken bei dynamischen Webseiten

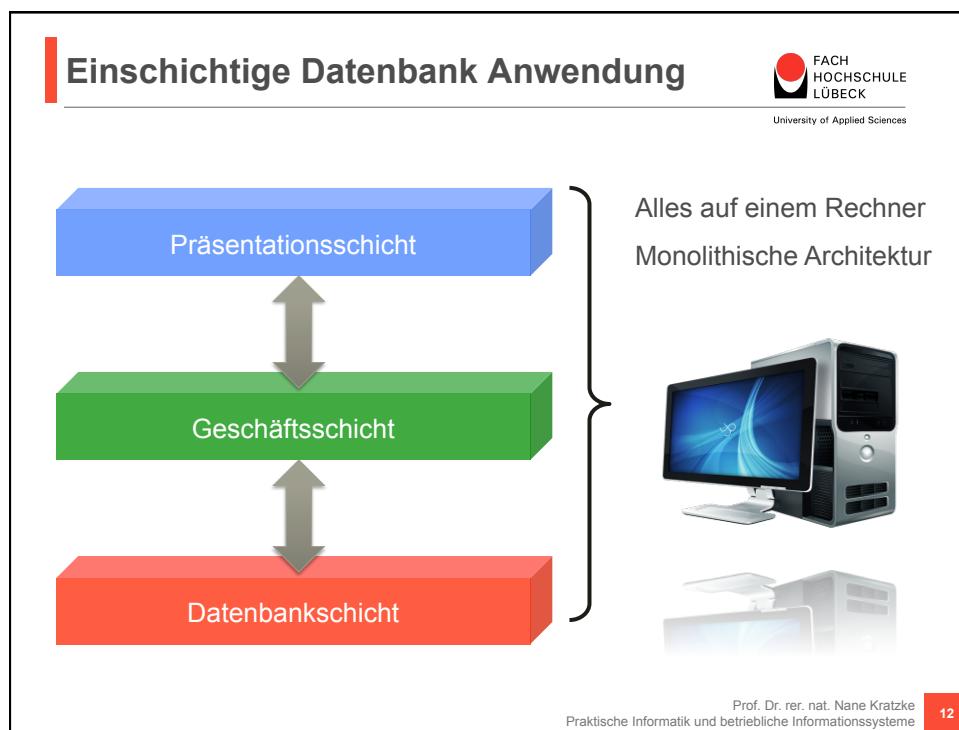
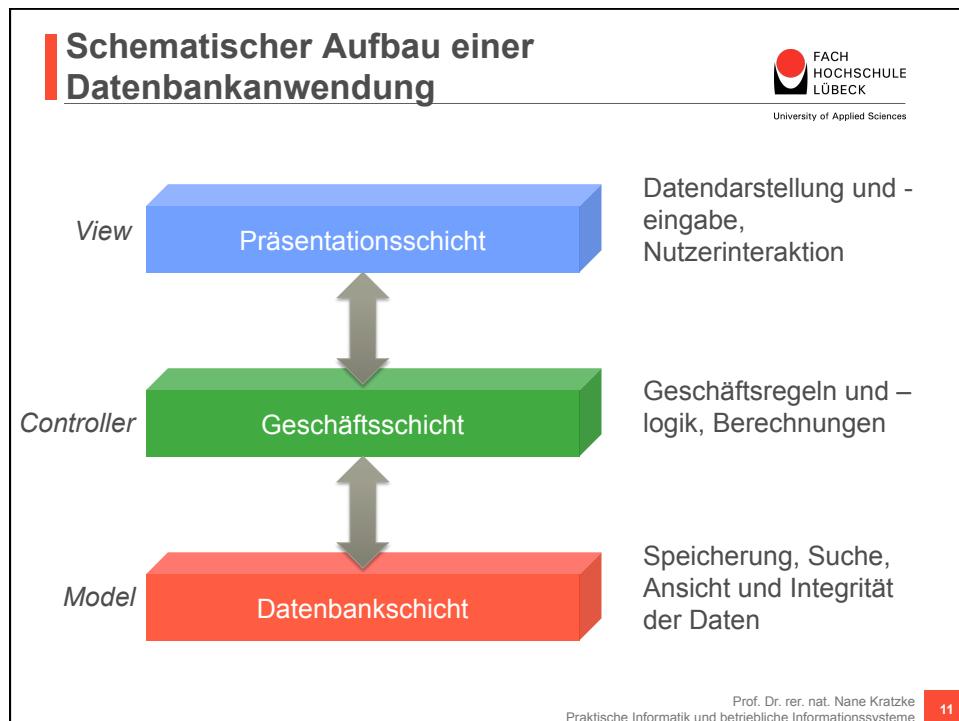
<http://praktische-informatik.fh-luebeck.de/node/50>

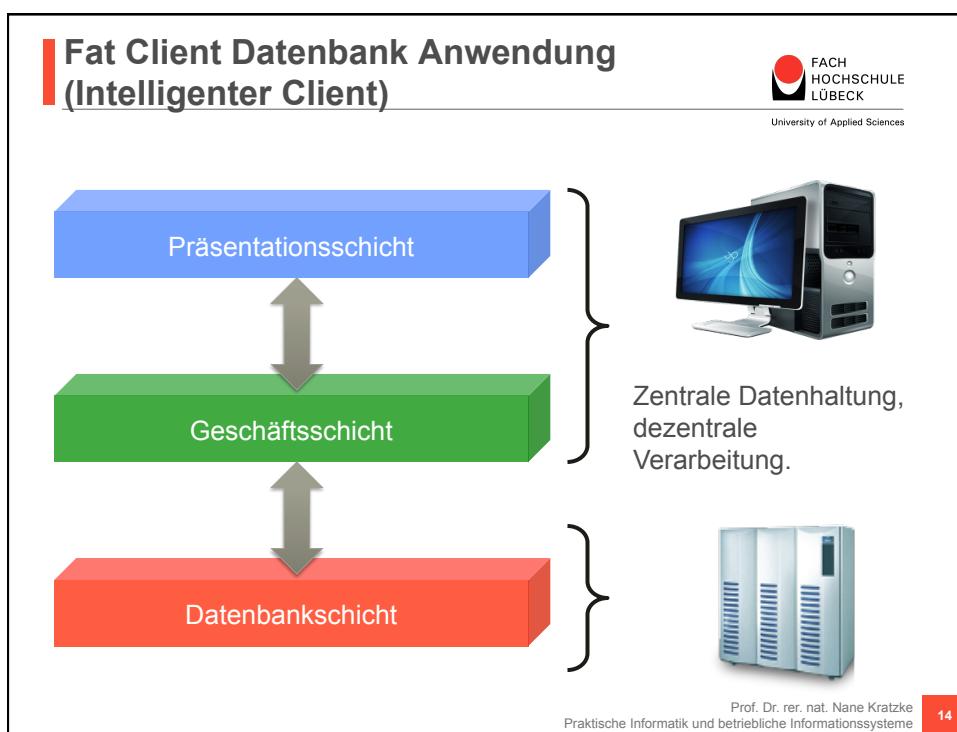
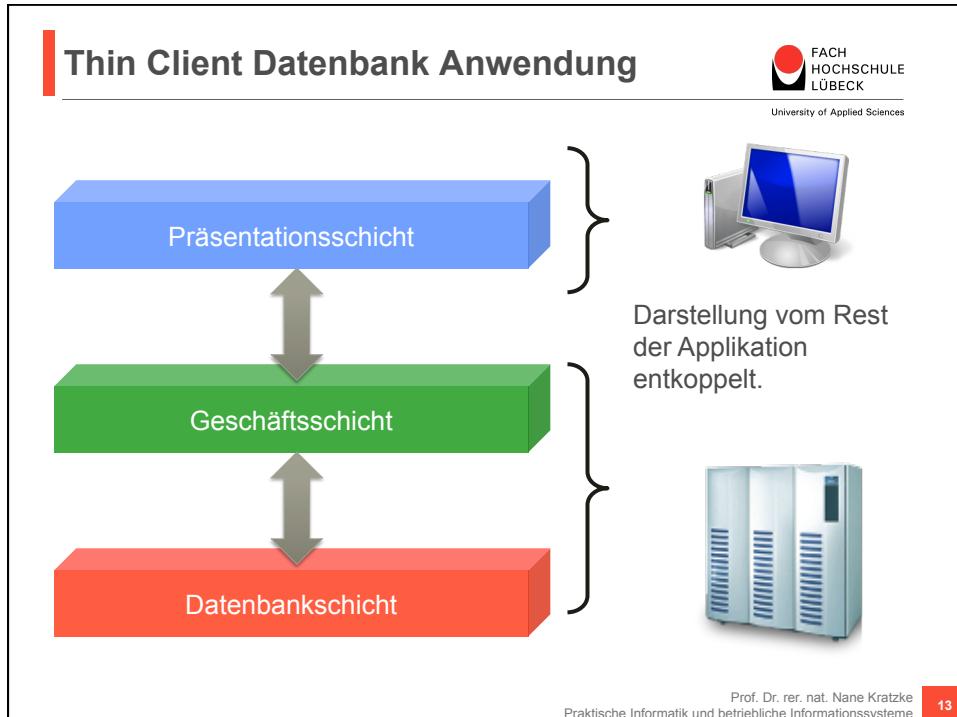
<http://praktische-informatik.fh-luebeck.de/node/39>

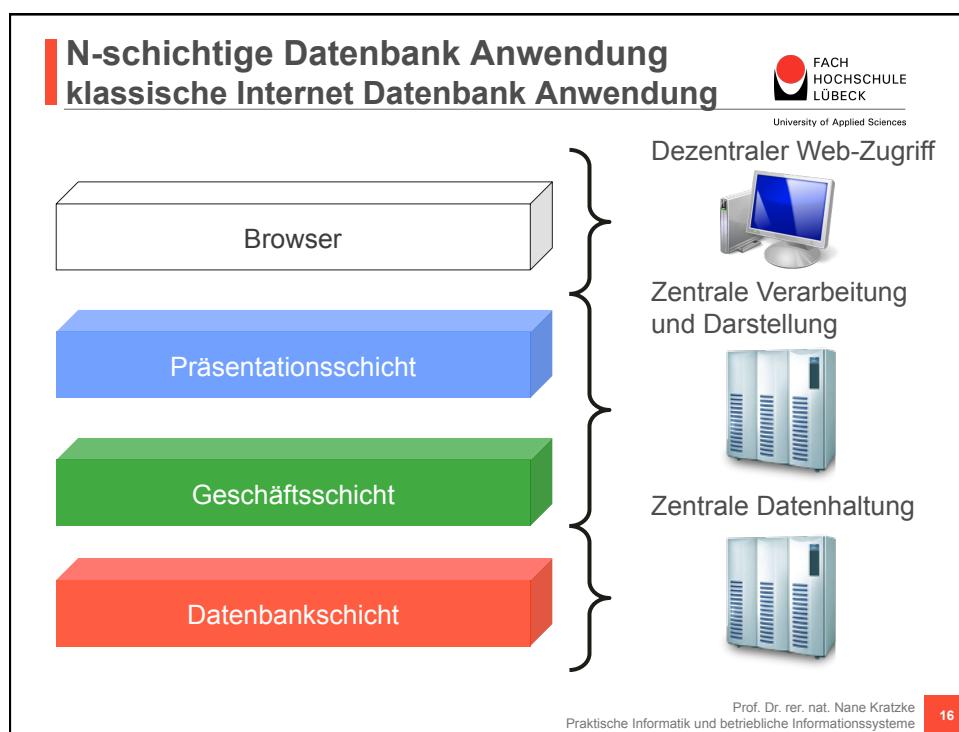
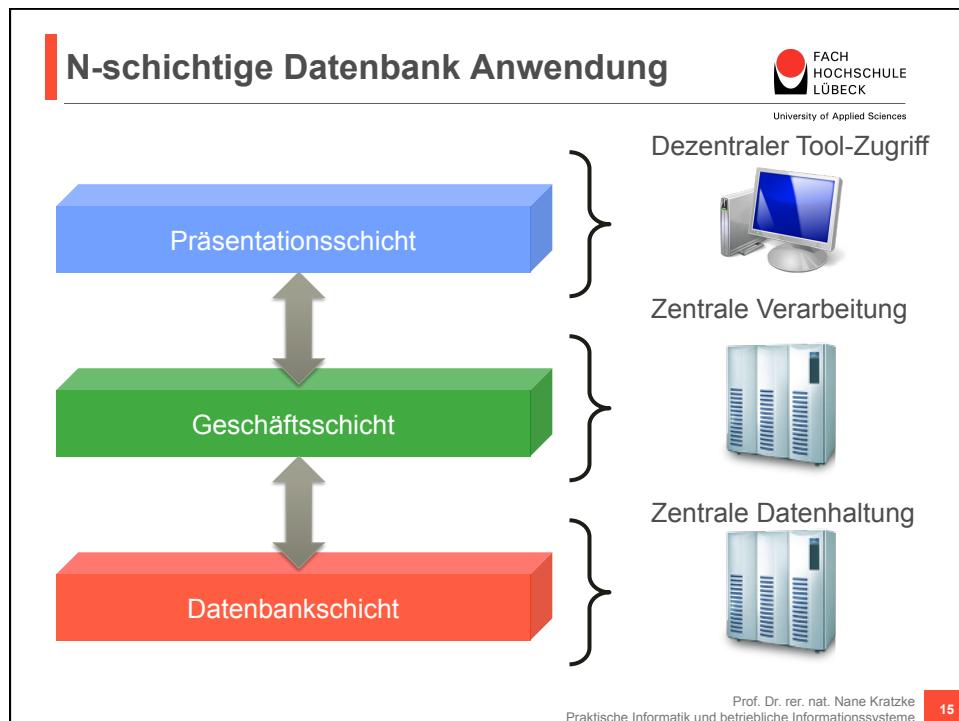
Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

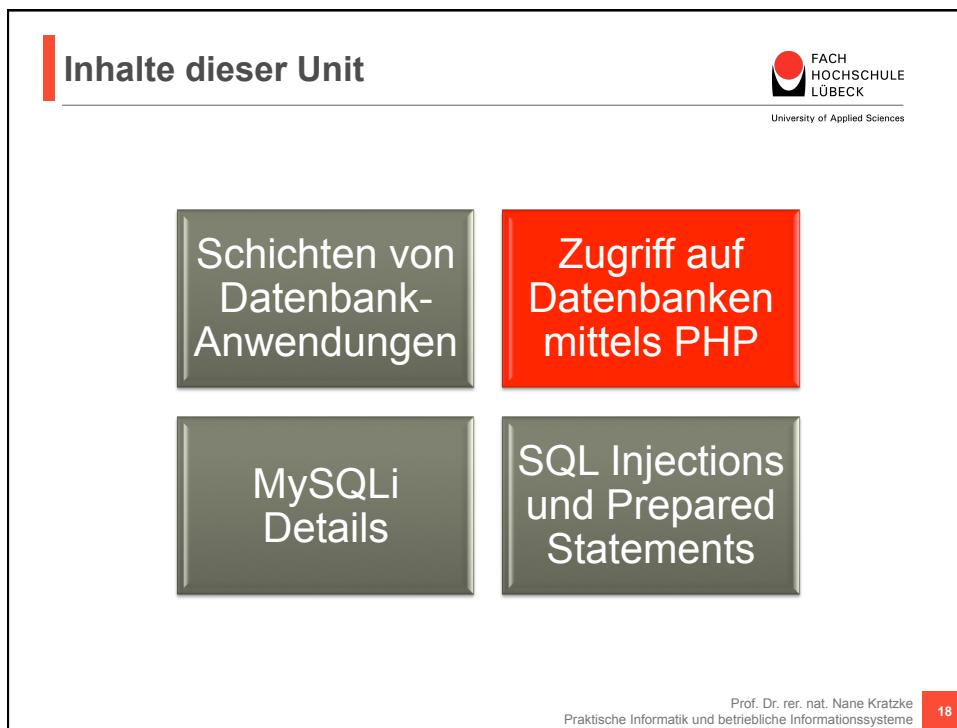
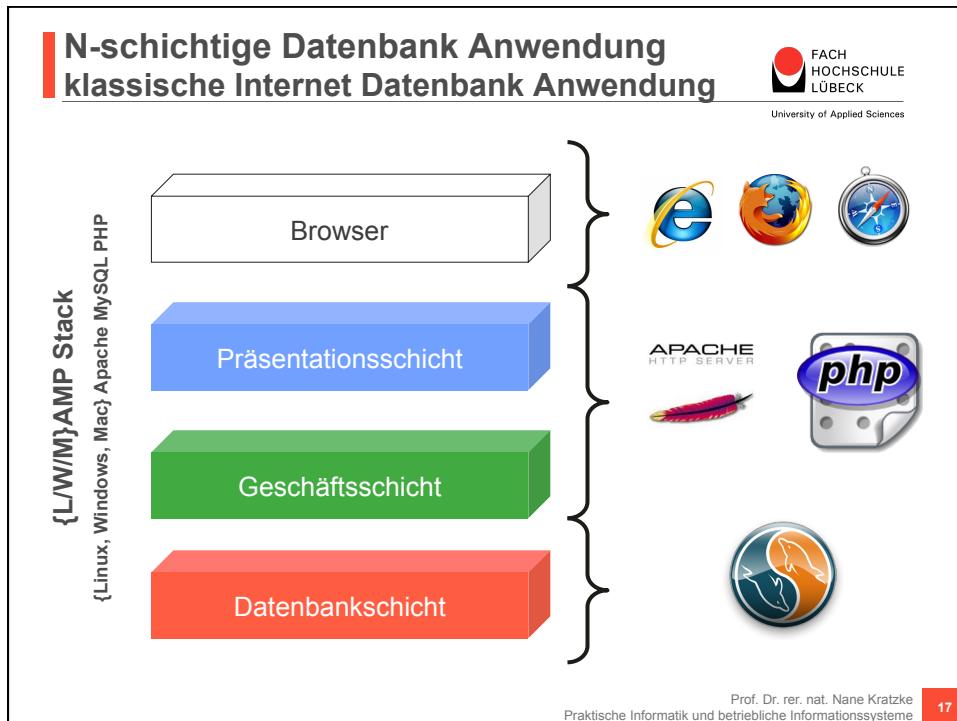
8

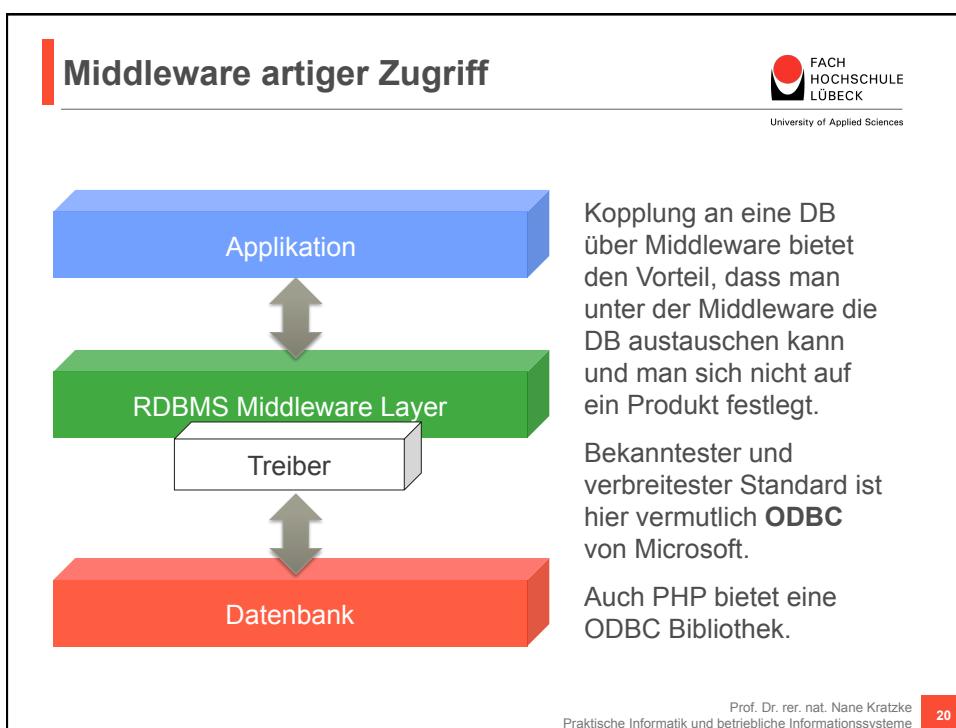
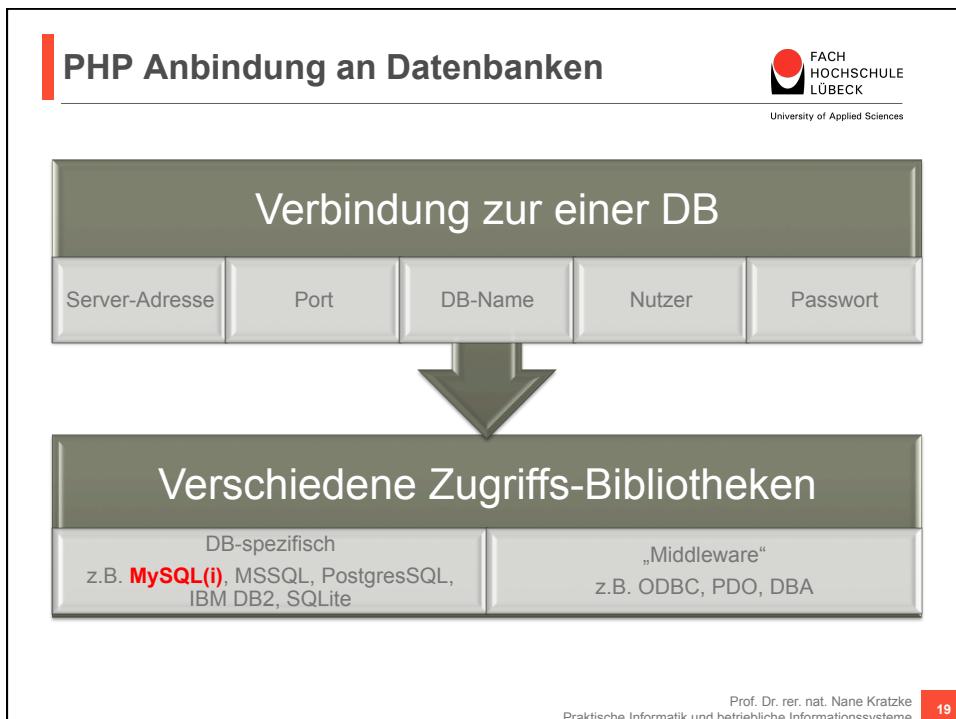












## DB-spezifischer Zugriff



University of Applied Sciences

- Verbreiteste und bekanntestes open source RDBMS ist vermutlich MySQL
- Die MySQL Integration in PHP ist eng und robust
- MySQL ist bei Open Source Projekten meist die erste Wahl
- Aufgrund der Verbreitung
- und der leichten Übertragbarkeit der DB-Zugriffsprinzipien auf andere RDBMS
- erfolgen die weiteren Erläuterungen am Beispiel von MySQL mittels der MySQLi Bibliothek

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

21

## Inhalte dieser Unit



University of Applied Sciences

Schichten von Datenbank-Anwendungen

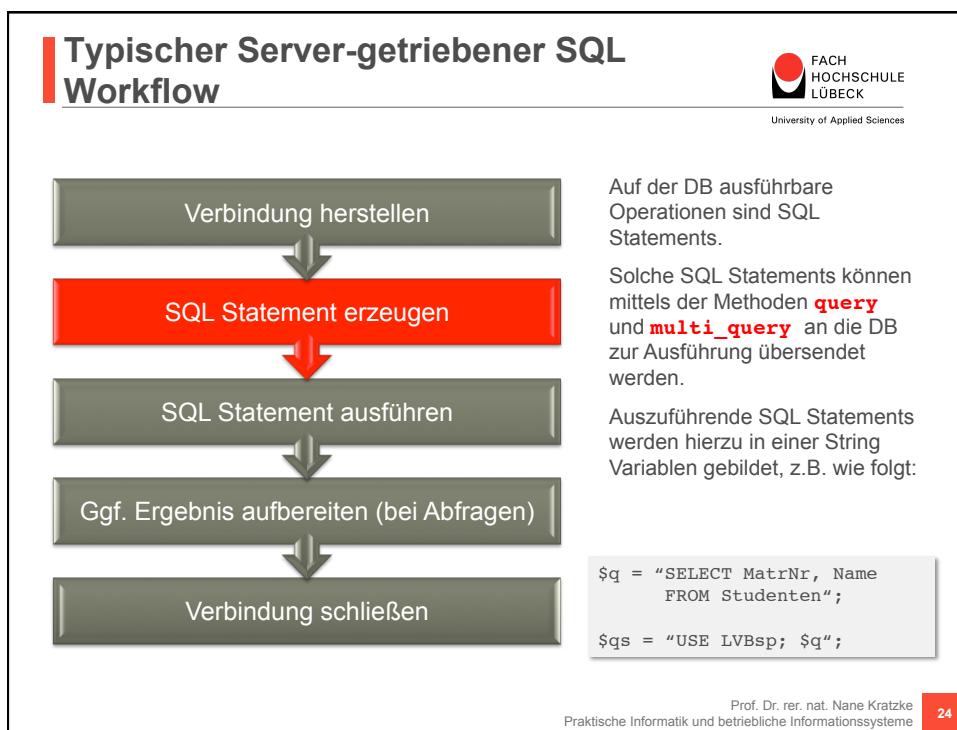
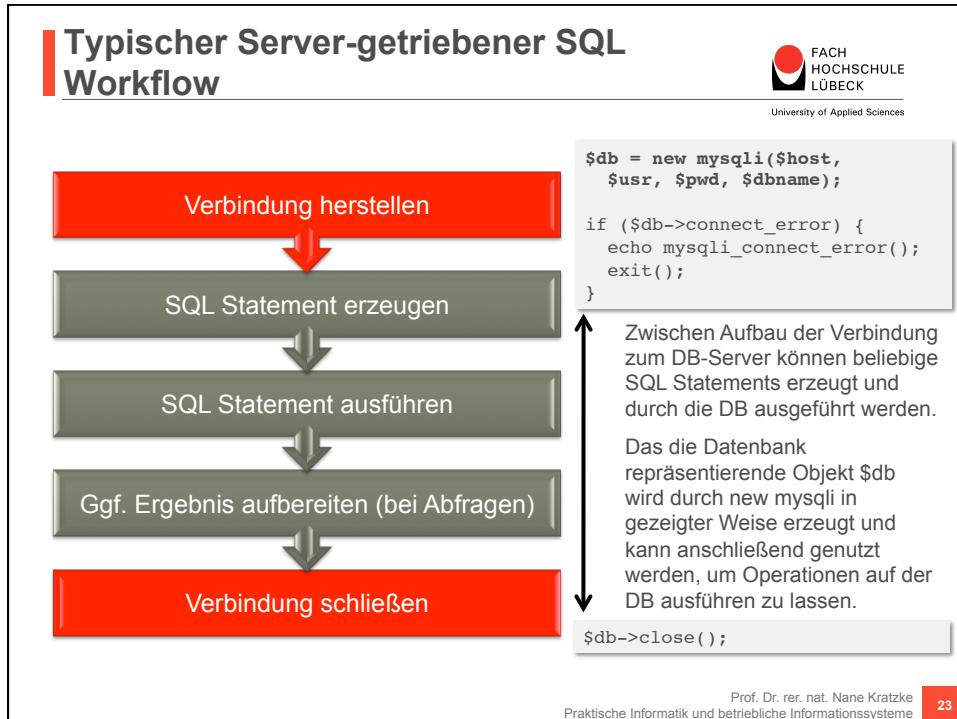
Zugriff auf Datenbanken mittels PHP

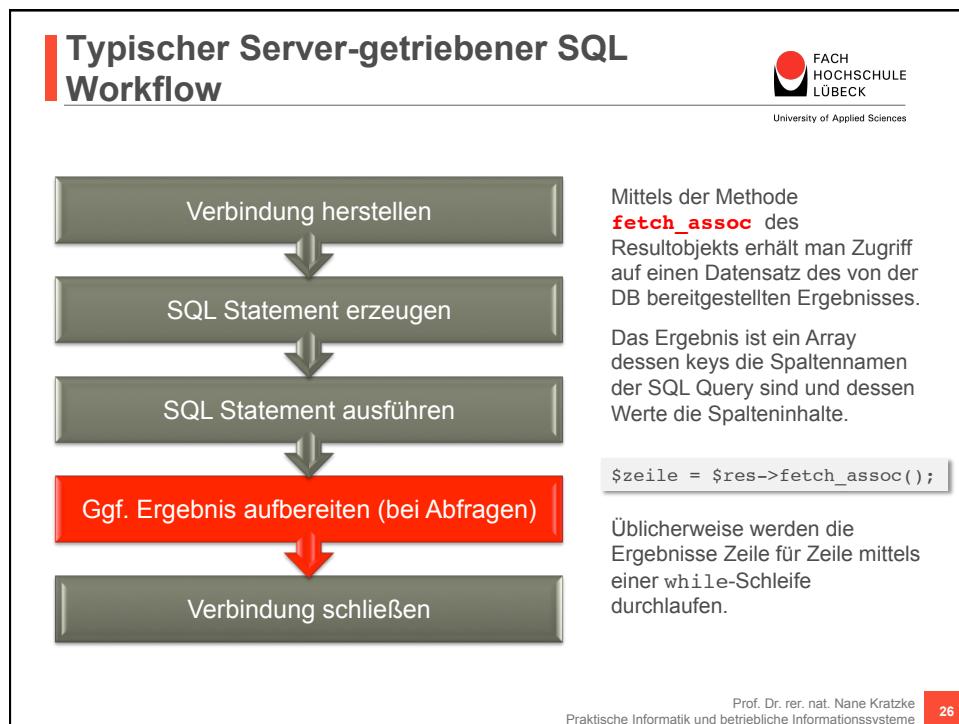
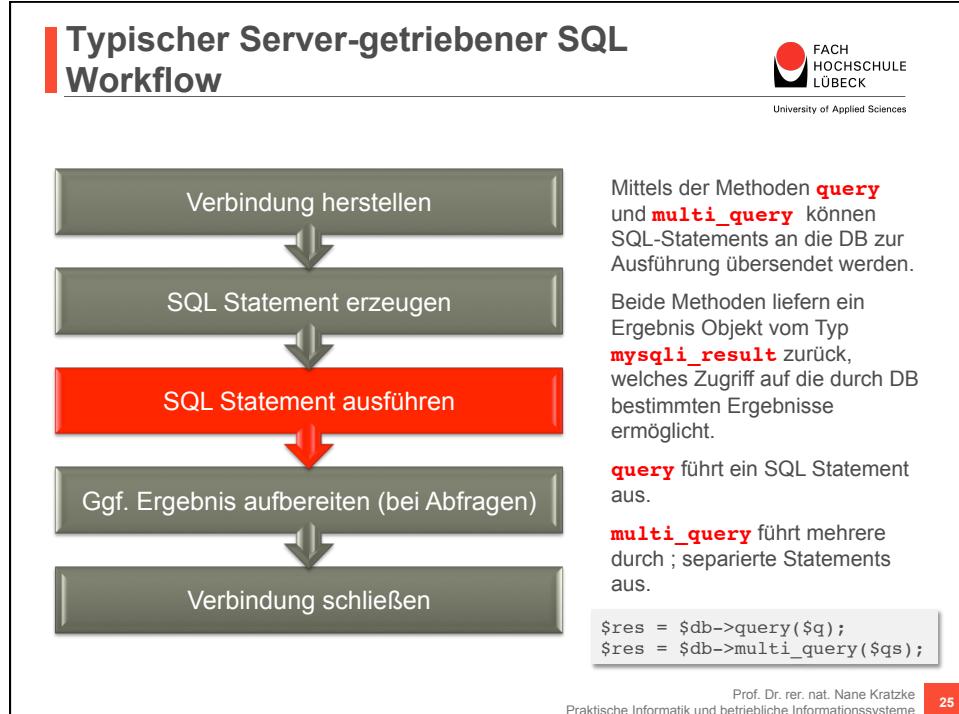
MySQLi Details

SQL Injections und Prepared Statements

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

22





## Beispiel einer typischen Ergebnisabfrage



University of Applied Sciences

```
$q = "SELECT MatrNr, Name
      FROM Studenten";

$res = $db->query($q);

while ($zeile = $res->fetch_assoc()) {
    $es = array();
    $es['MatrNr'] = $zeile['MatrNr'];
    $es['Name'] = $zeile['Name'];

    foreach ($es as $k => $v) {
        echo "$k -> $v<br/>";
    }
}
```

### Studenten:

MatrNr	Name	SG
12345	Max Mustermann	ESA
54321	Maren Musterfrau	INF
23451	Hansi Hase	IGi
56432	Hanna Montana	KIM

12345 -> Max Mustermann  
 54321 -> Maren Musterfrau  
 23451 -> Hansi Hase  
 56432 -> Hanna Montana

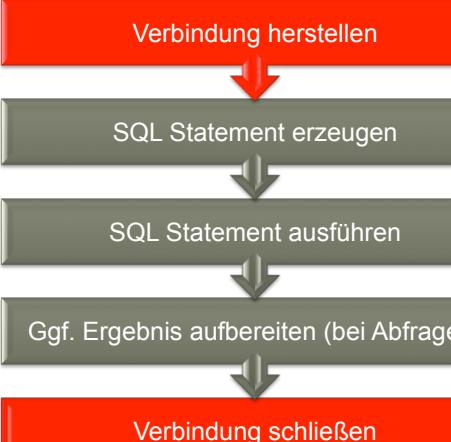
Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

27

## Typischer Server-getriebener SQL Workflow



University of Applied Sciences



```

$db = new mysqli($host,
                  $usr, $pwd, $dbname);

if ($db->connect_error) {
    echo mysqli_connect_error();
    exit();
}

Zwischen Aufbau der Verbindung
zum DB-Server können beliebige
SQL Statements erzeugt und
durch die DB ausgeführt werden.

Das die Datenbank
repräsentierende Objekt $db
wird durch new mysqli in
gezeigter Weise erzeugt und
kann anschließend genutzt
werden, um Operationen auf der
DB ausführen zu lassen.

$db->close();
    
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

28

## Sonderzeichen berücksichtigen



University of Applied Sciences

- Beim Eintragen von Daten in eine Datenbank
- oder beim Abfragen von Daten aus einer Datenbank
- mittels SQL-Statements
- müssen Zeichen mit einer besonderen Bedeutung „maskiert“ werden.
- Dies betrifft insbesondere die SQL Statements
  - **INSERT**
  - **SELECT**
  - **UPDATE**

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

29

## Sonderzeichen berücksichtigen Beispiel



University of Applied Sciences

```
$q = "INSERT INTO Studenten (MatrNr, Name, SGID) VALUES
      ('12345', 'Mac O'Brien', 'IGi')";
$res = $db->query($q);
```

Beispiel eines  
Sonderzeichens, dass  
mit SQL Syntax kollidiert

```
$q = "INSERT INTO Studenten (MatrNr, Name, SGID) VALUES
      ('12345', 'Mac O\'Brien', 'IGi')";
$res = $db->query($q);
```

Sonderzeichen durch  
Backslash \ maskieren.

```
$matrnr = $db->real_escape_string("12345");
$name = $db->real_escape_string("Mac O'Brien");
$sg = $db->real_escape_string("IGi");

$q = "INSERT INTO Studenten (MatrNr, Name, SGID) VALUES
      ('$matrnr', '$name', '$sg')";
$res = $db->query($q);
```

Alle Sonderzeichen  
können durch die  
Methode  
**real\_escape\_string**  
maskiert werden.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

30

## Filtere Werte die an die Datenbank transferiert werden!



University of Applied Sciences

- Welche Zeichen maskiert werden müssen
- kann von DBMS zu DBMS herstellerspezifisch verschieden sein
- daher bietet es sich an, zum „Escapen“ eine DBMS-spezifische Funktion zu nutzen
- MySQL => **real\_escape\_string**
- Sie sollten insbesondere Werte aus Formularen oder sonstigen Quellen nicht ungefiltert als HTML ausgeben oder in SQL Queries übernehmen
- Denn erinnern Sie sich ...

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

31

## Alle Eingaben sind BÖSE!

- **Frei nach Dr. House:**
  - Alle Nutzer sind böse, lügen, betrügen und wollen Deinem Server etwas Böses.
  - Traue keiner Nutzereingabe, die nicht auf dem eigenen Server geprüft, gefiltert
  - und für unschädlich befunden wurde
  - bzw. unschädlich gemacht wurde.

Was bei HTML als  
HTML Injection  
funktioniert,  
funktioniert auch bei  
SQL und heißt ...



SQL Injection

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

32

## Inhalte dieser Unit



University of Applied Sciences

Schichten von  
Datenbank-  
Anwendungen

Zugriff auf  
Datenbanken  
mittels PHP

MySQLi  
Details

SQL Injections  
und Prepared  
Statements

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

33

## Was sind SQL-Injections?



University of Applied Sciences

- Einschleusen von Datenbankbefehlen durch einen Angreifer
- Ziele
  - Zugriff auf geschützte Bereiche der Datenbank
  - Daten einsehen
  - Daten ändern
  - Daten einbringen – Schadcode hinterlassen
  - Anrichten von Schaden (Daten löschen)

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

34

## **SQL-Injections klassisches Beispiel**



University of Applied Sciences

- „Hacken“ einer DB-gestützten Nutzerverwaltung
  - In einer Tabelle `benutzer` werden Benutzernamen mit zugehörigen Kennwörtern verwaltet
  - Benutzername und Passwort werden mittels eines Login-Formulars übertragen
  - Mittels eines SELECT SQL Statement wird geprüft, ob die Kombination Benutzername und Passwort existiert
  - und es wird eine ID zurückgeliefert unter der dieser Nutzer geführt wird.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

35

## **SQL-Injections Beispiel für anfällige SQL-Abfrage**



University of Applied Sciences

Ungeschützte SQL-Query Generierung

```
$name = $_POST['name'];
$password = $_POST['password'];

$q = "SELECT id, name FROM benutzer
      WHERE name='$name' AND
            pwd='$password';";

$res = $db->query($q);
```

**Erzeugt für folgende Eingabe:**

**Name:** Dr. House

**Passwort:** EVIL

```
SELECT id, name FROM benutzer
WHERE name='Dr. House' AND
      pwd='EVIL';
```

**Erzeugt für folgende Eingabe:**

**Name:** Dr. House

**Passwort:** ' OR 'a'='a

```
SELECT id, name FROM benutzer
WHERE name='Dr. House' AND
      pwd=' OR 'a'='a';
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

36

## SQL-Injections

### Beispiel für anfällige SQL-Abfrage



University of Applied Sciences

Ungeschützte SQL-Query Generierung

```
$name = $_POST['name'];
$password = $_POST['password'];

$q = "SELECT id, name, FROM benutzer
      WHERE name='$name' AND
            pwd='$password';";

$res = $db->query($q);
```

Erzeugt für folgende Eingabe:

Name: Dr. House

Password: ' OR 'a'='a

```
SELECT id, name FROM benutzer
WHERE name='Dr. House' AND
      pwd=' ' OR 'a'='a';
```

Injezierter  
SQL-Code

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

37

## SQL-Injections

### Beispiel: Abarbeitung der injizierten Query



University of Applied Sciences

```
SELECT id, name FROM benutzer
WHERE name='Dr. House' AND pwd=' ' OR 'a'='a';
```

```
SELECT id, name FROM benutzer
WHERE TRUE AND FALSE OR 'a'='a';
```

```
SELECT id, name FROM benutzer
WHERE FALSE OR 'a'='a';
```

```
SELECT id, name FROM benutzer
WHERE FALSE OR TRUE;
```

D.h. der Angreifer muss nur einen registrierten Nutzernamen kennen, aber nicht dessen Passwort, um sich einzuloggen !!!

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

38

## Schutz vor SQL-Injections Was kann man tun?



University of Applied Sciences

Ungeschützte SQL-Query Generierung

```
$name = $_POST['name'];
$password = $_POST['password'];

$q = "SELECT id, name, FROM benutzer
      WHERE name='$name' AND
            pwd='$password';";

$res = $db->query($q);
```

Geschützte SQL-Query Generierung

```
$name = $db->real_escape_string($_POST['name']);
$password = $db->real_escape_string($_POST['password']);

$q = "SELECT id, name, FROM benutzer
      WHERE name='$name' AND
            pwd='$password';";

$res = $db->query($q);
```

Analog wie bei HTML-Injections  
Stattdessen **htmlspecialchars** nutzen von  
**real\_escape\_string**



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

39

## SQL-Injections Beispiel für anfällige SQL-Abfrage



University of Applied Sciences

Geschützte SQL-Query Generierung

```
$name = $db->real_escape_string($_POST['name']);
$password = $db->real_escape_string($_POST['password']);

$q = "SELECT id, name, FROM benutzer
      WHERE name='$name' AND
            pwd='$password';";

$res = $db->query($q);
```

Erzeugt für folgende Eingabe:

Name: Dr. House

Passwort: ' OR 'a='a

```
SELECT id, name FROM benutzer
WHERE name='Dr. House' AND
      pwd='\' OR \'a\'=\'\a';
```

**Sinn- aber harmlos!**

Geschützter  
SQL-Code

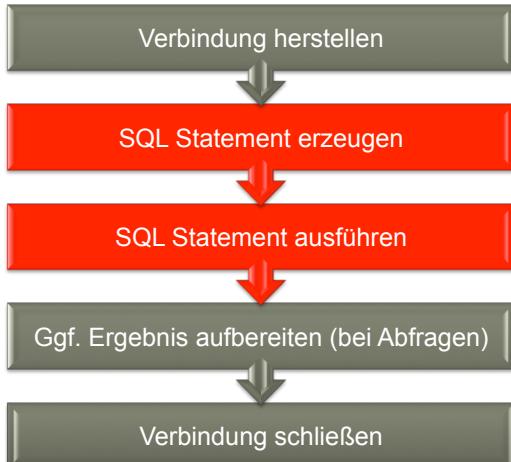
Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

40

## Prepared Statements



University of Applied Sciences



Eine weitere Art SQL Injections zu vermeiden, sind sogenannte **Prepared Statements**, die zu dem Geschwindigkeitsvorteile bieten und Eingaben inhärent prüfen.

Sie trennen die

- SQL Statementerzeugung
- Dateneinbettung
- und SQL Statement Ausführung.

## Prepared Statements



University of Applied Sciences

- Vorbereitete Anweisungen
- Die keine Werte sondern Platzhalter enthalten
- Sichere Methode zur Verhinderung von SQL Injections
  - Gültigkeit der Parameter wird vor Ausführung eines SQL Statements geprüft
  - Abfragestruktur wird von Parameter (Daten) getrennt
- Geschwindigkeitsvorteil, wenn Anweisungen mit unterschiedlichen Parametern mehrmals durchgeführt werden.
  - Z.B. mehrere gleich geformte Insert Statements
  - mit einer Vielzahl unterschiedlicher Daten (z.B. während Initial Befüllung oder Import einer DB).

## Prepared Statements und MySQL



University of Applied Sciences

Anweisung vorbereiten



Parameter binden



Anweisung ausführen



Ergebnisse binden



Ergebnisse holen

Mittels der **prepare** Methode lässt sich ein Prepared Statement Objekt vom Typ **mysqli\_stmt** erzeugen.

Das formulierte SQL-Statement enthält dabei noch keine Werte, sondern nur Platzhalter (?) – sogenannte Parameter des Prepared Statements.

Diese Platzhalter werden im nächsten Schritt mit Werten belegt (die Parameter werden hierzu an Variablen des Programms gebunden).

```
$stmt = $db->prepare(  
    "SELECT MatrNr, Name  
    FROM Studenten  
    WHERE SGID LIKE ?");
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

43

## Prepared Statements und MySQL



University of Applied Sciences

Anweisung vorbereiten



Parameter binden



Anweisung ausführen



Ergebnisse binden



Ergebnisse holen

Mittels der **bind\_param** Methode lassen sich Parameter in ein Prepared Statement einbinden.

```
$stmt->bind_param(  
    $format_str,  
    $p1, $p2, $p3, ...);
```

**\$format\_str** gibt hierbei pro Parameter an, welchen Typ die zu bindende Variable hat.

Zeichen	Bedeutung
i	Integer
d	Fließkommazahl (double)
b	BLOB (binary large object)
s	String

Der Format-String “**ssd**” gibt beispielsweise an, dass die ersten beiden Parameter vom Typ String sind und der dritte Parameter vom Typ double.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

44

## Prepared Statements und MySQL



University of Applied Sciences



Mittels der **execute** Methode wird nun das Prepared Statement mit eingebetteten Parametern an den DB-Server übertragen und ausgeführt.

Die Ergebnisse bleiben dabei solange auf dem Server bis sie vom Server mittels der **fetch** Methode geholt werden.

Ein typischer Aufruf sieht etwa wie folgt aus:

```

// Einbinden von Parameter in das Stmt
// Hier die Studiengangs-ID
$stmt->bind_param("s", $sgid);

// Ausführen des Statements
$stmt->execute();
  
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

45

## Prepared Statements und MySQL



University of Applied Sciences



Die Ergebnisse einer Query können an Ergebnis-Variablen im PHP Code gebunden werden.

Hierzu werden die Attribute der SELECT Klausel eines Prepared Statements Attribut für Attribut mittels der Methode **bind\_result** an Variablen gebunden.

```

$stmt = $db->prepare(  
    "SELECT MatrNr, Name  
    FROM Studenten  
    WHERE SGID LIKE ?");  
  
// Parameter binden  
// Statement ausführen  
  
$stmt->bind_result($matr, $name);
  
```

Mit jedem Aufruf von **fetch()** wird nun ein Ergebnis der Query in die gebundenen Ergebnis-Variablen geschrieben.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

46

## Prepared Statements und MySQL



University of Applied Sciences



Mit jedem Aufruf von **fetch()** wird nun ein Ergebnis der Query in gebundenen Ergebnis-Variablen geschrieben.

Üblicherweise ruft man **fetch()** so oft auf, bis es kein Ergebnis mehr liefert.

```

// Statement erzeugen
// Parameter binden
$stmt->execute();
$stmt->bind_result($matr, $name);

while ($stmt->fetch()) {
    echo "$matr, $name";
}

$stmt->close();
  
```

**fetch()** holt die Ergebnisse „Zeile für Zeile“ vom DB-Server. Diese ist zwar **spechereffizient**, jedoch bei großen Datenmengen **recht langsam**.

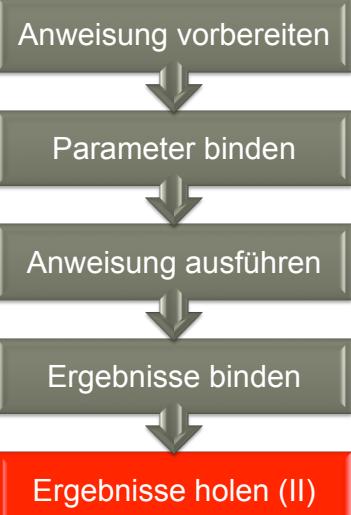
Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

47

## Prepared Statements und MySQL



University of Applied Sciences



**fetch()** holt die Ergebnisse „Zeile für Zeile“ vom DB-Server. Diese ist spechereffizient jedoch bei großen Datenmengen recht langsam.

Es können mit den Methoden **store\_result** und **free\_result** die Ergebnisse auch alle auf einmal geholt und im Speicher für schnelleren Zugriff zwischengespeichert werden. Dies ist bei großen Ergebnismengen häufig schneller, jedoch speicherintensiver.

**store\_result** holt alle Ergebnisdatensätze und legt sie im Hauptspeicher ab.

**free\_result** gibt den Hauptspeicher wieder frei.



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

48

## Prepared Statements und MySQL

FACH  
HOCHSCHULE  
LÜBECK  
University of Applied Sciences

```

graph TD
    A[Anweisung vorbereiten] --> B[Parameter binden]
    B --> C[Anweisung ausführen]
    C --> D[Ergebnisse binden]
    D --> E[Ergebnisse holen (III)]

```

**store\_result und free\_result** werden dabei gewöhnlicherweise so eingesetzt, dass sie den while fetch Block „klammern“.

```

// Statement erzeugen
// Parameter binden
// Statement ausführen
// Ergebnis-Variablen binden

$stmt->store_result();
while ($stmt->fetch()) {
    echo "$matr, $name";
}
$stmt->free_result();
$stmt->close();

```

← en bloc

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme | 49

## Prepared Statements Und jetzt alles zusammen

FACH  
HOCHSCHULE  
LÜBECK  
University of Applied Sciences

```

$stmt = $db->prepare("SELECT MatrNr, Name
                        FROM Studenten
                        WHERE SGID LIKE ?");

$sgid = "INF";
$stmt->bind_param("s", $sgid);

$stmt->execute();

$stmt->bind_result($matr, $name);

$stmt->store_result();
echo "<table>";
echo "<tr><th>Matrikelnr</th><th>Name</th></tr>";
while ($stmt->fetch()) {
    echo "<tr><td>$matr</td><td>$name</td></tr>";
}
echo "</table>";
$stmt->free_result();
$stmt->close();

```

Statement vorbereiten  
Parameter binden  
Statement ausführen  
Ergebnisse binden  
Ergebnisse holen (en bloc)

Welche Ausgabe erzeugt dieses Listing?

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme | 50

## Eine mit PHP und MySQL generierte Tabelle

FACH  
HOCHSCHULE  
LÜBECK  
University of Applied Sciences

Matrikelnummer Name

12345	Max Mustermann
1234	Maren Musterfrau
789456	Wilder Wahnsinn
567891	Tessa Loniki
111167	Holger Hilfreich
123478	Hanna Montana

Welche Eigenschaft haben diese Personen gemeinsam?

Gem. Code studieren sie alle INF ...

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

51

## Prepared Statements

FACH  
HOCHSCHULE  
LÜBECK  
University of Applied Sciences

- Sie haben Prepared Statements am Beispiel von SQL SELECT Queries kennen gelernt
- Für INSERT und UPDATE SQL Statements funktioniert das gezeigte Prinzip vollkommen analog

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme

52

## Zusammenfassung



University of Applied Sciences

- Logische Schichten von Datenbankanwendungen
- Mögliche Zugriffsarten auf Datenbanken mit PHP
- Datenbankzugriff am Beispiel der Bibliothek mysqli
- Prepared Statements
- SQL Injections (und wie man sich dagegen wappnet)