Datenbanken – Lab

PRAXISPROJEKT – DB-ZUGRIFFE AUS ANWENDUNGEN

Recap

- Theorie: DBMS, Datenmodelle, SQL, MariaDB
- Erstellen einer Datenstruktur (CREATE, ALTER, DROP
- Ändern von Datensätzen (INSERT, DELETE, UPDATE)
- Basics zu Normalisierung und referentielle Integrität
- Einstieg in Abfragen (SELECT FROM WHERE)
- Beispiel:
  - Tabellen students und classes zur Zuordnung der Studenten in den Kursen
  - Tabellen courses und students_in_course zur Zuordnung von Lehrveranstaltungen

Vorbereitung

Sofern noch nicht gemacht:

- Fügen Sie bitte die Lehrveranstaltungen aus Ihrem aktuellen Vorlesungsverzeichnis in die Tabelle courses ein (z.B. Angewandte Mathematik, Compilerbau, Systemnahe Programmierung, …)
- Fügen Sie außerdem noch einige Datensätze in die Tabelle student_in_course ein.
- Optional: Fügen Sie noch einige Studenten in students sowie ggf. einen oder zwei weitere Kurse in classes ein.

Todo: Einfache Abfragen

- 1. Nachname beginnt mit Buchstabe S.
- 2. Alle Kurse, deren Name das Wort "Grundlagen" enthält.
- 3. Alle Alumni (ehemalige Studenten).
- 4. Zeige den ersten Kurs in der Tabelle classes (nicht per WHERE id=1).
- 5. Alle Studenten, deren Nachname ein "e" enthält.
- 6. Alle Kurse mit einer ID größer 5.
- 7. Alle Studenten-IDs, die den Kurs mit der ID 3 besuchen.
- 8. Alle Studenten, die keinem Kurs zugeordnet sind.

Datenbanken in der Praxis

- Anwendungsbeispiele:
  - Online-Shops: Produkte, Benutezrdaten, Bestellungen
  - Bankensysteme: Transaktionen, Kontoinformationen, Historien
  - Content-Management-Systeme: Inhalte, Benutzerrollen, Kommentare
- Anforderungen:
  - Effizienz: Große Datenmengen schnell verarbeiten
  - Sicherheit: Schutz sensibler Daten vor unbefugtem Zugriff
  - Konsistenz: Keine widersprüchlichen Daten

Beispiel-Anwendung: Quiz-App

- Basis für die Anwendung: Quiz-App
  - Grundlegende Idee: Nutzer können Quizfragen beantworten und dabei Punkte sammeln. In der Bestenliste werden die besten Quizzer aufgelistet.
  - Rudimentäre Umsetzung ohne echt Quizfunktion → Fokus auf DB-Zugriffe
- Initiales DB-Schema:
  - Tabelle für users mit username und password_hash
  - Datensatz für Nutzer einfügen: admin, SHA2('password123', 256)
  - Wird später erweitert

Projektstruktur

```
quiz_app/
│
├── docker-compose.yml            # Docker-Compose-Konfigurationsdatei
│
└── html/                         # Haupt-Ordner, der Webanwendungs-Dateien enthält
    ├── public/                   # Öffentliche Dateien (Webzugriff)
    │   ├── index.php             # Einstiegspunkt / Home-Seite
    │   ├── login.php             # Login-Seite
    │   ├── logout.php            # Logout-Seite
    │   ├── dashboard.php         # Dashboard nach Login
    │   └── templates/            # HTML-Templates
    │       ├── header.php        # Header der Seiten
    │       └── footer.php        # Footer der Seiten
    │
    ├── src/                      # PHP-Quellcode (nicht direkt öffentlich)
    │   ├── auth.php              # Authentifizierungslogik für Login
    │   ├── db.php                # Datenbankverbindung (PDO)
    │   └── ConnectionPool.php    # Connection Pool (Datenbankverbindungen)
    │
    ├── sql/                      # SQL-Dateien (z. B. für Datenbank-Schemas)
    │   └── schema.sql            # SQL-Schema für die Datenbank
    │
    └── init.php                  # Initialisierungslogik (z.B. Session starten)
```

Umgebung aufsetzen

- Neue docker-compose für Quiz-App in Moodle
  - Herunterladen
  - In einem separaten Ordner quiz_app (o.ä.) über die Konsole ausführen: docker-compose up –d
  - Ggf. vorher andere Application stoppen (belegte Ports)
- DB-Nutzer:
  - root, root
  - user, password

DB-Treiber

- SW-Komponenten zur Kommunikation zwischen Anwendung und Datenbank
- Schnittstelle, um SQL-Anfragen an die Datenbank zu senden & Ergebnisse zu erhalten
- Verschiedene Treiber je nach Programmiersprache & DB (z.B. JDBC, ODBC)

→ Anwendung kann unabhängig von zugrunde liegender DB-Technologie arbeiten

- Treiber für PHP & MariaDB: PDO (PHP Data Objects):

  Einheitliche API für den Zugriff auf DBs wie MySQL, PostgreSQL, SQLite, MariaDB, …

PDO_MySQL-Treiber

- PHP-Anwendungen können mit Hilfe des Treibers
  - SQL-Anfragen an eine MariaDB senden
  - Ergebnisse von der MariaDB abrufen
- Aufgaben / Funktionen
  - Abstraktion der Datenbankzugriffe: konsistente API für den Zugriff auf Datenbanken (keine speziellen Funktionen für jede Datenbank)
  - Verbindungsaufbau und -verwaltung: Verbindung zur Datenbank, Abholen von Daten und Fehlerbehandlung
  - Sicherheit: Schutz vor SQL-Injection durch Möglichkeit für prepared Statements

Installation des Treibers

- Installation erfolgt hier direkt über die docker-compose
- Ausführung der Installationsanweisungen (command) in der Bash nach Containerstart
- Komponenten:
  - Paketliste aktualisieren:
    apt-get update
  - MariaDB-Bibliotheken installieren:
    apt-get install -y libmariadb-dev-compat libmariadb-dev
  - PDO und PDO_MySQL-Erweiterung installieren:
    docker-php-ext-install pdo pdo_mysql
  - Apache-Server im Vordergrund starten:
    apache2-foreground

Einfache Verbindung zur DB via PDO

Verbindungsaufbau in einem Try-catch-Block

- Im Try Verbindung mittels PDO und den benötigten Daten erstellen & zurückgeben:

```
$conn = new PDO("mysql:host=$server;dbname=$dbname", $username, $password);
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
return $conn;
```

- Im Catch für den Fehlerfall Ausführung stoppen und Benutzer benachrichtigen mittels die-Methode:

```
die("Verbindung fehlgeschlagen: " . $e->getMessage());
```

Abfrage machen

- Funktion für die Verbindung nutzen und Verbindung erstellen
- Beispiel: Abrufen aller Benutzer der Anwendung

```
$sql = "SELECT username FROM users";
$stmt = $conn->query($sql);
$user = $stmt->fetch(PDO::FETCH_ASSOC);
```

- Abrufen von Daten nach Benutzereingabe:

```
$sql = "SELECT * FROM users WHERE username=$username";
```

Was könnte hierbei das Problem sein?

SQL-Injections

- Was ist das Problem bei der Ausführung folgender Query?
  - SELECT * FROM users WHERE username='admin' AND password='' OR '1'='1';

    OR ist immer true → voller Zugriff auf admin-Account
  - Analog: SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '';

    Passwortprüfung wird umgangen → ggf. Zugriff auf alle Benutzerkonten / Daten
  - → SQL-Injection: Ungeprüfte Übernahme von Benutzereingaben
  - → Kann zu unbefugtem Zugriff, Manipulation / Diebstahl von Daten, ggf. Zerstörung der Datenbank (Verlust aller Daten) führen
- Lösung: Einsatz von Prepared Statements

Prepared Statements

- Vorgefertigte SQL-Statements mit Platzhaltern binden Benutzereingaben als Parameter statt sie direkt in die SQL-Anfrage einzufügen
- Vorteile:
  - Sicherheit: Schutz vor SQL-Injection, da Parameter automatisch "escaped" werden
  - Performance: Statement wird nur einmal geparst → mehrfache Ausführung mit unterschiedlichen Parametern möglich
  - Lesbarkeit: Trennung von SQL-Logik und Datenwerten

Beispiel: Prepared Statement

- Beispiel in PHP:

```
$stmt = $conn->prepare("SELECT * FROM users WHERE username = :name AND password = :pwd");
$stmt->bindParam(':name', $username);
$stmt->bindParam(':pwd', $password);
$stmt->execute();
$result = $stmt->getResult();
```

- Daten werden erst nach Abfrage-Analyse durch die DB eingefügt
- Benutzereingabe wird nicht als SQL-Code interpretiert sondern als reiner Wert gelesen

To Do: Login-Funktionalität [1]

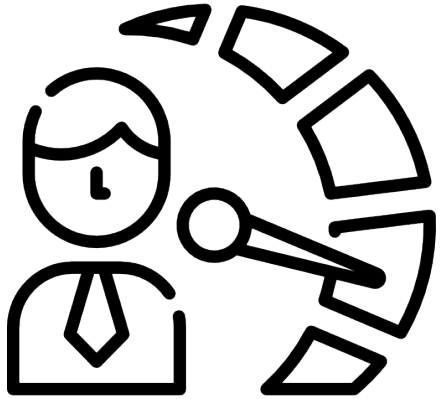
- Legen Sie das DB-Schema wie beschrieben an (users mit username & password_hash)
- Implementieren Sie die grundlegende Login-Funktionalität
  - Implementieren Sie eine Funktion für den Verbindungsaufbau zur Datenbank in db.php
  - Implementieren Sie eine Funktion für die Authentifizierung in auth.php
    - Aus DB password_hash zum übergebenen Username holen
    - Abgleich des übergebenen password_hash mit password_hash aus DB
    - Geben Sie true oder false zurück – je nach Erfolg
  - Nach Absenden des in der login.php enthaltenen Formulars:
    - Nutzer anhand der eingegebenen Daten versuchen zu authentifizieren
    - Im Erfolgsfall: Weiterleitung auf dashboard.php ⇔ Im Fehlerfall: Fehlermeldung ausgeben

To Do: Login-Funktionalität [2]

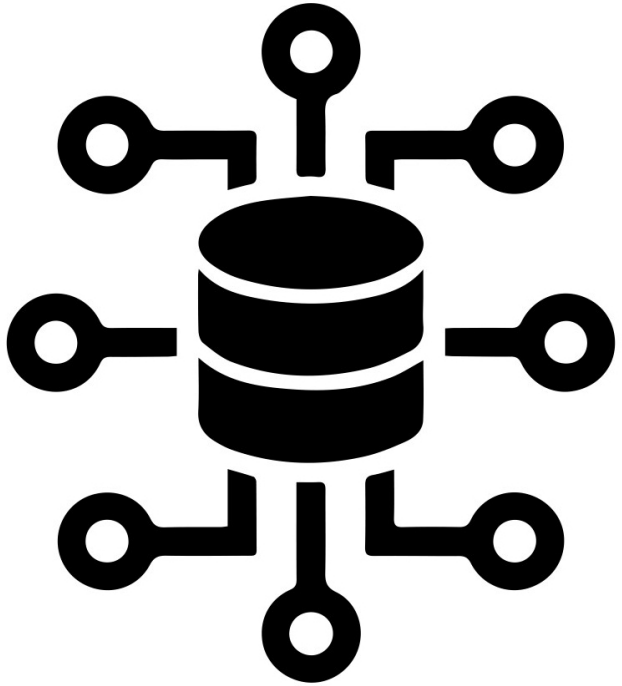
- Auf der dashboard.php begrüßen Sie den eingeloggten Nutzer mit seinem Namen und geben die Möglichkeit zum Logout via Button
- In der logout.php loggen Sie den Nutzer aus und leiten ihn anschließend weiter auf die login.php

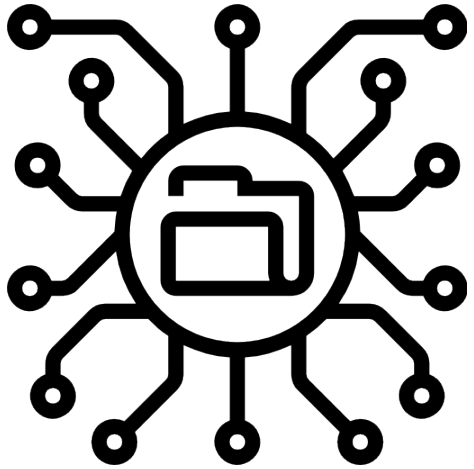
- Hinweise:
  - Im Projektordner ist die Stuktur angelegt sowie einige Basics und Kommentare eingefügt
  - Nutzen Sie die Templates header.php und footer.php, um die HTML-Struktur zu erhalten
  - Nutzen Sie Prepared Statements
  - Denken Sie an PHP-Konstrukte und Funktionen wie Session, require / include, header('location:...'), …

Herausforderungen



Performance bei
multiplen DB-Zugriffen





Abruf von
vielen Datensätzen

Connection Pooling

- Problem: Verbindungsaufbau ist zeitaufwändig
- Lösung: Wiederverwendung von Verbindungen aus einem Pool
- Vorteile: Performance, Ressourcenschonung, Skalierbarkeit
- Funktionsweise:
  - Beim Start der Anwendung wird ein Pool mit Verbindungen erstellt
  - Für Anfragen wird eine Verbindung aus dem Pool genutzt
  - Rückgabe der Verbindung in den Pool nach Nutzung

Beispiel: Connection Pool erstellen

- Implementierung vgl. Beispiel
- Kurze Erläuterung:
  - Implementierung des Pools als Klasse mit Singleton-Pattern und relevanten Funktionen
  - Einmalige Instanziierung des Pools in Methode getConnectionPool() in init.php
  - Einbindung der init.php auf jeweiligen Seiten & Zugriff auf Connections über Methode
    - Connection holen:

      ```
      $connectionPool = getConnectionPool();
      $connection = $connectionPool->getConnection();
      ```

    - Connection zurückgeben: $connectionPool->releaseConnection($connection);

Abrufen von Datensätzen: fetchAll

Möglichkeit 1: Alle Datensätze mittels fetchAll auf einmal holen

- Alle Datensätze gleichzeitig abgerufen & in Array geladen
- Vorteile: einfach, schnell, direkter Zugriff auf Daten
- Nachteile: Speicherintensiv → ggf. Performance-Probleme
- Fazit: Für kleine Datenmangen geeignet
- Beispiel:

```
$stmt = $conn->prepare("SELECT * FROM users");
$stmt->execute();
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

Abrufen von Datensätzen: fetch

Möglichkeit 2: Datensätze zeilenweise mittels fetch holen

- Jeder Aufruf von fetch() ruft einen Datensatz ab
- Vorteile: geringer Speicherverbrauch
- Nachteile: weniger effizient, fetch() muss ggf. mehrfach aufgerufen werden
- Fazit: Nützlich bei großen Datenmengen, Streaming-Daten oder wenn einzelne Datensätze nach und nach benötigt werden
- Beispiel:

```
$stmt = $conn->prepare("SELECT * FROM users");
$stmt->execute();
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    // Verarbeitung des Datensatzes }
```

Abrufen von Datensätzen: Pagination

Möglichkeit 3: Datensätze seitenweise auslesen mittels Pagination

- Abfragen werden in kleinere "Seiten" unterteilt (z.B. 10 oder 20 Datensätze pro Seite)
- Meist in Verbindung mit LIMIT und OFFSET verwendet
- Vorteile: effizient (Teilmenge abrufen), reduzierter Speicherverbrauch, vermeidet hohe Latenzzeiten
- Nachteile: Ausführung mehrerer Abfragen (zusätzlicher Overhead), ggf. Probleme bei dynamischen Datensätzen
- Beispiel: vgl. Quiz-App

Weitere Ansätze zum Abrufen von Daten

- Lazy Loading
- Eager Loading
- Batch Processing
- Asynchrone Abfragen

…

Erweiterung des Datenmodells

- Weitere Nutzer (mind. 4) in der Tabelle users hinzufügen
- Zusätzliche Tabelle für die Bestenliste hinzufügen
  - Tabelle leaderboard mit user_id, score und quiz_date
  - Fügen Sie 5 Datensätze ein
- Zusätzliche Tabelle für die Quizfragen hinzufügen
  - Tabelle questions mit id, question_text, option_a, option_b, option_c, option_d und correct_option
  - Fügen Sie die Beispieldatensätze über phpMyAdmin mit Hilfe des SQL-Skripts ein

To Do: Abrufen von Daten

- Leaderboard.php
  - Rufen Sie mit Hilfe einer Connection aus dem Pool die zugehörigen Daten ab
  - Nutzen Sie hierfür fetchAll, da die Bestenliste überschaubar ist
- Questions.php
  - Rufen Sie mit Hilfe einer Connection aus dem Pool die zugehörigen Daten ab
  - Nutzen Sie hierfür das zeilenweise Auslesen mittels Cursor über die fetch-Funktion
- Hinweis: In der questions_pagination.php finden Sie eine beispielhafte Implementierung für Pagination.