PHP 04

DATENBANKEN

INFO

- In den 70er Jahren entwickelte IBM die erste relationale Datenbank mit den Namen System R entwickelte dafür auch die Sprache SQL, damals SEQUEL
- 1986 veröffentlichte das US-Normungsinstitut ANSI erstmals einen entsprechenden Standard
- Im Folgejahr zog die internationale Standardisierungsorganisation (ISO) nach => Grundlage das SQL die am häufigsten verwendete Datenbanksprache sich durchsetzen konnte
- SQL Befehle in drei verschiedene Bereiche unterteilt
  - Data Definition Language (DDL): erlauben grobe Strukturen einer DB vorzugeben (Bsp.: `CREATE`, `DROP`, `Alter`)
  - Data Control Language (DCL): Zugriffsrechte. Kontrolle, welcher Anwender Zugriff auf Daten bekommt (`GRANT`, ...)
  - Data Manipulation Language (DML): beinhaltet alle Funktionen um Daten einzugeben, löschen, ändern (`SELECT`, `INSERT`, `UPDATE`, `DELETE`, ...)

PDO vs MySQLi

ZUSAMMENFASSUNG

- 3 verschiedene MySQL-APIs, für Verbindung zu MySQL-Datenbank aufzunehmen
  - ~~MySQL~~ (ACHTUNG: Veraltet!)
  - MySQLi
  - PDO (PHP Data Objects)
- API = Application Programming Interface -> Programmierschnittstelle, welche es verschiedenen Programmen erlaubt miteinander zu kommunizieren
- Auswahl der passenden API erfolgt nach persönlichen Vorlieben des Programmierers oder anhand der Eigenschaften der bereits bestehenden Anwendung
- Grobe Einteilung in Prozedurale und Objektorienteierte Programmierung
  - Prozedurale Programmierung bezeichnet die Programmierung von Algorithmen mittels einer sequentiellen Abfolge von Befehlen wobei im Gegensatz zur strukturierten Programmierung wiederverwendbare Teile in Funktionen (Prozeduren) ausgelagert werden.
  - Gerade bei großen Projekten ist die **objektorientierte Programmierung** (kurz: OOP) ein geeignetes Mittel, um einzelne Programmteile sauber zu verarbeiten. Dabei wird alles als Objekt angesehen. Jedes Objekt kann eigene Attribute (Eigenschaften) und Methoden haben. Attribute sind sozusagen Merkmale eines Objekts, etwa der Radius eines Kreises, während Methoden Funktionen sind, die an das Objekt gebunden sind, bspw. die Darstellung des Kreises. Oftmals werden Attribute als *privat* gekapselt, d. h. sie können nur durch Methoden des Objekts manipuliert werden.
- **verschiedenen APIs in sich exklusiv, d. h. es können nicht Funktionen von verschiedenen APIs gleichzeitig verwendet bzw. verschiedene APIs „gemischt“ werden**

Zusammenfassung

	PDO (PHP Data Objects)	MySQLi
Datenbanken	12 verschiedene	Nur MySQL
API	OOP	OOP + verfahrenstechnisch
Benannte Parameter	Ja	Nein

- https://wiki.selfhtml.org/wiki/Programmiertechnik/Programmierparadigma#Prozedurale_Programmierung
- https://wiki.selfhtml.org/wiki/PHP/Tutorials/Umstieg_von_der_veralteten_MySQL-API
- https://www.accentsconagua.com/articles/code/pdo-vs-mysqli-which-should-you-use.html

MySQLi objektorientiert

- Verbindung aufbauen

```php
<?php
$mysqli = new mysqli("localhost", "user", "password", "database");
if ($mysqli->connect_errno) {
    die("Verbindung fehlgeschlagen: " . $mysqli->connect_error);
}
?>
```

- SQL Query an Datenbank senden

```php
<?php
$mysqli = new mysqli("localhost", "user", "password", "database");
if ($mysqli->connect_errno) {
    die("Verbindung fehlgeschlagen: " . $mysqli->connect_error);
}

$sql = "UPDATE tabelle SET spalte = 'Wert' WHERE id = 1";
$mysqli->query($sql);
?>
```

- https://www.php-einfach.de/mysql-tutorial/crashkurs-mysqli/

Was sind Prepared Statements?

- Prepared Statements (dt. „vorbereitete Anweisungen“) = einsatzfertige Muster für Abfragen in SQL-Datenbanksystemen, die keine Werte für die einzelnen Parameter enthalten. Stattdessen arbeiten diese Anweisungsmuster (auch Anweisungstemplates genannt) mit Variablen bzw. Platzhaltern, die erst innerhalb des Systems durch die tatsächlichen Werte ersetzt werden – anders als bei der manuellen Eingabe, wo die Werte bereits zum Zeitpunkt der Ausführung eines Befehls zugeordnet sind.
- Der entscheidende Grund dafür, ist der Sicherheitsaspekt. Das wohl größte Problem an standardmäßigen Zugriffen auf Datenbanken ist die leichte Manipulation => SQL-Injection, bei der Code hinzugefügt oder angepasst wird, um an sensible Daten zu gelangen oder gänzlich die Kontrolle über das Datenbanksystem zu erlangen. Prepared Statements bieten eine solche Sicherheitslücke nicht, da ihnen erst innerhalb des Systems konkrete Werte zugeordnet werden.
- Weiterer Grund: Einmal analysiert und kompiliert, können diese vom jeweiligen Datenbanksystem im Anschluss immer wieder verwendet werden (mit den jeweiligen, abgeänderten Werten). Dadurch verbrauchen Prepared Statements wesentlich weniger Ressourcen und sind schneller als manuelle Datenbankabfragen, wann immer es um SQL-Aufgaben geht, die wiederholt ausgeführt werden müssen.

- https://www.ionos.at/digitalguide/websites/web-entwicklung/prepared-statements-in-phpmysql/
- https://www.php-einfach.de/mysql-tutorial/php-prepared-statements/

SQL-Injection

- => Ausnutzen einer Sicherheitslücke in relationalen Datenbankensystemen, die bei der Dateneingabe auf die Sprache SQL zurückgreifen. Oft tritt eine SQL-Injection in Zusammenhang mit PHP- und ASP-Programmen auf, die auf ältere Interfaces zurückgreifen.
- Mit gezielten Einsatz von Funktionszeichen schleust ein eigentlich unberechtigter Benutzer weitere SQL-Befehle ein und manipuliert die Einträge derart, dass er Daten verändern, löschen oder lesen kann. In gravierenden Fällen ist es sogar möglich, dass sich ein Angreifer auf diesem Wege den Zugriff auf die Kommandozeile des befehlsausführenden Systems und damit über den gesamten Datenbankserver verschafft

- https://www.ionos.at/digitalguide/server/sicherheit/sql-injection-grundlagen-und-schutzmassnahmen/

MySQLi & Prepared Statements

- Mit *bind_param()* Variable mit Parameter verbinden
- *ssi* sind die Typen der Parameter, gibt an, dass drei Parameter im Query sind, den ersten mit dem Typ **s**tring, den zweiten vom Typ **s**tring und den dritten vom Typ **i**nteger. Für Fließkommazahlen existiert noch der Wert *d*
- Werte den Variablen zuweisen
- mittels *$statement->execute()* wird der Prepared Statement an die Datenbank gesendet.

```php
<?php
$mysqli = new mysqli("localhost", "user", "Password", "database");
if ($mysqli->connect_errno) {
    die("Verbindung fehlgeschlagen: " . $mysqli->connect_error);
}
$sql = "UPDATE user SET email = ?, passwort = ? WHERE id = ?";
$statement = $mysqli->prepare($sql);
$statement->bind_param('ssi', $email, $passwort, $id);

//Variablen Werte zuweisen
$id= 1;
$email = "ein@beispiel.de";
$passwort = "neues passwort";
$statement->execute();
?>
```

DATENSÄTZE ABRUFEN

- Zum Abrufen und Ausgeben von Datensätzen existieren die Methoden $result->*fetch_assoc()* und *$result->fetch_object()*

```php
<?php
$mysqli = new mysqli("localhost", "user", "password", "database");
if ($mysqli->connect_errno) {
    die("Verbindung fehlgeschlagen: " . $mysqli->connect_error);
}
$id = 100;
$sql = "SELECT * FROM tabelle WHERE id < ?";
$statement = $mysqli->prepare($sql);
$statement->bind_param('i', $id);
$statement->execute();

$result = $statement->get_result();


while($row = $result->fetch_object()) {
    echo $row->spaltenname;
}

//Alternativ mit fetch_assoc():
while($row = $result->fetch_assoc()) {
    echo $row['spaltenname'];
}

?>
```

Datensatzabfrage ohne Parameter

- Da es keine Parameter gibt, kann `->query($sql)` verwendet werden ansonsten immer Prepared Statements verwenden

```php
<?php
$mysqli = new mysqli("localhost","root","","kundendaten");
if ($mysqli -> connect_errno) {
    die("Fehler: " . $mysqli -> connect_error);
}

$sql = "SELECT * FROM kunden";
$result = $mysqli->query($sql);

while($row = $result->fetch_assoc()) {
    print("Vorname: ".$row['kundenvorname'])."\n";
}
```

Anzahl der Zeilen

- Um Anzahl der betroffenen Zeilen von *UPDATE, DELETE* oder *INSERT*-Anweisungen zu erhalten, existiert das *Feld $statement->affected_rows* .

```php
<?php
$mysqli = new mysqli("localhost", "root", "", "php-einfach");
if ($mysqli->connect_errno) {
    die("Verbindung fehlgeschlagen: " . $mysqli->connect_error);
}

//Prepared statement mit INSERT/DELETE/UPDATE-Anweisung für $statement

$statement->execute();
$count = $statement->affected_rows;
echo $count;
?>
```

- Um die Anzahl der Zeilen einer SELECT-Anweisung zu erhalten, existiert das Feld $result->num_rows

```php
<?php
$mysqli = new mysqli("localhost", "root", "", "php-einfach");
if ($mysqli->connect_errno) {
    die("Verbindung fehlgeschlagen: " . $mysqli->connect_error);
}

//Prepared statement mit SELECT-Anweisung für $statement

$statement->execute();
$result = $statement->get_result();
$count = $result->num_rows;
echo $count;
?>
```

MySQL Fehlermeldung

- Sollte ein Query fehlschlagen, so erhalten wir die Fehlermeldung in dem Feld *$statement->error:*

```php
<?php
//Prepared statement in $statement
if(!$statement->execute()) {
    echo "Query fehlgeschlagen: ".$statement->error;
}
?>
```

ENDE

QUELLE: PHP & MYSQL FÜR EINSTEIGER
ISBN: 978-3-96645-009-6