1. SQL Übersicht und Datentypen

SQL (Structured Query Language) ist eine Datenbanksprache zum Erstellen, Abfragen und Bearbeiten von Daten in relationalen Datenbanken.

SQL Datentypen:

integer	ganze Zahl (positiv oder negativ)
serial	ganze positive Zahl, not null, auto_increment
varchar	(variabler) Charakterstring
datetime	Datum und/bzw. Zeit
float	Gleitkommazahl

Sonderbehandlung: NULL !=O

2. SQL Befehle

Die Datenmanipulationssprache besteht im Wesentlichen aus vier Primitiven: INSERT, SELECT, UPDATE und DELETE.
INSERT fügt neue Daten in die Datenbank ein, UPDATE verändert bereits vorhandene Datensätze, SELECT filtert Daten aus und DELETE löscht angegebene Daten.

Diese Befehle werden anhand folgender Beispiel-Tabelle erklärt:

```
TABLE Personen    id           INTEGER (unique, NOT NULL, primary key)
                  Zuname       VARCHAR(32) (NOT NULL)
                  Vorname      VARCHAR(32)
                  Funktion     VARCHAR(32)
                  Tel          VARCHAR(32)
                  Plz          VARCHAR(5)
                  Ort          VARCHAR(32) (unique)

TABLE Artikel     id           INTEGER (unique, NOT NULL, primary key)
                  Bezeichnung  VARCHAR(32) (unique, NOT NULL)
                  Preis        FLOAT (NOT NULL)
                  Gruppe       VARCHAR(32)
```

2.1. SELECT: Auswahl von Daten

```
SELECT * | column1,column2,.. FROM table
    [WHERE <bedingungsliste>]
    [GROUP BY column1,..]
    [ORDER BY column1,..
```

Beispiele:

```
SELECT * FROM artikel;
SELECT * FROM personen ORDER BY zuname,vorname;
SELECT zuname,vorname,ort FROM Personen WHERE Ort='wien' and funktion='leitung';
```

2.2. INSERT: Einfügen von Daten

Neue Datensätze (Zeilen) in eine Tabelle einzufügen.

```
INSERT INTO table (column1, column2, column3,...)
      VALUES (value1, value2, value3,...)
```

Anmerkung: Werden alle Felder einer Datenbank der Reihe nach eingefügt (value1,...) kann die Aufzählung der Spalten (column1,...) wegbleiben.

2.3. UPDATE: Aktualisierung vorhandener Datensätze

```
UPDATE table
      SET column1="value1", column2="value2",...
      [WHERE column0="value3"]

Column0 ist meistens die Spaltenbezeichnung für die ID.
```

2.4. DELETE: Löschen von Datensätzen

```
DELETE FROM table_name
      [WHERE column1= "value"]
```

ACHTUNG: bei numerischen Kriterien werden keine Hochkomma verwendet!

Eine Sammlung und Erklärung der SQL-Anweisungen sind in:
http://www.w3schools.com/sql/default.asp

3. Zugriff auf MySQL-Datenbank

3.1. Anlegen einer Datenbank und Import von Datensätzen

1. XAMPP(.bat): Start von Apache-Webserver und MySQL
2. localhost: Startseite von XAMPP (index.php im Verzeichnis C:\xampp\htdocs) wird angezeigt
3. Start von phpMyAdmin:
4. Anlegen der Datenbank dbZeltfest – Zeichensatz - und öffnen der Datenbank
5. Ausführen des SQL-Skripts createImportZeltfest.sql

3.2. Zugriff auf MySQL-Daten mittels PDO

PHP Data Objects oder kurz **PDOs** stellt eine Abstraktionsebene für den Datenbankzugriff dar und ermöglicht einen einheitlichen Zugang von PHP auf unterschiedliche SQL-basierte Datenbanken, wie zum Beispiel MySQL, PostgreSQL oder SQLite. Dabei wird unter anderem der Portierungsaufwand beim Umstieg auf eine andere Datenbank minimiert. Es wird nur der Datenbankzugriff abstrahiert, nicht die Datenbank selbst. Für die zu nutzende Datenbank wird ein datenbankspezifischer Treiber benötigt.

siehe „Rechte“ in phpMyAdmin

3.3. Prepared Statement

Ein Prepared Statement ist eine sogenannte vorbereitete Anweisung für ein Datenbanksystem. Die Anweisung enthält bei der Bearbeitung mittels des Parsers nur Platzhalter, keine Parameterwerte. Eventuelle injezierter Code wird also nicht interpretiert und kann somit auch nicht ausgeführt werden.

Mittels Prepared Statements können **SQL-Injections effektiv verhindert** werden, da das Datenbanksystem die Gültigkeit von Parametern prüft, bevor diese verarbeitet werden.

Soll ein Statement mit unterschiedlichen Parametern mehrere Male (z. B. innerhalb einer Schleife) auf dem Datenbanksystem ausgeführt werden, bringen Prepared Statements noch **zusätzlich einen Geschwindigkeitsvorteil**. Das Statement liegt schon vorübersetzt im Datenbanksystem und muss nur noch mit den neuen Parametern ausgeführt werden.

*Anmerkung: Bei **„dynamischem SQL“** müssen unbedingt prepared statements verwendet werden.*

statisches SQL: die Anweisung enthält nur feste Werte
z.B.: `SELECT * FROM artikel where preis>2;`

dynamisches SQL: die Anweisung bezieht sich auf einen zur Laufzeit einzugebenden Wert (z.B. über Textbox).

3.4. Öffnen der Datenbank mittels PDO

```
$user = 'root';
$pass = '';
$host = 'localhost';
$db_name = 'Datenbank Name';
try {
     $db = new PDO("mysql:host=$host;dbname=$dbname", $user,$pass);
}
catch       (Exception $fehler)    {
     echo "Datenbank nicht vorhanden! ";
     return;
}
```

3.5. SELECT: Auswahl von Daten

einen Datensatz auswählen:

```
$sh = "SELECT ...";
$res = $db->query($sh);
if ($res==NULL)
     echo "Problem mit der Tabelle";

else        {

     $row= $res->fetch();
     if (!$row)            {
            echo "Keinen Datensatz gefunden!";
            return;
     }
     // einzelne Felder mittels $row["feldname"] ansprechen
}
```

Loop über mehrere Datensätze:

```
$res = $db->query('SELECT * FROM artikel');
if ($res==NULL)
     echo "Problem mit der Tabelle";

else        {
     foreach ($res as $row) {

            // einzelne Felder des akuellen Datensatzes ($row) mittels $row[‘feldname’] ansprechen
     }
}
```

3.6. INSERT von Daten

INSERT: ohne prepared statement

```
try {
     $query = "insert into artikel (id,bezeichnung,preis,gruppe) ".
          "values($nid,\"g1\",8.88,\"XXX\")";
     $stmt= $db->prepare($query);
     $res = $stmt->execute();
}
catch(PDOException $e) {
     echo $e->getMessage();
}
```

INSERT: mit prepared statement---

```
try {
     // PREPARE ---
     $query = "insert into artikel (id,bezeichnung,preis,gruppe) ".
            "values(:id,:bezeichnung,:preis,:gruppe)";
     $stmt = $db->prepare($query);

     // 1. DATENSATZ ---
     $stmt->bindValue(":id",345);
     $stmt->bindValue(":bezeichnung","Bez1");
     $stmt->bindValue(":preis",7.77);
     $stmt->bindValue(":gruppe","ZZZ");
     $stmt->execute();

     // 2. DATENSATZ ---
     $stmt->bindValue(":id",346);
     $stmt->bindValue(":bezeichnung","Bez2");
     $stmt->bindValue(":preis",8.88);
     $stmt->bindValue(":gruppe","ZZZ");
     $stmt->execute();

     // bei wiederholten Eingaben mit konkreten Werten bringt das Prepared Statement nur
     // einen Geschwindigkeitsvorteil
     // bei Inserten von Benutzereingaben (z.B. von Textfeld) verhindert es noch dazu
     // SQL - Injection

}
catch(PDOException $e) {
     echo $e->getMessage();
}
```

3.7. DELETE von Daten

```
try {
     $query = "delete from artikel where id> :xid";
     $stmt = $db->prepare($query);
     $stmt->bindValue(":xid",$userinput);
     $res = $stmt->execute();
}
catch(PDOException $e) {
     echo $e->getMessage();
}

// $userinput: Benutzereingabe z.B. aus Textbox; da sind Prepared Statements unbedingt notwendig!
```

Anhang 1: Script-Vorlage zur Erzeugung einer Personen - Tabelle

```
DROP TABLE IF EXISTS personen;
CREATE TABLE personen
(
  id         SERIAL          NOT NULL,
  vorname    VARCHAR(32)  NOT NULL,
  zuname     VARCHAR(32)  NOT NULL,
  funktion   VARCHAR(32),
  tel        VARCHAR(32),
  plz        VARCHAR(5),
  ort        VARCHAR(32),
  PRIMARY KEY ( id )
);


INSERT INTO personen (id,zuname,vorname,funktion,tel,plz,ort)
VALUES (1,"Schneller","Gudrun","Service","0664-1819077","1040","Wien");

INSERT INTO personen (id,zuname,vorname,funktion,tel,plz,ort)
VALUES (2,"Gerber","Harald","Service","01-9077","1010","Wien");

INSERT INTO personen (id,zuname,vorname,funktion,tel,plz,ort)
VALUES (3,"Freitag","Franz","Service","0664-237865","3040","Neulengbach");

INSERT INTO personen (id,zuname,vorname,funktion,tel,plz,ort)
VALUES (4,"Dormann","Andrea","Leitung","0676-985428","3002","Purkersdorf");

INSERT INTO personen (id,zuname,vorname,funktion,tel,plz,ort)
VALUES (5,"Unger","Fitz","Leitung","01-7486","1180","Wien");

INSERT INTO personen (id,zuname,vorname,funktion,tel,plz,ort)
VALUES (6,"Hausner","Maria","Service","0680-9473210","1130","Wien");
```