MySQL lernen

http://mysql.lernenhoch2.de/lernen/

---

1. MySQL Einleitung

Ich erinnere mich noch, wie ich 2005 zum ersten Mal mit Datenbanken angefangen habe. Mit HTML und PHP war ich schon recht gut vertraut und konnte damit schon die ein oder andere Webseite zusammenbasteln. Doch das war mir irgendwann nicht mehr genug und ich wollte eine Webseite haben, auf der ich täglich Artikel zum Thema Webdesign schreiben konnte, um meine Fortschritte festzuhalten und anderen Fehler zu ersparen, die mir passiert sind.

Bevor ich auf das Thema Blogs und damit später zwangsläufig auf Wordpress gestoßen bin (was ultimativ in meinem **Webdesignblog** resultierte), habe ich versucht mein eigenes kleines Content Management System zu basteln und bin bei meiner Suche ständig auf das Thema Datenbanken gestoßen. Ich hatte damals keinerlei Vorstellung, wie man so eine Seite mit ganz ganz vielen Texten nur managen soll, für jede Seite eine HTML Datei anlegen? Oder die ganzen Texte in einzelne Textdateien auslagern, die dann von PHP an die entsprechenden Stellen geladen werden?

Die Lösung war natürlich eine Datenbank und da ich einen kleinen Shared Server mein Eigen nennen konnte, der von Haus aus mit 3 MySQL-Datenbanken kam, durchstöberte ich kurze Zeit später Tutorials zum Thema.

Warum möchtest du MySQL lernen?

Abgesehen davon, dass MySQL eine wirklich tolle (und kostenlose) Web-Datenbank ist, leicht zu erlernen und eine riesige Community hat, was ist

dein Grund MySQL zu lernen? Befindest du dich in der Ausbildung und “musst” es lernen? Hast du eine bislang statische Webseite die du mit einer Datenbank flexibler machen möchtest? Oder bist du wie ich damals an die Grenzen von HTML und PHP gestoßen und möchtest nun “größere” Seiten bauen, auf denen sich User registrieren, für Artikel voten und sich in einer Community austauschen können?

Was auch immer der Grund ist, ich hoffe dir mit diesem Kurs MySQL so einfach wie möglich vermitteln zu können. Solltest du etwas nicht verstehen, einen Fehler finden oder etwas auch nur minimal unklar sein, zöger nicht! Klick auf den Button auf der rechten Seite und schick mir dein Feedback, ich versuche dir so schnell wie möglich zu antworten, bzw. den Mangel zu beheben. Und jetzt wünsch ich dir viel Erfolg beim erlernen von MySQL.

1.1. Voraussetzungen für MySQL

Um MySQL nutzen zu können, brauchst du Kenntnisse in PHP. PHP ist einer der besten Skript-Sprachen und schon 2007 wurde PHP auf ca. 21 Millionen Domains [**Quelle**]. PHP wird unter anderem auch von ein paar sehr großen und bekannten Seiten wie Facebook und Wikipedia [**Quelle**] verwendet.

Also, für MySQL brauchst du PHP Kenntnisse, wenn du interessiert bist kannst du hier **PHP lernen**

1.2. Wofür braucht man eine Datenbank/MySQL?

Wie der Name schon andeutet, kann man in einer Datenbank Daten sammeln. Häufige Anwendungsfälle für eine Datenbank sind:

- **Registrierungen auf einer Webseite**: Wenn sich Besucher auf deiner Website registrieren und einloggen können, musst du ihre Logindaten irgendwo speichern, eine Datenbank bietet sich da für an
- **CMS (Content Management System)**: Eine Webseite, die ein Content Management System beinhaltet, hat häufig sehr viele Textinhalte. Die können prima in einer Datenbank abgelegt werden. Das hat unter anderem auch den Vorteil, dass man die Inhalte vom Design der Seite getrennt hat. Wenn also mal wieder ein Redesign ansteht, muss man nur das Design ändern und die Texte bleiben in der Datenbank unberührt
- **Inhalte generell**: Bastelt man an einer Fragen-Seite (siehe z.b.: **gutefrage.net**), kann man davon ausgehen mit der Zeit sehr viele Fragen und Antworten zu haben. Diese kann man prima in einer Datenbank ablegen.

Kleine Portfolio Seiten, bzw Webseiten mit nur ein paar HTML-Dateien kommen natürlich auch gut ohne eine Datenbank aus. Wenn man aber sehr viele Inhalte hat bzw erwartet, sollte man eine Datenbank in Erwägung ziehen.

Und warum MySQL? Gibt es da nichts anderes?

Doch, MySQL ist nur eine von vielen Datenbanken, aber im Internet weit verbreitet. Ausserdem ist sie kostenlos und geht Hand in Hand mit PHP. PHP und MySQL stehen bei den meisten Webhosting Paketen zur Verfügung, damit kann man auch mit kleinem Geldbeutel Großes erreichen.

Noch nicht verstanden wofür eine Datenbank nützlich ist? Dann frag einfach nach!

1.3. Unterschied von SQL und MySQL

SQL ist eine Datenbank-Abfrage-Sprache. Wir haben also Daten in einer Datenbank und mit SQL können wir Daten selektieren, neue Daten eintragen, vorhandene Daten aktualisieren oder löschen. MySQL ist nun der Weg, mittels PHP, SQL-Abfragen (Queries) auszuführen.

Schau dir aber auch das **SQL Tutorial** an, um alle Möglichkeiten von SQL kennenzulernen. Wenn du dir die Grundlagen von MySQL angeschaut hast, ist es ein leichtes auch die anderen SQL-Befehle in MySQL zu nutzen.

---

2. MySQL Anfänger

2.1. Datenbank erzeugen – Schnellstart mit PHPMyAdmin

Um MySQL mit PHP auf deinem eigenen Computer zu testen, brauchst du einen eigenen Server. Schau dir diesen Teil des **PHP Tutorial** an: **Dein eigener Server**. Installiere XAMPP oder XAMPP lite und du kannst PHP Skripte auf deinem Rechner ausführen und mit PHPMyAdmin Datenbanken anlegen, SQL ausführen, etc.

Server starten, Datenbank anlegen

Nachdem du deinen Server installiert hast, musst du ihn noch starten. Danach gib, in dem Browser deiner Wahl, folgendes als URL ein: **http://localhost/xampp/**. Damit gelangst du auf die XAMPP Startseite und im linken Menü unter **Tools** findest du den Link **PHPMyAdmin**.

Suche nun, in dem ganzen wirrwarr, den Text **Create new database**. Darunter befindet sich ein Eingabefeld in welches du den Namen deiner Datenbank reinschreibst. Nehmen wir als Namen einfach **tutorial**, das Feld **Collation** kannst du erstmal ignorieren.

Die nächsten Schritte

Glückwunsch, du hast deine erste eigene Datenbank angelegt. Doch das war erst der Anfang. In den folgenden Teilen wirst du in deiner Datenbank Tabellen anlegen, Werte eintragen und schonmal mit SQL ein bisschen rumspielen. Danach gehts dann direkt mit MySQL los. Wir werden mit PHP eine Verbindung zu unserer Datenbank herstellen und die Hauptoperationen von MySQL kennenlernen (SELECT, INSERT, UPDATE, etc.).

2.1.1. Tabelle anlegen

Der nächste Schritt erfolgt gleich nach dem Anlegen der Datenbank. Unter dem Text **Create new table…** befindet sich ein Eingabefeld, in welches wir den Namen unserer Tabelle eintragen. Beim Namen von Tabellen, Spalten, etc. haben wir an sich freie Hand, solange wir nicht die von MySQL reservierten Wörter nutzen: **Reservierte Wörter in MySQL**. In PHPMyAdmin kannst du zwar reservierte Wörter nutzen ohne eine Fehlermeldung zu erhalten, ich rate allerdings dringenst davon ab, denn es ist eine vorprogrammierte Fehlerquelle.

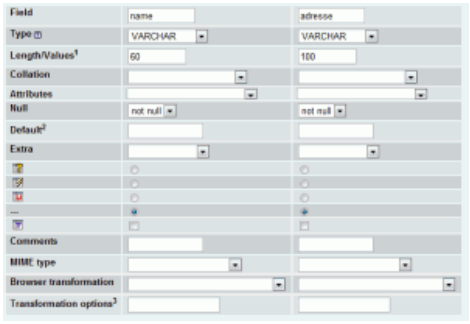
Wie auch immer, wir wollen unsere erste Tabelle in unserer Datenbank **kunden** nennen. Gib also als Tabellennamen **kunden** ein und bei “number of fields” den Wert **2**. Die 2 Felder müssen nun im folgenden Formular korrekt befüllt werden, erst danach wird unsere Tabelle erzeugt.

MySQL Spalten anlegen

Jede Tabelle kann beliebig von mindestens einer bis zu beliebig vielen Spalten haben. Die Spalten sind unsere Typen, also wenn wir eine Tabelle haben möchten, in der nur Namen gespeichert werden, dann brauchen wir nur eine Spalte mit der Bezeichnung **namen** und können darin beliebig viele Namen eintragen. Oder wir haben eine Tabelle **termine**, in die wir den Zeitpunkt des Termins eintragen (Spaltenname “datum”), eine Beschreibung des Termins (Spaltenname “beschreibung”) und den Ort wo der Termin stattfindet (Spaltenname “ort”). Je nachdem wieviele Werte wir in der Tabelle festhalten möchten, erhöht sich die Anzahl der benötigten Spalten.

Das Tabellen Formular ausfüllen

Für unsere Kunden-Tabelle hingegen brauchen wir erstmal nur 2 Spalten: **name** und **adresse**. Befülle das Formular wie im Screenshot gezeigt:



Beide Spalten sind vom **Type VARCHAR**, dass heißt die Spalten erwarten Strings als Input. Die Eingabe **Length/Values** ist abhängig vom Typ. Bei VARCHAR muss dort ein Wert eingetragen werden und zwar **eine Zahl zwischen 1 und 255**. Die Zahl gibt an, wie lang der jeweilige String jeweils sein darf. Der Name unserer Kunden darf nicht länger als 60 Zeichen sein, die Adresse nicht länger als 100 Zeichen. Die restlichen Werte kannst du erstmal ignorieren.

So, jetzt hast du schon eine Datenbank + Tabelle, dann müssen jetzt nur noch **Werte rein**.

2.1.2. Werte eintragen

Nun solltest du dich in der Tabelle “kunden” befinden und die beiden Spalten “name” und “adresse” sehen. Ist das nicht der Fall, wähle unter PHPMyAdmin im linken Menü deine Tabelle “tutorial” aus und klicke danach, ebenfalls im linken Menü, auf die Tabelle “kunden”.

Werte eintragen mit PHPMyAdmin

Als nächstes tragen wir mit PHPMyAdmin ein paar Werte in unsere MySQL Datenbank ein, dazu klicke oben auf den Reiter “Insert”. Je nach

Tabelle werden nun unterschiedlich viele Eingabefelder angezeigt, in unserem Falle 2. Standardmäßig gibt PHPMyAdmin immer 2 Formulare aus, für den Fall, dass du nicht nur einen sondern zwei Datensätze eintragen möchtest. Das nutzen wir direkt mal und tragen im ersten Formular, im ersten Feld, den Wert “Max Mustermann” ein. Im Feld “adresse” nehmen wir die Straße “Musterstraße 15″. Im zweiten Formular tragen wir den Namen “Peter Klein” ein und bei Adresse “Daten Alle 27″. Mit einem Klick auf den Button “Go” werden die Daten eingetragen, vorausgesetzt wir haben nichts falsch gemacht.

Als Ergebnis solltest du in etwa folgendes erhalten:

Du siehst hier den SQL-Code, den PHPMyAdmin für dich ausgeführt hat, um die Werte einzutragen. Im nächsten Teil wirst du sehen wie das geht, indem wir unsere eingetragenen Werte selektieren.

2.1.3. Bestimmte Werte selektieren (mittels SQL)

So, Datenbank anlegen, Tabellen erzeugen und Werte eintragen, soweit so gut. Bevor wir uns auf MySQL stürzen, werden wir erstmal kurz mit SQL etwas rumspielen. In PHPMyAdmin (wenn du deine Tabelle ausgewählt hast), findest du einen Reiter “SQL”. Klick drauf und du siehst ein großes Textfeld indem schon etwas SQL Code zu sehen ist:

```
1 SELECT * FROM `kunden` WHERE 1
```

Klick unten auf den Button “Go” und dir sollten unsere beiden eingetragenen Datensätze ausgegeben werden:

	name	adresse
	Max Mustermann	Musterstraße 5
	Peter Klein	Daten Allee 27

Der SQL-Befehl erklärt sich folgendermaßen:

- mit dem Schlüsselwort **SELECT** sagst du der Datenbank: “Hey, ich möchte jetzt ein paar Daten selektieren”
- mit dem ***-Zeichen sagst: “ich möchte das alle Spalten beim SELECT berücksichtigt werden”. Alternativ könntest du auch anstatt dem * einfach nur “name” schreiben, dann würden nur die Namen berücksichtigt werden oder “adresse”, dann nur die Adressen.
- mit **FROM** beziehst du dich auf die Tabelle. Normalerweise hat eine Datenbank mehr als eine Tabelle und muss wissen, von welcher Tabelle du Werte selektieren möchtest.
- als nächstes folgt die Tabelle, in unserem Fall “kunden”. Du kannst diese sogenannten **Backticks (`)** auch weglassen. Im Teil über **Tabelle in PHPMyAdmin anlegen** hab ich schon angedeutet, dass man auch von MySQL reservierte Wörter als Tabellennamen nutzen kann ohne einen Fehler zu erhalten. Das geht aber nur, wenn man bei allen Operationen (SELECT, INSERT, DELETE, etc.) den Tabellenname in Backticks fasst.
  Einerseits ist es guter Programmierstil, wenn man einfach immer die Backticks setzt. So kann man sich eine Fehlersuche von Stunden sparen (*Schande über mein Haupt*), andererseits ist es Geschmackssache. Ich hab es mir damals ohne Backticks angewöhnt und weiß mittlerweile, wo ich den Fehler suchen muss, wenn ich mal wieder ein reserviertes Wort verwendet habe. In diesem Tutorial werde ich auf Backticks verzichten, dass heißt aber nicht, das du das auch machen sollst, mach es wie es dir lieber ist.

- “WHERE 1″ kann man auch weglassen. Mit “WHERE” grenzt man normalerweise das Ergebnis ein, also anstatt alle Kunden zu selektieren, nur die die mit dem Buchstaben “M” beginnen oder ähnliches. “WHERE 1″ grenzt aber nichts ein und selektiert einfach alle, daher kann man es weglassen.

SELECT Auswahl eingrenzen

Da wir nicht mehr Tabellen haben, von denen wir Werte selektieren können, werden wir unsere Selektion mit WHERE eingrenzen:

```
SELECT * FROM kunden WHERE name = 'Max Mustermann'
```

Um dieses SELECT-Beispiel zu rechtfertigen, stell dir eine Tabelle mit 5000 Kundennamen vor und du möchtest prüfen, ob der Herr Max Mustermann immernoch Kunde bei dir ist (deine Sekretärin löscht Kunden, wenn sie zur Konkurrenz wechseln). Wenn Max Mustermann vorhanden ist, wird dir der Datensatz ausgegeben. Prüfe was passiert, wenn du deinen Namen selektieren möchtest (ausser du heißt “Max Mustermann” oder “Peter Klein”).

du solltest folgende Meldung erhalten:

```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0005 sec)
```

heißt soviel wie: “nix gefunden was auf deine Suche zutrifft”, genau was wir erwartet haben. Die WHERE-Klausel ist schnell erklärt:
Als erstes gibst du den Spaltennamen an, indem gesucht werden soll (bei uns “name”). Danach sagst du nach welchem Wert gesucht werden soll. Ähnlich wie bei den PHP Vergleichsoperatoren sind wir hier nicht auf einen direkten Vergleich beschränkt. Hätten wir eine Spalte mit dem Alter unserer Kunden, könnten wir nach allen Kunden suchen die unter 40 Jahre oder über 50 Jahre alt sind.

Wie es weitergeht

Im nächsten Teil werden wir eine mittels PHP eine Verbindung zur Datenbank herstellen, die Verbindung brauchen wir um danach unsere ersten Queries mittels PHP auszuführen.

2.2. Verbindung zur Datenbank herstellen und trennen

Jetzt gehts los, wir greifen mit PHP auf unsere Datenbank zu und führen SQL Befehle aus, kurz: MySQL.
Erstmal müssen wir eine Verbindung zur Datenbank herstellen, das geht mit folgendem PHP-Code:

```
<?php
    mysql_connect("localhost", "Benutzername","Passwort");
    mysql_select_db("Datenbank-Name");
?>
```

Erklärung:
Der PHP-Befehl “mysql_connect” öffnet eine Verbindung zu einem MySQL-Server. “localhost” ist in der Regel der richtige Wert, ausser dein PHP-Skript liegt auf einer anderen Domain, also du möchtest von “domain-a.de” (auf der dein Skript liegt) auf eine Datenbank von “domain-z.de” zugreifen, dann müsstest du anstatt “localhost” den Wert “domain-z.de” eintragen.
Benutzername und Passwort sind die Datenbank-Logindaten, die werden meistens automatisch generiert, wenn du die Datenbank anlegst. Wenn du PHPMyAdmin auf deinem Computer installiert hast, ist der default-Wert für den Benutzernamen “root” und das Passwort ist nicht gesetzt, also:

```
<?php
    mysql_connect("localhost", "root","") or die ("Verbindung nicht möglich");
?>
```

Mit dem PHP-Befehl “mysql_select_db” wählen wir die Datenbank aus, auf die wir zugreifen möchten. Dein MySQL-Server kann ja auch mehr als 1 Datenbank haben, deshalb müssen wir das noch mit angeben. Mit diesen beiden Befehlen würde eine Verbindung zur Datenbank stehen. Du kannst die Verbindungsdaten noch in einer Variablen speichern, ist aber erstmal nicht nötig.

Fehler abfangen

Wir sollten allerdings noch für den Fall, dass die Verbindung nicht klappt, eine Fehlermeldung ausgeben:

```
<?php
    mysql_connect("localhost", "Benutzername","Passwort")
        or die ("Verbindung nicht möglich");

    mysql_select_db("Datenbank-Name")
        or die ("Datenbank existiert nicht");
```

```
?>
```

Dadurch kannst du schnell den Fehler finden, falls du einen Wert falsch eingetragen hast oder die Datenbank nicht existiert.

Verbindungsdaten speichern und MySQL Verbindung schließen

Normalerweise musst du nicht explizit die Verbindung wieder schließen, denn das erfolgt automatisch am Ende des Scripts. Wenn du die Verbindung früher beenden möchtest, brauchst du erstmal eine Variable die die Verbindung speichert:

```php
<?php
    $link = mysql_connect("localhost", "Benutzername","Passwort")
        or die ("Verbindung nicht möglich");
    mysql_select_db("Datenbank-Name")
        or die ("Datenbank existiert nicht");

    mysql_close($link); //schließt die Verbindung zur Datenbank
?>
```

connect.php – Verbindungsdaten speichern

Da man häufig mehrere PHP-Skripte hat, die alle auf die Datenbank zugreifen, kann es sinnvoll sein die Verbindungsdaten in der Datei **connect.php** auszulagern und bei Bedarf einzubinden:

connect.php

```php
<?php
    mysql_connect("localhost", "Benutzername","Passwort") or die ("Verbindung nicht möglich");
    mysql_select_db("Datenbank-Name") or die ("Datenbank existiert nicht");
?>
```

Damit kannst du nun in jedem Skript eine Verbindung zu deiner Datenbank aufbauen. Jetzt wollen wir mal schauen, wie wir mit **MySQL Daten aus unserer Datenbank selektieren**.

2.3. SELECT – Daten selektieren und ausgeben

Im Artikel **Werte selektieren mit SQL** hast du schon erste Erfahrungen mit SQL gemacht und ich kann dir sagen, viel schwerer als das wird es nicht :)

MySQL ist quasi SQL in PHP gepackt, dazu ein Vergleich:

SQL

```sql
SELECT * FROM kunden
```

PHP

```php
<?php
    //Verbindung zur Datenbank herstellen
    mysql_connect("localhost", "Benutzername","Passwort") or die ("Verbindung nicht möglich");
    mysql_select_db("Datenbank-Name") or die ("Datenbank existiert nicht");

    //Alle Kundendaten selektieren
    $result = mysql_query("SELECT * FROM kunden");
?>
```

Das wirkt natürlich jetzt erstmal als wäre es sehr viel mehr Code, doch im Grunde ist nur **Zeile 7** entscheidend, denn dort befindet sich unser SQL-Query verpackt im PHP-Befehl “mysql_query”. Wenn wir obiges PHP-Skript ausführen, wird uns aber erstmal nichts ausgegeben. Das liegt daran, weil unser SELECT-Ergebnis in der Variablen “$result” gespeichert wird. “SELECT” selektiert schließlich nur die Daten, ausgeben müssen wir sie schon selbst.

SELECT Daten ausgeben

Jetzt haben wir schonmal die Daten in unserer Variable “$result”, doch wie können wir sie jetzt noch anzeigen? Dazu brauchen wir “mysql_fetch_row” und eine Schleife, die alle Einträge in $result durchläuft:

```
01 <?php
02     //Verbindung zur Datenbank herstellen
03     mysql_connect("localhost", "root","") or die ("Verbindung nicht möglich");
04     mysql_select_db("tutorial") or die ("Datenbank existiert nicht");
05 
06     //Alle Kundendaten selektieren
07     $result = mysql_query("SELECT * FROM kunden");
08 
09     while($row = mysql_fetch_row($result))
10         echo $row[0].' - '.$row[1].'<br />';
11 ?>
```

In Zeile 9 nutzen wir “mysql_fetch_row” in einer Schleife, um alle Datensätze zu durchlaufen. “mysql_fetch_row” holst sich jeweils eine Reihe aus $result und speichert diese in der Variablen “$row”. Nun können wir auf $row wie auf ein Array zugreifen. Da wir bislang nur zwei Spalten in unserer Tabelle “kunden” haben, nutzen wir auch nur “$row[0]” für den Kundennamen und “$row[1]” für die Kundenadresse:

Max Mustermann - Musterstraße 5
Peter Klein - Daten Allee 27

Weitere mysql_fetch Varianten

Es gibt neben “mysql_fetch_row” noch andere Möglichkeiten, Daten aus einem SQL-Ergebnis zu ziehen:

- **mysql_fetch_array** – abhängig vom Parameter, kann man nun anstelle von “$row[0]” auch “$row['name']” nutzen: **Beispiele**
- **mysql_fetch_assoc** – gibt ein Array mit benannten Feldern aus, also anstelle von “$row[0]” muss man nun $row['name'] nutzen
- **mysql_fetch_object** – dein Ergebnis wird nicht als Array, sondern als Object gespeichert. Deshalb greifst du auf die Werte auch anders zu. $row[0] bzw. $row['name] werden dann zu “$row->name” und die Kundenadresse zu “$row->adresse”: **Beispiele**
- **mysql_fetch_field** – diese Variante wirst du eher seltener brauchen, denn damit holst du dir nicht nur den Wert der Spalte, sondern auch noch alle Spalteninformationen. Also z.b. den Typ der Spalte, die maximale Länge, ob die Spalte der Primary Key ist, usw.
- **mysql_fetch_lengths** – ebenfalls eher seltener benötigt. Anstelle der Werte kannst du dir die Zeichenlänge des Wertes ausgeben lassen: **Beispiel**

Zusammenfassung

In diesem Teil hast du folgendes kennengelernt:

- **mysql_query** – Da packen wir unseren SQL Code rein. Bei einem SELECT-Query sollte das Ergebnis in einer Variablen gespeichert werden
- **mysql_fetch_row** – Mit dieser oder anderen Varianten holen wir die selektierten Datensätze aus der Variable heraus
- Mit einer Schleife können wir alle selektierten Datensätze nacheinander bearbeiten und ausgeben

Bislang haben wir nur zwei Datensätze zum selektieren, das wird langsam langweilig, deshalb werden wir im nächsten Teil mit dem SQL-Befehl **INSERT neue Werte in unsere Datenbank eintragen**.

2.4. INSERT – Daten einfügen

Mit **SELECT** können wir Werte aus der Datenbank selektieren und zum eintragen nutzen wir INSERT. Als erstes möchten wir einen weiteren Kunden in unsere Datenbank eintragen:

```
1 <?php
2     //Verbindung zur Datenbank herstellen
3     mysql_connect("localhost", "Benutzername","Passwort") or die ("Verbindung nicht möglich");
4     mysql_select_db("Datenbank-Name") or die ("Datenbank existiert nicht");
5 
6     $result = mysql_query("INSERT INTO kunden (name, adresse) VALUES ('Hans Meier', 'Wuppernweg 19')");
7 ?>
```

Eigentlich alles wie gewohnt, nur das der Teil in “mysql_query” nun anstelle eines SELECT’s ein INSERT ist. Schauen wir uns den SQL-Befehl mal genauer an:

```sql
INSERT INTO kunden (name, adresse) VALUES ('Hans Meier', 'Wuppernweg 19')
```

- **INSERT INTO kunden** – in welche Tabelle soll was eingetragen werden
- **(name, adresse)** – die Reihenfolge der Spalten, wie die Werte eingetragen werden sollen. Man kann sich das anordnen wie man mag, ich halte mich aber grundsätzlich an die Reihenfolge, die die Tabelle hat.
- **VALUES (‘Hans Meier’, ‘Wuppernweg 19’)** – hier kommen nun die Werte, die eingetragen werden sollen. Dabei ist die Reihenfolge abhängig von der Reihenfolge die man vorher festgelegt hat. Strings werden generell in Hochkommata gefasst, Zahlen nicht (ausser man möchte die Zahl als String speichern)

Führe das Skript mal aus (achte auf die korrekten Verbindungsdaten) und du solltest einen 3. Datensatz in deiner Tabelle “kunden” vorfinden.

INSERT mit Variablen

Natürlich müssen wir nicht jedes INSERT per Hand befüllen, wir können auch Variablen nutzen:

```php
<?php
    $kunde = 'Michael Weisel';
    $adresse = 'Charlottenstr. 7';

    $result = mysql_query("INSERT INTO kunden (name, adresse) VALUES ('".$kunde."', '".$adresse."')");
?>
```

Wenn man Variablen benutzt, darf man aber die Hochkommata nicht vergessen. Ausserdem muss man den String mit **“** beenden, dann die Variable mit **.** konkatenieren und dann den SQL String mit **“** wieder öffnen. Ist anfangs etwas verwirrend, aber man gewöhnt sich schnell dran.

Um im nächsten Teil WHERE – Auswahl eingrenzen ein paar mehr Daten haben, führe jetzt folgenden SQL-Query aus:

```sql
INSERT INTO kunden (name, adresse) VALUES ('Michael Meier', 'Saagengasse 13');
INSERT INTO kunden (name, adresse) VALUES ('Berta Brecht', 'Kalossenweg 8');
INSERT INTO kunden (name, adresse) VALUES ('Lilly Lagerfeld', 'Blumenweg 19');
INSERT INTO kunden (name, adresse) VALUES ('Peter Paulus', 'Kirchenstrasse 22');
INSERT INTO kunden (name, adresse) VALUES ('Sarah Seil', 'Roteneck 12');
INSERT INTO kunden (name, adresse) VALUES ('Adam Aldenau', 'Grüner Weg 5');
INSERT INTO kunden (name, adresse) VALUES ('Emil Entenich', 'Hinterhausen 12');
```

So, mit 10 Einträgen in der Datenbank sollte es ein bisschen mehr Spass mit **WHERE** machen.

2.5. WHERE – Auswahl eingrenzen

Mit dem **WHERE**-Befehl können wir unsere Auswahl bei einem SELECT begrenzen:

SQL

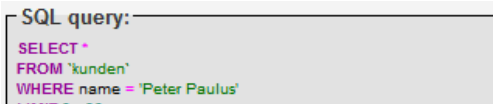
```sql
SELECT * FROM `kunden` WHERE name = 'Peter Paulus'
```

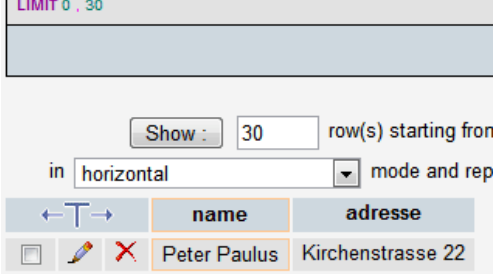
PHP

```php
<?php
    //Verbindung zur Datenbank herstellen
    mysql_connect("localhost", "Benutzername","Passwort") or die ("Verbindung nicht möglich");
    mysql_select_db("Datenbank-Name") or die ("Datenbank existiert nicht");

    //Alle Kundendaten selektieren
    $result = mysql_query("SELECT * FROM `kunden` WHERE name = 'Peter Paulus'");
?>
```

Ergebnis:





Anstelle einfach alle Einträge zu selektieren, können wir mit WHERE auswählen, welche Einträge wir haben wollen. Dabei können wir die einzelnen Spalten einer Tabelle ansprechen und nur Werte rausholen, die bestimmten Kriterien entsprechen. Möchtest du z.b. aus unserer Kunden-Tabelle alle Kunden, deren Name mit “M” beginnt, hilft folgender SQL-Query weiter:

```
1 SELECT * FROM  `kunden` WHERE name LIKE  'M%'
```

Da das “=”-Zeichen nur für genau zutreffende Werte steht, nutzen wir hier “LIKE”. Zusätzlich müssen wir mit dem Wildcard-Operator rumbasteln, also der erste Buchstabe soll ein “M” sein, alles was danach kommt ist egal (%). Wenn einfach irgendwo im Namen des Kunden ein “M” vorkommen soll, sieht der Query so aus (“M” von “%” eingeschlossen):

```
1 SELECT * FROM  `kunden` WHERE name LIKE  '%M%'
```

Leider haben wir in unserer Kundentabelle keine Zahlen, denn mit WHERE könnte man z.b. alle Kunden selektieren, die jünger als 50 sind:

```
1 SELECT * FROM  `kunden` WHERE alter < 50
```

(funktioniert natürlich nicht, da wir keine Spalte “Alter” haben)

Wir können auch mit AND und OR arbeiten und alle Kunden selektieren, die zwischen 40 und 49 Jahre alt sind:

```
1 SELECT * FROM  `kunden` WHERE alter >= 40 AND alter < 50
```

Oder alle, deren Name mit einem “L” oder “M” beginnt:

```
1 SELECT * FROM `kunden` WHERE name > 'L%' AND name < 'O%'
```

Oder deren Adresse entweder mit einem “D” oder einem “W” beginnt:

```
1 SELECT * FROM `kunden` WHERE adresse LIKE 'D%' OR adresse LIKE 'W%'
```

WHERE Fazit

WHERE wirst du sehr häufig brauchen, denn die Grundbefehle (SELECT, UPDATE, DELETE) sind ohne WHERE schlicht zu allgemein. Lediglich das INSERT kommt ganz gut ohne WHERE aus.

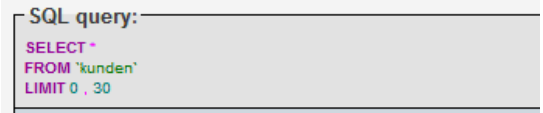
2.6. ORDER BY – Daten sortieren

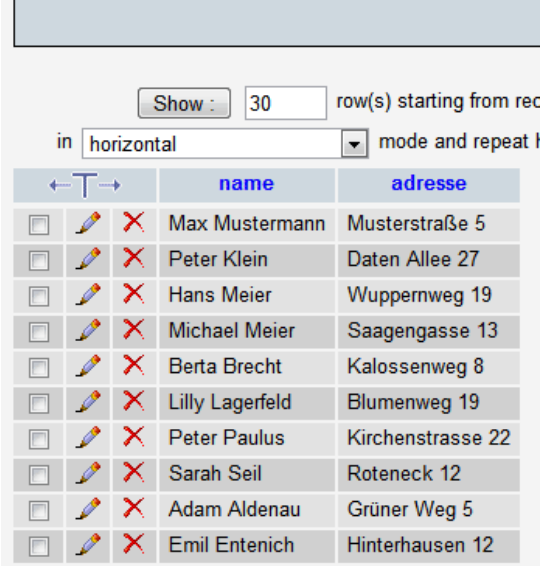
Häufig möchte man das SELECT-Ergebnis sortieren: **alphabetisch** (a-z), **chronologisch** (alt nach neu), **numerisch** (nach Zahlen), dafür gibt es **ORDER BY**.

Ohne ORDER BY erhalten wir bei folgendem Query:

```
1 SELECT * FROM kunden
```

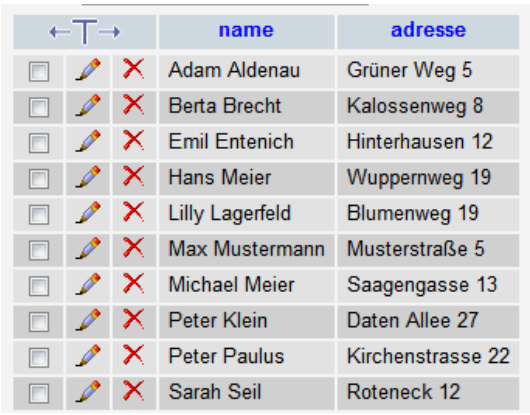
folgendes Ergebnis:





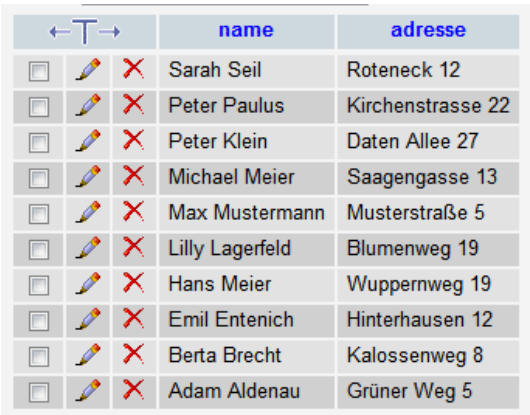
Möchte man aber seine Kundenliste alphabetisch sortiert ausgeben, von a bis z, setzt man noch ein ORDER BY *spaltenname* ans Ende des Queries:

```
1 SELECT * FROM `kunden` ORDER BY name ASC
```



Das “ASC” am Ende bedeutet “Ascending” und heißt “aufsteigend”. Das Ergebnis soll also abhängig von der Spalte “name” aufsteigend (ASC) sortiert werden. Möchte man es absteigend sortieren, nutzt man DESC (descending):

```
1 SELECT * FROM `kunden` ORDER BY name DESC
```



2.7. UPDATE – Daten ändern

Bislang können wir Daten selektieren (SELECT) und Daten eintragen (INSERT), jetzt wollen wir vorhandene Daten ändern (UPDATE).

```
1 UPDATE kunden SET name = 'Hans Müller' WHERE name = 'Hans Meier'
```

Wir sagen also erstmal, welche Tabelle wir aktualisieren wollen. Danach der Spaltenname und welcher Wert er erhalten soll. Das nun folgende WHERE ist wichtig, um unser UPDATE auf eine Zeile zu beschränken, denn ohne WHERE, würden alle Zeilen der Tabelle “Kunden” aktualisiert

werden.

Wir sind aber nicht auf eine Spalte pro Update beschränkt:

```
1 UPDATE kunden SET name = 'Donald Duck', adresse = 'Entenhausen' WHERE name = 'Emil Entenich'
```

In PHP würde das ganze folgendermaßen aussehen

```
1 <?php
2     //Verbindung zur Datenbank herstellen
3     mysql_query("UPDATE kunden SET name = 'Donald Duck', adresse = 'Entenhausen' WHERE name = 'Emil
  Entenich'");
4 ?>
```

2.8. DELETE – Daten löschen

Der letzte Teil im MySQL Tutorial behandelt das Löschen von Datensätzen

```
1 DELETE FROM kunden WHERE name = 'Donald Duck'
```

Im Grunde sagt man nur aus welcher Tabelle man etwas löschen möchte und dann, welcher Datensatz genau. Auch hier ist es wieder wichtig, die WHERE-Bedingung einzusetzen, sonst werden alle Datensätze einer Tabelle gelöscht.

In PHP sieht der obige SQL-Query so aus:

```
1 <?php
2     //Verbindung zur Datenbank herstellen
3     mysql_query("DELETE FROM kunden WHERE name = 'Donald Duck'");
4 ?>
```

←T→	name	adresse
	Max Mustermann	Musterstraße 5
	Peter Klein	Daten Allee 27
	Hans Müller	Wuppernweg 19
	Michael Meier	Saagengasse 13
	Berta Brecht	Kalossenweg 8
	Lilly Lagerfeld	Blumenweg 19
	Peter Paulus	Kirchenstrasse 22
	Sarah Seil	Roteneck 12
	Adam Aldenau	Grüner Weg 5

Wenn du noch mehr mit MySQL machen möchtest, schau dir das **SQL Tutorial** an. Alles was du darin lernst, kannst du auch mit MySQL nutzen.
