SQL lernen

http://sql.lernenhoch2.de/lernen/

---

1. SQL Tutorial als PDF

Das komplette SQL Tutorial gibt es auch als kostenloses PDF zum download. Lad dir das SQL Ebook runter und druck es aus, um es wie ein Buch zu lesen. Sollten sich Bereiche verändern oder Teile des Tutorials aktualisiert werden, wird auch das SQL Tutorial aktualisiert:

Download SQL Tutorial als PDF

zuletzt aktualisiert: 26.03.2010

Immer aktuell: **HTML Version**

Hinweis: Bitte verlinke nicht direkt auf das PDF oder die HTML Version, sondern auf diese Seite. Damit kann der Besucher selbst wählen, welche Version er möchte und sollte sich die URL des PDF's oder der HTML Version ändern, kommt man über diese Seite hier trotzdem noch zum Ziel. Danke!

---

2. SQL Einleitung

Bevor wir uns über die SQL-Abfragen hermachen, erstmal ein paar Grundlagen. Was ist eigentlich SQL, was sind Tabellen, Spalten und Zeilen und wie richtet man sich auf seinem Computer eine Datenbank ein, um all die schönen Beispiele direkt auszuprobieren? Das alles findet ihr im Kapitel **SQL Einleitung**.

2.1. Was ist SQL?

SQL steht für *Structured Query Language* und mit ihr kann man Daten aus einer Datenbank selektieren, eintragen, aktualisieren und löschen. Zudem kann man mit SQL Datenbank-Tabellen erstellen, modifzieren und löschen. Wenn du also eine Datenbank hast und die Datenbank benutzen möchtest, brauchst du SQL!

SQL ist recht einfach zu lernen, zumindest die Grundzüge. Umso komplexere Anfragen du an deine Datenbank stellst, desto komplexer die SQL-Queries (die Abfragen an die Datenbank). Aber keine Sorge, wir starten einfach für SQL Anfänger und behandeln auch den Bereich für SQL Fortgeschrittene so einfach wie möglich. Wenn dir irgendwas im SQL Tutorial fehlt, du eine Frage oder einen Fehler gefunden hast, melde dich über den Feedback Button auf der rechten Seite.

2.2. Tabellen erklärt

Jede Datenbank hat mindestens 1 Tabelle, in der Regel aber mehr. In Tabellen werden Daten gespeichert und jede Tabelle gruppiert bestimmte Daten. So kann man zum Beispiel eine Tabelle “Kunden” haben, in der alle Kunden-relevanten Daten gespeichert werden. Oder man hat eine Tabelle “Produkte”, in der alle Produkte + ihre spezifischen Daten **(wie Preis und Anzahl)** gespeichert werden.

Tabellen Normalisierung

Welche Tabellen man benötigt, hängt immer vom jeweiligen Anwendungsfall an. Die Anzahl der Tabellen sollte keine Einschränkung sein, man kann beliebig viele anlegen und die Performance leidet darunter auch nicht wirklich. Viel wichtiger ist es, die Tabellen korrekt zu normalisieren, zumindest bis zur 3. Normalform. Tabellen normalisieren ist für Anfänger keine leichte Kost, vorallem wenn man noch nichtmal eine Datenbank oder Tabelle angelegt hat. Daher empfehle ich, die folgenden Links erstmal zu überspringen, SQL anhand der Beispiele zu lernen und wenn du dann deine eigene Datenbank anlegst, dich vorher mit Tabellen Normalisierung zu beschäftigen:

- **Normalisierung** – super erklärt, übersichtlich und vorallem anhand von schönen Beispielen. Absolut lesenswert!
- **Der Königsweg: Normalisierung** – Vollständige Erklärung, ebenfalls mit Beispielen
- **Datenbank Normalisierung auf Wikipedia** – zur Vollständigkeit den Artikel auf Wikipedia zur Normalisierung, zum lernen finde ich aber die ersten beiden Links besser.

Meiner Meinung nach reicht es die 3. Normalform zu beherrschen, zumindest für Anfänger. Solltest du später auf einen Mangel der 3. Normalform für dein Projekt stoßen, kannst du dich immernoch mit den beiden höheren Formen (also 4. und 5. Normalform) beschäftigen.

2.2.1. Spalten und Zeilen

Jede Tabelle besteht aus einer Menge aus Spalten und Zeilen. Spalten sind dabei die Wert-Gruppen, während Zeilen die eigentlichen Werte beinhalten. Eine Kunden-Tabelle wird zum Beispiel folgende Spalten haben:

Name, Adresse, PLZ

Mit eingetragenen Werten sieht das ganze folgendermaßen aus:

name	adresse	plz
Max Mustermann	Musterstraße 5	03728
Peter Klein	Daten Allee 27	15324
Hans Müller	Wuppernweg 19	42553
Michael Meier	Saagengasse 13	42553
Berta Brecht	Kalossenweg 8	50677
Lilly Lagerfeld	Blumenweg 19	37280
Peter Paulus	Kirchenstrasse 22	20193
Sarah Seil	Roteneck 12	41211
Adam Aldenau	Grüner Weg 5	50677

Eine Tabelle hat in der Regel zwischen 2 und 20 Spalten, kann aber auch lediglich eine oder weit mehr als 20 Spalten haben. Die Anzahl an Zeilen ist im Grunde auch nicht beschränkt, eine Tabelle mit 1 Millionen Zeilen ist daher eher Standard als Ausnahme. Aber sogar eine Zeilenanzahl von 1 Milliarden und mehr ist hin und wieder in der Datenbankwelt anzutreffen.

2.2.2. Primary Key und Auto Increment

Der Primary Key ist ein uniquer Schlüssel, der jede Zeile einer Tabelle eindeutig identifiziert. Wenn man zum Beispiel einer Tabelle “Kunden” hat kann es vorkommen, dass zwei oder mehr Kunden den gleichen Namen und Nachnamen haben. Anstatt nun die Kunden anhand ihrer Namen eindeutig zu identifizieren (was problematisch wäre) gibt man jeder Zeile eine eindeutige ID, den Primary Key. Dieser wird beim eintragen einer neuen Zeile automatisch gesetzt und mittels **Auto Increment** nach jedem INSERT erhöht. Dadurch wird verhindert, dass zweimal die gleiche ID vergeben wird.

Foreign Key und Tabellen verknüpfen

Ein weiterer praktischer Vorteil des Primary Keys ist die Performance. Wenn man sein **Datenbank Schema normalisiert** hat, sind Daten die eine gewiße Verbindung haben auf mehreren Tabellen verteilt. Anhand des Primary Keys und des Foreign Keys innerhalb der Tabellen kann man diese Daten mittels **JOINS** bei einer Abfrage verknüpfen.

2.3. Deine eigene Datenbank

Um SQL vernünftig zu lernen brauchst du eine eigene Datenbank, die auf deinem Computer läuft und mit der du das gelernte Wissen praktisch vertiefen kannst. Die Installation geht leicht von der Hand, schau dir dazu einfach folgendes Tutorial an: **Datenbank installieren**. Schau dir ausserdem die beiden Unterteile an:

- **Tabelle anlegen in PHPMyAdmin**
- **Werte eintragen mit PHPMyAdmin**

---

3. SQL Anfänger

Hier erhälst du das absolute Basis-Wissen über SQL. Die Hauptbefehle sind:

- SELECT – Daten auswählen
- INSERT- Daten eintragen
- UPDATE – Daten aktualisieren
- DELETE – Daten löschen

Keine Sorge, dieses Kapitel ist wirklich einfach, aber probiere die gezeigten Beispiele aufjedenfall an deiner eigenen Datenbank aus!
Und jetzt wünsch ich dir viel Spass beim lernen von SQL :)

3.1. Vorwort

Um die Beispiele in den folgenden Kapiteln direkt auszuprobieren, brauchst du die Tabelle “users” mit ein paar Startwerten. Die folgenden Schritte können mithilfe der **Installations-Anleitung** gemeistert werden *(wenn Fragen sind, Feedback Button nutzen!)*.

- Starte deinen Xampp Server
- Gehe zu PHPMyAdmin
- Erzeuge die Datenbank “tutorial”

Nun führe folgenden SQL-Code **innerhalb** der Datenbank “tutorial” aus:

```sql
CREATE TABLE IF NOT EXISTS `users` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `username` varchar(60) NOT NULL,
  `punkte` mediumint(8) unsigned NOT NULL,
  `created` datetime NOT NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=7 ;

INSERT INTO `users` (`id`, `username`, `punkte`, `created`) VALUES
(1, 'liesel34', 2314, '2010-01-06 19:35:58'),
(2, 'michaelwesser', 195, '2010-01-12 21:15:58'),
(3, 'ano85', 8445, '2010-01-31 16:25:58'),
(4, 'Meerlis', 5278, '2010-02-04 11:55:58'),
(5, 'joooschi', 327, '2010-02-16 09:42:58'),
(6, 'Ginko5', 12, '2010-03-06 13:15:58');
```

Jetzt solltest du in der linken Seite die Tabelle “users” sehen. Klicke auf das **Icon links neben “users”**

	id	username	punkte	created
	1	liesel34	2314	2010-01-06 19:35:58
	2	michaelwesser	195	2010-01-12 21:15:58
	3	ano85	8445	2010-01-31 16:25:58
	4	Meerlis	5278	2010-02-04 11:55:58
	5	joooschi	327	2010-02-16 09:42:58

und du solltest folgendes erhalten

☐	✎	✕	6	Ginko5	12	2010-03-06 13:15:58

:

3.2. SELECT – Daten selektieren

Mit dem SELECT Befehl kann man Daten aus der Datenbank selektieren.

SELECT Syntax

```
1 SELECT spalten_name FROM tabellen_name
```

Alle Spalten aus der Tabelle selektieren

```
1 SELECT * FROM users
```

id	username	punkte	created
1	liesel34	2314	2010-03-02 13:36:44
2	michaelwesser	195	2010-03-04 17:16:45
3	ano85	8445	2010-01-09 12:06:36
4	Meerlis	5278	2010-02-12 19:42:42
5	joooschi	327	2010-02-27 07:01:59
6	Ginko5	12	2010-01-17 23:14:08

Nur bestimmte Spalten selektieren

```
1 SELECT username,created FROM users
```

username	created
liesel34	2010-03-02 13:36:44
michaelwesser	2010-03-04 17:16:45
ano85	2010-01-09 12:06:36
Meerlis	2010-02-12 19:42:42
joooschi	2010-02-27 07:01:59
Ginko5	2010-01-17 23:14:08

Spaltennamen umbenennen mit “AS”

Wenn man möchte, kann man auch den Namen einer Spalte umbenennen (die Tabelle ändert sich dadurch nicht, nur das selektierte Ergebnis hat den geänderten Namen):

```
1 SELECT id, punkte AS score FROM users
```

id	score
1	2314
2	195
3	8445
4	5278
5	327
6	12

3.2.1. ORDER BY – Sortieren

Mit **ORDER BY** kann man das Ergebnis einer Selektion auf- oder absteigend sortieren.

```
1 SELECT * FROM tabellen_name ORDER BY spalten_name ASC|DESC
```

ASC = aufsteigend sortieren *(kleinster Wert zuerst)*, ***default***

DESC = absteigend sortieren *(größter Wert zuerst)*

Anhand der User-Tabelle zeige ich dir was ORDER BY macht:

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
2	michaelwesser	195	2010-01-12 21:15:58
3	ano85	8445	2010-01-31 16:25:58
4	Meerlis	5278	2010-02-04 11:55:58
5	joooschi	327	2010-02-16 09:42:58
6	Ginko5	12	2010-03-06 13:15:58
7	friedel85	0	2010-03-06 17:39:37

Wir wollen alle Mitglieder-Daten selektieren und dabei die Liste aufsteigend nach den Punkten sortieren *(der Benutzer mit dem kleinsten Punktestand kommt zuerst)*:

```
1 SELECT * FROM users ORDER BY punkte ASC
```

id	username	punkte	created
7	friedel85	0	2010-03-06 17:39:37
6	Ginko5	12	2010-03-06 13:15:58
2	michaelwesser	195	2010-01-12 21:15:58
5	joooschi	327	2010-02-16 09:42:58
1	liesel34	2314	2010-01-06 19:35:58
4	Meerlis	5278	2010-02-04 11:55:58
3	ano85	8445	2010-01-31 16:25:58

*(Hinweis: der Wert "friedel85" wurde im **INSERT Tutorial** eingefügt. Lass dich davon nicht irritieren oder mache zuerst den INSERT-Teil)*

Nun wollen wir auf unserer Webseite eine Liste mit den neuesten Mitgliedern anzeigen, deshalb sortieren wir die Liste anhand des Datums absteigend *(der User mit dem neuesten Account kommt zuerst)*:

```
1 SELECT * FROM users ORDER BY created DESC
```

id	username	punkte	created
7	friedel85	0	2010-03-06 17:39:37
6	Ginko5	12	2010-03-06 13:15:58
5	joooschi	327	2010-02-16 09:42:58
4	Meerlis	5278	2010-02-04 11:55:58
3	ano85	8445	2010-01-31 16:25:58
2	michaelwesser	195	2010-01-12 21:15:58
1	liesel34	2314	2010-01-06 19:35:58

mehrere Spalten mit ORDER BY sortieren

Wenn man seine Selektion abhängig von 2 oder mehr Spalten sortieren möchte, kann man das wie folgt machen:

```
1 SELECT * FROM kunden ORDER BY plz, name
```

name	adresse	plz
Max Mustermann	Musterstraße 5	03728
Peter Klein	Daten Allee 27	15324
Peter Paulus	Kirchenstrasse 22	20193
Lilly Lagerfeld	Blumenweg 19	37280
Sarah Seil	Roteneck 12	41211
Hans Müller	Wuppernweg 19	42553
Michael Meier	Saagengasse 13	42553
Adam Aldenau	Grüner Weg 5	50677
Berta Brecht	Kalossenweg 8	50677

Wenn keine Sortierreihenfolge angegeben ist *(also ASC oder DESC)* wird bei default "ASC" verwendet. Bei einem mehrspaltigen ORDER BY, wird zuerst die erste Spalte sortiert *(in diesem Fall "plz" aufsteigend, also von 0-9 an)* und dann, wenn doppelte Werte vorhanden sind, die zweite

Spalte *(in diesem Fall "name" aufsteigend, also von a-z)*

3.2.2. DISTINCT

Wenn man eine Tabelle hat, in der viele Werte doppelt vorkommen, kann man mit dem Schlüsselwort **DISTINCT** die Selektion von doppelten Werten befreien.

DISTINCT Syntax

```
1 SELECT DISTINCT spalten_name FROM tabellen_name
```

Unabhängig davon, wieviel Sinn diese Tabelle macht, haben wir Deutschland zweimal drin:

country
Deutschland
Frankreich
Polen
Österreich
Schweiz
Dänemark
Luxemburg
Italien
Deutschland

Mit DISTINCT bekommen wir eine Liste von Ländern ohne doppelten Eintrag:

```
1 SELECT DISTINCT country FROM countries
```

country
Deutschland
Frankreich
Polen
Österreich
Schweiz
Dänemark
Luxemburg
Italien

3.3. WHERE – Auswahl eingrenzen

Mit WHERE kann man das Ergebnis einer Selektion begrenzen.

WHERE Syntax

```
1 SELECT * FROM tabellen_name WHERE spalten_name = wert
```

Für die Beispiele nutzen wir folgende Tabelle:

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
2	michaelwesser	195	2010-01-12 21:15:58
3	ano85	8445	2010-01-31 16:25:58
4	Meerlis	5278	2010-02-04 11:55:58
5	joooschi	327	2010-02-16 09:42:58
6	Ginko5	12	2010-03-06 13:15:58
7	friedel85	0	2010-03-06 17:39:37

WHERE bei numerischen Werten

```
1 SELECT * FROM users WHERE id = 4
```

id	username	punkte	created
4	Meerlis	5278	2010-02-04 11:55:58

```
1 SELECT * FROM users WHERE punkte > 1000
```

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
3	ano85	8445	2010-01-31 16:25:58
4	Meerlis	5278	2010-02-04 11:55:58

WHERE auf Strings anwenden

```
1 SELECT * FROM users WHERE username = 'joooschi'
```

id	username	punkte	created
5	joooschi	327	2010-02-16 09:42:58

Wenn man keinen bestimmten User selektieren möchte, sondern z.b. alle User deren username mit dem Buchstaben “M” beginnt, kann man dafür das Schlüsselwort “LIKE” mit einer Wildcard (%) nutzen:

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

id	username	punkte	created
2	michaelwesser	195	2010-01-12 21:15:58
4	Meerlis	5278	2010-02-04 11:55:58

Die Wildcard besagt, dass alles selektiert wird, solange der erste Buchstabe ein “M” ist. Die Wildcard steht dabei für alle anderen Zeichen. Wenn man alle User selektieren möchte, in deren Namen irgendwo ein “L” vorkommt, kann man das folgendermaßen machen:

```
1 SELECT * FROM users WHERE name LIKE '%L%'
```

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
2	michaelwesser	195	2010-01-12 21:15:58
4	Meerlis	5278	2010-02-04 11:55:58
7	friedel85	0	2010-03-06 17:39:37

AND, OR

Mit AND und OR kann man seine Auswahl noch verfeinern:

AND

```
1 SELECT * FROM users WHERE punkte > 1000 AND punkte < 6000
```

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
4	Meerlis	5278	2010-02-04 11:55:58

OR

```
1 SELECT * FROM users WHERE punkte < 1000 OR punkte > 6000
```

id	username	punkte	created
2	michaelwesser	195	2010-01-12 21:15:58

3	ano85	8445	2010-01-31 16:25:58
5	joooschi	327	2010-02-16 09:42:58
6	Ginko5	12	2010-03-06 13:15:58
7	friedel85	0	2010-03-06 17:39:37

3.4. INSERT – Daten einfügen

Mit dem INSERT Befehl werden Daten in die Datenbank eingetragen.

INSERT Syntax

```
1 INSERT INTO tabellen_name (spalte1, spalte2, spalte3, etc.) VALUES ('Wert1', 'Wert2', 'Wert3', etc.)
```

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
2	michaelwesser	195	2010-01-12 21:15:58
3	ano85	8445	2010-01-31 16:25:58
4	Meerlis	5278	2010-02-04 11:55:58
5	joooschi	327	2010-02-16 09:42:58
6	Ginko5	12	2010-03-06 13:15:58

```
1 INSERT INTO users (username, punkte, created) VALUES ('friedel85', 0, NOW())
```

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
2	michaelwesser	195	2010-01-12 21:15:58
3	ano85	8445	2010-01-31 16:25:58
4	Meerlis	5278	2010-02-04 11:55:58
5	joooschi	327	2010-02-16 09:42:58
6	Ginko5	12	2010-03-06 13:15:58
7	friedel85	0	2010-03-06 17:39:37

Auto Increment Spalten *(in diesem Fall **ID**)* müssen/dürfen nicht aufgelistet und befüllt werden, da die Datenbank diese Spalte selbst befüllt. Numerische Werte *(Spalte "Punkte")* werden ohne Hochkomma eingetragen, da sie keine Strings sind. Die Spalte "created" erwartet ein Datum + die Zeit. Dafür nutzen wir die SQL Funktion "NOW()", die für uns den genauen Zeitpunkt einträgt, wann der Eintrag erstellt wurde.

3.5. UPDATE – Daten aktualisieren

Mit UPDATE kannst du Werte aktualisieren.

INSERT Syntax

```
1 UPDATE tabellen_name SET tabellen_spalte = wert1 WHERE tabellen_spalte = wert2
```

Wenn man bei der UPDATE-Operation keine Eingrenzung mit **WHERE** setzt, werden alle Werte der Spalte für die jeweilige Tabelle auf den neuen Wert geändert. In der Regel möchte man aber nur den Wert einer bestimmten Zeile ändern, daher sollte man unbedingt darauf achten, die WHERE-Bedingung zu setzen.

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
2	michaelwesser	195	2010-01-12 21:15:58
3	ano85	8445	2010-01-31 16:25:58
4	Meerlis	5278	2010-02-04 11:55:58
5	joooschi	327	2010-02-16 09:42:58
6	Ginko5	12	2010-03-06 13:15:58
7	friedel85	0	2010-03-06 17:39:37

Als erstes wollen wir die Punktzahl des Users “friedel85” von 0 auf 37 setzen:

```
1 UPDATE users SET punkte = 37 WHERE id=7
```

Als nächstes soll der Name des Users “joooschi” auf “joschi” geändert und gleichzeitig seine Punktzahl um 50 Punkte erhöht werden:

```
1 UPDATE users SET username = "joschi", punkte = punkte + 50 WHERE username = "joooschi"
```

3.6. DELETE – Daten löschen

Mit DELETE kann man Einträge aus der Datenbank löschen.

DELETE Syntax

```
1 DELETE FROM tabellen_name WHERE spalten_name = wert
```

Mittels DELETE kann man beliebige Einträge aus einer Tabelle löschen, doch vorsicht: Wenn man die WHERE-Bedingung nicht setzt, werden alle Einträge innerhalb der Tabelle gelöscht:

```
1 DELETE FROM tabellen_name
```

Diesen Query kann man nicht rückgängig machen, wenn alle Einträge gelöscht wurden hilft nur noch ein vorher erstelltes Backup.

Nun wollen wir aus der Tabelle users den Benutzer mit Namen “Ginko5” löschen.

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
2	michaelwesser	195	2010-01-12 21:15:58
3	ano85	8445	2010-01-31 16:25:58
4	Meerlis	5278	2010-02-04 11:55:58
5	joschi	377	2010-02-16 09:42:58
6	Ginko5	12	2010-03-06 13:15:58
7	friedel85	37	2010-03-06 17:39:37

Dazu können wir einen der folgenden SQL-Queries nutzen:

```
1 DELETE FROM users WHERE username = 'Ginko5'
```

```
1 DELETE FROM users WHERE id = 6
```

```
1 DELETE FROM users WHERE username = 'Ginko5' AND punkte = 12
```

Das Resultat ist folgende User Tabelle:

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
2	michaelwesser	195	2010-01-12 21:15:58
3	ano85	8445	2010-01-31 16:25:58
4	Meerlis	5278	2010-02-04 11:55:58
5	joschi	377	2010-02-16 09:42:58
7	friedel85	37	2010-03-06 17:39:37

Weitere Beispiele:

Mehrere Einträge anhand der ID löschen:

```
1 DELETE FROM tabellen_name WHERE id IN (wert1, wert2, wert3, wert4, wert5)
```

3.7. SQL Datentypen

4. SQL Fortgeschritten

Nun wollen wir tiefer in SQL einsteigen. Es kommen zwar einige Befehle dazu, doch im Grunde dreht sich immernoch alles um die 4 Hauptbefehle **SELECT**, **INSERT**, **UPDATE** und **DELETE**. So werden wir z.b JOINS kennenlernen, mit deren Hilfe wir Tabelle zusammenführen können, um daraus zu selektieren. Ausserdem lernen wir noch weitere Möglichkeiten kennen, Daten zu selektieren und schauen uns hilfreiche SQL-Funktionen an.

Vorbereitung

Um die folgenden Beispiele direkt auszuprobieren, müssen wir noch ein paar weitere Tabellen mit Inhalten erzeugen. Bislang haben wir die User-Tabelle:

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
2	michaelwesser	195	2010-01-12 21:15:58
3	ano85	8445	2010-01-31 16:25:58
4	Meerlis	5278	2010-02-04 11:55:58
5	joschi	377	2010-02-16 09:42:58
7	friedel85	37	2010-03-06 17:39:37

Nun brauchen wir noch folgende Tabellen *(ganz unten steht die Erklärung wie die Tabellen zueinanderpassen)*:

questions-Tabelle

```
01 CREATE TABLE IF NOT EXISTS `questions` (
02   `id` int(10) unsigned NOT NULL auto_increment,
03   `user_id` int(10) unsigned NOT NULL,
04   `question` text character set utf8 NOT NULL,
05   `created` datetime NOT NULL,
06   PRIMARY KEY  (`id`)
07 ) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=6 ;
08
09 INSERT INTO `questions` (`id`, `user_id`, `question`, `created`) VALUES
10 (1, 1, 'Wofür ist der DELETE Befehl in SQL da?', '2010-02-12 19:26:22'),
11 (2, 1, 'Wer kennt den Unterschied zwischen SQL und MySQL?', '2010-02-27 12:11:05'),
12 (3, 2, 'Kennt jemand ein gutes SQL Tutorial?', '2010-03-01 02:01:44'),
13 (4, 3, 'Muss ich mich in PHP auskennen für MySQL?', '2010-03-12 14:51:13'),
14 (5, 7, 'Wie hoch ist der Mount Everest?', '2010-03-15 19:14:56');
```

id	user_id	question	created
1	1	Wofür ist der DELETE Befehl in SQL da?	2010-02-12 19:26:22
2	1	Wer kennt den Unterschied zwischen SQL und MySQL?	2010-02-27 12:11:05
3	2	Kennt jemand ein gutes SQL Tutorial?	2010-03-01 02:01:44
4	3	Muss ich mich in PHP auskennen für MySQL?	2010-03-12 14:51:13
5	7	Wie hoch ist der Mount Everest?	2010-03-15 19:14:56

question-votes-Tabelle

```
01 CREATE TABLE IF NOT EXISTS `question_votes` (
02   `id` int(10) unsigned NOT NULL auto_increment,
03   `question_id` int(10) unsigned NOT NULL,
04   `user_id` int(10) unsigned NOT NULL,
05   `created` datetime NOT NULL,
06   PRIMARY KEY  (`id`)
07 ) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=8 ;
08
09 INSERT INTO `question_votes` (`id`, `question_id`, `user_id`, `created`) VALUES
10 (1, 1, 2, '2010-03-16 14:25:29'),
11 (2, 1, 4, '2010-03-16 14:25:29'),
12 (3, 1, 5, '2010-03-16 14:25:37'),
13 (4, 1, 7, '2010-03-16 14:25:37'),
14 (5, 2, 1, '2010-03-16 14:26:07'),
```

```
(6, 2, 7, '2010-03-16 14:26:07'),
(7, 4, 1, '2010-03-16 14:26:39');
```

id	question_id	user_id	created
1	1	2	2010-03-16 14:25:29
2	1	4	2010-03-16 14:25:29
3	1	5	2010-03-16 14:25:37
4	1	7	2010-03-16 14:25:37
5	2	1	2010-03-16 14:26:07
6	2	7	2010-03-16 14:26:07
7	4	1	2010-03-16 14:26:39

answers-Tabelle

```
CREATE TABLE IF NOT EXISTS `answers` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `question_id` int(10) unsigned NOT NULL,
  `user_id` int(10) unsigned NOT NULL,
  `answer` text character set utf8 NOT NULL,
  `created` datetime NOT NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=10 ;

INSERT INTO `answers` (`id`, `question_id`, `user_id`, `answer`, `created`) VALUES
(1, 1, 3, 'Mit DELETE kannst du Einträge/Zeilen aus einer Tabelle löschen.', '2010-03-16 14:12:24'),
(2, 1, 7, 'Hier gibt es auch ein Tutorial zu DELETE:\r\nhttp://sql.lernenhoch2.de/lernen/sql-anfanger/delete-daten-loschen/', '2010-03-16 14:12:24'),
(3, 1, 1, 'Vielen dank für die Antworten!', '2010-03-16 14:12:38'),
(4, 2, 5, 'Schau mal hier: http://mysql.lernenhoch2.de/lernen/mysql-einleitung/alternativen-zu-mysql/', '2010-03-16 14:13:36'),
(5, 3, 4, 'Hier: http://sql.lernenhoch2.de/lernen/', '2010-03-16 14:14:45'),
(6, 4, 1, 'MySQL ist quasi der Zusammenschluss von PHP und MySQL, von daher musst du dich schon in PHP auskennen.', '2010-03-16 14:17:12'),
(7, 4, 7, 'Wenn ich mal fragen darf: Wofür brauchst du MySQL, wenn du dich nicht in PHP auskennst? Normalerweise lernt man doch erst PHP und nutzt dann MySQL um auf aus seinen PHP Skripten auf eine Datenbank zuzugreifen?', '2010-03-16 14:17:12'),
(8, 4, 3, 'Mein Chef meinte unsere Firma braucht jetzt eine MySQL Datenbank und ich solle mich da mal reinarbeiten...', '2010-03-16 14:19:50'),
(9, 4, 7, 'Oh.... ja die Story kenne ich, da hat der Chef in einem Magazin was über MySQL gelesen und das es im Web weit verbreitet ist und nun braucht er das auch...\r\n\r\nFrag ihn einfach mal, warum eure Firma eine MySQL Datenbank braucht, mich würde die Antwort interessieren ^^', '2010-03-16 14:19:50');
```

id	question_id	user_id	answer	created
1	1	3	Mit DELETE kannst du Einträge/Zeilen aus einer Tab...	2010-03-16 14:12:24
2	1	7	Hier gibt es auch ein Tutorial zu DELETE: http://...	2010-03-16 14:12:24
3	1	1	Vielen dank für die Antworten!	2010-03-16 14:12:38
4	2	5	Schau mal hier: http://mysql.lernenhoch2.de/lernen...	2010-03-16 14:13:36
5	3	4	Hier: http://sql.lernenhoch2.de/lernen/	2010-03-16 14:14:45
6	4	1	MySQL ist quasi der Zusammenschluss von PHP und My...	2010-03-16 14:17:12
7	4	7	Wenn ich mal fragen darf: Wofür brauchst du MySQL,...	2010-03-16 14:17:12
8	4	3	Mein Chef meinte unsere Firma braucht jetzt eine M...	2010-03-16 14:19:50
9	4	7	Oh.... ja die Story kenne ich, da hat der Chef in ...	2010-03-16 14:19:50

answer-votes-Tabelle

```
CREATE TABLE IF NOT EXISTS `answer_votes` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `answer_id` int(10) unsigned NOT NULL,
  `user_id` int(10) unsigned NOT NULL,
  `created` datetime NOT NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 AUTO_INCREMENT=16 ;

INSERT INTO `answer_votes` (`id`, `answer_id`, `user_id`, `created`) VALUES
```

```
(1, 1, 1, '2010-03-16 14:31:52'),
(2, 2, 1, '2010-03-16 14:31:52'),
(3, 2, 3, '2010-03-16 14:32:04'),
(4, 4, 2, '2010-03-16 15:38:56'),
(5, 4, 7, '2010-03-16 16:38:56'),
(6, 4, 1, '2010-03-16 17:38:56'),
(7, 6, 1, '2010-03-16 18:12:56'),
(8, 7, 1, '2010-03-16 18:32:56'),
(9, 7, 2, '2010-03-16 18:54:56'),
(10, 7, 4, '2010-03-16 19:10:56'),
(11, 9, 3, '2010-03-16 20:18:56'),
(12, 9, 5, '2010-03-16 21:01:56'),
(13, 9, 1, '2010-03-16 21:28:56'),
(14, 9, 4, '2010-03-16 22:38:56'),
(15, 9, 2, '2010-03-16 23:38:56');
```

id	answer_id	user_id	created
1	1	1	2010-03-16 14:31:52
2	2	1	2010-03-16 14:31:52
3	2	3	2010-03-16 14:32:04
4	4	2	2010-03-16 15:38:56
5	4	7	2010-03-16 16:38:56
6	4	1	2010-03-16 17:38:56
7	6	1	2010-03-16 18:12:56
8	7	1	2010-03-16 18:32:56
9	7	2	2010-03-16 18:54:56
10	7	4	2010-03-16 19:10:56
11	9	3	2010-03-16 20:18:56
12	9	5	2010-03-16 21:01:56
13	9	1	2010-03-16 21:28:56
14	9	4	2010-03-16 22:38:56
15	9	2	2010-03-16 23:38:56

Tabellen Erklärung

Als erstes was zu den Tabellennamen: ich nutze meist die englische Übersetzung im Plural, also anstelle der Tabelle “Benutzer” habe ich “users”, anstelle von “Frage” oder “Fragen” habe ich “questions”. Diese Form habe ich gelernt, als ich mir das PHP Framework **CakePHP** beigebracht habe und da sie sehr gut funktioniert, halte ich mich seitdem an diese Form.

Unser Datenbank-Schema hat eine Tabelle “users”, in die alle Benutzer reinkommen. Die Spalte “ID” ist unser Primary Key, den wir brauchen um die anderen Spalten korrekt zu referenzieren. In die Tabelle “questions” kommen alle Fragen, die unsere Benutzer stellen, jede Frage hat als Primary Key eine “ID” und als Foreign Key, bzw. Referenz Key, die Spalte “user_id”. Diese Form habe ich ebenfalls von CakePHP übernommen, dadurch lassen sich foreign-keys sehr schnell in der Tabelle erkennen. Dazu nimmt man den Namen der Tabelle, die man referenzieren möchte, im singular und setzt den Spalten Name des Primary Key der referenzierten Tabelle nach einem “_” daran. Hört sich komplizierter an als es ist:

Tabelle “users” hat “ID” => Tabelle “questions” bekommt die Spalte “user_id”, so einfach!

Im Prinzip funktioniert das für die anderen Tabellen ähnlich. Die Tabelle “answers” braucht natürlich auch einen Primary Key, wiedermal *ID*. Und natürlich muss man die User-ID referenzieren *(user_id)*, damit man weiß, welcher User die Antwort geschrieben hat. Aber natürlich muss man noch wissen, zu welcher Frage die Antwort geschrieben wurde, deshalb: questions -> ID => “question_id”.

Ich habe zwei Vote-Tabellen erstellt (damit wir wissen welche Fragen und Antworten von den Usern gut bewertet wurden). Jeweils eine für Fragen und eine für Antworten. Gespeichert wird der User der gevotet hat und für welche Frage / Antwort er gevotet hat und wann. Ob das für eine wirkliche Webanwendung die beste Umsetzung ist mag dahingestellt sein, aber für die folgenden SQL-Queries sind sie zumindest lehrreich.

4.1. JOIN – Tabellen zusammenfügen

Mit JOINS kann man zwei oder mehr Tabellen zusammenfügen, solange es eine Verbindung zwischen den Tabellen gibt.

JOIN Syntax

```
SELECT spalten_name FROM tabelle1 JOIN tabelle2 ON tabelle1.spalten_name = tabelle2.spalten_name
```

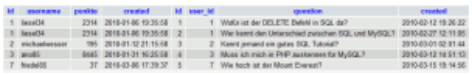
Versuchen wir das ganze mal an einem konkreten Beispiel. Im Teil **SQL für Fortgeschrittene** haben wir ein paar neue Testtabellen angelegt, mit denen wir jetzt arbeiten werden. Als erstes machen wir ein einfachen SELECT, indem wir alle Fragen aus der Tabelle “questions” selektieren:

```
1 SELECT * FROM questions
```

id	user_id	question	created
1	1	Wofür ist der DELETE Befehl in SQL da?	2010-02-12 19:26:22
2	1	Wer kennt den Unterschied zwischen SQL und MySQL?	2010-02-27 12:11:05
3	2	Kennt jemand ein gutes SQL Tutorial?	2010-03-01 02:01:44
4	3	Muss ich mich in PHP auskennen für MySQL?	2010-03-12 14:51:13
5	7	Wie hoch ist der Mount Everest?	2010-03-15 19:14:56

Nun möchten wir auf unserer Website alle Fragen ausgeben + den Benutzernamen des Benutzers, der die Frage gestellt hat. Doch leider haben wir in der Tabelle “questions” keinen Benutzernamen, sondern nur die ID des “users” und an dieser Stelle kommt **JOIN** ins Spiel. Da wir mit JOIN zwei Tabellen zusammenfügen können, werden wir nun die Tabelle “users” und die Tabelle “questions” komplett selektieren. Das können wir machen, da die Tabellen eine Verbindung haben, und zwar die ID des Benutzers.

```
1 SELECT * FROM users JOIN questions ON users.id = questions.user_id
```



Bei einem JOIN ALL (*) werden alle Spalten der einen Tabelle mit allen Spalten der anderen Tabelle zusammengepackt, solange eine Übereinstimmung gefunden wurde. Unser JOIN hat geklappt und wir haben nun die Fragen mit den Benutzernamen assoziiert. Doch es gibt hier einige Mängel:

1. Die Spalte “created” ist in der Ergebnis-Tabelle zweimal vorhanden, einmal das **users.created** (Benutzeraccount erstellt) und einmal **questions.created** (Frage gestellt).
2. Außerdem haben wir vielmehr Spalten im Result Set *(Ergebnis Tabelle)* als wir eigentlich brauchen und man sollte immer nur das selektieren, was man auch braucht.

Deshalb werden wir unseren Query überarbeiten:

```
1 SELECT users.username, questions.question,
2 questions.created AS frage_erstellt
3 FROM users JOIN questions
4 ON users.id = questions.user_id
```

username	question	frage_erstellt
liesel34	Wofür ist der DELETE Befehl in SQL da?	2010-02-12 19:26:22
liesel34	Wer kennt den Unterschied zwischen SQL und MySQL?	2010-02-27 12:11:05
michaelwesser	Kennt jemand ein gutes SQL Tutorial?	2010-03-01 02:01:44
ano85	Muss ich mich in PHP auskennen für MySQL?	2010-03-12 14:51:13
friedel85	Wie hoch ist der Mount Everest?	2010-03-15 19:14:56

Dieses Result Set hat genau die Werte die wir brauchen, die Frage und den Benutzernamen des Fragestellers + Wann die Frage gestellt wurde. Dies ist ein recht einfacher JOIN der dennoch häufig gebraucht wird (gerade bei Webanwendungen). Doch es gibt noch weitere JOIN-Typen die im folgenden behandelt werden.

INNER JOIN = JOIN

Das obige JOIN-Beispiel wird eigentlich als **INNER JOIN** bezeichnet. Ob man nun im SQL-Query JOIN oder INNER JOIN schreibt, macht für das Ergebnis aber keinen Unterschied.

4.1.1. LEFT JOIN

LEFT JOIN funktioniert ähnlich wie **INNER JOIN** mit dem Unterschied, dass Einträge der linken Tabelle keine Verbindung zu den Daten der rechten Tabelle haben müssen, um selektiert zu werden.

kurz: Selektiere alles von der linken Tabelle, auch wenn in der rechten kein übereinstimmender Wert vorhanden ist.

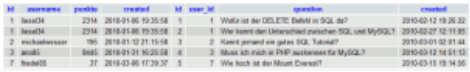
LEFT JOIN Syntax

```
SELECT * FROM tabelle1 LEFT JOIN tabelle2 ON tabelle1.spalten_name = tabelle2.spalten_name
```

Die Auswirkungen von LEFT JOIN wollen wir anhand des Beispiels aus **JOIN – Tabellen zusammenfügen** veranschaulichen.

INNER JOIN

```
SELECT * FROM users JOIN questions ON users.id = questions.user_id
```



LEFT JOIN

```
SELECT * FROM users LEFT JOIN questions ON users.id = questions.user_id
```

Im Gegensatz zum **INNER JOIN**, bei dem in beiden Tabellen der verknüpfende Wert vorhanden sein muss, ist das beim **LEFT JOIN** nicht der Fall. Beim *LEFT JOIN* werden alle Werte der linken Tabelle mit ins Result Set gepackt, auch wenn kein übereinstimmender Wert in der rechten Tabelle vorhanden ist.

Dazu noch ein passendes Beispiel. Wir wollen alle User selektieren, die bislang noch keine Frage gestellt haben.

```
SELECT users.username
FROM users LEFT JOIN questions
ON users.id = questions.user_id
WHERE questions.user_id IS NULL
```

username
Meerlis
joschi

Da das LEFT JOIN Result Set eine Tabelle erzeugt in der auch User gelistet werden, die keine Frage gestellt haben, können wir mittels **IS NULL** prüfen ob eine bestimmte Spalte den Wert **NULL** hat *(wenn kein Wert vorhanden ist, wird automatisch NULL gesetzt)*. Dadurch ziehen wir uns aus dem LEFT JOIN die NULL Zeilen und erhalten damit die User, die noch keine Frage gestellt haben.

4.1.2. RIGHT JOIN

RIGHT JOIN funktioniert genau wie LEFT JOIN, nur in diesem Fall ist alles umgedreht. Beim **RIGHT JOIN** werden die Einträge der rechten Tabelle selektiert, auch wenn keine Verbindung zu den Daten der linken Tabelle besteht.

RIGHT JOIN Syntax

```
SELECT * FROM tabelle1 RIGHT JOIN tabelle2 ON tabelle1.spalten_name = tabelle2.spalten_name
```

Im Prinzip kann man auf das RIGHT JOIN auch gänzlich verzichten (zumindest bei zwei Tabellen), indem man einfach die Position der Tabellen im JOIN vertauscht.

4.1.3. UNION

Mit dem UNION Befehl kann man die Result Sets von zwei oder mehr SELECT's kombinieren. Doppelte Werte werden dabei allerdings ignoriert.

UNION Syntax

```
1 SELECT spalten_name FROM tabelle1 UNION SELECT spalten_name FROM tabelle2
```

Bei UNION muss man darauf achten, dass die selektierten Spalten beider Tabellen vom gleichen Typ sind, dazu ein Beispiel: Es sollen alle User ID's selektiert werden, die schon aktiv waren. Ein User war aktiv, wenn er entweder eine Frage gestellt oder eine Antwort abgegeben hat.

id	user_id	question	created
1	1	Wofür ist der DELETE Befehl in SQL da?	2010-02-12 19:26:22
2	1	Wer kennt den Unterschied zwischen SQL und MySQL?	2010-02-27 12:11:05
3	2	Kennt jemand ein gutes SQL Tutorial?	2010-03-01 02:01:44
4	3	Muss ich mich in PHP auskennen für MySQL?	2010-03-12 14:51:13
5	7	Wie hoch ist der Mount Everest?	2010-03-15 19:14:56

id	question_id	user_id	answer	created
1	1	3	Mit DELETE kannst du Einträge/Zeilen aus einer Tab...	2010-03-16 14:12:24
2	1	7	Hier gibt es auch ein Tutorial zu DELETE: http://...	2010-03-16 14:12:24
3	1	1	Vielen dank für die Antworten!	2010-03-16 14:12:38
4	2	5	Schau mal hier: http://mysql.lernenhoch2.de/lernen...	2010-03-16 14:13:36
5	3	4	Hier: http://sql.lernenhoch2.de/lernen/	2010-03-16 14:14:45
6	4	1	MySQL ist quasi der Zusammenschluss von PHP und My...	2010-03-16 14:17:12
7	4	7	Wenn ich mal fragen darf: Wofür brauchst du MySQL,...	2010-03-16 14:17:12
8	4	3	Mein Chef meinte unsere Firma braucht jetzt eine M...	2010-03-16 14:19:50
9	4	7	Oh.... ja die Story kenne ich, da hat der Chef in ...	2010-03-16 14:19:50

```
1 SELECT user_id FROM questions UNION SELECT user_id FROM answers
```

user_id
1
2
3
7
5
4

UNION ALL

Möchte man alle Werte im Result Set haben, also auch solche die doppelt vorkommen, muss man **UNION ALL** verwenden.

```
1 SELECT user_id FROM questions UNION ALL SELECT user_id FROM answers
```

user_id
1
1
2
3
7
3
7
1
5
4
1
7
3
7

4.2. SELECT für Fortgeschrittene

Im **SQL Anfänger Tutorial** haben wir den **SELECT Befehl** schon kennengelernt. Doch nun wollen wir ein paar komplexere Abfragen mittels SELECT durchführen, dazu lernen wir Subqueries kennen, gruppieren Werte mittels **GROUP BY** und schränken SELECT's mittels **HAVING** weiter ein.

4.2.1. Subquery

Ein Subquery ist eine Abfrage, innerhalb einer Abfrage.

Subquery Syntax

```
SELECT * FROM tabellen1 WHERE id IN (SELECT id from tabellen2)
```

Im folgenden Beispiel möchten wir alle User selektieren, die schonmal eine Frage gestellt haben:

```
SELECT * FROM users WHERE id IN (3,5,7)
```

Wir möchten alle User mit einer bestimmten ID selektieren. Mittel “IN” können wir eine komma-separierte Liste mit ID’s angeben, die Benutzer mit diesen ID’s werden selektiert. Nun möchten wir aber mit einem Subquery alle User ID’s selektieren, die in der Tabelle “questions” vorkommen und mit dieser Liste die User selektieren, die mindestens eine Frage gestellt haben:

```
SELECT * FROM users WHERE id IN (SELECT DISTINCT user_id FROM questions)
```

id	username	punkte	created
1	liesel34	2314	2010-01-06 19:35:58
2	michaelwesser	195	2010-01-12 21:15:58
3	ano85	8445	2010-01-31 16:25:58
7	friedel85	37	2010-03-06 17:39:37

Meistens kann man einen Subquery auch durch einen JOIN ersetzen, aber Subqueries sind in der Regel einfacher zu programmieren und performanter:

```
SELECT DISTINCT users.* FROM users JOIN questions ON (users.id = questions.user_id)
```

4.2.2. GROUP BY – Werte gruppieren

Mittels GROUP BY kann man Zeilen gruppieren und mit den **SQL Funktionen** weitere Berechnungen durchführen.

GROUP BY Syntax

```
SELECT count(*) FROM tabellen_name GROUP BY id
```

Als erstes wollen wir eine Liste mit den ID’s der Fragen haben und wie häufig für die jeweilige Frage gevotet wurde, dazu nutzen wir die Tabelle “question_votes”:

```
SELECT question_id, COUNT(*) FROM question_votes GROUP BY question_id
```

question_id	count(*)
1	4
2	2
4	1

Mit “GROUP BY question_id” werden die gleichen ID’s miteinander gruppiert. Doch die Gruppierung alleine bringt uns nichts, wir müssen auch noch eine Funktion anwenden. In diesem Fall nutzen wir COUNT, denn wir wollen ja die Anzahl der Einträge zählen, welche in jeder Gruppe sind.

Nun wollen wir eine Liste mit allen Benutzern, die schonmal eine Frage gestellt haben und zusätzlich die Anzahl der Fragen, die sie gestellt haben:

```
SELECT users.*, count(questions.user_id)
FROM users JOIN questions
ON (users.id = questions.user_id)
GROUP BY questions.user_id
```

id	username	punkte	created	count(questions.user_id)
1	liesel34	2314	2010-01-06 19:35:58	2
2	michaelwesser	195	2010-01-12 21:15:58	1
3	ano85	8445	2010-01-31 16:25:58	1
7	friedel85	37	2010-03-06 17:39:37	1

Nun sollen alle Fragen selektiert werden und zusätzlich die Anzahl der Antworten, für die jeweilige Frage:

```
1 SELECT questions.*, count(questions.id)
2 FROM questions JOIN answers
3 ON (questions.id = answers.question_id)
4 GROUP BY questions.id
```

id	user_id	question	created	count(questions.id)
1	1	Wofür ist der DELETE Befehl in SQL da?	2010-02-12 19:26:22	3
2	1	Wer kennt den Unterschied zwischen SQL und MySQL?	2010-02-27 12:11:05	1
3	2	Kennt jemand ein gutes SQL Tutorial?	2010-03-01 02:01:44	1
4	3	Muss ich mich in PHP auskennen für MySQL?	2010-03-12 14:51:13	4

Am besten man macht erstmal einen einfachen JOIN (oder LEFT/RIGHT JOIN, je nach Bedarf) und schaut dann, anhand welcher Spalten man die Werte gruppiert. Dadurch vermeidet man auch Fehler oder findet schneller heraus, warum ein bestimmter Query nicht zum Ergebnis führt.

4.2.3. HAVING

HAVING ist eine Bedingung, die auf aggregierte Werte angewendet werden kann. Die WHERE Bedingung kann zum Beispiel auf gruppierte Werte (**GROUP BY**) nicht angewendet werden, dafür muss man HAVING verwenden.

HAVING Syntax

```
1 SELECT spalten_name, aggregations_funktion(spalten_name)
2 FROM tabelle1
3 GROUP BY spalten_name
4 HAVING aggregations_funktion(spalten_name) operator wert
```

HAVING = WHERE für gruppierte Daten

Im vorigen Teil haben wir alle Fragen selektiert und mittels GROUP BY noch die Anzahl der Antworten zu jeder Frage hinzugefügt:

```
1 SELECT questions.*, count(questions.id)
2 FROM questions JOIN answers
3 ON (questions.id = answers.question_id)
4 GROUP BY questions.id
```

id	user_id	question	created	count(questions.id)
1	1	Wofür ist der DELETE Befehl in SQL da?	2010-02-12 19:26:22	3
2	1	Wer kennt den Unterschied zwischen SQL und MySQL?	2010-02-27 12:11:05	1
3	2	Kennt jemand ein gutes SQL Tutorial?	2010-03-01 02:01:44	1
4	3	Muss ich mich in PHP auskennen für MySQL?	2010-03-12 14:51:13	4

Das wollen wir nun wieder machen, nur diesmal sollen nur die Fragen angezeigt werden, die mehr als 1 Antwort haben. Dazu nutzen wir HAVING, welches wir auf die aggregierten Werte anwenden *(die Anzahl der Antworten sind die aggregierten Werte)*:

```
1 SELECT questions.*, count(questions.id) AS anzahl_antworten
2 FROM questions JOIN answers
3 ON (questions.id = answers.question_id)
4 GROUP BY questions.id
```

```
5 HAVING anzahl_antworten > 1
```

id	user_id	question	created	anzahl_antworten
1	1	Wofür ist der DELETE Befehl in SQL da?	2010-02-12 19:26:22	3
4	3	Muss ich mich in PHP auskennen für MySQL?	2010-03-12 14:51:13	4

Mit WHERE kann man nur einen bestimmten Bereich einschränken, sobald man Werte gruppiert und Einschränkungen anhand der Gruppierung machen möchte, benötigt man HAVING. Gerade für die weiteren **SQL Funktionen**, auf die wir im nächsten Teil eingehen werden, wird HAVING häufig benötigt.

4.3. SQL Funktionen

SQL Funktionen bieten die Möglichkeit Rechenoperationen auf den selektierten Daten auszuführen. Dabei wird unterschieden zwischen Funktionen für aggregierte Daten, also zum Beispiel AVG(), COUNT() oder SUM(), die auf eine Menge von Werten angewendet werden und einen Wert zurückgeben (z.b. die Summe der Gehaltskosten aller Mitarbeiter). Im Gegensatz dazu stehen die Funktionen für skalare Daten, also anstelle einer Menge von Daten nur auf einzelne Zeilen. So kann man z.b. mit ROUND() die Werte einer Spalte runden oder NOW() das aktuelle Datum in einer Spalte speichern (wird standardmäßig bei INSERT verwendet, wenn die Tabelle eine Spalte “created” hat).

Mit SQL Funktionen kann man viel mehr Informationen aus den vorhandenen Daten extrahieren *(wieviel zahle ich pro Monat an meine Mitarbeiter)* oder temporäre Veränderungen an den vorhandenen Daten vornehmen *(wandel alle Werte für Spalte X mittels LCASE() in Kleinbuchstaben um)*.

4.3.1. AVG() – Durchschnittswert berechnen

4.3.2. COUNT() – Zeilen zählen

4.3.3. FIRST() – den ersten Wert einer Spalte selektieren

4.3.4. LAST() – den letzten Wert einer Spalte selektieren

4.3.5. MAX() – den höchsten Wert einer Spalte selektieren

4.3.6. MIN() – den niedrigsten Wert einer Spalte selektieren

4.3.7. SUM() – Zeilenwerte summieren

4.3.8. UCASE() – Werte einer Spalte in Großbuchstaben

4.3.9. LCASE() – Wert einer Spalte in Kleinbuchstaben

4.3.10. MID() – Zeichen extrahieren

4.3.11. LEN() – die Länge einer Zeichenkette einer Spalte

4.3.12. ROUND() – selektierte Werte runden

4.3.13. NOW() – aktuelles Datum und Zeit