Proseminar: Some Website Building Techniques

Thema: MySQL

erstellt von: Andreas Ens
Raimund Hartmann

Matr. Nr. 201210299
201210297

Universität Koblenz Landau
Fachbereich 4: Informatik
Institut für Informatik

Dozent: Dr. Manfred Jackel
Semester: Wintersemester 05/06

Einführung

Im Folgenden möchten wir versuchen einen Einblick in das zu geben, was MySQL leistet und welchem Datenbankkonzept es unterliegt.
Dies soll jedoch keine vollständige Leistungsbeschreibung oder gar eine Referenz sein, sondern lediglich der Versuch, einige wichtige Aspekte von MySQL zu verdeutlichen.

Wir werden kurz auf die Syntax von MySQL eingehen und danach an einem konkreten Beispiel, Schritt für Schritt, zeigen, wie man eine Datenbank erstellen und sie verwalten würde.

Zum Schluss gehen wir noch auf die Stärken und Schwächen von MySQL ein, und ziehen ein subjektives Resume über die Arbeit, die wir mit MySQL hatten.

Datenbanken und ihre Konzepte

Um zu verstehen, wie MySQL arbeitet, sollte man zunächst ein Grundwissen über Datenbaken allgemein, aber in erster Linie über ***relationale Datenbanken*** haben, denn genau dieses Konzept steckt hinter MySQL und auch hinter einigen seiner Konkurrenten.
Daher möchten wir kurz erklären was eine Datenbank überhaupt ist:

Der Begriff Datenbank ist sehr unspezifisch, so dass es auch keine allgemeine und allumfassende Definition dafür gibt, oder wir sie zumindest nicht finden konnten.
So wird eine simple Adressliste, die man z.B. mit *Excel* erstellt und speichert, ebenso als Datenbank bezeichnet, wie ein „Programm“, welches die Lagerverwaltung eines Großunternehmens darstellt.
Ein solches Unternehmen könnte z.B. das *Metro Warenhaus* sein, welches weltweit operiert und eine gigantische Menge an Waren, Warenbestand, Warenversand usw. verwalten muss.
So etwas ist natürlich nur mit den Mitteln, die Excel zur Verfügung stellt nicht mehr realisierbar, denn man braucht zumindest einen Hauptrechner (Server), damit einzelne Filialen (Clients), verteilt in der ganzen Welt, auf die Informationen des Hauptrechners (über das Internet) zugreifen können, um z.B. Waren zu bestellen oder ihren aktuellen Warenbestand in der Zentrale anzugeben.

Somit gibt es nicht DIE Definition aber es gibt EINE mögliche Definition, welche folgendermaßen lautet:

Eine Datenbank ist eine geordnete Menge von Daten, die normalerweise in einer oder in mehreren Dateien gespeichert ist.

Die Daten werden dabei in ***Tabellen*** strukturiert und abgelegt. Beim ***Relationalen Datenbankkonzept*** werden mehrere Tabellen, die eine Beziehung zueinander haben oder haben sollen, häufig miteinander verknüpft.
Das geschieht über ***Schlüssel (keys),*** die in einer oder mehreren Spalten der Tabelle enthalten sein können.

Im kommenden Beispiel möchten wir eine einfache DVDShop –Datenbank zumindest zum Teil realisieren, die z.B. die Tabellen ***Order*** und ***Customer*** haben könnte.
In Order könnten Informationen über Bestellungen enthalten sein (Bestellnummer, Bestelldatum…) und Customer würde Informationen über den Kunden beherbergen.
Beide Tabellen sind offensichtlich abhängig voneinander, denn es gibt ja keine Bestellung ohne einen Kunden. Daher würde man beide Tabellen auf einander beziehen wollen, um z.B. die Bestellungen eines bestimmten Kunden zu erfahren.

Die berühmtesten Vertreter relationaler Datenbanksysteme, abgekürzt: ***RDBMS***, sind natürlich MySQL oder aber auch ***Oracle*** und ***Microsoft SQL-Server***.
Seinen Ursprung hat dieses Konzept in den 70ern und entwickelt wurde es von E. F. Codd.
Es galt zu seiner Zeit als revolutionär, da es einfach nichts Vergleichbares gab.

Ein anderes Datenbankkonzept, welches ganz im Trend der großen Programmiersprachen wie ***Java*** oder ***C++,*** ist das der ***OODBS*** Datenbanken.
Übersetzt bedeutet das soviel wie: Objektorientierte Datenbanksysteme.
Der Vorteil einet objektorientierten Datenbank liegt in der Möglichkeit, Objekte ineinander zu schachteln, um Strukturen abbilden zu können.
Die Datenbasis eines OODBS besteht dabei aus einer Sammlung von Objekten, wobei jedes Objekt einen physischen Gegenstand, ein Konzept, eine Idee usw. repräsentiert.

Obwohl dieses Konzept dem relationalen Datenbankkonzept auf den ersten Blick überlegen ist, hat es sich dennoch noch nicht durchgesetzt.
Unsere Vermutung, warum dem so ist, ist dass dieses Konzept nicht sonderlich Einsteigerfreundlich ist.

Die berühmtesten Vertreter dieses neuen Konzeptes sind ***Object-Store, O2*** oder ***Versant.***

An dieser Stelle möchten wir den groben Überblick beenden und genauer auf das relationale Datenbankkonzept eingehen, da es ja auch MySQL zugrunde liegt.

Aufbau einer relationalen Datenbank

Wie bereits erwähnt, werden Daten in Form von zweidimensionalen Tabellen verwaltet, die über Schlüssel (Primärschlüssel, Fremdschlüssel) miteinander verknüpft werden können.

Die Zeilen einer Tabelle werden als ***Datensatz (record)*** bezeichnet und ihr Aufbau ist durch die Definition der Tabelle vorgegeben.
Die Datensätze sind wiederum unterteilt in ***Felder (fields).***
In unserem Beispiel könnte die Tabelle Customer, Felder für den Namen, Vornamen, Wohnort usw. enthalten.
Die einzelnen Felder müssen genau definiert werden. So dürften in dem Feld, Name, keine Zahlen eingetragen werden.
Diese Definition geschieht über ***Datentypen***, wie man sie auch aus anderen Programmiersprachen kennt. Sie bilden somit eine Vorschrift und legen genau fest, was in ein bestimmtes Feld eingetragen werden darf. Eine genaue Auflistung der in MySQL vorhandenen Datentypen erfolgt in einem der folgenden Kapitel.

Die Daten, die in Tabellen abgelegt werden, müssen nicht zwingend geordnet sein. Wenn man neue Werte einfügt, so wird die Tabelle automatisch verlängert und der neue Wert am Ende eingefügt. Diese Unordnung ist aber nur auf dem ersten Blick ein Hindernis für den User, kann jedoch zumindest für den Administrator in bestimmten Fällen problematisch werden. Z.B. wenn nur ein ganz bestimmter Datensatz gelöscht werden soll, und dieser zu löschende Datensatz viele Teile/Werte gemeinsam hat mit anderen Datensätzen.

Für den normalen User ist jedoch in den meisten Fällen nur der Zugriff (also das Anzeigen) auf die Daten eines Servers von Interesse.
Und genau das ist es, was MySQL in seinem Kern ist, nämlich eine Sprache für strukturierte Anfragen (SQL = Structured Query Language).

In unserem DVDShop –Beispiel könnte ein Lagerarbeiter eine Liste aller Artikel abfragen wollen, die ein bestimmter Kunde bestellt hat.
Er würde z.B. eine Anfrage über den Namen des Kunden erbitten und würde eine geordnete Liste aller bestellten Produkte erhalten.
Solche listen nennt man auch ***temporäre Listen***, da sie nur im Speicher existieren und physisch nicht vorhanden sind.

Die Geschwindigkeit mit der solche Anfragen ablaufen hängt in relationalen Datenbanken extrem stark von der Qualität ihres ***Index*** ab, bzw. wie dieser (diese) angeordnet ist.
Ein Index oder auch Schlüssel ist eine zusätzliche Tabelle, die nur Informationen über die Reihenfolge enthält.
Diese Indextabelle ist das, was MySQL so erfolgreich gemacht hat.
Sie bietet nämlich den Vorteil extrem schneller Anfragen, selbst bei großen Datenbeständen, weißt jedoch auch zwei Nachteile auf, von denen einer jedoch zu vernachlässigen sein sollte.

Dieser ist nämlich, dass die Indextabelle zusätzlichen Speicher, sprich Festplattenplatz belegt. Der größere Nachteil ist aber, dass bei jeder Veränderung der Daten auch der Index entsprechend zu verändern ist, was einen Zeitverlust bedeutet.

Das relationale Datenbankkonzept hat noch weitere Nachteile, die wir hier zwar erwähnen aber nicht genauer drauf eingehen möchten.
Segmentierung führt zu unübersichtlichen Abfragen, ***künstliche Schlüsselatribute*** erzeugen unnötige Attribute.
Ein weiterer Nachteil ist die ***externe Programmierschnittstelle,*** was das Einbinden von mächtigeren Programmiersprachen deutlich erschwert.

An dieser Stelle möchten wir dieses Kapitel verlassen und einen genaueren Blick auf MySQL selbst werfen.

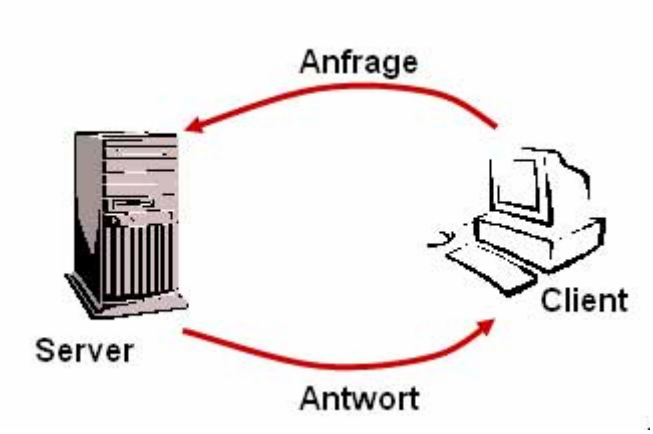
Wesentliche Eigenschaften von MySQL

Wir haben bereits einige Kokurrenzprodukte von MySQL genannt, mit denen man ebenfalls leistungsstarke Datenbanken realisieren kann.
Welches Datenbankkonzept das Bessere ist, können wir nicht beantworten, da beide vorgestellten Modelle ihre eigenen spezifischen Vor – und Nachteile haben.
Allerdings möchten wir in diesem Kapitel einige wichtige Merkmale von MySQL nennen und diese etwas genauer beleuchten.

Eines der Hauptargumente, warum man sich eventuell für MySQL entscheiden würde, ist jenes, dass es zumindest für nicht kommerzielle Zwecke ***kostenlos*** ist.
Und auch für den kommerziellen Bereich ist MySQL meist deutlich günstiger als vergleichbare Produkte.
Darüber hinaus handelt es sich hier um ein ***Open Sorce*** Produkt, was eine ganze Reihe von Vorteilen, sowohl für den privaten als auch für den kommerziell orientierten Anwender, mit sich bringt.
Der freizugängliche Quellcode macht es beispielsweise möglich, das Datenbanksystem durch Änderung im Quelltext und anschließende Kompilierung auf eigene, individuelle Bedürfnisse anzupassen. Solche Bedürfnisse treten meist im kommerziellen Bereich auf. Ein konkretes Beispiel wäre die Implementierung und Informationasbearbeitung von Barcodesystemen.
Aus genau diesem Grund existiert eine breite ***Community,*** die nahezu täglich ***Tutorials*** zur Lösung von ganz bestimmten Problemen veröffentlicht oder aber Fragen in zahlreichen ***Supportforen*** beantworten kann.

Bereits erwähnt, und daher möchten wir auch nicht mehr weiter darauf eingehen, ist, das MySQL ein relationales Datenbankkonzept verfolgt.
Natürlich mit all seinen Vor –und Nachteilen.

Ein weiterer wichtiger Punkt, auf den wir etwas genauer eingehen möchten, ist das MySQL ein ***Client/Server–System*** ist.
In diesem Zusammenhang spricht man auch von der so genannten ***Client/Server-Architektur*** oder abgekürzt: C/S – Architektur.
Etwas vereinfacht dargestellt, zeigt folgendes Bild wie dieses Prinzip funktioniert:



Die C/S-Architektur bezeichnet ein Systemdesign, bei dem die Verarbeitung einer Anwendung in zwei separate Teile aufgespaltet wird. Ein Teil läuft auf dem Server (dem Datenbank-Server, also MySQL), der andere Teil auf einer Workstation (Client).
Beide Teile werden über Netzwerke (z.B. das Internet) zu einem System zusammengefügt.
Der Client gibt auf dem Server die Bearbeitung von Daten in Auftrag und nimmt die Leistungen des Servers in Anspruch. Dabei bekommt jeder Client einen eigenen ***Thread.***
In diesem Zusammenhang spricht man auch vom ***Multi-Threaded Server.***

Aufgrund der stark gestiegenen Leistungsfähigkeit von Desktoprechnern und gleichzeitig immer größer werdenden Datenbeständen, geht man heute mehr und mehr dazu über, den Server nur noch als Datenlieferant zu nutzen und die Rechenarbeit von den einzelnen Desktops durchführen zu lassen.
Dieses Prinzip steht im krassen Gegensatz zu der ***Host-Basierten Architektur***, bei der der Server sowohl für die Lagerung der Datenbestände als auch für die Berechnung zuständig ist.
In diesem Zusammenhang spricht man auch von den so genannten ***File-Server-Systemen***.
Diese Architektur hat den großen Nachteil einer einbrechenden Performance bei steigender Benutzeranzahl.

Wie der name MySQL schon suggeriert, wird hier die Standardsprache ***SQL*** und der ***ANSI/SQL 92 Standard*** zur Anfrage und Verarbeitung von Daten und zur Administration der Datenbank verwendet.
Dies geschieht nicht völlig konsequent, da es einige Abweichung vom SQL Standard gibt.

Allgemein sehen wir dieses Argument nicht als MySQL spezifischen Kaufgrund an, da andere Konzepte wie z.B. Microsoft SQL, ebenfalls SQL, wenn auch in veränderter Form, nutzen.

Ein anderes Merkmal von MySQL, welcher durchaus als ein guter Kaufgrund anzusehen ist, ist die ***Plattformunabhängigkeit*** dieses Produktes.
Es unterstützt die wichtigsten Plattformen wie ***Windows, Linux, Mac OS X*** und ist somit extrem vielseitig.
Die Vorteile der Plattformunabhängigkeit liegen auf der Hand: So kann ein Windows-Client ebenso auf einen MySQL Server zugreifen und Daten bearbeiten wie ein Linux-Client.

Starten mit MySQL

Da diese Ausarbeitung sich nicht zu sehr im Detail verlieren soll, verzichten wir an dieser Stelle auf eine Installationsanleitung von MySQL, zumal es unter Windows, denn das ist die Plattform, die wir für das gleich kommenden Beispiele benutzt haben, nicht wirklich eine Herausforderung darstellt.

Eingabe von Anfragen geschieht bei MySQL nicht über eine GUI Umgebung sondern mittels Kommandozeile.
Es gibt zwar diverse grafische Entwicklungsumgebungen für MySQL, welche jedoch meist von 3t Anbietern stammen und sich letztlich auch der Kommandozeileneingabe bedienen, jedoch einer angenehmeren als die, die man vielleicht von Windows kennt.
Ein solches GUI Programm ist z.B. das ***MySQL Control Center.***

Ein Windowsverwöhnter Anwender wird am Anfang Schwierigkeiten haben sich an die Konsole zu gewöhnen, aber nach einer gewissen Einarbeitungszeit werden die Vorteile überwiegen.
So wird man sich nicht an verschiedene grafische Anwendungen gewöhnen müssen um von verschiedenen Plattformen aus mit MySQL arbeiten zu können.

Nach der Installation von MySQL muss man es zunächst konfigurieren.
Dazu muss man in das ***bin-Verzeichnis*** gehen und den ***Configuration Wizard*** aufrufen.
Hier konfiguriert man Schritt für Schritt seinen MySQL Server und legt für den ***root Account*** ein Passwort fest.
Nach erfolgreichem Einrichten kann man sich als root Account mit dem folgenden Statement
MySQL –u root –p password
anmelden und mit der Arbeit beginnen.

MySQL Syntax und Anweisungen

Bevor wir zu konkreteren Beispielen kommen, möchten wir noch kurz erläutern wie die MySQL Syntax aussieht.

Eine SQL Anweisung hat immer folgende Struktur:

Anweisung - Datenobjekte - Daten - Klauseln.

SQL Anweisungen können aus einer oder mehreren Zeilen bestehen, werden mit einem Semikolon abgeschlossen und können auch mehrere Anweisungen enthalten. Die Unterscheidung zwischen Groß -und Kleinschreibung ist nicht zwingend, in der Regel werden die Anweisungen zwecks besserer Lesbarkeit Groß geschrieben.
Es gibt eine Reihe von Schlüsselwörtern, die nicht zwingend mit einem Semikolon angeschlossen werden müssen.
Darunter fällt beispielsweise das ***USE*** statement, welches man unter anderem benutzt um eine bestimmte Datenbank zu mounten.
Die Schachtelung bzw. Reihung von Anweisungen ist ebenfall möglich und die Trennung geschieht mittels Komma:

SELECT version();
SELECT current_date;

In diesem Fall würden wir eine Tabelle mit der MySQL Version und nach Eingabe der zweiten Anweisung eine Tabelle mit dem aktuellen Datum als Ausgabe bekommen.
Man kann beide Anweisungen aber auch aneinanderreihen:

SELECT version(),current_date;

und so erhalten wir eine größere Tabelle, die beide Informationen beinhaltet:

```
mysql> SELECT VERSION(), CURRENT_DATE;

+------------+--------------+
| VERSION()  | CURRENT_DATE |
+------------+--------------+
| 5.0.15-nt  | 2005-09-03   |
+------------+--------------+
1 row in set (0.01 sec)
```

Wie man am Beispiel sehen kann, gibt MySQL das Ergebnis einer korrekten Anfrage in tabellarischer Form aus.
Dabei steht in der ersten Zeile stets die ***Beschriftung (label)*** für die dazugehörige Spalte (also: *VERSION()* und *CURRENT_DATE).*
In den einzelnen ***Spalten (rows)*** unter den bezeichnenden label stehen die Ergebnisse der Anfrage.
Die Angaben unter der Spalte zeigen an, wie viele *rows* zurückgegeben wurden und wie viel Zeit der Bearbeitungsprozess beansprucht hat.
Damit kriegt man vielleicht einen kleinen Eindruck, wie schnell MySQL ist.

Da MySQL eine Sprache ist, hat es auch eine Reihe von reservierten Wörtern, von denen wir einige nennen möchten:

CREATE, DROP, ALTER, DELETE, INSERT, UPDATE, SELECT, SET, FROM, INTO, WHERE, JOIN, LEFT JOIN,CROSS JOIN, RIGHT JOIN, FULL JOIN, INNER JOIN, ON, ORDER BY und GROUP BY.

Bevor man mit konkreten Aufgaben beginnen kann, sollte man noch etwas über die Datentypen wissen, die MySQL zur Verfügung stellt.

Datentypen in MySQL

Es gibt eine ganze Reihe von Datentypen in MySQL. Einige von ihnen sind zu mindest vom Namen auch schon aus anderen Programmiersprachen wie Java oder C++ bekannt.
In diesem Kapitel möchten wir auf die wichtigsten eingehen und sie kurz beschreiben.

Integer-Zahlen:

Es gibt 5 Größenordnungen bei Integer –Zahlen:

TINYINT	8-Bit
SMALLINT	16-Bit
MEDIUMINT	24-Bit
INT	32-Bit
BIGINT	64-Bit

Bei den Int -Datentypen sind sowohl positive als auch negative Zahlen erlaubt. Mit dem Attribut ***UNSIGNED*** kann der Zahlenbereich aber auch auf nur positive Zahlen eingeschränkt werden.
Bei ***TINYINT*** sind also Zahlen zwischen -128 und +127 erlaubt. Mit UNSIGNED wäre der Zahlenbereich somit 0-255.

Optional kann bei der Definition eines Integer –Felds die gewünschte Stellenzahl angegeben werden: z.B. INT(3)

Man bezeichnet diesen Wert als ***maximum display size***.
Dieser Wert hilft MySQL bei der übersichtlichen Formatierung von Abfrageergebnissen, schränkt jedoch weder den Zahlenbereich noch die erlaubte Stellenzahl ein.

Fließkommazahlen:

FLOAT	(8 Stellen Genauigkeit)	4-Byte
DOUBLE	(16 Stellen Genauigkeit)	8-Byte

Es gibt noch den Typ ***REAL,*** welcher aber eigentlich nur den Typ ***DOUBLE*** darstellt.

Die Anzahl der Stellen bei FLOAT und DOUBLE können durch 2 Parameter eingestellt werden:
FLOAT(M, D)
M ist für die Formatierung der Zahlen zuständig und D bewirkt eine Rundung der Zahlen beim Speichern.
Damit würde ein Feld mit FLOAT(3, 3) die zahl 367,56709 auf den Wert 367,567 runden.

Festkommazahlen:

DECIMAL(P, S)

Die Festkommazahl wird als Zeichenkette gespeichert. Damit ist eine beliebige Stellenzahl möglich.

P gibt die gesamte Stellenzahl an und S die Anzahl der Stellen hinter dem Dezimalpunkt.
Der positive Zahlenbereich ist bei DECIMAL um eine Stelle größer als der negative, da für das negative Vorzeichen eine Stelle reserviert ist.

Dieser Zahlentyp ist nur dann zu empfehlen, wenn Rundungsfehler ausgeschlossen werden sollen.

Zeit / Datum:

DATE	(Datum in der Form YYYY-MM-DD)	3-Byte
TIME	(Zeit in der Form 23:59:59)	3-Byte
DATETIME	(Kombination aus DATE und TIME)	8-Byte
YEAR	(Jahreszahl)	1-Byte

Wie DATE und DATETIME korrekt arbeiten liegt in der Verantwortung des Client – Programms.
So ist z.B. auch der 0 für den Monatswert erlaubt, was so in der Realität nicht vorkommt.

Es gibt über die erwähnten Daten –und Zeittypen noch den Typ ***TIMESTAMP***, welcher die „Zeit der letzten Änderung“ angibt.
Felder dieses Typs werden bei jeder Änderung des Datensatzes automatisch an die aktuelle Zeit angepasst.

Zeichenketten:

CHAR(N)	(Zeichenkette mir vorgegebener Länge N, max. 255)
VARCHAR(N)	(Zeichenkette mit variabler Länge,N<256)
TINYTEXT	(Zeichenkette mit variabler Länge, max. 255 Zeichen)
TEXT	(Zeichenkette mit variabler Länge, max. 2^16-1)
MEDIUMTEXT	(Zeichenkette mit variabler Länge, max. 2^24-1)
LONGTEXT	(Zeichenkette mit variabler Länge, max. 2^32-1)

Bei CHAR ist die Länge der Zeichenkette fest vorgegeben und Leerzeichen am Beginn der Zeichenkette werden vor dem Speichern eliminiert und zu kurze Ketten werden am Ende mit Leerzeichen erweitert.

Binärdaten:

Zum Speichern binärer Daten gibt es in MySQL 4 Datentypen:

TINYBLOB	(Binärdaten mit variabler Länge, max. 255Byte)
BLOB	(Binärdaten mit variabler Länge, max. 2^16-1Byte)
MEDIUMBLOB	(Binärdaten mit variabler Länge, max. 2^24-1Byte)
LONGBLOB	(Binärdaten mit variabler Länge, max. 2^32-1Byte)

BLOB –und TEXT –Datentypen sind fast identisch. Ihr einziger Unterschied liegt beim Sortieren und Vergleichen der Daten.

Im Grunde braucht man die BLOB Datentypen nicht, aber sie haben einige Vorteile, daher finden sie noch Verwendung.
Ihr größter Vorteil ist die Integration in die Datenbank, was zu einer höheren Sicherheit führt.

Aufzählungen:

ENUM	(Auswahl einer von max. 65535 Zeichenketten)
SET	(Kombination von max. 255 Zeichenketten)

Man kann mit diesen Datentypen eine Liste von Zeichenketten verwalten, denen durchlaufende Nummern zugeordnet sind.
Damit kann man eine dieser Zeichenketten auswählen.

Um in einem Feld eine Kombination von mehreren Zeichenketten zu speichern, muss man diese durch Kommas trennen.
Dabei ist zu beachten, dass keine Leerzeichen angegeben werden dürfen und die Reihenfolge ist nicht relevant.

Erstellen einer kleinen Datenbank

Im Allgemeinen beginnt man mit einem Projekt, wie z.B. das Erstellen eines 3D Berechnungsmodells mit Java, nicht ohne einen konkreten Plan zu haben, wie die Software aussehen und was sie leisten soll. Wenn aber doch, so wird man insbesondere bei größeren Projekten schnell merken, dass man die Übersicht verliert und dadurch auf der einen Seite der Zeitaufwand stark ansteigt und auf der anderen Seite die Qualität und die Nachvollziehbarkeit der Software sinkt.

Genauso ist es auch bei der Erstellung einer Datenbank.
Insgesamt werden in der Literatur 5 Punkte genannt, die man zu mindest bei wirklich großen Projekten auch Schritt für Schritt befolgen sollte:

- Definieren des aktuellen Geschäftsprozesses
- Definieren der Geschäfts-Objekte
- Definieren der Geschäftsregeln
- Skizzieren der Datenbank
- Definieren der Beziehungen

Mit ***Geschäftsprozess*** ist der Ablauf gemeint, in den die Datenbank eingefügt werden soll.
Geschäftsobjekte sind Komponenten, aus denen der Geschäftsprozess zusammengesetzt ist.
Eine ***Geschäftsregel*** ist eine Anweisung oder eine Reihe von Anweisungen, die bestimmt, wie ein Geschäft geführt wird.
Das ***Modellieren,*** also das Skizzieren der Datenbank dient der Übersichtlichkeit.
Beziehungen bilden in dieser Reihe den wichtigsten Teil der Arbeit, denn hier muss man entscheiden, ob bestimmte Tabellen eine Beziehung untereinander haben oder nicht.
Beziehungen können die Formen 1:1, 1:n oder m:n haben.

Allein über das Design einer Datenbank gibt es in den meisten Fachbüchern eine Reihe von Kapiteln, die dieses Thema ausführlich und von verschiedensten Blickwinkeln aus beleuchten. Doch alle haben etwas gemeinsam, sie münden im handwerklichen Teil des Entwicklungsprozesses.

Dieser beginnt (mal abgesehen von der Anmeldung als privilegierter user) stets mit der Erstellung einer Datenbank, was in MySQL mit dem statement

CREATE DATABASE databaseName;

geschieht. In unserem Falls also:

```
mysql> CREATE DATABASE DVDShop;
Query OK, 1 row affected (0.00 sec)
```

Mit ***SHOW DATABASES;*** können wir uns alle vorhandenen Datenbanken anzeigen lassen:

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| dvdshop            |
| mysql              |
| temp               |
+--------------------+
4 rows in set (0.00 sec)
```

Damit hat man zunächst lediglich eine leere Datenbankhülle.
Falls man sich einer bereits erstellten Datenbank wieder entledigen möchte, benutzt man den ***DROP-Befehl****:*

DROP DATABASE databaseName;

Das Erstellen der Datenbank allein befähigt einen aber noch nicht direkt auf die Datenbank zugreifen zu können, z.B. um Tabellen einzufügen.

Daher muss man die gewünschte Datenbank erst einmal mounten, was mit dem statement: ***USE databaseName;*** geschieht.

```
mysql> USE DVDShop;
Database changed
```

Nun haben wir das Grundgerüst und können damit beginnen dieses Gerüst mit Tabellen zu füllen, in denen später unsere Daten abgelegt werden.

Erstellen und bearbeiten von Tabellen

Unser DVDShop könnte eine Reihe von Tabellen enthalten. Beispielsweise eine Tabelle Customer, für die Kundendaten und eine Tabelle Orders, für Aufträge und Inventory, für den Lagerbestand.
Customer soll zunächst nur den Namen, Vornamen, Geburtsdatum, Geschlecht, Anzahl erworbener Artikel und den Gesamtumsatz der Kunden enthalten.

Erstellen kann man Tabellen mit dem statement:

CREATE TABLE tableName();

In der Klammer werden Spalten deklariert und mit einem Datentyp versehen.
Wie üblich geschieht die Trennung mittels Komma.

```
mysql> CREATE TABLE Customer(Name VARCHAR(20), Vorname VARCHAR(20),
    -> Geburtsdatum DATE, Geschlecht ENUM('M','F') DEFAULT 'F',
    -> Artikelanzahl INT(4), Umsatz INT(5));
Query OK, 0 rows affected (0.11 sec)
```

Name und Vorname sollen vom Typ Varchar sein und eine maximale Länge von 20 Zeichen haben.
Das Geburtsdatum soll vom Typ Date sein. Anders als vom deutschen gewohnt, wird hier die amerikanische Form der Zeit benutzt: YYYY-MM-DD.
Das Geschlecht ist vom Typ Enum, man hat also die Wahl zwischen männlich und weiblich, wobei der Standardwert weiblich ist.
Die restlichen Werte sind einfache Int Typen.

Mit ***DESCRIBE tableName;*** können wir uns anschauen was wir gerade erstellt haben:

```
mysql> DESCRIBE Customer;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| Name          | varchar(20)   | YES  |     | NULL    |       |
| Vorname       | varchar(20)   | YES  |     | NULL    |       |
| Geburtsdatum  | date          | YES  |     | NULL    |       |
| Geschlecht    | enum('M','F') | YES  |     | F       |       |
| Artikelanzahl | int(4)        | YES  |     | NULL    |       |
| Umsatz        | int(5)        | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Damit haben wir bereits den Grundaufbau der Tabelle Customer und könnten sie theoretisch bereits mit Daten füllen.
Das Löschen der Tabelle geschieht analog zum Löschen einer Datenbank.

Es kann natürlich passieren, dass man eine Tabelle erweitern oder modifizieren muss, und genau für diesen Zweck existiert das ***ALTER TABLE*** statement, welches wir auch aufgrund seiner Wichtigkeit etwas genauer betrachten wollen.
Die Möglichkeiten, die ALTER TABLE liefert sind sehr groß. So kann man Spalten hinzufügen oder löschen, einen Index erzeugen oder löschen, den Datentyp einer Spalte ändern und die Tabelle oder aber nur einzelne Spalten umbenennen.

Der interne Vorgang bei diesem Prozess ist (wie für MySQL üblich) der, dass zunächst eine temporäre Kopie der zu ändernden Tabelle erzeugt wird und danach die Operation darauf durchgeführt wird. Kommt es dabei zu keinen Komplikationen, wird die Originaltabelle durch die temporäre Tabelle ersetzt.

Als erstes möchten wir unsere Tabelle Customer um eine weitere Spalte ***Herkunft*** erweitern.

```
mysql> ALTER TABLE customer ADD Herkunft VARCHAR(5);
Query OK, 0 rows affected (0.24 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Das Ergebnis:

```
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| Name          | varchar(20)   | YES  |     | NULL    |       |
| Vorname       | varchar(20)   | YES  |     | NULL    |       |
| Geburtsdatum  | date          | YES  |     | NULL    |       |
| Geschlecht    | enum('M','F') | YES  |     | F       |       |
| Artikelanzahl | int(4)        | YES  |     | NULL    |       |
| Umsatz        | int(5)        | YES  |     | NULL    |       |
| Herkunft      | varchar(5)    | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
```

Einen Datentyp ändern geschieht ähnlich wie das Hinzufügen einer zusätzlichen Spalte.
Wir könnten beispielsweise den Typ der Spalte Herkunft auf VARCHAR(3) setzen.
Diese Operation ist jedoch mit Vorsicht zu genießen, da durch die Reduzierung der maximalen Länge, Datensätze abgeschnitten werden können.

```
mysql> ALTER TABLE customer CHANGE Herkunft Herkunft VARCHAR(3);
Query OK, 0 rows affected (0.64 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Das Ergebnis:

```
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| Name          | varchar(20)   | YES  |     | NULL    |       |
| Vorname       | varchar(20)   | YES  |     | NULL    |       |
| Geburtsdatum  | date          | YES  |     | NULL    |       |
| Geschlecht    | enum('M','F') | YES  |     | F       |       |
| Artikelanzahl | int(4)        | YES  |     | NULL    |       |
| Umsatz        | int(5)        | YES  |     | NULL    |       |
| Herkunft      | varchar(3)    | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
```

Die Syntax aller Alter Table Befehle ähnelt sich sehr stark, daher verzichten wir an dieser Stelle auf weitere Beispiele und zeigen stattdessen eine Übersicht über die wichtigsten Anfragen:

```
ALTER TABLE tabelle änderungsangaba [,änderungsangabe]

änderungsangabe:


     ADD [COLUMN] create_definition [FIRST | AFTER spalten_name]
oder ADD [COLUMN] (create_definition, create_definition,...)
oder ADD INDEX [index_name] (index_spalten_name,...)
oder ADD PRIMARY KEY (index_spalten_name,...)
oder ADD UNIQUE [index_name] (index_spalten_name,...)
oder ADD FULLTEXT [index_name] (index_spalten_name,...)
oder ADD [CONSTRAINT symbol] FOREIGN KEY index_name(index_spalten_name,...)
oder ALTER [COLUMN] spalten_name {SET DEFAULT literal | DROP DEFAULT}
oder CHANGE [COLUMN] alter_spalten_name create_definition
oder MODIFY [COLUMN] create_definition
oder DROP [COLUMN] spalten_name
oder DROP PRIMARY KEY
oder DROP INDEX index_name
oder DISABLE KEYS
oder ENABLE KEYS
oder RENAME [TO] neue_tabelle
oder ORDER BY spalte
oder tabellen_optionen
```

*entnommen aus der: „official MySQL documentation“.
Quelle: www.MySQL.com

Im Zusammenhang mit dem Alter Table statement ist vielleicht noch zu sagen, dass nur user mit den Privilegien ***ALTER, INSERT,*** und ***CREATE*** ihn verwenden können.
Zum Privilegiensystem von MySQL kommen wir noch in einem späteren Kapiteln.

Füllen einer Tabelle mit Daten

In MySQL gibt es diverse Möglichkeiten eine Tabelle mit Daten zu füllen. Die statements derer man sich dabei bedient sind ***LOAD, DATA*** und ***INSERT.***

Die einfachste Möglichkeit bietet dabei das INSERT statement, mit dem man jeweils einen einzelnen record einfügen kann.
Die Eingabe der Werte geschieht dabei in derselben Reihenfolge wie die Felder in der betreffenden Tabelle angeordnet sind und Werte müssen natürlich auch konform zum Feldspezifischen Datentyp sein. Um das zu demonstrieren, möchten wir in die Tabelle Customer einen Customer mit den dazugehörigen Daten einfügen:

```
mysql>  INSERT INTO Customer
    ->  VALUES('Hannelore','Musterfrau','1923-3-21','F','34','2345','DE');
Query OK, 1 row affected (0.05 sec)
```

Um sich die Daten, die in einer Tabelle abgelegt zu betrachten oder auszuwählen bedient man sich nicht wie vielleicht erwartet der DESCRIBE sondern der ***SELECT*** Anweisung:

```
mysql> select * from customer;
+-----------+------------+--------------+------------+---------------+--------+----------+
| Name      | Vorname    | Geburtsdatum | Geschlecht | Artikelanzahl | Umsatz | Herkunft |
+-----------+------------+--------------+------------+---------------+--------+----------+
| Hannelore | Musterfrau | 1923-03-21   | F          |            34 |   2345 | DE       |
+-----------+------------+--------------+------------+---------------+--------+----------+
2 rows in set (0.00 sec)
```

Das Ergebnis ist, wie zu erwarten war, wieder eine Tabelle, die nach Feldern geordnet ist, und unter jedem Feldbezeichner stehen die entsprechenden Werte.
In obigen Fall existiert bis jetzt nur ein einziger Customer namens Hannelore Musterfrau aus Deutschlang. Geboren ist sie am 21.03.1923 und hat bis dato 34 Artikel in einem Gesamtwert von 2345 erworben.

Eine andere Methode ist das Einlesen von Werten aus einer Datei. Das gelingt jedoch nicht ohne weiteres mit jedem Datenformat. Mit Einem .doc File ist es nach unseren Recherchen nicht ohne weiteres möglich.
Halbwegs gut geeignet sind .txt oder .csv Formate.
Auch in diesem Fall müssen die Werte in der passenden Reihenfolge vorliegen und das Zeilenende muss MySQL bekannt sein oder bekannt gemacht werden, was z.B. mit dem Anhang

LINES TERMINATED BY `\R\N`;

geschieht.

Im .txt Format würde man, und das ist auch das was wir gemacht haben, die Daten in der Form:

```
Anton Harz  1980-03-08  M     10    230   DE
Frauke Kant 1978-12-10  F     29    890   DE
usw.
```

speichern und anschließend das File mit folgender Anweisung einlesen:

```
mysql>  LOAD DATA LOCAL INFILE 'g:\customer1.txt' INTO TABLE customer;
Query OK, 3 rows affected (0.00 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0
```

Das Ergebnis:

```
mysql> SELECT * FROM Customer;
+-----------+------------+--------------+------------+---------------+---------+----------+
| Name      | Vorname    | Geburtsdatum | Geschlecht | Artikelanzahl | Umsatz  | Herkunft |
+-----------+------------+--------------+------------+---------------+---------+----------+
| Hannelore | Musterfrau | 1923-03-21   | F          |            34 |    2345 | DE       |
| Anton     | ausTirol   | 1980-03-08   | M          |            10 |     230 | DE       |
| Frauke    | Kant       | 1980-03-08   | F          |            29 |     890 | DE       |
| Mike      | Kirch      | 1956-05-17   | M          |           101 |    4000 | DE       |
+-----------+------------+--------------+------------+---------------+---------+----------+
4 rows in set (0.00 sec)
```

Am Ergebnis kann man sehr schön sehen, dass Daten ohne weitere Ordnung außer der Reihenfolge mit der sie eingefügt wurden, in der Tabelle gespeichert werden.
Ganz nach dem Motto: Was zuerst da war steht auch weiter oben.

Ganz ohne Probleme ist diese Form des Einlesens nicht, es kann leicht zu Fehlern kommen, die dann die ganze Tabelle ruinieren können.

Alternativ bietet MySQL auch ein eigenes kleines Tool, welches im bin -Verzeichnis zu finden ist und den Namen ***mysqlimport.exe*** hat. Es handelt sich dabei um ein Kommandozeilen - Utility, das 2 Parameter und eine Anzahl von Optionen beinhaltet. Dieses Hilfsprogramm importiert eine Textdatei in die angegeben Datenbank und Tabelle. Die Syntax für einen solchen Befehl sieht so aus:

mysqlimport databaseName textFileName.fileFormat

in unserem Fall also:

mysqlimport DVDShop Customer.txt

Dieser befehl überträgt den Inhalt der Textdatei in die Tabelle, die durch den Dateinamen bis zum ersten Punkt bestimmt wird.
Falls es bei diesem Vorgang zu Fehlern kommt, werden diese angezeigt.

Im nächsten Kapitel möchten wir eine Einführung in die Operationen geben, die man auf bereits vorhandene Daten anwenden kann.

Arbeiten mit den Daten einer Tabelle

Es gibt eine ganze Reihe von Operationen, die man auf Daten anwenden kann.
Auf einige wichtige möchten wir eingehen und etwas näher beleuchten.
Dazu gehören folgende Operationen:

- Auswählen von Daten
- Modifizieren und Löschen von Daten
- Sortieren und Zählen von Daten
- Pattern Matching

Auswählen von Daten:

Das Auswählen und Anzeigen aller Daten haben wir bereits einige Male praktiziert und möchten daher an dieser Stelle nur noch einmal die Syntax explizit vorstellen:

SELECT * FROM tableName;

Der kleine Stern repräsentiert alle Felder, man kann aber auch nur ein (oder auch eine Reihe) Bestimmtes Feld auswählen:

SELECT Name FROM Customer;

Das würde uns alle Werte aus dem Feld ***Name*** liefern:

```
mysql> SELECT Name FROM Customer;
+-----------+
| Name      |
+-----------+
| Hannelore |
| Anton     |
| Frauke    |
| Mike      |
+-----------+
4 rows in set (0.09 sec)
```

Man kann auch mehrere Felder, getrennt durch Kommas, angeben.
Des Weiteren gibt es eine ***WHERE*** **–**Klausel, die es erlaubt die Auswahl von Zeilen mit der Auswahl von Spalten zu kombinieren.

Modifizieren und Löschen von Daten:

Das komplette Löschen von Daten geht natürlich auch indem man die Tabelle löscht.
Alternativ und auch weniger brachial gibt es dafür in MySQL folgende Anweisung:

DELETE FROM tableName;

Man kann aber natürlich auch nur einzelne ausgewählte Werte (Felder) löschen oder diese modifizieren.
Was man alles löschen kann, ersieht man an der Auflistung der ALTER TABLE Anweisungen die im Kapitel „Erstellen und bearbeiten von Tabellen“ zu finden ist.

In der Praxis dürfte das Modifizieren von Daten, wenn auch in automatisierter Form am häufigsten vorkommen.
Bei unserem DVDShop würde man z.B. in der Product Tabelle den Lagerbestand aller Artikel ständig auf den neuesten Stand haben wollen, oder in der Tabelle Customer die Anzahl der erworbenen Artikel inkrementieren, falls ein Kunde mehr Artikel erworben hat. Hierfür benutzt man das ***UPDATE*** statement.

Im nächsten Beispiel soll unser Customer Anton einen weiteren Artikel erworben haben. Dazu müssen wir die Anzahl der Artikel um 1 inkrementieren und den Umsatz um den Wert des Artikels erhöhen:

```
mysql> UPDATE Customer SET Artikelanzahl = Artikelanzahl+1 WHERE Name = 'Anton'`
Query OK, 1 row affected (0.42 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Das Ergebnis:

```
mysql> SELECT * FROM Customer;
+-----------+------------+--------------+------------+---------------+---------+----------+
| Name      | Vorname    | Geburtsdatum | Geschlecht | Artikelanzahl | Umsatz | Herkunft |
+-----------+------------+--------------+------------+---------------+---------+----------+
| Hannelore | Musterfrau | 1923-03-21   | F          |            34 |   2345 | DE       |
| Anton     | ausTirol   | 1980-03-08   | M          |            12 |    288 | DE       |
| Frauke    | Kant       | 1980-03-08   | F          |            29 |    890 | DE       |
| Mike      | Kirch      | 1956-05-17   | M          |           101 |   4000 | DE       |
+-----------+------------+--------------+------------+---------------+---------+----------+
4 rows in set (0.00 sec)
```

Auch hier kann man wieder mehrere Felder gleichzeitig ändern:

mysql> **UPDATE Customer SET Artikelanzahl = Artikelanzahl+1, Umsatz = Umsatz+29 WHERE Name = 'Anton';**
Query OK, 1 row affect,d (0.’8 sec)
Rows matched: 1 Changed: 1 Warnings: 0

Damit haben wir die Artikelanzahl um 1 inkrementiert und den Umsatz um 29 erhöht.
Das Problem dieser Art der Modifikation ist, das alle Spalten, also alle Customer mit dem Namen Anton betroffen wären von der Modifikation.
Um das zu umgehen, kann man die Anfrage weiter spezialisieren, indem man das Kriterium ***Name*** noch um das Kriterium ***Vorname*** erweitert:

UPDATE Customer SET Artikelanzahl = Artikelanzahl+1, Umsatz = Umsatz+29 WHERE Name = 'Anton' && Vorname = ’ausTirol’,

Sortieren und Zählen von Daten:

In MySQL sind die Daten nicht sortiert und die Ergebniszeilen (Ausgabe der Anfrage) bildet da zunächst keine Ausnahme. Jedoch gibt es die Möglichkeit das Ergebnis nach bestimmten Kriterien zu sortieren. Man benutzt dafür z.B. die ***ORDER BY*** Klausel.

```
mysql> SELECT Name, Geburtsdatum FROM Customer ORDER BY Geburtsdatum;
+-----------+--------------+
| Name      | Geburtsdatum |
+-----------+--------------+
| Hannelore | 1923-03-21   |
| Mike      | 1956-05-17   |
| Anton     | 1980-03-08   |
| Frauke    | 1980-03-08   |
+-----------+--------------+
4 rows in set (0.03 sec)
```

In dieser Anfrage haben wir eine Tabelle aller Customer aufsteigend geordnet nach ihrem Geburtstag bekommen. In absteigender Richtung ginge es mit dem zusätzlichen Schlüsselwort ***DESC.*** Hier ist jedoch zu beachten, dass DESC sich nur auf die Spalte bezieht, die unmittelbar davor steht!

Man kann natürlich auch über mehrere Spalten sortieren lassen. Um beispielsweise nach Herkunft und dann nach dem Geburtsdatum innerhalb der Herkunft zu sortieren usw.

Datenbanken werden oft benutzt, um die Frage zu beantworten, wie oft eine bestimmte Art von Daten in einer Tabelle erschein.
Man könnte z.B. wissen wollen, wie viel Customer es gibt oder wie viele Customer einer bestimmten Herkunft sind usw.
Für solchen Anfragen stellt MySQL die ***COUNT()*** Funktion.
Diese Funktion zählt lediglich die Anzahl von nicht NULL Ergebnissen.

Wir möchten jetzt die Anzahl aller Customer haben:

```
mysql> SELECT COUNT(*) FROM Customer;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.03 sec)
```

Wir sehen es der Wert stimmt.

Oder Anzahl der Customer pro Kombination von Name und Umsatz.

```
mysql> SELECT Name, Umsatz, COUNT(*) FROM Customer GROUP BY Artikelanzahl;
+-----------+--------+----------+
| Name      | Umsatz | COUNT(*) |
+-----------+--------+----------+
| Anton     |    288 |        1 |
| Frauke    |    890 |        1 |
| Hannelore |   2345 |        1 |
| Mike      |   4000 |        1 |
+-----------+--------+----------+
4 rows in set (0.02 sec)
```

Pattern Matching ist ein Suchalgorithmus. Das Verfahren ermittelt in endlicher Zeit, ob sich ein gegebenes Muster (Pattern) in einem (begrenzten) Suchbereich wieder findet.
Das Pattern muss dabei vorher angegeben werden.

MySQL unterstützt 2 Sorten von Pattern Matching. Nämlich das ***Standard SQL Pattern Matching*** und einer anderen Form, die auf ***regulären Ausdrücken*** basiert.

Das Standard SQL Pattern Matching verwendet die Vergleichoperatoren ***LIKE*** und ***NOT LIKE.*** Darüber hinaus gibt es noch die beiden Zeichen ’_’ und ’%’.

Die Anwendung beider möchten wir am DVDShop demonstrieren:

```
mysql> SELECT * FROM Customer WHERE Name LIKE '_____';
+--------+---------+--------------+------------+---------------+--------+----------+
| Name   | Vorname | Geburtsdatum | Geschlecht | Artikelanzahl | Umsatz | Herkunft |
+--------+---------+--------------+------------+---------------+--------+----------+
| Frauke | Kant    | 1980-03-08   | F          |            29 |    890 | DE       |
+--------+---------+--------------+------------+---------------+--------+----------+
1 row in set (0.02 sec)Customer
```

Ausgegeben wurden die Werte, deren Name eine Länge von genau 6 Zeichen hat, denn genau 6 ’_’ wurden von uns eingegeben.

Das zweite Zeichen, ’%’, ist etwas interessanter. Man hat damit die Möglichkeit nach einem beliebigen Zeichen (auch Zahl) suchen zu lassen. Dabei kann angeben, ob sich das Zeichn am Anfang, in der Mitte oder am Ende eines Wortes (Zahl) befindet.
Also: %x oder x% oder %x%, wobei x eine Zahl oder ein Buchstabe ist.

Im folgenden Beispiel möchten wir eine Liste aller Customer, deren Name mit dem Buchstaben a beginnt:

```
mysql> SELECT * FROM Customer WHERE Name LIKE 'a%';
+-------+----------+--------------+------------+---------------+--------+---`--’---+
| Name  | Vorname  | Geburtsdatum | Geschlecht | Artikelanzahl | Umsatz | Herkunft |
+-------+----------+--------------+------------+---------------+--------+----------+
| Anton | ausTirol | 1980-03-08   | M          |            12 |    288 | DE       |
+-------+----------+--------------+------------+---------------+--------+----------+
1 row in set (0.00 sec)
```

NOT LIKE würde an diesem Beispiel alle Namen ausgeben, die nicht mit dem Buchstaben a beginnen.

Der zweite Pattern Matching Typ, den MySQL unterstützt, verwendet die Operatoren ***RLIKE*** und ***NOT RLIKE*** und nutzt explizit reguläre Ausdrücke, die wir schon an anderer Stelle verwendet haben, wie z.B. der kleine Stern * .
Es gibt eine ganze Reihe von regulären Ausdrücken in MySQL, wir können aber nicht auf alle eingehen und erwähnen daher nur einige von ihnen, die wir auch benutzen werden.

’.’ Der einfache Punkt repräsentiert ein beliebiges Zeichen und wäre somit im Standard Pattern Matching gleichzusetzen mit dem Zeichen ’_’.

[..] Eckige Klammern bilden kann man interpretieren als: a | b | c .D.h. entweder a oder b oder c oder auch alle 3 Buchstaben in beliebiger Mischung Und Anzahl müssen vorkommen.

* Der kleine Stern wäre in der theoretischen Informatik die geschweifte Klammer {} und bedeutet so viel wie: Etwas kommt von 0…x Mal vor.
x* würde eine beliebige Anzahl der Zeichen x treffen.

^ Das ist ein Marker, mit dem man den ersten vorkommenden Buchstaben definieren kann: ^x damit würde MySQL nach Werten suchen, die mit dem Zeichen x beginnen.

$ Analog zu ^, nur dass es den letzten Buchstaben markiert; x$

Neben diesen kleinen Unterschieden sehen die Anfragen jedoch genauso aus und verhalten sich auch äquivalent zu den bereits im ersten Teil vorgestellten Patterns, daher verzichten wir an dieser Stelle auf weitere Beispiele.

Indizes in MySQL

Wenn man einen bestimmten Datensatz einer Tabelle suchen möchte oder eine geordnete Tabelle einer Reihe von Datensätzen erstellen möchte, muss MySQL alle Datensätze der Tabelle laden.
Bei großen Tabellen würde aufgrund dieses Sachverhaltes die Performance deutlich leiden, daher wurden, um dem vorzubeugen, die ***Indizes*** eingeführt.

Pro Tabelle sind dabei bis zu 16 Indizes erlaubt und können prinzipiell jeder Spalte (sogar mehrmals) zugeordnet werden.

Indizes werden in einer Indextabelle abgelegt, was eine Reihe von Vor –aber auch Nachteilen mit sich bringt, die wir bereits in einem der ersten Kapitel genannt haben.
Es gibt eine Reihe von Indextypen, auf die wir im Folgenden kurz eingehen möchten:

Der einfache Index:

> Die einzige Aufgabe eines solchen Index ist es, den Zugriff auf Daten zu beschleunigen.
> Der einfache Index erlaubt es, dass mehrere Datensätze im indizierten Feld denselben Wert aufweisen.

Der Unique -Index:

> Wenn feststeht, dass eine Spalte eindeutige Werte enthält, sollte man den Index mit dem Schlüsselwort ***UNIQUE*** definieren.
> Das führt dazu, dass der Index effizienter wird und keine Datensätze (im indizierten Feld) doppelt vorkommen werden.

Der Primärindex:

> Der Primärindex ist im Grunde ein gewöhnlicher UNIQUE Index, jedoch mit der Besonderheit, jedoch mit der Besonderheit, dass er den Namen PRIMARY hat.

Zusammengesetzte Indizes:

Der zusammengesetzte Index, ist ein Index der mehrere Spalten umfasst.
Die Besonderheit daran ist jedoch, dass MySQL ihn auch selektiv einsetzen kann, falls nach dem ersten Teilindex (oder einem zusammengesetzten Index, indem der erste Teilindex vorkommt) gesucht wird.

Volltextindex:

Wenn man in Textfeldern ganze Texte, die auch aus mehreren Wörtern bestehen können, suchen lassen möchten, bleibt ein herkömmlicher Index wirkungslos.
Und genau für solche Fälle hat man den Volltextindex eingeführt.

Wir verzichteten in diesem Kapitel extra auf Syntaxbeispiele, da wir bereits weiter vorne eine Auflistung der ALTER TABLE Anfragen gegeben haben, die auch die Erstellung von Indizes und Keys umfasst.

Benutzerverwaltung von MySQL

Die Accountverwaltung der MySQL -Datenbank die eine der wichtigsten Aufgaben eines Administrators.
Der Administrator muss in der Lage sein, wann immer es nötig ist neue Accounts zu erstellen oder wieder zu löschen. Des Weiteren darf nicht jeder Account alle Rechte besitzen.
So sollen manche die Daten nur lesen können, während andere diese auch verändern dürfen.
Jeder Account sollte selbst ebenfalls geschützt sein und das nicht nur durch den Namen des Accounts sondern auch durch ein Passwort, welches wiederum selbst verschlüsselt sein sollte.

In MySQL gibt es 2 Möglichkeiten Accounts zu erstellen:

- Indirekt, über die GRANT statements
- Direkt, durch Manipulation der MySQL *Grant-Tabellen*

Die erste Methode ist die sicherere, da sich nur schwer Fehler einschleichen können.
Änderungen dieser Art darf nur der *root –user* durchführen, welcher über die Privilegien INSERT und RELOAD verfügt.

GRANT:

GRANT *privileges* ON *database* TO '*accname*'@'*location*' IDENTIFIED BY '*password*' WITH GRANT OPTION;

Privileges gibt an, mit welchen Privilegien der Account ausgestattet werden soll. Dabei gibt's es die Möglichkeit alle Privilegien zu vergeben mit *ALL PRIVILEGES* oder auch einige bestimmte wie z.B. *RELOAD* oder *PROCESS.*
Hier ist es wie bekannt auch erlaubt mehrere Privilegien anzugeben, welche getrennt durch ein Komma anzugeben sind.

Database ist der Name einer bestimmten Datenbank. Soll ein User auf alle enthaltenen Datenbanken zugreifen dürfen, so kann man anstelle eines spezifischen Namens die Syntax: *.* zu benutzen.
An dieser Stelle arbeitet MySQL nicht sehr konsequent, denn: Will man dem Account nur den Zugriff auf eine bestimmte Datenbank gewähren, so muss man als *root* diese Datenbank vorher SELECTEN. Also:

```
mysql> use dvdshop;
Database changed

mysql>  GRANT ALL PRIVILEGES ON dvdshop TO 'test'@'localhost' IDENTIFIED BY
'testpasswort' WITH GRANT OPTION;
Query OK, 0 rows affected (0.02 sec)
```

In der Datenbankübersicht kann dieser Account dann auch nur die Datenbanken sehen, über die er auch Rechte verfügt.

Der *Accname* unterliegt nur einigen wenigen Einschränkungen.
So darf z.B. die maximale Länge des Namens 16 Zeichen nicht überschreiten und eine Änderung dieses Sachverhaltes ist nicht erlaubt.

Die *Location* besagt, von welchem Standort ein bestimmter Account sich einloggen darf. Hat ein Account z.B. als Location nur das Attribut *localhost*, so wird er, falls er versuchen würde sich von einem anderen Host anzumelden, als anonymer Account ohne Rechte behandelt werden.
Mit dem Attribut ‚%' kann sich der Account jedoch von jedem beliebigen Host verbinden.

Die Angabe eines Passwortes ist nicht zwingend erforderlich jedoch sehr zu empfehlen, falls man MySQL im Netz betreiben möchte.
Bei der GRANT –Anweisung wird das Passwort automatisch verschlüsselt.

INSERT:

Insert ist die Alternative zu der GRANT –Anweisung. Sie verlangt jedoch einen etwas höheren Wissenstand des Administrators über die Interna von MySQL. So muss er z.B. wissen, wie die Granttabellen aufgebaut sind, wenn er nur bestimmte Privilegien Verteilen möchte.

INSERT INTO user VALUES(*'location','accname'*,PASSWORD(*'password'*),
'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
FLUSH PRIVILEGES;

user ist die Tabelle, in der die Accountinformationen abgelegt werden. Sie befindet sich in der Datenbank *mysql*.

location und *accname* sind äquivalent zu denen aus dem GRANT –statement, beim Passwort muss man sich der *PASSWORD()* –Funktion bedienen, welche das Passwort verschlüsselt.

Die vielen 'Y'-Werte stellen bestimmte Privilegien dar, die dem Account gewährt werden. Mit 'N' kann man bestimmte Privilegien auch nicht zulassen, daher ist es wichtig zu wissen, in welcher Reihenfolge sich die Privilegien in der Tabelle befinden.

Host	char(60)	NO	PRI		
User	char(16)	NO	PRI		
Password	char(41)	NO			
Select_priv	enum('N','Y')	NO		N	
Insert_priv	enum('N','Y')	NO		N	
Update_priv	enum('N','Y')	NO		N	
Delete_priv	enum('N','Y')	NO		N	
Create_priv	enum('N','Y')	NO		N	
Drop_priv	enum('N','Y')	NO		N	
Reload_priv	enum('N','Y')	NO		N	
Shutdown_priv	enum('N','Y')	NO		N	
Process_priv	enum('N','Y')	NO		N	
File_priv	enum('N','Y')	NO		N	
Grant_priv	enum('N','Y')	NO		N	
References_priv	enum('N','Y')	NO		N	
Index_priv	enum('N','Y')	NO		N	
Alter_priv	enum('N','Y')	NO		N	
Show_db_priv	enum('N','Y')	NO		N	
Super_priv	enum('N','Y')	NO		N	
Create_tmp_table_priv	enum('N','Y')	NO		N	
Lock_tables_priv	enum('N','Y')	NO		N	
Execute_priv	enum('N','Y')	NO		N	
Repl_slave_priv	enum('N','Y')	NO		N	
Repl_client_priv	enum('N','Y')	NO		N	
Create_view_priv	enum('N','Y')	NO		N	
Show_view_priv	enum('N','Y')	NO		N	
Create_routine_priv	enum('N','Y')	NO		N	
Alter_routine_priv	enum('N','Y')	NO		N	
Create_user_priv	enum('N','Y')	NO		N	

Flush Privileges wird im Zusammenhang mit dem INSERT –statement benutzt, damit der Server die Granttabellen direkt neu einliest.
Würde man das nicht machen, wären die Änderungen erst beim nächsten Neustart des Servers wirksam.

Wenn man Accounts erstellen kann, möchte man auch Accounts löschen können. Dies geschieht wie gewohnt mit dem DROP –Statement.

```
mysql> DROP USER accname @ location;
Query OK, 0 rows affected (0.06 sec)
```

Literaturverzeichnis

Internetquellen:

Client – Server – Architektur:
http://www.bond-online.de/client_server.htm

Objektorientierte Datenbanken:
http://de.wikipedia.org/wiki/Objektorientierte_Datenbank

MySQL (offizielle Dokumentation):
http://dev.mysql.com/

Literatur:

MySQL official documentation
(ebenfalls von MySQL.com)