MySQL in 90 Minuten

Referat zu MySQL 4.0.13 anlässlich lug-camp.ch in Felsberg, 30. Mai 2003

Inhaltsverzeichnis

1 Einleitung **2**
- 1.1 MySQL in 90 Minuten – geht das? 2
- 1.2 Copyright-Notiz 2
- 1.3 MySQL-Hilfestellung 2
- 1.4 Das benötigen wir 2

2 Installation von MySQL 4.0.13 **3**
- 2.1 Installation unter Debian 3.0 3
- 2.2 Andere Linux-Versionen . . 5

3 Grundlagen **6**
- 3.1 MySQL-Dienst starten und beenden 6
- 3.2 Konsole starten und beenden sowie Passwort setzen . 6
- 3.3 Aufzeichnen einer SQL-Sitzung 7
- 3.4 Erstellen einer Datenbank . 7
- 3.5 Erstellen einer Tabelle 7
- 3.6 Ändern von Tabellen 8
- 3.7 Hinzufügen und Abfragen von Datensätzen 9
- 3.8 Limit-Argument 10
- 3.9 Datum- und Zeitstempel . . 10
- 3.10 Bearbeiten bestehender Datensätze 11
- 3.11 Löschen von Datensätzen . . 11
- 3.12 Datentabellen verknüpfen . 12
- 3.13 Grenzen von MySQL? 13

4 Zugriffsberechtigungen **14**
- 4.1 Wer, wann, was, aber wozu? . 14
- 4.2 Die Datenbank 'mysql' . . . 14
- 4.3 Grant-Kommando 14
- 4.4 Revoke-Kommando 16
- 4.5 Verfeinern von Zugriffsrechten 16
- 4.6 MySQL gegen Fremdzugriffe absichern 17
- 4.7 Zugriff im Netz sperren . . . 17

5 Tabellen-Typen von MySQL **18**
- 5.1 MyISAM: Schnell und bewährt 18
- 5.2 InnoDB: Jung und dynamisch 18
- 5.3 HEAP: Immer auf Abruf . . . 21
- 5.4 Tabellen sperren (Backup) . 22

6 Volltext-Datenbanken **23**
- 6.1 Volltext hinzufügen 23
- 6.2 Arbeiten mit dem Volltext . . 23
- 6.3 Einige Beispiele 24

7 Export und Import **26**
- 7.1 File_priv-Rechte 26
- 7.2 Export von Tabellen 27
- 7.3 Import von Tabellen 28
- 7.4 Arbeiten mit Bildern 28

8 Perl-Beispiel (DBI) **30**
- 8.1 Demo 'halloperl.pl' 30
- 8.2 Perl Database Admin (pDBA) 31

9 Performence und Optimierung **32**
- 9.1 Cache-Parameter 32
- 9.2 Blob-Objekte in eigenen Tabellen 33

10 Zukunft und Würdigung **35**
- 10.1 Was bringt die Gegenwart? . 35
- 10.2 Was erwartet uns in Zukunft? 35

1 Einleitung

1.1 MySQL in 90 Minuten – geht das?

Dieses Skript ist ein Experiment. Es wurde für ein MySQL-Referat anlässlich des lug-camp.ch 2003 in Felsberg erstellt und soll eine Kurzeinführung in die Materie MySQL ermöglichen.

Für dieses Skript stehen dem Verfasser zwei Arbeitstage zur Verfügung. Wunder dauern bekanntlich im allgemeinen länger, ich versuche aber trotzdem das Unmögliche erträglich zu machen.

1.2 Copyright-Notiz

1.3 MySQL-Hilfestellung

Falls ein Web-Zugang zur Verfügung steht, dürfte www.mysql.com die erste Wahl darstellen. Insbesondere das durch die User kommentierte offizielle Handbuch bietet gute Hilfestellung.

Die MySQL-Doku ist recht griffig geschrieben, wenn z.T. auch für Einsteiger etwas gar umfassend geraten. Besonders empfohlen werden kann das Kapitel 'Tutorial', das behutsam die wichtigsten Funktionen zu MySQL aufzeigt.

➲ Daneben existieren mittlerweile eine ganze Menge von Büchern. Persönlich bin ich aber der Ansicht, dass die offizielle Dokumentation von MySQL als Nachschlagewerk ausreichen sollte.

1.4 Das benötigen wir

Für dieses Tutorial benötigen wir eine Linux-Distribution, wobei wir nachfolgend mit Debian 3.0 arbeiten werden. Seitens von MySQL wird eine Version 4.0.x erwartet. Wie wir diese Version unter Debian installieren, wird nachfolgend beschrieben.

2 Installation von MySQL 4.0.13

2.1 Installation unter Debian 3.0

Nachfolgend möchte ich die Installation eines Binär-Paketes der aktuellen Version 4.0.13 beschreiben. Im konkreten Fall steht auf dem Rechner Debian 3.0 zur Verfügung. Wir gehen zudem davon aus, dass bereits eine frühere Version von MySQL (z.B. 3.23.x bei Debian 3.0) aufgespielt worden ist.

➲ Damit alle Linux-Distributionen eine Chance erhalten, verwenden wir nicht die Debian-Packages, sondern das Standard-Binär-Paket ab MySQL-Homepage.

Bevor wir die Installation der Version 4.0.13 vornehmen, sollten wir einen gestarteten MySQL-Server mit **`/etc/init.d/mysql stop`** herunterfahren.

➲ Debian-User sollten zudem die MySQL-Debian-Packages mit 'dselect' auf 'Hold' setzen.

Nach dem Download müssen wir mit root-Rechten einige Dinge tun, für die wir besser gleich ein Skript schreiben:

```
mv mysql-standard-4.0.13-pc-linux-i686.tar.gz /usr/local
cd /usr/local
tar xvfz mysql-standard-4.0.13-pc-linux-i686.tar.gz
ln -s mysql-standard-4.0.13-pc-linux-i686 mysql
cd mysql
chown -R root .
chown -R mysql data
chgrp -R mysql .
cd /etc
ln -s ./mysql/my.cnf my.cnf
cd init.d
```

Ok, wir können das auch Zeile für Zeile eingeben, aber ein Shell-Skript ist trotzdem eleganter. Die Installation ist nun beinahe abgeschlossen. Im Verzeichnis '/usr/local/mysql' finden wir die Datei 'INSTALL-BINARY', welche eine Binär-Installation recht gut beschreibt, wenn auch aus dem Blickwinkel, dass zuvor noch kein MySQL installiert wurde. Die wichtigsten Zeilen der Datei hier zur Information:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> cd /usr/local
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> scripts/mysql_install_db
shell> chown -R root  .
shell> chown -R mysql data
shell> chgrp -R mysql .
shell> bin/mysqld_safe --user=mysql &
```

Wichtig für uns ist die Zeile 'scripts/mysql/install_db'. Mit dieser Zeile kann die Passwort- bzw. User-Datenbank 'mysql' komplett neu aufgebaut werden. Es ist zudem möglich, den gewünschten Installationspfad für die MySQL-Datenbanken mit anzugeben:

```
scripts/mysql_install_db --datadir=/mnt/archiv
```

Danach müssten noch die Rechte für das entsprechende Verzeichnis auf den User und die Gruppe 'mysql' gelegt werden. Und damit beim Start von MySQL der gewünschte Datenpfad (in unserem Beispiel '/mnt/archiv') auch gefunden wird, müssen wir in /etc/my.cnf den gewünschten Eintrag vornehmen:

```
[mysqld]
datadir=/mnt/archiv
```

Da wir bei Debian 3.0 aber i.d.R. bereits eine MySQL-Installation vorfinden, möchte ich einen anderen Weg vorschlagen. Im Prinzip genügt es, die beiden Dateien '/etc/mysql/my.cnf' sowie '/etc/init.d/mysql' mit einigen Änderungen zu versehen, damit die Version 4.0.13 bereits beim Hochfahren des Systems sauber und automatisch gestartet werden kann. Zunächst ändern wir die Datei 'my.cnf' wie folgt:

```
# aus 'language = /usr/share/mysql/english' wird
language = /usr/local/mysql/share/mysql/english
```

➲ Achtung: Ohne den Link von '/etc/mysql/my.cnf' nach '/etc/my.cnf' übernimmt MySQL 4.0.13 unter Debian 3.0 keine Einstellungen. Wir erstellen deshalb einen Link:

```
ln -s /etc/mysql/my.cnf /etc/my.cnf
```

Weiter müssen wir die Datei '/etc/init.d/mysql' an einigen Stellen anpassen:

```
...
#aus 'test -x /usr/sbin/mysqld || exit 0' wird
test -x /usr/local/mysql/mysqld_safe || exit 0
...
# aus 'MYADMIN="/usr/bin/mysqladmin -u root ..."' wird
MYADMIN="/usr/local/mysql/bin/mysqladmin -u root ..."
...
# Start daemon
# aus '/usr/sbin/mysqld ...' wird (in zwei Zeilen):
cd /usr/local/mysql
./bin/mysqld_safe > /dev/null 2>&1 &
...
```

Überall, wo '...' steht, muss die Datei '/etc/init.d/mysql' gemäss der Vorlage belassen werden. Damit ist die Installation von MySQL 4.0.13 abgeschlossen; wir können nun den MySQL-Server ganz normal starten, genauso wie beim Starten von Debian 3.0 nun anstelle der Version 3.23.x die Version 4.0.13 hochgefahren wird.

2.2 Andere Linux-Versionen

Bei kommerziellen Distributionen dürfte MySQL mittlerweile bereits in der Version 4.0.x vorliegen. Sollte dies nicht der Fall sein, so sei (aufgrund eigener Erfahrungen) gesagt, dass das Installieren eines Binär-Paketes an sich nicht problematisch ist (wir können wie oben beschrieben vorgehen). Nur das Hochfahren dürfte (analog zu Debian) manchmal nicht ganz so trivial sein. Zentral sind die richtigen Werte für die Verzeichnisse 'tmp', 'bin' sowie 'datadir' sowie der Speicherort der PID- und Socket-Dateien.

Im übrigen sei darauf hingewiesen, dass Knoppix (siehe www.knoppix.net mit Version vom 16.5.2003) bereits die Version 4.0.x von MySQL enthält.

3 Grundlagen

3.1 MySQL-Dienst starten und beenden

Am einfachsten starten wir den MySQL-Server über das Init-Skript '/etc/init.d/mysql'. Wir erhalten dabei eine Bestätigungsmeldung, dass der MySQL-Server gestartet werden konnte.

Genauso einfach können wir den MySQL-Serverdienst wieder stoppen. Dies erreichen wir durch '/etc/init.d/mysql stop'. Es gilt zu beachten, dass sowohl das Starten als auch das Beenden 'root'-Privilegien erfordert.

3.2 Konsole starten und beenden sowie Passwort setzen

In diesem Tutorial arbeiten wir ausschliesslich mit dem Konsolenprogramm 'mysql'. Wer mit der Kommando-Zeile arbeitet, der wird sich auch mit 'mysql' kaum schwertun.

Selbstverständlich gibt es eine ganze Reihe von grafischen Front-Ends, doch das würde den Rahmen dieses Tutorials sprengen. Also ran an die Konsole. Wir starten MySQL mit 'mysql'. Falls die Verbindung klappt, sehen wir die MySQL-Eingabeaufforderung: **`mysql>`**. Allerdings funktioniert das nur, wenn wir uns lokal mit dem aktuellen Usernamen und ohne Passwort anmelden. Spätestens nach dem Absichern von MySQL dürfte der normale Anmeldevorgang so aussehen:

```
mysql -u user -h host -p
```

'-u' steht für User, '-h' für Host und '-p' bedeutet, dass ein Passwort abgefragt werden soll. Nun melden wir uns als Superuser 'root' am lokalen Datenbank-Server mit einer Passwortabfrage an:

```
mysql -u root -h localhost -p
```

Da wir dem User 'root' noch kein Passwort vergeben haben, müssen wir die Passwortabfrage mit der Enter-Taste bestätigen. Als erstes sollten wir dem User 'root' ein Passwort 'genehmigen'. Dies erreichen wir mit:

```
set password for root@localhost=Password('secret');
```

Der Strichpunkt am Ende ist zwingend erforderlich, damit der SQL-Befehl definitiv an den MySQL-Server weitergereicht werden kann. Nachdem wir das Passwort vergeben haben, gelangen wir mit 'quit' zurück zur Linux-Shell.

3.3 Aufzeichnen einer SQL-Sitzung

Um unsere SQL-Grundlagen etwas aufzufrischen, möchten wir nachfolgend eine kleine SQL-Sitzung wiedergeben. Und damit wir das Erstellen einer SQL-Sitzung auch gleich lernen, sei hier auf die Capture-Möglichkeit innerhalb der Konsole hingewiesen:

```
mysql>\T /home/up/log.sql
Logging to file '/home/up/log.sql'
mysql>
```

Sobald wir genügend aufgezeichnet haben, schalten wir die Log-Datei wieder ab mit:

```
mysql>\t
Outfile disabled.
mysql>
```

Wir können nun die Sitzungsdatei in jedem beliebigen Editor einsehen.

3.4 Erstellen einer Datenbank

Bevor wir eine neue Datenbank erstellen, wollen wir kurz nachsehen, welche Datenbanken es bereits gibt:

```
mysql> show databases;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.00 sec)
```

Nun eröffnen wir eine neue Datenbank mit dem nachfolgenden Befehl:

```
mysql> create database archiv;
Query OK, 1 row affected (0.00 sec)
```

3.5 Erstellen einer Tabelle

Bevor wir in der Datenbank 'archiv' eine Tabelle eröffnen können, müssen wir die Datenbank aktivieren:

```
mysql> use archiv;
Database changed
```

Nun können wir in der Datenbank 'archiv' eine erste Tabelle einrichten:

```
mysql> create table akten (Titel varchar(30), Datum datetime,
    -> Seiten int not null default 0,
    -> ID int not null default 0 primary key auto_increment);
Query OK, 0 rows affected (0.00 sec)
```

Werfen wir einen Blick auf die Zeilen 2 und 3. Diese beginnen mit **->**. MySQL zeigt damit an, dass der Befehl noch nicht abgeschlossen ist, d.h. MySQL erwartet ein weiteres SQL-Fragment bzw. das Abschlusszeichen ';', ehe der Befehl zum Server übermittelt wird.

Die Felder 'Titel', 'Datum' und 'Seiten' sollten klar sein, das Feld 'ID' dürfte schwieriger zu verstehen sein, weil wir diesem Feld gleich auch einen Primärindex (primary key) sowie einen automatischen Zähler (auto_increment) verpassen. Mit dem Primärindex erreichen wir, dass auf das Feld 'ID' schneller als auf jedes andere Feld ein Zugriff möglich ist und mit 'auto_increment' stellen wir sicher, dass bei jedem Hinzufügen eines Datensatzes das Feld 'ID' einen eindeutigen Wert erhält.

Damit wir etwas Übung erhalten, eröffnen wir gleich noch eine zweite Tabelle:

```
mysql> create table Seiten (AkteID int not null default 0,
    -> Seite int not null default 0,
    -> Text mediumtext,
    -> Bild mediumblob);
Query OK, 0 rows affected (0.00 sec)
```

3.6 Ändern von Tabellen

MySQL-Tabellen (zumindest jene im MyISAM-Format) haben die Eigenheit, dass diese als Datei auf Betriebssystemebene erstellt und verwaltet werden. Damit später keine Namenskonflikte auf anderen Betriebssystemen (z.B. unter Windows) entstehen, empfiehlt es sich, die Tabellen mit Kleinbuchstaben zu verwalten.

Wir könnten nun die Konsole verlassen und die Tabellendateien 'Seiten' mit 'mv ...' umbenennen. Das ist aber unschön, stehen zum Ändern von Tabellen doch mächtige Befehle zur Verfügung. Wir verwenden dazu 'alter table ...'. Nachfolgend benennen wir die Tabelle 'Seiten' in 'seiten' um:

```
mysql> alter table Seiten rename seiten;
Query OK, 0 rows affected (0.00 sec)
```

Normalerweise sollten wir beim Erstellen von Tabellen darauf achten, dass wir bei zentralen Feldern einen Index anlegen. Da wir dies beim Erstellen der Tabelle nicht gemacht haben, holen wir das jetzt nach:

```
mysql> alter table akten add index TitelI (Titel);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Ebenso möchten wir dem Feld 'Titel' mehr Platz einräumen, d.h. anstelle der bisherigen 30 Zeichen, soll das Feld maximal bis zu 100 Zeichen (möglich wären max. 255) aufnehmen.

```
mysql> alter table akten change Titel Titel varchar(100);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Wir können jederzeit ein Feld vergrössern, ohne dass dies einen Datenverlust zur Folge hat. Das Verkleinern von Feldern ist weitaus kritischer, da wir dabei unter Umständen Feldwerte zerstören. Damit wir auch bei 'alter table' Übung erhalten, geben wir nacheinander die folgenden SQL-Befehle ein:

```
mysql> alter table akten add index DatumI (Datum);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table seiten add primary key (AkteID,Seite);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Nun können wir den Aufbau der Tabelle mit 'describe seiten' begutachten:

```
mysql> describe seiten;
+--------+------------+------+-----+---------+-------+
| Field  | Type       | Null | Key | Default | Extra |
+--------+------------+------+-----+---------+-------+
| AkteID | int(11)    |      | PRI | 0       |       |
| Seite  | int(11)    |      | PRI | 0       |       |
| Text   | mediumtext | YES  |     | NULL    |       |
| Bild   | mediumblob | YES  |     | NULL    |       |
+--------+------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

3.7 Hinzufügen und Abfragen von Datensätzen

Nachdem die Struktur der beiden Tabellen in Ordnung ist, gehen wir nun dazu über, die Tabellen mit Werten zu füllen:

```
mysql> insert into akten (Titel,Datum) values ('Meine erste Akte',
    -> '2003-05-28');
Query OK, 1 row affected (0.00 sec)
```

Im Prinzip ist der Eintrag ok, wenn wir 'Query OK,...' erhalten. Zur Sicherheit können wir die Tabelle aber abfragen. Dies erledigen wir mit dem SQL-Befehl 'select':

```
mysql> select * from akten;
+-----------------+---------------------+--------+----+
| Titel           | Datum               | Seiten | ID |
+-----------------+---------------------+--------+----+
| Meine erste Akte | 2003-05-28 00:00:00 |      0 |  1 |
+-----------------+---------------------+--------+----+
1 row in set (0.38 sec)
```

Die Abfrage 'select * from akten' zeigt sämtliche Datensätze der Tabelle 'akten' an. Fügen wir nun einen zweiten Datensatz hinzu:

```
mysql> insert into akte (Titel,Datum) values ('Meine zweite Akte',
    -> \c
mysql> insert into akten (Titel,Datum) values ('Meine zweite Akte',
    -> '2003-05-01');
Query OK, 1 row affected (0.00 sec)
```

Bei der ersten Abfrage ist mir ein Fehler (welcher?) unterlaufen, deshalb habe ich den Befehl mit '\c' gestoppt, damit ich den Befehl nochmals richtig eingeben kann. Bei dieser Gelegenheit sei erwähnt, dass einmal eingetippte Befehle bequem mit den Pfeiltasten nochmals zur Ausführung gebracht werden können.

3.8 Limit-Argument

Bei grossen Datenbeständen ist eine Abfrage 'select * from akten' problematisch, weil damit sämtliche Datensätze angezeigt werden. Bei einigen Millionen könnte bis zum letzten Datensatz eine längere Wartezeit entstehen. Diese kann vermieden werden, indem wir mit dem Limit-Argument arbeiten:

```
mysql> select * from akten limit 1;
+-----------------+---------------------+--------+----+
| Titel           | Datum               | Seiten | ID |
+-----------------+---------------------+--------+----+
| Meine erste Akte | 2003-05-28 00:00:00 |      0 |  1 |
+-----------------+---------------------+--------+----+
1 row in set (0.38 sec)
```

Obwohl wir mittlerweile zwei Datensätze haben, wird (aufgrund des Limit-Arguments nur der erste Datensatz angezeigt). Möglich wäre z.B. auch 'limit 1,1' gewesen. Damit würde genau der zweite Datensatz angezeigt. Ausnahmsweise ist das kein Schreibfehler, denn limit 0,1 würde den ersten Datensatz zurückgeben, d.h. die Null wird mitgezählt.

3.9 Datum- und Zeitstempel

Die Option 'auto_increment' für Zahlfelder haben wir bereits besprochen, etwas ähnliches gibt es auch für Datumsfelder. Wenn wir möchten, dass MySQL immer gleich das aktuelle Datum erfasst, können wir ein 'timestamp'-Feld anlegen.

```
mysql> alter table akten add DatumErfasst timestamp;
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from akten;
+-----------------+---------------------+--------+----+----------------+
| Titel           | Datum               | Seiten | ID | DatumErfasst   |
+-----------------+---------------------+--------+----+----------------+
| Meine erste Akte  | 2003-05-28 00:00:00 |      0 |  1 | 20030526214715 |
| Meine zweite Akte | 2003-05-01 00:00:00 |      0 |  2 | 20030526214715 |
+-----------------+---------------------+--------+----+----------------+
2 rows in set (0.00 sec)
```

Damit wir den gewünschten Effekt sehen können, müssen wir erneut einen Datensatz hinzufügen:

```
mysql> insert into akten (Titel,Datum) values ('Jetzt mit Zeitstempel',
    -> '2003-05-28');
Query OK, 1 row affected (0.00 sec)

mysql> select * from akten;
+----------------------+---------------------+--------+----+----------------+
| Titel                | Datum               | Seiten | ID | DatumErfasst   |
+----------------------+---------------------+--------+----+----------------+
| Meine erste Akte     | 2003-05-28 00:00:00 |      0 |  1 | 20030526214715 |
| Meine zweite Akte    | 2003-05-01 00:00:00 |      0 |  2 | 20030526214715 |
| Jetzt mit Zeitstempel | 2003-05-28 00:00:00 |      0 |  3 | 20030526214830 |
+----------------------+---------------------+--------+----+----------------+
3 rows in set (0.00 sec)
```

Bei der Abfrage sehen wir, dass der dritte Datensatz einen aktuelleren Zeitstempel besitzt als die ersten beiden Datensätze. Der Zeitstempel wird bei jedem Verändern der Datentabelle gesetzt.

➲ Pro Tabelle können wir zwar mehrere 'timestamp'-Felder anlegen, automatisch ausgefüllt wird aber immer nur das erste Feld. Ein zweites 'timestamp'-Feld könnten wir z.B. dafür verwenden, um neben dem Änderungs- auch das Erfassungsdatum festzuhalten.

3.10 Bearbeiten bestehender Datensätze

Datensätze können wir mit 'update' aktualisieren. Falls wir nicht sämtliche Datensätze einer Tabelle bearbeiten wollen (dies dürfte regelmässig der Fall sein), sollten wir immer eine Einschränkung 'where' übermitteln. Nachfolgend ein Beispiel:

```
mysql> update akten set Titel='Akte mit Zeitstempel' where ID=3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Dank 'where ID=3' wird nur ein Datensatz geändert, was i.d.R. dem entspricht, was sinnvoll sein dürfte.

3.11 Löschen von Datensätzen

Genau gleich verfahren wir beim Löschen von Datensätzen:

```
mysql> delete from akten where ID=2;
Query OK, 1 row affected (0.01 sec)
```

Kleine Quizfrage am Rande: Wieviele Datensätze haben wir nun in der Tabelle? Drei minus eins ergibt hoffentlich keine zwei Fragezeichen. Wir müssen ja nicht gleich 'Erbsen' zählen, aber es ist beruhigend zu wissen, wie gross (mindestens in etwa) unsere Datentabellen sind.

➲ Wer Mühe mit Kopfrechnen hat, möge mit 'select count(*) from akten' kurz nachzählen.

Wenden wir uns noch kurz der zweiten Tabelle zu. Alles, was wir mit der Tabelle 'akten' gemacht haben, können wir auch mit der Tabelle 'seiten' anstellen. Fügen wir zunächst zwei Datensätze hinzu:

```
mysql> insert into seiten (AkteID,Seite,Text) values
    -> (1,1,'Ferien, Felsberg, Schweiz');
Query OK, 1 row affected (0.00 sec)

mysql> insert into seiten (AkteID,Seite,Text) values
    -> (1,2,'Wir erfassen eine zweite Seite');
Query OK, 1 row affected (0.00 sec)
```

3.12 Datentabellen verknüpfen

Warum gibt es in der Tabelle 'seiten' das Feld 'AkteID'? Ganz einfach, weil wir die beiden Tabellen miteinander verknüpfen wollen. Ohne, dass wir das beim Anlegen der Datenstrukturen an die grosse Glocke gehängt haben, soll in unserer Datenhaltung gelten: Eine Akte enthält mehrere Seiten und die Tabelle 'seiten' nimmt die einzelnen Seiten letztlich auf. Wir wollen nun nachschauen, wieviele Seiten wir in der ersten Akte haben:

```
mysql> select akten.ID,akten.Titel,seiten.Text from akten,
    -> seiten where akten.ID=seiten.AkteID;
+----+-----------------+-------------------------------+
| ID | Titel           | Text                          |
+----+-----------------+-------------------------------+
|  1 | Meine erste Akte | Ferien, Felsberg, Schweiz     |
|  1 | Meine erste Akte | Wir erfassen eine zweite Seite |
+----+-----------------+-------------------------------+
2 rows in set (0.00 sec)
```

Damit die Abfrage gelingt, sagen wir MySQL einfach 'where akten.ID=seiten.AkteID'. So einfach geht das. Falls wir nur an der Anzahl interessiert sind, könnten wir auch nur die Anzahl (und nicht die Feldwerte) abrufen:

```
mysql> select count(akten.ID) from akten,seiten
    -> where akten.ID=seiten.AkteID;
+-----------------+
| count(akten.ID) |
+-----------------+
|               2 |
+-----------------+
1 row in set (0.00 sec)
```

Aus kosmetischen Gründen würde ich die nachfolgende Variante empfehlen:

```
mysql> select count(akten.ID) as AnzAkten from akten,
    -> seiten where akten.ID=seiten.AkteID;
+----------+
| AnzAkten |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

Mit dem Zusatz 'as AnzAkten' erreichen wir, dass eine 'unleserliche' Spaltenüberschrift lesbarer wird.

3.13 Grenzen von MySQL?

Betrachten wir die nachfolgende Ausgabe:

```
mysql> select akten.ID,akten.Titel,akten.Seiten,
    -> seiten.Text from akten,seiten where akten.ID=seiten.AkteID;
+----+-----------------+--------+-------------------------------+
| ID | Titel           | Seiten | Text                          |
+----+-----------------+--------+-------------------------------+
|  1 | Meine erste Akte |      0 | Ferien, Felsberg, Schweiz     |
|  1 | Meine erste Akte |      0 | Wir erfassen eine zweite Seite |
+----+-----------------+--------+-------------------------------+
2 rows in set (0.01 sec)
```

Das Feld 'Seiten' der Tabelle 'akten' führt uns in die Irre, bestehen doch in der Tabelle 'seiten' bereits zwei Seiten für die erste Akte. Im Prinzip könnten wir immer über die Tabelle 'seiten' die aktuelle Anzahl der Seiten je Akte abfragen; und so würde es wohl auch im Lehrbuch stehen. Abfragen über mehrere Tabellen können aber bei umfangreichen Datenbeständen langsam werden. Damit dies später nicht auftritt, legen wir die Anzahl der verfügbaren Seiten direkt in der Tabelle 'akten' im Feld 'Seiten' ab. Allerdings handeln wir uns dabei das Problem ein, dass wir bei jedem Hinzufügen einer Seite in der Tabelle 'seiten' den Wert 'Seiten' in der Tabelle 'akten' anpassen müssen. Das würde letztlich in etwa so aussehen:

```
mysql> insert into seiten (AkteID,Seite,Text)
    -> values (1,3,'Jetzt haben wir die dritte Seite');
Query OK, 1 row affected (0.01 sec)

mysql> update akten set Seiten=3 where ID=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from akten;
+---------------------+---------------------+--------+----+---------------+
| Titel               | Datum               | Seiten | ID | DatumErfasst  |
+---------------------+---------------------+--------+----+---------------+
| Meine erste Akte    | 2003-05-28 00:00:00 |      3 |  1 | 20030526215835 |
| Akte mit Zeitstempel | 2003-05-28 00:00:00 |      0 |  3 | 20030526214942 |
+---------------------+---------------------+--------+----+---------------+
2 rows in set (0.01 sec)
```

Ob das Verfahren Sinn macht, darüber kann gestritten werden. Letztlich entspricht das Feld 'Seiten' in der Tabelle 'akten' genau dem gleichen Wert, wie wenn wir ihn direkt aus der Tabelle 'seiten' abfragen würden. Die meisten Datenbanksysteme bieten für solche Fälle Referenzierungen (references) an. MySQL bietet eine solche Option derzeit noch nicht an für die MyISAM-Tabellen (davon später mehr) und solange dies der Fall ist, ergibt die hier angeführte Lösung m.E. durchaus Sinn.

4 Zugriffsberechtigungen

4.1 Wer, wann, was, aber wozu?

Natürlich können wir die Zugriffsrechte von MySQL so belassen, wie diese MySQL bei der Installation vergibt. Wir könnten MySQL gar so konfigurieren, dass Otta und Otto uneingeschränkt das Warenlager (mindestens in der Datenbank) leerräumen können, ohne auch nur einen Gegenstand in die Hand nehmen zu müssen. Es gibt aber auch Gründe, dass wir das nicht möchten. Und immer dann sollten wir (wenigstens) in Grundlagen die Zutrittsmöglichkeiten zu MySQL kennen und entsprechend einstellen können.

4.2 Die Datenbank 'mysql'

In der Datenbank 'mysql' werden die Zugriffsrechte verwaltet. Betrachten wir zunächst einmal die Tabellen, die uns dazu zur Verfügung stehen:

```
mysql> use mysql;

Database changed
mysql> show tables;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| func            |
| host            |
| tables_priv     |
| user            |
+-----------------+
6 rows in set (0.00 sec)
```

In der Tabelle 'user' können wir globale Rechte für User festlegen. Die Tabellen 'db', 'host', 'tables_priv' und 'columns_priv' dienen dazu, die Rechte auf Stufe Datenbank(en), Rechner (Host), Tabelle(n) sowie Spalte(n) festzulegen.

Im Prinzip könnten wir die entsprechenden Tabellen mit 'insert' bzw. 'update' Kommandos entsprechend bearbeiten. Einmal davon abgesehen, dass das eine relativ mühsame Sache ist, dürften wir dabei nicht vergessen, dass wir nach jeder Änderung das Kommando 'flush privileges' eingeben müssen; erst dann werden die Änderungen aktiv geschaltet. Wesentlich eleganter sind die Befehle 'grant' und 'revoke', weil wir so mit einem SQL-Befehl eine effiziente Rechtevergabe vornehmen können.

4.3 Grant-Kommando

Nehmen wir einmal an, dass wir in unserer Datenbank 'archiv' ein Gast-Konto benötigen. User 'gast' soll alles lesen, aber keine Änderungen vornehmen dürfen. Dazu geben wir den folgenden Befehl ein:

```
mysql> grant select on archiv.* to gast@localhost identified by '';
Query OK, 0 rows affected (0.01 sec)
```

Hinter 'grant' stehen zunächst einmal jene Rechte, welche wir dem entsprechenden User erteilen möchten. Typischerweise werden wir zwischen 'all', 'select,insert,update,delete' sowie 'select' wählen.

Danach (d.h. bei 'on') bestimmen wir die Datenbank(en) sowie die Tabelle(n). Die Kombination '*.*' bedeutet dabei, dass wir Rechte für sämtliche Datenbanken und Tabellen vergeben möchten; seien wir etwas vorsichtig mit den Sternen, damit wir später nicht 'blaue' Wunder erleben.

Mit 'to' legen wir fest, wer von den Rechten profitieren soll. Wir müssen uns merken, dass nebem dem User (z.B. 'gast') immer auch noch der Zugriff geografisch einzugrenzen ist. Im einfachsten Falle werden wir die Rechte für gast@localhost oder gast@'%' (beim zweiten Beispiel benötigen wir zwingend die Apostroph-Zeichen) festlegen. 'localhost' bedeutet nur jener Rechner, auf dem der Server installiert ist, das Prozentzeichen '%' dagegen umfasst die übrige Welt. Das kann eine IP-Adresse sein, oder auch nur ein bestimmter IP-Adressenbereich. Ebenfalls denkbar (und sinnvoll) ist die Angabe eines Domain- bzw. Rechnernamens (sofern ein DNS-Dienst zur Verfügung steht).

➲ Ich gebe zu, unter Linux dürfte 'localhost' allgemein bekannt sein, aber fast jeder Windows-User (bzw. Linux-Umsteiger) wird für eine solche Erklärung dankbar sein.

Und falls das alles zu kompliziert klingt, so helfen ganz sicher einige Beispiele:

```
mysql> grant all on archiv.* to up@localhost
    -> identified by 'secret' with grant option;
Query OK, 0 rows affected (0.01 sec)
```

Wir geben dem User 'up@localhost' sämtliche Rechte in der Datenbank 'archiv'. Eine Anmeldung erfolgt mit dem Password 'secret' und der User 'up' kann seinerseits Rechte der Datenbank 'archiv' an andere Benutzer vergeben. Um dies zu prüfen, verlassen wir kurz die Konsole und melden uns mit 'mysql -u up -p -D archiv' erneut an. Nun kann 'up' weitere Rechte für die Datenbank 'archiv' vergeben:

➲ Mit der Option '-D datenbank' können wir beim Start direkt eine Datenbank aktivieren, was uns das Kommando 'use archiv' erspart.

```
mysql> grant select on archiv.* to gast@'%';
Query OK, 0 rows affected (0.01 sec)
```

Der User 'up' kann für die Datenbank 'archiv' auch neue User anlegen; nicht aber für diese neuen User Passwörter vergeben, da dies Rechte in der Datenbank 'mysql' erfordert. Möglich ist aber, dass der neu angelegte User 'gast@localhost' jederzeit selber das eigene Passwort setzt bzw. ändert:

```
mysql> set password=Password('secret');
```

4.4 Revoke-Kommando

Das Gegenstück zu 'grant' stellt 'revoke' dar. Die Syntax ist plus/minus gleich, wenn wir davon absehen, dass aus 'grant ... on ... to ...' ein 'revoke ... on ... from ...' wird. Konkret wollen wir unserem Gast die Rechte wieder entziehen. Dies erreichen wir mit:

```
mysql> revoke select on archiv.* from gast@'%';
Query OK, 0 rows affected (0.01 sec)
```

Wir sollten uns klar sein, dass der User 'gast' von irgendwoher nun nicht mehr in die Datenbank 'archiv' reinkommt, dass er sich aber gleichwohl noch in MySQL anmelden kann. Ist das tragisch? In gewisser Weise schon, steht doch per Default die Datenbank 'test' sämtlichen Usern zur Verfügung. Unser mittlerweile unerwünschte externe Gast könnte nun in dieser Datenbank 'test' wahllos Tabellen eröffnen und unsere Festplatte in die Knie zwingen. Dies sollten wir vermeiden, indem wir z.B. mit 'root@localhost' den User 'gast@%' definitiv aus der Tabelle 'user' entfernen:

```
mysql> use mysql;
Database changed
mysql> delete from user where User='gast' and Host='%';
Query OK, 1 row affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

Da wir bei diesem Beispiel direkt eine Änderung in der Tabelle 'user' vornehmen, benötigen wir anschliessend das Kommando 'flush privileges'; erst damit wird die Änderung aktiviert.

Ein Spezialfall besteht bei den 'grant'-Rechten. Hat ein User 'grant'-Rechte ('with grant option') erhalten, so müssen wir diese mit 'revoke grant option ...' zusätzlich zum normalen 'revoke'-Kommando aus der 'mysql'-Datenbank entfernen.

4.5 Verfeinern von Zugriffsrechten

Nachfolgend möchten wir dem User 'gast@localhost' genau das Feld 'Titel' der Tabelle 'akten' in der Datenbank 'archiv' zum Ändern freigeben. Dies erreichen wir mit:

```
mysql> grant update(Titel) on archiv.akten to gast@localhost;
Query OK, 0 rows affected (0.01 sec)
```

Und nun melden wir uns kurz als als 'gast@localhost' an und machen die Probe aufs Exempel:

```
mysql> select * from akten;
+----------------------+---------------------+--------+----+----------------+
| Titel                | Datum               | Seiten | ID | DatumErfasst   |
+----------------------+---------------------+--------+----+----------------+
| Meine erste Akte     | 2003-05-28 00:00:00 |      3 |  1 | 20030526215835 |
| Akte mit Zeitstempel | 2003-05-28 00:00:00 |      0 |  3 | 20030526214942 |
+----------------------+---------------------+--------+----+----------------+
```

```
2 rows in set (0.01 sec)

mysql> update akten set Titel='Gast war hier' where ID=1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from akten;
+---------------------+---------------------+--------+----+----------------+
| Titel               | Datum               | Seiten | ID | DatumErfasst   |
+---------------------+---------------------+--------+----+----------------+
| Gast war hier       | 2003-05-28 00:00:00 |      3 |  1 | 20030527144213 |
| Akte mit Zeitstempel | 2003-05-28 00:00:00 |      0 |  3 | 20030526214942 |
+---------------------+---------------------+--------+----+----------------+
2 rows in set (0.01 sec)
```

Selbstverständlich kann der User 'gast@localhost' nur das Feld 'Titel' bearbeiten, im Feld 'Seiten' kann er keine Änderungen vornehmen:

```
mysql> update akten set Seiten=4 where ID=1;
ERROR 1143: UPDATE command denied to user: 'gast@localhost'
            for column 'Seiten' in table 'akten'
```

4.6 MySQL gegen Fremdzugriffe absichern

Nach einer Installation von MySQL sollten wir die Zugriffsrechte einschränken, damit niemand Unfug mit MySQL treiben kann.

Zunächst müssen wir dem User 'root@localhost' ein Passwort vergeben:

```
set password for root@localhost=Password('secret');
```

Danach sollten wir zusätzlich den User '' entfernen, und zwar sowohl unter 'localhost' als auch unter '%'. Dies erreichen wir mit:

```
mysql> use mysql;
Database changed
mysql> delete from user where User='';
Query OK, 2 rows affected (0.01 sec)
```

Damit stellen wir sicher, dass niemand eine Verbindung mit MySQL aufbauen kann; die Datenbank 'test' wird es uns danken!

➲ Unter Windows müssten wir zusätzlich 'root@%' mit einem Passwort versehen bzw. das Konto komplett löschen.

4.7 Zugriff im Netz sperren

Es gibt eine weitere Möglichkeit, MySQL-Hackern das Leben ein bisschen schwerer zu machen. In der Startdatei 'my.cnf' können wir mit 'skip-networking' festlegen, dass (egal welche Rechte wir in der Datenbank 'mysql' definiert haben) nur User vom Rechner 'localhost' aus sich am MySQL-Server anmelden können.

➲ Unter Debian 3.0 ist das nach der Installation erst einmal aktiviert. Das sollten wir uns merken, falls wir im Intranet mal nicht wissen sollten, weshalb niemand reinkommt.

5 Tabellen-Typen von MySQL

Wozu sollen wir uns mit Tabellen-Typen herumschlagen? Oder anders herum gefragt, weshalb kann das MySQL nicht selber entscheiden?

Ganz einfach deshalb, weil wir mit der Wahl des richtigen Tabellenformats ein grosses Potential an Optimierung erreichen können. Weshalb sollten wir den ganzen Balast von COMMIT/ROLLBACK (eine Erklärung dazu folgt gleich) immer mit uns tragen, wenn wir in unserer Datenbank kaum Änderungen vornehmen, dafür aber umso mehr Abfragen verarbeiten müssen?

Nachfolgend schauen wir auf die drei wichtigsten Tabellen-Typen von MySQL und werden kurz in einem Beispiel aufzeigen, wie wir Tabellen im entsprechenden Format anlegen.

5.1 MyISAM: Schnell und bewährt

In alten Tagen (so um Version 3.22) gab es in MySQL nur die ISAM-Tabellen. Damit war es z.B. nicht möglich, eine Datentabelle über 2 GByte zu eröffnen, ebenso konnten die Daten nicht ohne weiteres zwischen den verschiedenen Betriebssystemen hin und her kopiert werden.

Glücklicherweise sind diese Zeiten mit den MyISAM-Tabellen längst passé, ohne dass sie merklich langsamer geworden wären. Kurz und gut, meine MySQL-'Laufbahn' ist mit den MyISAM-Tabellen derart eng verzahnt, dass ich diese einfach nicht missen möchte, Transaktionen hin oder her!

Bei den bisherigen Beispielen haben wir immer mit den MyISAM-Tabellen gearbeitet, ohne dass uns das aufgefallen wäre. An dieser Stelle sei zu den MyISAM-Tabellen einzig noch gesagt, dass wir den Tabellentyp beim Erstellen einer Tabelle explizit angeben können:

```
mysql> create table quick (Feld1 int) type=MyISAM;
Query OK, 0 rows affected (0.00 sec)
```

5.2 InnoDB: Jung und dynamisch

Mit den InnoDB-Tabellen können wir transaktionsfähige Tabellen verwalten. Was heisst das konkret? Oft werden zunächst Datensätze erfasst, um aufgrund der erfassten Daten umfangreiche Berechnungen durchzuführen. Die Resultate verwalten wir selbstverständlich wieder in einer Tabelle bzw. entsprechenden Feldern.

Nun könnte es ja bei der Verarbeitung zu Problemen mit den Daten kommen, MySQL könnte auch von einem Stromausfall betroffen sein, sodass wir nur einen Teil der Resultate vorliegen haben. In diesem Fall kann es sinnvoller sein, die Berechnung nochmals von vorne zu beginnen. Ohne Transaktionen kann das Problem auftauchen, dass wir nicht mehr (oder nur äusserst mühsam) zum Ausgangspunkt zurückkehren können, mit Transaktionen senden wir MySQL ganz einfach eine Storno-Nachricht (rollback), und die Sache hat sich erledigt.

Als Preis für diese Annehmlichkeiten nehmen wir in Kauf, dass die Verarbeitung insgesamt länger dauern wird, weil jeder Schritt protokolliert werden muss; aber irgendeinen Grund benötigen wir ja auch, um neue Rechner anschaffen zu können.

5.2.1 Aktivieren der InnoDB-Tabellen

➲ Das Arbeiten mit dem Tabellentyp 'InnoDB' setzt voraus, dass der MySQL-Server mit diesem Tabellentyp hochgefahren wird.

Unter Debian 3.0 d.h. falls wir MySQL gemäss dieser Anleitung aufgesetzt haben, steht der InnoDB-Tabellentyp zunächst nicht zur Verfügung. Nicht weil MySQL 4.0.13 das nicht unterstützen würde, sondern weil wir die my.cnf-Datei von der Version 3.23.x übernommen haben und das dort nicht aktiviert war. Um die InnoDB-Tabellen nun zu aktivieren, müssen wir einige Änderungen in der Datei '/etc/mysql/my.cnf' vornehmen:

```
# Read the manual if you want to enable InnoDB!
# skip-innodb
innodb_data_home_dir =
innodb_data_file_path = /var/lib/mysql/ibdata/ibdata1:100M:autoextend
```

Zunächst muss 'skip-innodb' deaktiviert werden. Danach müssen wir den Speicherort für die Tabellen angeben. Falls wir 'innodb_data_home_dir' leer belassen, verwendet MySQL den normalen Datenpfad der Datenbanken. Dort werden einige Temporär-Dateien erstellt. Den Speicherort für die InnoDB-Tabellen legen wir mit dem Wert 'innodb_data_file_path' fest. Wir verwenden wiederum den Speicherort für die MySQL-Datenbanken sowie das Unterverzeichnis 'ibdata'. Dieses Verzeichnis muss von Hand angelegt werden:

```
mkdir /var/lib/mysql/ibdata
chown -R mysql /var/lib/mysql/ibdata
chgrp -R mysql /var/lib/mysql/ibdata
```

Um die InnoDB-Tabellen endgültig nutzen zu können, müssen wir den MySQL-Server neu initialisieren:

```
/etc/init.d/mysql restart
```

Der Neustart dauert einige Sekunden, weil die InnoDB-Tabellen zunächst initialisiert werden müssen. Um zu überprüfen, ob die InnoDB-Tabellen nun aktiviert sind, können wir in '/var/lib/mysql/ibdata' nachsehen. Falls Sie dort eine Datei vorfinden, ist alles ok.

5.2.2 Arbeiten mit den InnoDB-Tabellen

Um eine InnoDB-Tabelle anzulegen, müssen wir beim Erstellen der Tabelle die Option 'type=InnoDB' verwenden:

```
mysql> create table statistik (Feld1 int) type=InnoDB;
Query OK, 0 rows affected (0.41 sec)
```

Mit 'show table status from archiv' können wir überprüfen, ob MySQL den Tabellentyp 'InnoDB' auch akzeptiert hat. Bei der Tabelle 'statistik' bzw. dem Feld 'Type' sollten wir nun 'InnoDB' vorfinden. Falls dies nicht der Fall ist, d.h. wenn dort z.B. 'MYISAM' steht, stehen uns die InnoDB-Tabellen noch nicht zur Verfügung; in diesem Fall muss nochmals die Konfiguration überprüft werden.

Nun, nachdem wir die InnoDB-Tabelle erstellt haben, folgt an dieser Stelle ein einfaches Beispiel:

```
mysql> start transaction;
Query OK, 0 rows affected (0.01 sec)

mysql> insert into statistik () values(1);
Query OK, 1 row affected (0.00 sec)
mysql> insert into statistik () values(2);
Query OK, 1 row affected (0.00 sec)
mysql> insert into statistik () values(1);
Query OK, 1 row affected (0.00 sec)

mysql> select * from statistik;
+------+
| Feld |
+------+
|    1 |
|    2 |
|    1 |
+------+
3 rows in set (0.01 sec)

mysql> rollback;
Query OK, 0 rows affected (0.00 sec)
mysql> select * from statistik;
Empty set (0.00 sec)

mysql> insert into statistik () values(1);
Query OK, 1 row affected (0.01 sec)
mysql> insert into statistik () values(4);
Query OK, 1 row affected (0.01 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from statistik;
+------+
| Feld |
+------+
|    1 |
|    4 |
+------+
2 rows in set (0.00 sec)
```

Ich gebe zu, das Beispiel hat eine gewisse Länge erhalten, dürfte dafür aber beinahe selbsterklärend sein. Zunächst schalten wir die Transaktionen mit 'start transaction' ein. Nun fügen wir einige Datensätze der Tabelle 'statistik' hinzu. Mit einer 'select'-Abfrage sehen wir, dass die Datensätze vorhanden sind. Anschliessend zerstören wir mit 'rollback' unser 'Werk' und können feststellen, die Datensätze wurden doch nicht definitiv gespeichert.

Zum Abschluss haben wir erneut zwei Datensätze erfasst, wobei wir diese nun mit 'commit' definitiv ablegen. Selbst ein darauffolgendes Kommando 'rollback' ändert daran nichts mehr.

➲ Merken müssen wir uns im Prinzip nur, dass auf 'start transaction' immer entweder ein 'commit' (Annehmen) oder 'rollback' (Verwerfen) folgen muss.

5.3 HEAP: Immer auf Abruf

Die Heap-Tabellen werden direkt im Speicher verwaltet und sind daher äussert schnell im Zugriff, weil keine Zeit benötigt wird, um die Festplatte anzuwerfen. Wir müssen einzig sicherstellen, dass Heap-Tabellen nicht derart gross werden, dass unser Speicher überquillt. Bei den derzeitigen RAM-Preisen dürften HEAP-Tabellen rein finanziell kaum Grenzen gesetzt sein.

➲ Ebenfalls sollten wir uns im Klaren sein, dass z.B. bei einem Stromunterbruch die Daten weg sind. Ok, wir wissen, worauf wir uns einlassen und legen eine Heap-Tabelle an:

```
mysql> create table speicher type=heap select * from akten;
Query OK, 2 rows affected (0.42 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Auch wenn das nicht direkt mit den Heap-Tabellen zusammenhängt, so sei an dieser Stelle darauf hingewiesen, dass wir mit dem obenstehenden Kommando auf äusserst einfache Weise eine Kopie der Tabelle 'akten' erstellt haben.

Zusammen mit 'drop table if exists speicher' können wir jederzeit eine Kopie der Tabelle 'akten' ziehen. Wozu das Ganze? Unter bestimmten Umständen können wir damit eine select-Abfrage beschleunigen, weil eine 1:1-Kopie schneller sein kann, als das Zusammentragen der Resultate quer über alle Index- und Tabellendateien. Die Heap-Tabellen haben derzeit aber einen praktischen Nachteil:

```
mysql> create table problem type=heap select * from seiten;
ERROR 1163: The used table type doesn't support BLOB/TEXT columns
```

Leider ist es noch nicht möglich, Text- und Blob-Felder in einer Heap-Tabelle zu speichern. Zugegeben, solche Felder können den Speicher relativ schnell zum Bersten bringen, aber angenommen in einer Adresskartei gibt es ein Notizfeld, und dieses wird nur hin und wieder benutzt, dann wäre eine Heap-Kopie doch eine angenehme Sache! Geht derzeit leider nicht; wir müssten jedes Feld im 'select'-Teil einzeln anführen, um wenigstens eine 'Rumpf'-Kopie (ohne Blob- und Text-Felder) fahren zu können. Immerhin, das Feature ist angekündigt.

Und damit wir uns richtig verstehen, eine Heap-Tabelle kann auch komplett neu aufgebaut werden:

```
mysql> create table speicher (Feld1 int) type=heap;
Query OK, 0 rows affected (0.01 sec)

mysql> insert into speicher () values (1);
Query OK, 1 row affected (0.00 sec)

mysql> select * from speicher;
+-------+
| Feld1 |
+-------+
|     1 |
+-------+
1 row in set (0.00 sec)
```

5.4 Tabellen sperren (Backup)

Solange der MySQL-Server läuft, sind die Datentabellen für Linux gesperrt. Das kann (insbesondere beim Backup) ärgerlich sein. Wir können aber MySQL-Tabellen temporär auf 'lesend' schalten. In diesem Zustand können wir keine Änderungen an den Tabellen mehr vornehmen und dabei gleichzeitig ein Backup durchführen. Auch dazu ein Beispiel:

```
mysql> lock table akten read;
Query OK, 0 rows affected (0.00 sec)

mysql> lock table seiten read;
Query OK, 0 rows affected (0.00 sec)

mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)
```

Beachten müssen wir insbesondere, dass jede einzelne Tabelle zu sperren ist, während 'unlock tables' sämtliche Tabellen auf einmal wieder freigibt.

6 Volltext-Datenbanken

Die Volltextindexierung ist bereits seit längerer Zeit erhältlich, aber m.E. kann die Volltextindexierung erst ab Version 4.0.x wirklich sinnvoll eingesetzt werden. Das gilt in erster Linie für das Erstellen von Volltext-Datenbanken; die Indexierung war bei der Version 3.23.x bereits bei einigen zehntausend Seiten zum Verzweifeln langsam. Die Version 4.0.x ist hier ein Vielfaches schneller und bietet auch sonst recht viele nützliche Funktionen.

6.1 Volltext hinzufügen

Zunächst fügen wir einen Volltext der Tabelle 'seiten' hinzu:

```
mysql> select * from seiten;
+--------+-------+---------------------------------+------+
| AkteID | Seite | Text                            | Bild |
+--------+-------+---------------------------------+------+
|      1 |     1 | Ferien, Felsberg, Schweiz       | NULL |
|      1 |     2 | Wir erfassen eine zweite Seite  | NULL |
|      1 |     3 | Jetzt haben wir die dritte Seite | NULL |
+--------+-------+---------------------------------+------+
3 rows in set (0.00 sec)

mysql> alter table seiten add fulltext (Text);
Query OK, 3 rows affected (0.09 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

Derzeit können wir einen Volltext nur auf eine MyISAM-Tabelle anwenden, wobei der Index ein oder mehrere Felder des Typs 'Char', 'Varchar' sowie 'Text' umfassen kann, nicht aber 'Blob'-Felder.

Im übrigen kann es sich lohnen, den Volltext-Index erst nachträglich zu erstellen bzw. vor grossen Updates gänzlich zu entfernen und nach den Updates neu zu erstellen. Ein komplett neu aufgebauter Volltext-Index wird (nach meinen Beobachtungen) ca. 2 bis 3 mal schneller aufgebaut, als dies der Fall ist, wenn x-tausende von Datensätzen hintereinander erfasst werden.

6.2 Arbeiten mit dem Volltext

Nachdem wir den Volltext aufgebaut haben, schreiten wir zur Tat und stellen vermutlich gleich fest, dass MySQL gar keine Treffer zurückgibt:

```
mysql> select * from seiten where match Text against('Seite');
Empty set (0.44 sec)
```

Noch seltsamer wird die Sache, wenn wir die folgende Abfrage absetzen:

```
mysql> select * from seiten where match Text
    -> against('Schweiz');
+--------+-------+---------------------------+------+
| AkteID | Seite | Text                      | Bild |
+--------+-------+---------------------------+------+
|      1 |     1 | Ferien, Felsberg, Schweiz | NULL |
+--------+-------+---------------------------+------+
1 row in set (0.01 sec)
```

Wir haben es hier nicht mit einer 'patriotisch' gesinnten MySQL-Version zu tun, sondern vielmehr mit einer Eigenheit der MySQL-Abfragetechnik, die am Anfang verwirren kann. MySQL zeigt bei der normalen Volltext-Abfrage 'match Feld against('Text') nur Treffer an, sofern diese nicht in mehr als der Hälfte der Datensätze vorkommen. Mit anderen Worten gesagt, was allzu häufig auftritt, wird einfach unterschlagen. Ob das Sinn macht, bleibe dahingestellt, viel lieber möchte ich die Variante zeigen, bei der wir in jedem Falle sämtliche Treffer erhalten:

```
mysql> select * from seiten where match Text
    -> against('Seite' in boolean mode);
+--------+-------+---------------------------------+------+
| AkteID | Seite | Text                            | Bild |
+--------+-------+---------------------------------+------+
|      1 |     2 | Wir erfassen eine zweite Seite  | NULL |
|      1 |     3 | Jetzt haben wir die dritte Seite | NULL |
+--------+-------+---------------------------------+------+
2 rows in set (0.01 sec)
```

Sobald wir den Zusatz 'in boolean mode' verwenden, erhalten wir auch beim Suchen nach 'Seite' in unserem Beispiel korrekt zwei Treffer angezeigt. Persönlich ziehe ich 'in boolean mode' der Standard-Variante meist vor.

6.3 Einige Beispiele

Nachfolgend wollen wir einige (d.h. eine kleine Auswahl) der Möglichkeiten und Grenzen des Volltextes aufzeigen. Zunächst gibt es eine Art 'Ranking', d.h jene Treffer die besser passen, erhalten einen höheren Rückgabewert:

```
mysql> select AkteID, Text, match(Text)
    -> against('Seite zweite' in boolean mode)
    -> as Treffer from seiten where match(Text)
    -> against ('Seite' in boolean mode);
+--------+---------------------------------+---------+
| AkteID | Text                            | Treffer |
+--------+---------------------------------+---------+
|      1 | Wir erfassen eine zweite Seite  |       2 |
|      1 | Jetzt haben wir die dritte Seite |       1 |
+--------+---------------------------------+---------+
2 rows in set (0.01 sec)
```

Leider muss der Suchtext zweimal aufgeführt werden. Immerhin entsteht keine Performance-Einbusse (es erfolgt intern nur eine Abfrage). Das nächste Beispiel zeigt, wie wir zwei Wörter miteinander verknüpfen können (AND-Bedingung):

```
mysql> select * from seiten where match Text
    -> against('+Seite +zweite' in boolean mode);
+--------+-------+--------------------------------+------+
| AkteID | Seite | Text                           | Bild |
+--------+-------+--------------------------------+------+
|      1 |     2 | Wir erfassen eine zweite Seite | NULL |
+--------+-------+--------------------------------+------+
1 row in set (0.00 sec)
```

Hätten wir die beiden Suchbegriffe ohne das '+'-Zeichen eingegeben, so wäre aus der AND-Verknüpfung eine ODER-Verknüpfung entstanden. In der Praxis wird das problematisch sein, da die User nur schwer davon zu überzeugen sein dürften, jedesmal ein '+'-Zeichen einzugeben. Zwar könnte MySQL so kompiliert werden, dass MySQL 'normal' tickt, es dürfte aber (beim Programmieren von Applikationen) einfacher sein, die Eingabe der User zu parsen und mit einem '+'-Zeichen zu versehen.

Einen Vorteil hat das Pluszeichen aber dennoch. Wir können damit sehr einfach Wörter ausschliessen:

```
mysql> select * from seiten where match Text
    -> against('+Seite -zweite' in boolean mode);
+--------+-------+---------------------------------+------+
| AkteID | Seite | Text                            | Bild |
+--------+-------+---------------------------------+------+
|      1 |     3 | Jetzt haben wir die dritte Seite | NULL |
+--------+-------+---------------------------------+------+
1 row in set (0.01 sec)
```

➲ An dieser Stelle sei noch darauf hingewiesen, dass wir auch nach Wortteilen suchen können, wenn auch nur am Anfang, wie das zweite Beispiel unschwer zeigt:

```
mysql> select * from seiten where match Text
    -> against('Fels*' in boolean mode);
+--------+-------+--------------------------+------+
| AkteID | Seite | Text                     | Bild |
+--------+-------+--------------------------+------+
|      1 |     1 | Ferien, Felsberg, Schweiz | NULL |
+--------+-------+--------------------------+------+
1 row in set (0.00 sec)

mysql> select * from seiten where match Text
    -> against('*berg' in boolean mode);
Empty set (0.00 sec)
```

Zum Schluss möchte ich noch anfügen, dass uns die zahlreichen 'Aber' nicht davon abhalten sollten, die MySQL-Volltextindexierung positiv zu werten. Im Mengentest hat MySQL respektable Ergebnisse erreicht. Auf einem P4-Computer (512 MByte Ram) dauerte das Indexieren von ca. 2,2 GByte Textmasse ca. 80 Minuten. Und sofern ich 'in boolean mode' verwendet habe, erzielte ich Trefferzeiten im tiefen Sekundenbereich.

7 Export und Import

7.1 File_priv-Rechte

Wenn es darum geht, Daten aus MySQL zu exportieren oder fremde Daten direkt in eine MySQL-Tabelle zu importieren, dann müssen wir zunächst sicherstellen, dass wir die entsprechenden Rechte haben.

```
mysql> select * from akten order by Titel into outfile '/home/up/export.tb';
ERROR 1045: Access denied for user: 'up@localhost' (Using password: YES)
```

Erinnern wir uns, wir haben dem User 'up' nur Rechte für die Datenbank 'archiv' gegeben. Beim Exportieren (und auch Importieren) benötigen wir zusätzliche Rechte, weil wir die abgeschlossene MySQL-Welt verlassen und direkt mit Linux sprechen. Wir haben gelernt, dass sämtliche Rechte in der Tabelle 'mysql' verwaltet werden. Da unser User 'up' mehr Rechte erhalten soll, sollten wir das in der Tabelle 'user' finden. Wir melden uns mit 'root@localhost' an, geben 'use mysql' ein und lassen uns die Struktur anzeigen, um herauszufinden, welche Einstellung tangiert werden könnte:

```
mysql> describe user;
+-----------------+-----------------+------+-----+---------+-------+
| Field           | Type            | Null | Key | Default | Extra |
+-----------------+-----------------+------+-----+---------+-------+
| Host            | char(60) binary |      | PRI |         |       |
| User            | char(16) binary |      | PRI |         |       |
| Password        | char(16) binary |      |     |         |       |
| Select_priv     | enum('N','Y')   |      |     | N       |       |
| Insert_priv     | enum('N','Y')   |      |     | N       |       |
| Update_priv     | enum('N','Y')   |      |     | N       |       |
| Delete_priv     | enum('N','Y')   |      |     | N       |       |
| Create_priv     | enum('N','Y')   |      |     | N       |       |
| Drop_priv       | enum('N','Y')   |      |     | N       |       |
| Reload_priv     | enum('N','Y')   |      |     | N       |       |
| Shutdown_priv   | enum('N','Y')   |      |     | N       |       |
| Process_priv    | enum('N','Y')   |      |     | N       |       |
| File_priv       | enum('N','Y')   |      |     | N       |       |
| Grant_priv      | enum('N','Y')   |      |     | N       |       |
| References_priv | enum('N','Y')   |      |     | N       |       |
| Index_priv      | enum('N','Y')   |      |     | N       |       |
| Alter_priv      | enum('N','Y')   |      |     | N       |       |
+-----------------+-----------------+------+-----+---------+-------+
17 rows in set (0.00 sec)
```

Es sollte nicht schwer sein, herauszufinden, dass wir dem User 'up' wohl das Recht 'File_priv' geben müssen. Um sicher zu gehen, schauen wir kurz nach, was für Rechte er hat:

```
mysql> select Host,User,File_priv from user where User='up';
+-----------+------+-----------+
| Host      | User | File_priv |
+-----------+------+-----------+
| localhost | up   | N         |
+-----------+------+-----------+
1 row in set (0.00 sec)
```

Ok, wir wissen nun, was zu tun ist und geben dem User 'up' das entsprechende Recht:

```
mysql> update user set File_priv='Y' where User='up';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select Host,User,File_priv from user where User='up';
+-----------+------+-----------+
| Host      | User | File_priv |
+-----------+------+-----------+
| localhost | up   | Y         |
+-----------+------+-----------+
1 row in set (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.01 sec)
```

So, nun kann sich der User 'up' anmelden und den Export durchführen. Für den Exportvorgang verwenden wir 'select ... into outfile ...':

```
mysql> select * from akten into outfile '/home/up/export.tb';
ERROR 1: Can't create/write to file '/home/up/export.tb' (Errcode: 13)
```

Die Fehlermeldung mag auf den ersten Blick irritieren. Wir haben dem User 'up' die Rechte doch korrekt erteilt? Schon, aber der Export wird durch MySQL durchgeführt und deshalb benötigen wir im Export-Verzeichnis zumindest Schreib-Rechte für 'Alle'. Das erledigen wir direkt in der Linux-Konsole mit:

```
shell> mkdir /home/up/mysql
shell> chmod -R a+rw /home/up/mysql
```

➲ Nach dem Export sollten wir die Rechte wieder mit 'chmod -R go-rw /home/up/mysql' zurücksetzen, damit die Datei nicht von Dritten eingesehen werden kann. Alternativ könnten wir auch mit dem User 'mysql' und der Gruppe 'mysql' arbeiten, was mir aber aus Sicherheitsüberlegungen weniger sinnvoll erscheint.

7.2 Export von Tabellen

Jetzt endlich können wir den Export durchführen. Sinnvollerweise sortieren wir die Daten auch gleich richtig:

```
mysql> select * from akten order by Titel
    -> into outfile '/home/up/mysql/export.tb';
Query OK, 2 rows affected (0.01 sec)
```

In der Exportdatei finden wir nun auf jeder Zeile einen Datensatz. Die einzelnen Felder sind durch Tabulatoren voneinander getrennt.

7.3 Import von Tabellen

Zunächst möchte ich darauf hinweisen, dass wir die MySQL-Konsole jederzeit im Batch-Modus starten können. Damit können SQL-Befehle als Batch verarbeitet werden. Auch dazu ein Beispiel:

```
mysql -u up -p <do.sql
```

Das Passwort wird ganz normal abgefragt, doch danach gelangen wir nicht auf die Konsole, sondern der Inhalt der Datei 'do.sql' wird an die MySQL-Konsole weitergereicht. Nach dem Durcharbeiten dieser gelangen wir zurück zum Betriebssystem.

Falls wir eine ANSI-Datei importieren möchten, so können wir das Kommando 'load infile ... into table ...' verwenden. Da wir etwas faul sind, importieren wir ganz einfach die zuvor exportierte Datei wieder:

```
mysql> load data infile '/home/up/mysql/export.tb'
    ->  into table akten;
ERROR 1062: Duplicate entry '3' for key 1
```

Leider gibt es auch hier ein Problem. Die Tabelle 'akten' enthält ein Schlüsselfeld, das automatisch ausgefüllt wird (primary key sowie auto_increment). Logischerweise hat die zuvor exportiere Datei natürlich jene Schlüssel, die wir auch in der Tabelle haben. So gesehen darf die Fehlermeldung nicht mehr erstaunen, MySQL sagt uns ganz einfach, die Nummer drei gibt es schon. In diesem Beispiel sind wir sicher dankbar für den Hinweis, denn duplizierte Datenbestände sind definitiv keine gute Sache. Es ist aber durchaus denkbar, dass es ungewollte Konflikte gibt, wenn wir mehrere Importvorgänge aus verschiedenen Quellen durchführen. Vielleicht hilft in einem solchen Fall die folgende Variante:

```
mysql> load data infile '/home/up/mysql/export.tb'
    -> into table akten (Titel,Datum,Seiten);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 2
```

Nun wird die Datei eingelesen, allerdings funktioniert auch das nur, weil wir auf die Felder 'ID' sowie 'DatumErfasst' verzichten. Alles in allem ist der Export und Import mit Bordmitteln nicht ganz trivial. Wer es einfacher haben möchte, der möge ein Dritt-Tool bzw. ein eigenes Skript herbeiziehen.

7.4 Arbeiten mit Bildern

Einfacher als der Export/Import von und zu Tabellen ist das Arbeiten mit Blob-Feldern, d.h. Feldern, welche Binär-Informationen aufnehmen.

In der Tabelle 'seiten' haben wir bereits ein Blob-Feld mit dem Namen 'Bild' eröffnet. Sollten uns auf der Konsole keine Bilder zur Verfügung stehen, so importieren wir an dieser Stelle einfach eine beliebige Text-Datei:

```
mysql> update seiten set Bild=load_file('/home/up/mysql/export.tb')
    -> where AkteID=1 and Seite=2;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Und weil das so einfach ging (vorausgesetzt wir haben die 'File_priv'-Rechte), möchten wir das Blob-Feld auch gleich wieder in eine Datei exportieren:

```
mysql> select Bild into dumpfile '/home/up/mysql/blobcopy'
    -> from seiten where AkteID=1 and Seite=2;
Query OK, 1 row affected (0.01 sec)
```

Ich gebe zu, es gibt bequemere (und wohl auch effizientere) Arten, Dateien zu kopieren! Auch sollten wir uns bewusst sein, dass es verschiedene Arten von Blob-Feldern gibt. Der Unterschied besteht aber nur darin, wie viele Daten ein Blob-Feld aufnehmen kann. In einem Standard-Blob-Feld (BLOB) können wir 'nur' 64 KByte speichern, bei MEDIUMBLOB sind es bereits 16 MByte und bei LONGBLOB stehen immerhin 4 GByte zur Verfügung.

8 Perl-Beispiel (DBI)

Eine der grossen Stärken von MySQL ist, dass es gut dokumentierte Schnittstellen zu fast allen Programmiersprachen gibt. Als Beispiel wollen wir hier eine Lösung mit Perl besprechen.

8.1 Demo 'halloperl.pl'

Da wir an dieser Stelle keinen Perl-Kurs durchführen können/wollen, möchte ich vorschlagen, dass wir einfach mal den Code kurz anschauen:

```
#!/usr/bin/perl
# Programm halloperl.pl -- (c) Demo MySQL-Vortrag lug-camp.ch

use strict;
use DBI;
my ($dbh,$sth,$sql,$err,$c,@res);

print "Anmeldung bei MySQL erfolgt...\n";
$dbh=DBI->connect("DBI:mysql:host=localhost;database=archiv",
  "up","secret",RaiseError=>0,PrintError=>0);

print "Prüfvorgang, ob Tabelle 'cachetest' ok ist!\n";
$sql = "create table if not exists cachetest (id int not null ";
$sql .= "primary key auto_increment, text varchar(30))";

$dbh->do($sql);
if ($dbh->err)
    $err=$dbh->errstr;
    print "Fehlermeldung: $err\n";


print "Hinzufügen von Datensätzen\n";
for($c=0;$c<100000;$c++)
    $dbh->do("insert into cachetest () values ()");
    if (($c % 2000) == 0)  print ".";

print "\n";

$sql="select count(*) from cachetest";
@res=$dbh->selectrow_array($sql);
print "Die Tabelle enthält nun: $res[0] Datensätze!\n";

$dbh->disconnect();
print "Verbindung zu MySQL beenndet...Programmende\n";
```

Was macht nun 'halloperl.pl'? Zunächst stellt das Programm eine Verbindung zu MySQL her. Diese erfolgt bei Perl i.d.R. mit DBI, sagen wir mal höchst ungenau, dass DBI eine Perl-Konsole für MySQL realisiert. Da es sich dabei um ein normales Perl-Paket handelt, kann davon ausgegangen werden, dass überall wo MySQL und Perl installiert ist, wir auch das DBI-Paket finden sollten. Wir können üblicherweise also die Datenbank mit Perl ansprechen, ohne zusätzliche Software installieren zu müssen.

Bei 'DBI->connect(...)' melden wir uns am MySQL-Server an. Zugegeben, dass Passwort im Klartext ist unschön, ich wollte aber den gesamten Code in einem Beispiel haben. Danach

setzen wir einen SQL-Befehl ab, d.h. wir eröffnen eine neue Tabelle; allerdings nur, wenn diese nicht bereits existiert.

Sofern wir bis dahin keinen Fehler haben, fügen wir der Tabelle 100'000 Datensätze hinzu. Ich gebe gerne zu, dass dies auf den ersten Blick nicht besonders sinnvoll erscheinen mag, aber keine Bange, wir werden doch noch etwas Gescheites mit der Tabelle anstellen.

Nachdem wir die Datensätze haben, prüfen wir die Anzahl der Datensätze. Beim ersten Start von 'halloperl.pl' sollten wir 100'000 erhalten, bei jedem weiteren Aufruf dürften jeweils weitere 100'000 Datensätze hinzukommen.

Wir können das Beispiel ausführen, indem wir den Inhalt in der Datei '/home/up/halloperl.pl' speichern und anschliessend auf der Linux-Konsole (nicht unter MySQL!) die folgende Zeile eingeben:

```
perl /home/up/halloperl.pl
```

Das Programm gibt uns die folgende Ausgabe:

```
Anmeldung bei MySQL erfolgt...
Prüfvorgang, ob Tabelle 'cachetest' ok ist!
Hinzufügen von Datensätzen
..................................................
Die Tabelle enthält nun: 100000 Datensätze!
Verbindung zu MySQL beenndet...Programmende
```

Wir können uns nun erneut auf der MySQL-Konsole anmelden, die Datenbank 'archiv' wählen (use archiv) und mit 'select count(*) from cachetest' die Anzahl der Datensätze abfragen.

8.2 Perl Database Admin (pDBA)

An dieser Stelle möchte ich auf ein OpenSource-Projekt verweisen, das die Fähigkeiten von MySQL im Zusammenspiel mit Perl eindrücklich aufzeigt. 'Perl Database Admin (pDBA)' ist komplett in Perl programmiert und bietet ein Front-End für MySQL-Datenbanken an.

Unter pdba.wilabs.ch gibt es einige Demo-Datenbanken (Anmeldung mit User 'demo' und Passwort 'demo'), in denen wir bequem über ein Web-Interface mit einer MySQL-Datenbank kommunizieren können. Den Source-Code zum Download finden wir unter www.wilabs.ch unter der gleichen Adresse gibt es auch ein Forum zu pDBA sowie ebenfalls den Zugriff auf die Demo-Datenbanken.

9 Performence und Optimierung

Es gibt für den MySQL-Server eine grosse Fülle von Parametern, um die Ausführungsgeschwindigkeit zu optimieren. Im Rahmen einer Einführung können wir das Thema aber nur anschneiden. Ich muss auch zugeben, dass ich bis jetzt nicht allzu viele Optionen verwendet habe.

Als erste Anlaufstelle empfiehlt sich das Studium der Dateien, die im Unterverzeichnis 'support-files' liegen. Dort finden wir einige Start-Konfigurationsdateien für verschiedene Anwendungszwecke.

Nachfolgend möchte ich konkret auf zwei spezifische Problemfälle eingehen, welche mir bisher in der Praxis begegnet sind.

9.1 Cache-Parameter

Um mit den Cache-Parametern herumspielen zu können, müssen wir uns mit 'root'-Rechten einloggen. Zunächst schauen wir nach, welche Optionen MySQL überhaupt kennt. Wir können dazu einfach 'show variables' eingeben; allerdings erhalten wir dabei eine Riesenliste, und müssten die interessanten Werte mühsam herausklauben. Einfacher geht es mit dem folgenden Befehl:

```
mysql> show variables like '%cache%';
+-----------------------+------------+
| Variable_name         | Value      |
+-----------------------+------------+
| binlog_cache_size     | 32768      |
| have_query_cache      | YES        |
| max_binlog_cache_size | 4294967295 |
| query_cache_limit     | 1048576    |
| query_cache_size      | 0          |
| query_cache_type      | ON         |
| table_cache           | 64         |
| thread_cache_size     | 0          |
+-----------------------+------------+
8 rows in set (0.00 sec)
```

Konkret möchte ich den Wert 'query_cache_size' vorstellen. Damit können wir erreichen, dass MySQL einen bestimmten Speicherbereich reserviert, um einmal beantwortete 'select'-Anfragen temporär zwischenzuspeichern. Das bringt ab der zweiten Abfrage ganz massiv schnellere Resultate, weil die Informationen direkt aus dem Speicher kommen. Damit wir uns davon überzeugen können, schalten wir zunächst den Cache mal ein:

```
mysql> set global query_cache_size = 10000000;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'query_cache_size';
+------------------+---------+
| Variable_name    | Value   |
```

```
+------------------+---------+
| query_cache_size | 9999360 |
+------------------+---------+
1 row in set (0.01 sec)
```

Die Differenz von einigen Bytes (im Beispiel 640 Bytes) sollte uns nicht weiter beunruhigen, letztlich braucht die Verwaltung auch noch einige Bytes. Damit wir den Effekt veranschaulichen können, sollten wir Zugang zu einer möglichst grossen Datensammlung haben.

Ich habe versprochen, dass wir die Tabelle 'cachetest' nicht umsonst erstellt haben; nun ist die Zeit gekommen und wir öffnen die Tabelle 'cachetest' nochmals wie untenstehend:

```
mysql> use archiv;

Database changed
mysql> select count(*) from cachetest order by ID desc;
+----------+
| count(*) |
+----------+
|   100000 |
+----------+
1 row in set (0.18 sec)

mysql> select count(*) from cachetest order by ID desc;
+----------+
| count(*) |
+----------+
|   100000 |
+----------+
1 row in set (0.00 sec)
```

Was erkennen wir daraus? Bei der ersten Abfrage haben wir eine Reaktionszeit von 0.18 Sekunden erreicht, bei der zweiten Abfrage haben wir 0.00 Sekunden erhalten. Ok, irgendwelche Millisekunden wird auch diese Abfrage gebraucht haben, aber letztlich ist die zweite Abfrage schneller, weil das Resultat direkt aus dem Cache kommt.

Der Effekt kann verstärkt gezeigt werden, indem das Demoprogramm 'halloperl.pl' einige Male hintereinander gestartet wird (noch einfacher setzen wir im Programm selber den Zählobergrenze für $c entsprechend hoch), sodass wir weit mehr Datensätze für die Abfrage haben.

9.2 Blob-Objekte in eigenen Tabellen

Zum Abschluss möchte ich noch eine Problematik ansprechen, welche das Geschwindigkeitsverhalten von MySQL massgeblich beeinflussen kann.

Dazu sei folgender Sachverhalt gegeben: Wir haben eine Tabelle, die Tausende von Blob-Feldern (Bild- oder Sound-Daten) enthält. Zu jedem Blob-Feld wird eine Textzusammenfassung gespeichert. Die Blob-Felder benötigen etwa 95% des Datenvolumens (sagen wir mal 4 GByte). Der Text und die übrigen Statusinformationen belegen die restlichen 5%.

Nehmen wir weiter an, dass wir mit einen Volltext (Fulltext) arbeiten. Nun geben unsere User vorzugsweise 'gemeingefährliche' Wörter wie z.B. Linux ein. Dieses Wort kommt in unserer Tabelle relativ oft und quer über die gesamte Tabelle verteilt vor. Wir möchten nun immer einige Treffer gleichzeitig anzeigen. Soweit so gut, aber warum erhalten wir plötzlich saulahme Ergebnisse, wenn die User 'Linux' eingeben?

Keine Bange, das ist kein Microsoft-Werbespot, falls wir Microsoft ebenso oft in unserer Tabelle halten, würde auch 'Microsoft' erst mit einer Riesenverzögerung erscheinen; und auch das wäre kein Linux-Werbespot. Warum ist MySQL hier also langsam?

Das Problem liegt darin begraben, dass wir die Texte und Blob-Informationen in der gleichen Tabelle ablegen. Dadurch müssen wir beim Zusammenstellen der Treffer quer über die ganze Festplatte hüpfen (ok, mit 4 GByte sollte die Platte noch nicht gefüllt sein), um die Texte zusammenzutragen. Leisten wir uns den Luxus von zwei Tabellen, die wir über ein gemeinsames Feld verwalten, so liegen sämtliche Textinformationen praktisch nebeneinander (einige MBytes), sodass unser Lama umgehend wieder zum Jaguar mutiert.

➲ Wir können uns daher als Regel merken, dass Blob-Felder aus Performance-Gründen sinnvollerweise in eine einzige Tabelle gelegt werden sollten.

10 Zukunft und Würdigung

10.1 Was bringt die Gegenwart?

In den letzten beiden Jahren haben sich ganz wesentliche Dinge bei MySQL geändert. Wir haben nun eine Datenbank ohne 2 GByte-Limiten, mit Query-Cache, mit Volltext, mit Transaktionen und einigem mehr.

➲ Für mich persönlich entscheidend(er) ist aber, dass MySQL stabil und mühelos unter Linux läuft.

10.2 Was erwartet uns in Zukunft?

In der nahen Zukunft wird die Version 4.1 erwartet und angekündigt ist ja auch bereits die Version 5. So sehr ich mich auf all die Features freue, so bleibt zumindest eine gewisse Unsicherheit, ob das alles mit der gleichen Stabilität, Kompatibilität und Einfachheit machbar ist, wie dies bisher der Fall war bzw. noch immer ist. Es wäre schade, wenn wir in einigen Jahren einen Fork ziehen müssten, um aus MySQL wieder mSQL zu machen. In der Zwischenzeit wünsche ich beim Ausreizen der vielen Features von MySQL viel Spass.