

Sichere und effiziente Backups mit MySQL im laufenden Betrieb

Kommando-Sache: Backup

Wer ein Backup einer MySQL-Datenbank durchführen will, hat die Qual der Wahl zwischen zahlreichen Kommandos und Werkzeugen. Für große Datenbanken erweist sich *mylvmbackup* als praktikable Lösung. **Michael Kofler**

Auf einen Blick

Inhalt

MySQL-Administratoren haben die Wahl zwischen einer Vielzahl von Backup-Kommandos und -Verfahren.

- *mysqldump* ist das klassische MySQL-Backup-Kommando. Es liefert portablen SQL-Code, ist aber relativ langsam.

- Das kostenlose Skript *mylvmbackup* kommt dem Ideal eines Hot Backups sehr nahe. Es stört den Datenbankbetrieb kaum, setzt allerdings Linux und den Einsatz von LVM voraus.

- Für MySQL 6 verspricht Sun ein neues, betriebssystem- und Engine-unabhängiges Hot-Backup-Verfahren.

Plattform

MySQL

Technik/Anwendung

Datenbank-Design/
Administration

Voraussetzungen

MySQL 5.1

Autor

Dr. Michael Kofler lebt in Graz und ist MySQL-Anwender der ersten Stunde sowie Autor zahlreicher Bücher zu Visual Basic, MySQL, PHP und Linux.

Grundsätzlich gibt es zwei Methoden, um ein Backup einer Datenbank zu erstellen: Sie können eine Sicherungskopie der Datenbankdateien vornehmen, oder Sie können über den Server die eigentlichen Daten auslesen und so speichern, dass später ein einfaches Wiedereinspielen möglich ist (zumeist in Form von *INSERT*-Kommandos).

Bei beiden Varianten gibt es ein Problem: Die Dateien respektive Daten dürfen sich während des Backups nicht ändern! Andernfalls besteht die Gefahr, dass das Backup nicht konsistent ist (also beispielsweise eine Tabelle auf nicht mehr existente Datensätze einer anderen Tabelle verweist) oder die gesicherten Dateien gar korrupt sind. Je nach Verfahren gibt es eine einfache Lösung: Sie fahren die Datenbank herunter, bevor Sie die Dateien kopieren, oder Sie stellen durch *LOCK TABLE* (MyISAM) beziehungsweise durch eine Transaktion (InnoDB) sicher, dass sich die Daten während des Auslesens nicht ändern. Der Nachteil: Während des Backups ist der Datenbankbetrieb mehr oder weniger stark beeinträchtigt. Da größere Backups unter Umständen stundenlang dauern, ist das oft nicht akzeptabel.

Nachdem nun klar ist, warum ein Datenbank-Backup nicht so einfach ist, wie man zunächst denken könnte, lohnt ein Blick auf **Tabelle 1**: Sie sehen, dass es zwar viele Backup-Kommandos und -Verfahren gibt, aber keine optimale Lösung. *mysqldump* ist langsam und blockiert den Server, *mylvmbackup* funktioniert nur unter Linux und erfordert eine LVM-Partition, InnoDB Hot Backup ist kostenpflichtig und zudem für MyISAM-Tabellen ungeeignet, und MySQL Online Backup ist Zukunftsmusik. Dieser Artikel konzentriert sich im Folgenden auf zwei Kommandos: *mysqldump*, weil es der De-facto-Standard ist und jeder MySQL-Administrator damit umgehen können muss, und *mylvmbackup*, weil es die zurzeit ideale Backup-Lösung für Linux-Anwender ist.

mysqldump

Ein Backup ist ein statisches Abbild der Datenbank. Außer durch einfache Backups können die Daten auch durch Logging und Replikation gesichert werden. Die Unterschiede zwischen diesen Verfahren sind im Kasten „Backup, Logging und Replikation“ zusammengefasst.

mysqldump liest die zu sichernden Daten vom MySQL Server und liefert das Backup in Form einer SQL-Datei. Das Kommando wird durch unzählige Optionen gesteuert. **Listing 1** fasst die Syntax zusammen, **Tabelle 2** die wichtigsten Optionen. Details zu noch mehr Optionen geben *mysqldump --help* oder *man mysqldump*.

Der Einsatz von *mysqldump* variiert etwas, je nachdem, ob eine Datenbank MyISAM- oder InnoDB-Tabellen enthält. Standardmäßig gilt ein ganzes Set von Optionen, das speziell für MyISAM-Tabellen optimiert ist. Das vereinfacht den Aufruf des Kommandos ein wenig (siehe **Listing 2**).

Wesentlich mehr Optionen brauchen Sie für ein Backup einer Datenbank mit InnoDB-Tabellen.

Listing 6: mylvmbackup-Einstellungen

```
# /etc/mylvmbackup.conf
[mysql]
user      = root
password  = *****
host      = localhost
port      = 3306
socket    =
mycnf     = /etc/mysql/my.cnf

[lvm]
vgname    = vg1
lvname    = mysql
backuplv  =
lvsize    = 5G

[fs]
xfs=0
mountdir  = /var/cache/mylvmbackup/mnt/
backupdir = /var/cache/mylvmbackup/backup/
relpath   =

[tools]
... (normalerweise keine Änderungen)

[misc]
backuptype = tar
prefix     = backup
tararg     = cvzf
tarsuffixarg =
rsyncarg   = -avWP
datefmt    = %Y%m%d_%H%M%S
innodb_recover = 1
pidfile    = /var/tmp/mylvmbackup_
recoverserver.pid
```

len: Mit `--skip-opt` deaktivieren Sie die speziell für MyISAM gedachten Standardoptionen. Ein Teil dieser Optionen ist aber auch für InnoDB-Tabellen zweckmäßig und muss deswegen explizit wieder aktiviert werden. Hinzu kommen einige InnoDB-spezifische Optionen. Das Ergebnis ist ein dreizeiliges Monsterkommando

Listing 1: mysqldump-Syntax

```
mysqldump -u root -p [optionen] daten-
bankname > backup.sql
```

(siehe Listing 3). Beachten Sie, dass `mysqldump` standardmäßig weder Stored Procedures noch Trigger sichert. Wenn Sie das wünschen, müssen Sie zusätzlich die Optionen `--routines` und `--triggers` angeben. Um ein Backup später wieder einzuspielen, erzeugen Sie zuerst die betreffende Datenbank (falls es sie noch nicht gibt). Anschließend übergeben Sie die Backup-Datei an das Kommando `mysql` (siehe Listing 4). Dabei stellt die Option `--default-character-set` sicher, dass die im UTF8-Zeichensatz gespeicherten Zeichenketten wieder richtig eingelesen werden. ▶

Listing 2: MyISAM-Tabellen mit mysqldump sichern

```
mysqldump -u root -p --lock-all-tables dbname > backup.sql
```

Listing 3: InnoDB-Tabellen mit mysqldump sichern

```
mysqldump -u root -p --skip-opt --single-transaction \
--disable-keys --create-options --quick \
--extended-insert --add-drop-table dbname > backup.sql
```

Listing 4: Mit mysqldump gesicherte Datenbank wieder einlesen

```
mysqladmin -u root -p create dbname
mysql -u root -p --default-character-set=utf8 dbname < backup.sql
```

Listing 5: Komprimierte mysqldump-Backups

```
mysqldump [optionen] dbname | gzip -c > backup.sql.gz
gunzip -c backup.sql.gz | mysql [optionen] dbname
```

Listing 7: mylvmbackup-Backup durchführen

```
root# mylvmbackup
20090306 14:50:23 Info: Connecting to database...
20090306 14:50:23 Info: Flushing tables with read lock...
20090306 14:50:23 Info: Taking position record...
20090306 14:50:23 Info: Taking snapshot...
File descriptor 3 left open
Logical volume "mysql_snapshot" created
20090306 14:50:24 Info: Unlocking tables...
20090306 14:50:24 Info: Disconnecting from database...
20090306 14:50:24 Info: Mounting snapshot...
20090306 14:50:24 Info: Recovering innodb...
```

Listing 8: mylvmbackup-Backup wieder einspielen

```
root# /etc/init.d/mysql stop
root# rm -rf /var/lib/mysql/* oder mv /var/lib/mysql/* /bak/
root# mv /etc/mysql/my.cnf /etc/mysql/my.cnf.bak
root# tar -x -f backup.tar.gz -C /var/lib/mysql --strip 1 backup
root# tar -x -f backup.tar.gz -C /etc/mysql --strip 1 backup-pos
root# mv /etc/mysql/backup-*_my.cnf /etc/mysql/my.cnf
root# /etc/init.d/mysql start
```

Mit *mysqldump* erzeugte Backup-Dateien sind aufgrund des Textformats sehr groß. Dieses Problem umgehen Sie unter Unix/Linux am einfachsten dadurch, dass Sie die Backups sofort komprimieren (*gzip*) oder beim Wiedereinspielen direkt dekomprimieren (*gunzip*). Die entsprechenden Kommandos sehen Sie in **Listing 5**.

mylvmbackup

mylvmbackup ist ein kostenloses Backup-Skript, das bei einigen Linux-Distributionen gleich als Paket mitgeliefert wird. Ist das nicht der Fall, laden Sie das Skript von der in **Tabelle 1** genannten Website herunter. Das Skript kann nur eingesetzt werden, wenn die folgenden Voraussetzungen erfüllt sind:

- Der MySQL Server läuft unter Linux.
 - Das Verzeichnis */var/lib/mysql* mit den Datenbankdateien befindet sich in einer LVM-Partition (siehe Kasten „Logical Volume Manager“).
 - Die InnoDB-Logging-Dateien befinden sich in derselben LVM-Partition (standardmäßig werden diese Logging-Dateien ebenfalls im Verzeichnis */var/lib/mysql* erstellt).
 - Im LVM-Speicherpool ist noch ausreichend Speicher, um einen LVM-Snapshot zu erstellen.
- Kurz einige Worte zur Funktionsweise von *mylvmbackup*: Das Skript erstellt ein Backup aller

Datenbankdateien. Nun habe ich eingangs bereits darauf hingewiesen, dass sich die Datenbankdateien im laufenden Betrieb ständig ändern und daher nicht einfach kopiert werden können. *mylvmbackup* umgeht dieses Problem in zwei Schritten.

Zuerst führt es *FLUSH TABLES WITH WRITE LOCK* aus, um sicherzustellen, dass alle MyISAM-Tabellen in einem konsistenten Zustand sind. Anschließend erstellt *mylvmbackup* einen Snapshot der LVM-Partition, die */var/lib/mysql* enthält. Dieser Vorgang dauert nur Sekundenbruchteile. Danach gibt *mylvmbackup* den Lock wieder frei (*UNLOCK TABLES*), und MySQL kann wieder ungehindert weiterarbeiten. Die InnoDB-Tabellen waren überhaupt nie blockiert, weil der *WRITE LOCK* lediglich für MyISAM-Tabellen galt.

mylvmbackup kopiert nun die Datenbankdateien aus dem Snapshot in ein komprimiertes *tar*-Archiv und löscht schließlich den Snapshot wieder.

mylvmbackup wird durch die Konfigurationsdatei */etc/mylvmbackup.conf* gesteuert (siehe **Listing 6**): Die meisten Parameter können Sie in der Voreinstellung belassen. Eigene Werte müssen Sie für *user* und *password* angeben, damit *mylvmbackup* eine Verbindung zum MySQL Server herstellen und die *LOCK*-Kommandos aus-

Tabelle 1: MySQL-Backup-Werkzeuge und -Verfahren

Werkzeug	Beschreibung
mysqldump	Dieses mit MySQL mitgelieferte Kommando ist der Klassiker unter den Backup-Werkzeugen. Sein größter Vorteil: Die resultierenden SQL-Dateien sind zwischen unterschiedlichen MySQL-Versionen weitgehend portabel. Leider ist das Kommando für große Datenbanken ungeeignet: Die Backup-Dateien sind riesig, sowohl das Backup als auch das Wiedereinspielen der Daten dauert schier endlos.
mylvmbackup	Das Open-Source-Skript <i>mylvmbackup</i> kann nur unter Linux eingesetzt werden, und auch da nur, wenn sich das Verzeichnis <i>/var/lib/mysql</i> in einer LVM-Partition befindet. Sind diese Voraussetzungen erfüllt, ist <i>mylvmbackup</i> aber eine extrem effiziente Backup-Lösung, die gleichermaßen für InnoDB- und MyISAM-Tabellen geeignet ist. (http://lenzg.net/mylvmbackup/)
mysqlhotcopy	Dieses früher sehr populäre Perl-Skript wird mit MySQL mitgeliefert. Es führt <i>LOCK TABLE</i> aus und kopiert dann die Datenbankdateien. Es ist aber nur für MyISAM- und Archive-Tabellen geeignet, läuft nur unter Unix/Linux und gilt als veraltet. Details zur Verwendung des Skripts gibt das MySQL-Handbuch: (http://dev.mysql.com/doc/refman/5.1/en/mysqlhotcopy.html)
InnoDB Hot Backup	Dieses kostenpflichtige Backup-Werkzeug von Oracle erlaubt es, ein Backup von InnoDB-Tabellen im laufenden Betrieb durchzuführen. Innobackup bietet damit eine ähnliche Funktionalität wie <i>mylvmbackup</i> , ist aber nicht auf LVM oder ein anderes, Snapshot-fähiges Dateisystem angewiesen. Sein größter Nachteil ist der Preis (zurzeit ca. 400 Euro pro Server pro Jahr). (www.innodb.com/)
MySQL Online Backup	MySQL 6.0 soll ein neues Backup-Verfahren erhalten, das unabhängig von der Tabellen-Engine und dem Betriebssystem ein Backup im laufenden Betrieb und ohne längere Blockierung der Tabellen ermöglichen soll. Das einzige Problem: Noch steht die Fertigstellung von MySQL 6.0 in den Sternen. Optimisten hoffen auf Ende 2010, wahrscheinlicher ist aber ein späterer Termin. (http://forge.mysql.com/wiki/OnlineBackup)
GUIs	Nahezu jedes MySQL-Administrations-Werkzeug mit grafischer Benutzeroberfläche bietet die Möglichkeit, Backups durchzuführen (MySQL Administrator, phpMyAdmin etc.). Für MySQL-Einsteiger ist das zumeist die einfachste Form, ein Backup zu erstellen. Die Backup-Methode ist aber selten im Hinblick auf maximale Effizienz optimiert und lässt sich zumeist nicht automatisieren (z.B. ein Backup jede Sonntagnacht). Außerdem können die Backups vielfach nur mit dem jeweiligen Administrations-Werkzeug wieder eingelesen werden, was ein gravierender Nachteil ist. Achten Sie darauf, dass Sie Ihre Backups jederzeit ohne (womöglich kostenpflichtige) Zusatzwerkzeuge nutzen können!
Replikation	Replikation bezeichnet die Synchronisierung von Datenbanken auf unterschiedlichen Servern. Wie im Kasten „Backup, Logging, Replikation“ erläutert, ist Replikation kein Ersatz für Backups, manchmal aber eine zweckmäßige Ergänzung.

Backup, Logging und Replikation

Die drei genannten Begriffe stiften bisweilen Verwirrung, weil sie alle in der einen oder anderen Form mit der Rekonstruktion von Daten zu tun haben.

Am einfachsten ist sicherlich der Begriff Backup zu verstehen: Er bezeichnet eine zu einem bestimmten Zeitpunkt durchgeführte Sicherung aller Daten. Ein Backup ist bei großen Datenbanken ein aufwendiger Prozess, der zumeist nur einmal täglich oder gar nur einmal wöchentlich ausgeführt wird.

Was aber ist mit allen Änderungen, die zwischen dem letzten Backup und dem Ausfall eines Datenbankservers durchgeführt wurden? Bei MySQL können Sie alle Datenänderungen durch das sogenannte Update Logging protokollieren. Die resultierenden Logging-Dateien in einem kompakten, binären Format ermöglichen es, alle Änderungen seit dem letzten Backup Schritt für Schritt nachzuvollziehen. Es ist zweckmäßig, die Logging-Dateien regelmäßig auf einen anderen Rechner zu kopieren, damit bei einem Totalausfall des Servers nicht auch

die Logging-Dateien zerstört werden. (MySQL kennt auch andere Formen des Loggings, die hier aber nicht von Interesse sind.)

Von Replikation spricht man, wenn zwei Datenbanken auf unterschiedlichen Rechnern synchron gehalten werden. Jede Änderung am Master wird nahezu sofort auch am Replikations-Slave ausgeführt. Entgegen landläufiger Meinung ersetzt Replikation kein Backup: Wenn Sie am Master irrtümlich `DELETE * FROM table` ausführen, wird die Tabelle Sekunden später auch am Slave gelöscht! Replikation kann aber eine ideale Ergänzung zu regelmäßigen Backups sein, wenn jederzeit ein Ersatz-Server einsatzfähig sein soll: Fällt der Haupt-Server aus, kann der bisherige Replikations-Slave diese Rolle praktisch sofort übernehmen. Als Administrator müssen Sie nur ein paar Konfigurationszeilen verändern. Dagegen dauert das Einspielen eines großen Backups auf einem anderen Rechner unter Umständen Stunden oder Tage. Replikation wird das Thema eines eigenen Artikels in einer der nächsten database-pro-Ausgaben sein.

führen kann. *vgroup* und *lvname* bezeichnen die *Volume Group* und den *Logical Volume Name* der LVM-Partition, auf der sich die MySQL-Datenbankdateien befinden. Daraus setzt das Skript den LVM-Device-Name `/dev/vgroup/lvname` zusammen.

lvsize gibt die Größe des Snapshot-Puffers für geänderte Datenblöcke an. Der Puffer muss so groß sein, dass darin alle Datenblöcke der Partition Platz finden, die sich während des Backups ändern. 5 GByte sind schon recht großzügig bemessen. *mylvmbackup* verrät zum Schluss, wie viel Speicher während des Backups tatsächlich verwendet wurde. Mit dieser Information können Sie die Einstellung optimieren.

backupdir gibt an, wo *mylvmbackup* die Backup-Dateien speichern soll. Das Verzeichnis sollte sich möglichst in einem anderen *Logical Volume* (LV) befinden als `/var/lib/mysql`.

relpath gibt an, wo sich das Verzeichnis `/var/lib/mysql` relativ zum LV-Mount-Punkt befindet. Wenn es ein eigenes LV für `/var/lib/mysql` gibt, bleibt *relpath* leer. Wenn das LV hingegen das gesamte `/var`-Verzeichnis erfasst, so müssen Sie *relpath=lib/mysql/* angeben.

Vielleicht haben Sie sich schon darüber gewundert, dass zwar die MyISAM-Datenbankdateien durch `FLUSH TABLES` vor dem Backup synchronisiert wurden,

nicht aber die InnoDB-Dateien. Bei InnoDB ist eine derartige Synchronisation nicht erforderlich, weil das Dateiformat Crash-sicher ist. Alle abgeschlossenen Transaktionen, deren Änderungen im InnoDB-Tablespace noch nicht gespeichert wurden, sind zumindest in den InnoDB-Transaktions-Logging-Dateien verzeichnet. Daher kann der InnoDB-Tabellentreiber beim ersten Start des MySQL Server nach dem Wiederherstellen eines Backups diese Transaktionen nachvollziehen. Dieser Recovery-Vorgang kostet allerdings Zeit. Wenn Sie möchten, dass die Datenwiederherstellung so schnell wie möglich erfolgen kann, müssen Sie die Einstellung `innodb_recover=1` vornehmen: *mylvmbackup* überprüft nun bereits während des Backups, ob der InnoDB-Masterspace und die dazugehörigen Logging-Dateien synchron sind. Ist das nicht der Fall, erfolgt im Rahmen des Backups ein InnoDB-Recovery-Durchlauf. Das verlängert die Zeit für das Backup, verkürzt aber die Zeit, um später ein neues System auf der Basis des Backups einzurichten.

Sind die Einstellungen von *mylvmbackup.conf* einmal abgeschlossen, stößt ein simples *mylvmbackup*-Kommando das Backup an (Listing 7).

Etwas aufwendiger ist es, ein Backup später wieder einzuspielen: Dazu ►

1/3 hoch rechts
55 x 270

X: 131,824
Y: -5

Logical Volume Manager (LVM)

Der Logical Volume Manager ist ein Linux-Modul, das eine logische Schicht zwischen das Dateisystem und die physikalischen Partitionen der Festplatte setzt. Via LVM verwaltete Bereiche („LVM-Partitionen“) und die darauf enthaltenen Dateisysteme können im laufenden Betrieb vergrößert oder verkleinert werden. Zudem kann der LVM-Pool jederzeit durch eine zusätzliche Festplatte vergrößert werden. Aus diesem Grund kommt LVM auf vielen Linux-Servern zum Einsatz, die große Datenmengen verwalten.

Die aus der Sicht dieses Artikels interessanteste Funktion von LVM sind aber Snapshots: Damit wird eine LVM-Partition scheinbar dupliziert. Das Original bleibt weiter verwendbar, während das Duplikat „eingefroren“ wird. Aus Effizienzgründen dupliziert LVM tatsächlich nur jene Blöcke der LVM-

Partition, die während der Lebensdauer des Snapshots verändert werden. (Für die duplizierten Blöcke muss ausreichend Speicherplatz im LVM-Pool reserviert werden. Ist dieser Speicherplatz erschöpft, endet die Lebensdauer des Snapshots vorzeitig!)

Für den Datenbankbetrieb bedeutet das: Während die Datenbank weiterläuft und im Dateisystem der ursprünglichen LVM-Partition Änderungen durchführt, kann das Backup-Programm in aller Ruhe die Dateien aus dem Duplikat lesen.

LVM wird idealerweise bereits während der Installation des Servers eingerichtet. Eine nachträgliche Aktivierung für bereits genutzte Partitionen ist nicht möglich. LVM kann aber jederzeit für noch freie Partitionen oder für zusätzliche Festplatten eingerichtet werden.

Noch eine Erklärung zu den beiden *tar*-Kommandos: Das erste Kommando extrahiert nur diejenigen Dateien, die sich innerhalb des Archivs im Verzeichnis *backup* befinden, und schreibt sie – ohne vorangestelltes *backup*-Verzeichnis – in das Verzeichnis */var/lib/mysql*. Das zweite *tar*-Kommando extrahiert analog die Archivdateien *backup-pos/** direkt in das Verzeichnis */etc/mysql*.

Fazit: Die Größe bestimmt das ideale Werkzeug

Wer überwiegend mit kleinen Datenbanken zu tun hat und keine Probleme damit hat, wenn während des Backups einige Tabellen für ein paar Minuten blockiert sind, kann bei *mysqldump* bleiben. Das Kommando liefert portablen SQL-Code, der vermutlich noch in vielen Jahren lesbar ist. *mysqldump* ist zudem das ideale Werkzeug, wenn nur bestimmte Daten (zum Beispiel alle Trigger einer Datenbank) gesichert werden sollen.

Um große, stark beanspruchte Datenbanken zu sichern, ist *mylombackup* das ideale Werkzeug – sofern die Voraussetzungen (im Einsatz mit Linux, LVM) erfüllt sind. Ist das nicht der Fall, können InnoDB-Anwender auf das kostenpflichtige Werkzeug InnoDB Hot Backup ausweichen. [ef]

müssen Sie die im Backup-Archiv enthaltenen Dateien in die Verzeichnisse */etc/mysql* und */var/lib/mysql* extrahieren. Der MySQL Server muss während dieser Operationen heruntergefahren und anschließend neu gestartet werden. **Listing 8** fasst die erforderlichen Kommandos zusammen.

Tabelle 2: Wichtige *mysqldump*-Optionen

Option	Beschreibung
--add-drop-tables	Fügt der Backup-Datei SQL-Kommandos hinzu, um beim Wiedereinlesen der Daten bereits vorhandene, gleichnamige Tabellen zu löschen.
--all-databases	Führt ein Backup aller Datenbanken durch (nicht nur einer Datenbank).
--create-options	Ergänzt die SQL-Kommandos in der Backup-Datei um MySQL-spezifische Besonderheiten. Diese Option gilt standardmäßig, sofern nicht --skip-opt angegeben wird.
--databases	Sichert die am Ende des Kommandos angegebenen Datenbanken (nicht nur eine Datenbank).
--disable-keys	Fügt der Backup-Datei SQL-Kommandos hinzu, die den Index erst nach allen INSERTs aktualisiert. Das erhöht die Effizienz beim Wiedereinlesen der Daten.
--events	Speichert auch Events (ab MySQL 5.1).
--extended-insert	Fasst die Daten mehrerer Datensätze in einem INSERT-Kommando zusammen. Das führt zu etwas kompakteren Backup-Dateien. Diese Option gilt standardmäßig, sofern nicht --skip-opt angegeben wird.
--hex-blob	Speichert binäre Daten in einem hexadezimalen Format.
--lock-all-tables	Blockiert alle Tabellen, bis das gesamte Backup der Datenbank abgeschlossen wird. Standardmäßig wird immer nur eine Tabelle blockiert, was zwar den Datenbankbetrieb weniger beeinträchtigt, aber zu inkonsistenten Backups führen kann. Die Option ist nur für MyISAM-Tabellen geeignet.
--no-data	Erzeugt nur CREATE-TABLE-Kommandos. Damit wird nur das Schema der Datenbank, nicht aber ihr Inhalt gesichert.
--quick	Vermeidet ein Server-internes Zwischenspeichern der zu sichernden Tabelle. Das ist bei großen Tabellen unbedingt erforderlich. Die Option gilt standardmäßig, sofern nicht --skip-opt angegeben wird.
--routines	Speichert auch Stored Procedures.
--single-transaction	Führt das Backup innerhalb einer Transaktion aus. Das verhindert, dass sich während des Backups Daten ändern können. Die Option ist allerdings nur für Datenbanken mit InnoDB-Tabellen geeignet.
--skip-opt	Deaktiviert die für MyISAM-Tabellen konzipierten Standardoptionen. Diese Option muss verwendet werden, wenn InnoDB-Tabellen gesichert werden sollen.
--triggers	Speichert auch Trigger.
"--where=ID<1000"	Sichert nur die Datensätze, die die angegebene Bedingung erfüllen.