MySQL-Datenbanken sichern – Teil 1

Gesichert wie eine Bank

Die beliebte Open-Source-Datenbank MySQL kennt verschiedene Möglichkeiten der Datensicherung. Jede hat Vor- und Nachteile, die jedoch nur unter bestimmten Umständen gravierend sind. Thomas Wölfer

Die MySQL-Datenbanken nehmen alle möglichen Daten auf. Darin lassen sich Artikel genauso verwalten wie Daten für Werbeeinblendungen. Diese Daten müssen regelmäßig gesichert werden, um bei einem Defekt auf der sicheren Seite zu sein. MySQL [1] enthält für die Datensicherung eine ganze Reihe an Werkzeugen, die sich für verschiedene Anwendungsbereiche eignen. Der Anwender kann zwischen schnellen und sicheren Möglichkeiten wählen.

Option 1: Select into outfile

Wenn die Datenbank aus wenigen Tabellen besteht und zudem ein Shellzugang zum MySQL-Server vorhanden ist, sind die Daten mit der Option »select into outfile« einfach zu sichern. So lässt sich eine Tabelle respektive ein Teil von ihr auf die Schnelle in eine Datei schreiben. Das »select«-Statement ist dabei in vollem Umfang nutzbar, um die sicherungsbedürftigen Dateien in die Auswahl einzubeziehen. Eine ganze Tabelle speichert man einfach mit dem Kommando »select *«. Den kompletten Aufruf von »select into outfile« zeigt Listing 1.

»Datei« steht für den Namen der Datei, in die die Tabelle gesichert wird. Die Anweisung »fields terminated by« enthält das Zeichen, mit dem die einzelnen Spaltenwerte in der Datei getrennt sind. Gängig dafür ist ein Komma. Im Beispiel kommt eine mit Kommas separierte Liste heraus, die einzelne Werte in Hochkommas einschließt.

Zu beachten ist dabei, dass die Ausgabedatei nicht vorhanden sein darf. Dies dient MySQL als einfacher Sicherheitsmechanismus, der das irrtümliche Überschreiben einer Sicherungskopie verhindert. Außerdem muss der Account, der das »select into outfile« auf MySQL anwendet, File-Rechte auf dem MySQL-Server-Host besitzen. Der größte Vorteil des Verfahrens ist die enorme Geschwindigkeit, mit der eine einzelne Tabelle gesichert wird.

Die gleiche Geschwindigkeit legt auch das Gegenstück an den Tag: »load date infile«. Mit dieser Option lässt sich eine Datei laden, die mit »select into outfile« gesichert wurde. »load data infile« erhält unter anderem die einzulesende Datei als Parameter:

```
load data info 'Datei' ↵
 [replace | ignore] into table Tabelle
```

Im Wesentlichen lädt es den Inhalt der Datei in die Tabelle mit dem angegebenen Namen. Wenn die Kombination aus »select into outfile« und »load data info« für ein Tabellen-Backup zum Einsatz kommt, muss vor dem »load data« die Zieltabelle leer sein. Alternativ funktioniert auch der »ignore«-Parameter. Er verwirft Zeilen mit doppelten Primary-Keys, sodass selbst dann keine Fehler auftreten, wenn die Zieltabelle bereits dieselben Daten enthält.

Allerdings bringen »select into outfile« und »load data infile« auch einen Nachteil mit sich, der ihren Einsatz für die meisten Anwender zu sehr beschränkt. Denn es lassen sich immer nur einzelne Tabelle laden oder speichern. Daher muss der Anwender bei einer neu angelegten Tabelle jedes Mal auch die Sicherungsskipte ergänzen, damit die neue Tabelle auch ins Backup einfließt. Ideal ist »select into outfile« hingegen, um eine einzelne Tabelle zu sichern.

Option 2: Backup Table

Backup Table funktioniert ähnlich wie »select into outfile«, ist aber vom Befehl her klarer strukturiert: Der Anwender gibt direkt im SQL-Statement an, was er sichern möchte. Die Option »backup table« erhält als Parameter eine oder mehrere Tabelle sowie den Namen einer Datei, in die die Tabellen gesichert werden sollen: »backup table *Tabelle1* [,*Weitere*] to '*Datei*'«. Mit diesem Befehl lassen sich mehrere Tabellen gleichzeitig sichern,

die mit einem Komma voneinander getrennt anzugeben sind.
In aktuellen MySQL-Dokumentationen ist zwar zu lesen, dass Backup Table veraltet sei und die Entwickler gerade an einer neuen Variante arbeiten. Diesen Hinweis kann der Anwender jedoch getrost ignorieren, denn die Option – ob veraltet oder nicht – funktioniert ganz hervorragend. Der große Vorteil im Vergleich zu »select into outfile« ist, dass sie mehrere Tabellen gleichzeitig sichert. Sie verlangt jedoch wie die Select-Funktion einen Shellzugriff auf den MySQL-Server.

Option 3: Mysqldump

Mysqldump macht die Sache mit der Sicherungskopie gleich wesentlich einfacher: Das Programm kann eine komplette Datenbank oder auch mehrere Datenbanken gleichzeitig von einem MySQL-Server auf die Festplatte schreiben. Praktischerweise braucht man dazu keinen Shellaccount auf dem Rechner mit dem MySQL-Server. Der Port muss nur für MySQL zugänglich sein, sonst kann sich Mysqldump nicht mit dem Server verbinden. Der kurze Aufruf von Mysqldump lautet:

```
mysqldump [Optionen] Datenbank [Tabellen]
```

»*[Tabellen]*« bedeutet eine Auflistung aller Tabellen einer Datenbank, die in die Datensicherung gehören. Sind sämtliche Tabellen gemeint, lässt man diese Angabe weg. Der Befehl gibt alle Daten auf dem Bildschirm aus. Um die Datenbank in eine Datei zu speichern, reicht die Umleitung der Ausgabe in eine Datei, die natürlich nicht existieren darf:

```
mysqldump Datenbank > sicherung.sql
```

Die Sicherungskopie »sicherung.sql« enthält sowohl die Struktur der Tabellen als auch die zugehörigen Daten. Um beide Informationen aus der Sicherungskopie wieder zu laden, reicht:

```
mysql < sicherung.sql
```

Falls auf dem MySQL-Server kein lokaler Zugriff möglich ist, lässt der Verwalter Mysqldump die Sicherungsdaten von dem entfernten Server abholen. Zu diesem Zweck enthält das Programm die Option »-host = *RemoteHost*«. Allerdings erzeugt dieses Verfahren besonders bei großen Datenbanken jede Menge Traffic – genau dies ist überhaupt das größte Problem beim Sichern mit Mysqldump übers Netz.
Für eine lokale Sicherung ist das Programm hingegen das optimale Werkzeug. Wie nicht anders zu erwarten, ist Mysqldump mit jeder Menge Optionen ausgestattet (Abbildung 1), mit denen sich das Programm optimal auf die eigenen Bedürfnisse einstellen lässt. Eine komplette Auflistung aller Kommandozeilenschalter mit Beispielen enthält die MySQL-Dokumentation im Web [2].

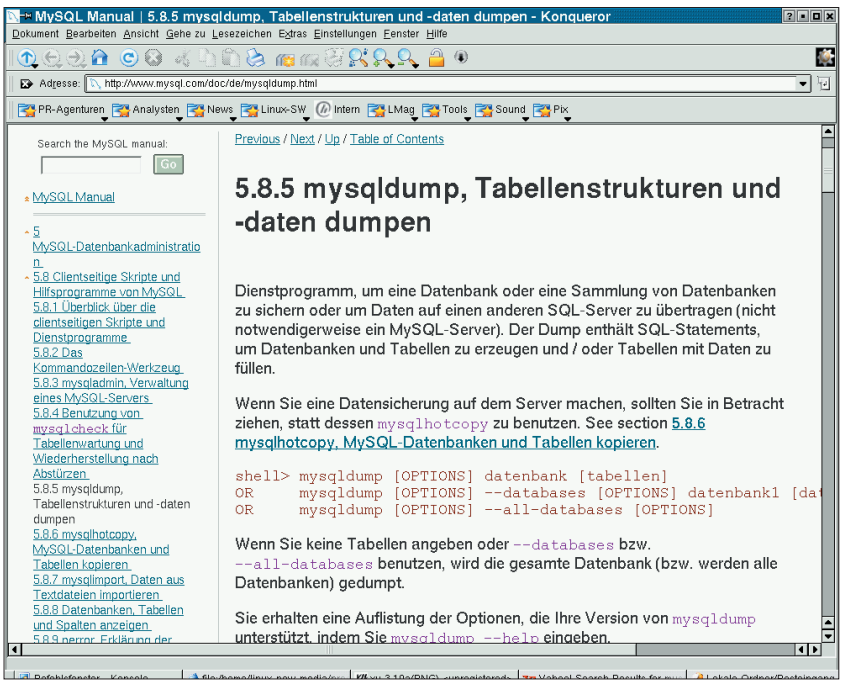

Abbildung 1: Mysqldump bietet viele Optionen an und lässt sich auf verschiedene Umgebungen abstimmen.

Option 4: Manuelles Backup

Wer den MySQL-Server für die Zeit, die der Sicherungsprozess in Anspruch nimmt, anhalten darf, kann eine extrem einfache Methode anwenden, um die Daten zu sichern. Dazu stoppt der Admin den MySQL-Daemon zunächst, kopiert alle Datenbank-Dateien und fährt den Dämon wieder hoch. Die kopierten Dateien lassen sich dann sicher per SSH auf einen anderen Rechner transferieren. Damit dies funktioniert, sind alle FRM-, MYD- und MYI-Dateien aus dem Datenbankverzeichnis zu kopieren. Um die Daten zu restaurieren, folgt die Prozedur dem diesem Weg: Daemon anhalten, Datenbankdateien zurückkopieren, Server wieder starten.
Das Verfahren ist jedem Einsteiger klar und zudem schön handlich. Natürlich bringt so eine einfache Sicherung auch Einschränkungen mit sich. Den MySQL-Daemon anhalten mag bei kleinen Datenbanken, Websites mit wenig Traffic oder Intranet-Sites mit klaren Wartungszeiten kein Problem sein. Große oder öffentlichen Sites hingegen müssen jederzeit verfügbar sein, sodass diese Methode nicht praktikabel ist.
Zudem benötigt der Administrator Shellzugriff auf den Datenbankserver. Dadurch fällt das manuelle Backup bei kleineren Webhosting-Angeboten ebenfalls aus. Wer hingegen einen Shellzugang besitzt, erhält mit dem manuellen Kopieren eine einfache und zuverlässige Si-

Listing 1: Select into outfile

```
01 select * into outfile 'Datei.txt'
02 fields terminated by ','
03 enclosed by '"'
04 lines terminated by 'n' from Tabelle;
```

Listing 2: Verbindung zum lokalen Rechner

```
01 $cL = mysql_connect( "Lokaler_Server", "Write_Username",
   "Passwort");
02 if( ! $cL)
03 {
04    print 'no local connection.';
05    return;
06 }
```

cherungsmethode. Das ist auch anderen aufgefallen – und die haben die ganze Prozedur in das komfortable Perl-Skript »mysqlhotcopy« verpackt [2].
Wer Perl-Skripte auf dem Server ausführen darf, für den ist Mysqlhotcopy ein angenehmer Weg, um das manuelle Backup zu automatisieren. Es hat aber seinerseits eine Einschränkung: Das Skript funktioniert nur für ISAM- und MyISAM-Datenbanken.

Option 5: PHP-Skripte

MySQL-Daten ganz nach persönlichen Vorlieben sichern funktioniert mit einer Skriptsprache wie PHP. Die folgende Lösung mit PHP ist relativ einfach gehalten und stellt lediglich ein Gerüst dar, um das sich ein eigenes Backup-Skript bauen lässt. Das vollständige Skript liegt zum Download unter [3]. Sicherlich ist auch eine allgemein gültige Lösung zur Datenübertragung zwischen zwei MySQL-Datenbanken denkbar. Das wäre dann bereits eine Erweiterung für PHP MyAdmin [4], was als Ad-hoc-Lösung jedoch zu umfangreich ist.
Das Skript erwartet, dass bereits zwei funktionstüchtige MySQL-Server bereitstehen und zudem einer der Rechner als Apache-Server mit PHP-Unterstützung fungiert. Für PHP soll auch der MySQL-Support installiert sein, was im Rahmen der PHP-Distribution standardmäßig der Fall ist. Weiterhin muss der Sicherungsrechner Zugriff auf die MySQL-Ports haben. Normalerweise verwendet MySQL die Ports 3306 und 6000 für UDP und TCP. Gelingt der Zugriff später nicht, blockt wahrscheinlich eine Firewall die Requests ab.
Wichtig für die korrekte Datensicherung mit dem Beispielskript ist vor allem, wie die Daten auf Server organisiert sind. Wenn vorhandene Datensätze immer wieder zu ändern sind, dann sind mehr Daten zu sichern, als wenn lediglich neue Daten hinzukommen. Im ersten Fall müssen sowohl die neue Daten, die seit der letzten Sicherung angefallen sind, als auch die während diese Zeitraums geänderten Daten in die Sicherung einfließen. Im zweiten Fall reicht es aus, einfach nur die neuen Datensätze zu sichern.
Wenn das Skript per Browser oder WGET starten soll, muss es sich um eine normale PHP-Seite mit allen für HTML benötigten Anweisungen handeln. Das ist relativ einfach zu erreichen:

```
<html><head></head><body>
<?php Update(); ?>
</body></html>
```

Eine HTML-Minimalseite, in der das PHP-Statement die Update-Funktion aufruft, reicht aus. Diese Funktion wiederum startet anschließend alle weiteren Update-Funktionen nacheinander. Die Update-Funktion für ein inkrementelles Update, bei dem nur neue Datensätze abgerufen werden, wird im Folgenden erläutert:

```
function UpdateNew()
{
   print "Begin Update New <br>";
```

Zunächst gibt die Funktion eine kurze Meldung aus. Das ist praktisch, wenn man das Update-Skript per Browser aufruft. Dadurch sieht der Anwender sofort, welche Schritte im Update bereits durchgelaufen sind:

```
$cR = mysql_connect("Remote_Server",
"Read_Username", "Passwort");
```

Im nächsten Schritt soll das Skript eine Verbindung zum entfernten Datenbankserver aufbauen. (Der entfernte Server ist der Hauptserver, auf dem die zu sichernden Daten bereits vorliegen.) Dazu ist als »*Remote_Server*« der Name oder die IP-Adresse des Servers anzugeben. Normalerweise sind solche Rechner respektive das MySQL auf solchen Rechnern so konfiguriert, dass es unterschiedliche Usernamen und Passwörter für Lese- und Schreibzugriffe gibt. Da das Skript im Folgenden lesend auf diesen Rechner zugreift, müssen beim Verbindungsaufbau der Account-Name des leseberechtigten Users sowie das zugehörige Passwort (»*Read_Username*«, »*Passwort*«) stehen:

```
if( ! $cR)
{
   print 'No remote connection';
   return;
}
```

Da zumindest zu Beginn der Implementierung der Verbindungsaufbau gern mal scheitert, testet das Skript vorsichtshalber den Rückgabewert der »connect«-Funktion. Sollte die Verbindung nicht zustande kommen, beendet sich das Skript an dieser Stelle.
Nun wird ebenfalls mit einer Fehlerbehandlung die Verbindung zum lokalen Rechner aufgebaut (Listing 2). Der lokale Rechner speichert die kopierten Daten. Auch dieser muss mit Rechnername oder IP-Adresse sowie Benutzernamen und Passwort im Skript stehen. Voraussetzung ist allerdings ein User mit Schreibrechten.

Neue Datensätze ermitteln

Die beiden Verbindungen zu den beteiligten Datenbankservern stehen, jetzt ist der trickreiche Teil an der Reihe. Da das erste Beispielskript nur neue Datensätze kopiert, muss es ermitteln, welche Datensätze denn überhaupt neu sind. Dazu nutzt das Skript die fortlaufende Nummer oder eine eindeutige ID der Tabellenzeilen innerhalb der Tabelle. Es geht davon aus, dass die höchste ID der höchsten Nummer der Datensätze entspricht (Listing 3).
Im Beispiel erfragt die SQL-Query zunächst die höchste ID aus der Tabelle. Der Anwender muss lediglich die Platzhalter »*Tabelle*« und »*Datenbank*« seiner Umgebung anpassen: »*Tabelle*« steht dabei für die Tabelle innerhalb der Datenbank und »*Datenbank*« ist der Name jener Datenbank, auf die innerhalb des

Listing 3: IDs aussuchen

```
01 $qs = "select id from Tabelle order by id desc limit 1";
02 $rLocal = mysql_db_query( "Datenbank", $qs, $cL);
03 if( ! $rLocal)
04 {
05    print "query error: $qs <br>";
06    return;
07 }
```

Listing 4: Entfernten Server abfragen

```
01 $qsr = "select * from Tabelle where id>$num order by id";
02 $rRemote = mysql_db_query( "Datenbank", $qsr, $cR);
03 if( ! $rRemote)
04 {
05    print "query error: $qsr <br>";
06    return;
07 }
```

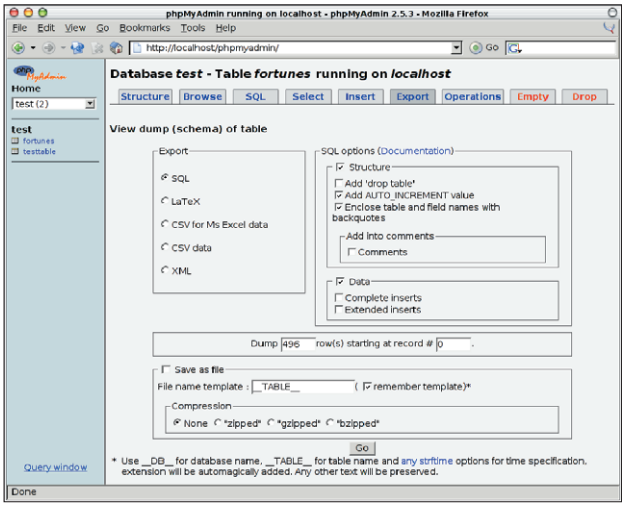

Abbildung 2: Auch PHP MyAdmin bietet Backupfunktionen für MySQL-Tabellen.

MySQL-Servers zugegriffen wird. Auch bei dieser Abfrage empfiehlt sich eine Fehlerbehandlung, die darauf achtet, ob die Query tatsächlich beantwortet wurde respektive überhaupt beantwortbar ist. Kommt keine Antwort, gibt das Skript den verwendeten Query-String aus. So lässt sich leicht im Browser prüfen, wo diese spezielle Abfrage das Problem verursacht hat:

```
$row = mysql_fetch_row( $rLocal);
$num = $row[0];
```

Nun wird die mit Hilfe der Query ermittelte Nummer ausgelesen. PHP bietet dafür mehrere Funktionen. Da hier nur ein einzelner Wert interessant ist, reicht »mysql_fetch_row()« völlig aus. Die Nummer steht an der ersten Zelle im gelieferten Array und wird mit »[0]« indiziert und ausgelesen. Die höchste lokale Datensatznummer ist somit bekannt und es folgt eine ähnliche Abfrage für den entfernten Server (Listing 4).

Bei dieser zweiten Abfrage interessieren nur noch jene Nummern, die größer als die zuvor ermittelte sind. Entsprechend ist die Abfrage auch leicht verändert. Die Bedingung ist dabei hinter dem »where«-Statement formuliert. Die gelieferte Nummer steht dann wieder im Query-Resultat, das sich mit PHP auslesen lässt:

```
$count =
mysql_num_rows(
$rRemote);
```

Damit steht fest, wie viele Datensätze kopiert werden müssen. Das Kopieren kommt jetzt als nächster Schritt (Listing 5). Es wird einfach so oft iteriert, wie es für die Komplettkopie der neuen Datensätze nötig ist. Für jeden Datensatz ruft das Skript einmal »mysql_fetch_row()«auf; die Daten liegen dann in einem Array vor.

Anschließend muss die Tabelle diese Daten aufnehmen. Dazu ist der Platzhalter »*Tabelle*« wieder durch den Namen der eigentlichen Tabelle zu ersetzen. An die Stelle von »*Feldern*« kommt eine durch Kommas getrennte Liste der Felder. Die eingefügten Werte sind also »$row[0]«, »$row[1]« und so weiter. Schließlich wird mit dem auf diese Weise zusammengesetzten Query-String noch die MySQL-Query abgesetzt. Sie kümmert sich dann darum, dass die Daten auch tatsächlich in die Tabelle eingetragen werden.

Vollständiges Update

Das Skript für ein vollständiges Update unterscheidet sich nicht wesentlich von dem bisher vorgestellten:

```
$strDel = "delete from Tabelle";
$r = mysql_db_query("Datenbank",
$strDel, $cL);
```

Allerdings sind noch ein paar Dinge zu beachten: Zunächst ist sicherzustellen, dass die Tabelle, die kopiert werden soll, auf dem Zielrechner zurückgesetzt wird. Daher sind in der Tabelle keine Daten mehr vorhanden und man kann beim späteren Einfügen ein »insert into«-Statement verwenden. Das Löschen der Tabelle erfolgt mit »delete from«. Es ist sehr wichtig, dass die richtige Tabelle aus der richtigen Datenbank gelöscht wird, bevor das Skript läuft. Danach selektiert das Skript alle Werte aus der Quelltabelle und fügt sie in die Zieltabelle ein (Listing 6). *(jre)*

Infos

[1] MySQL: [http://www.mysql.com]
[2] Mysqldump: [http://www.mysql.com/doc/de/mysqldump.html]
[3] HP-Skript: [ftp://ftp.linux-magazin.de/pub/listings/magazin/2004/05/MySQL]
[4] PHP MyAdmin: [http://www.phpmyadmin.net]

Der Autor

Thomas Wölfer ist seit 20 Jahren Software-Entwickler und administriert die Mail-, Web- und Datenbankserver des Portals Nickles.de.

Listing 5: Datensätze kopieren

```
01 for( $i=0; $i<$count; $i++)
02 {
03    print "Updating: $i ... <br> ";
04    $row = mysql_fetch_row( $rRemote);
05    $qsLocal = "insert into Tabelle(Felder) values(Werte)";
06    $rLocal = mysql_db_query("Datenbank",
   $qsLocal, $cL);
07    if( ! $rLocal)
08   {
09       print "query error: $qsLocal <br>";
10       return;
11    }
12 }
```

Listing 6: Daten in Zieltabelle einfügen

```
01 $qs = "select * from Tabelle";
02 $r = mysql_db_query("Datenbank", $qs, $cR);
03 $num = mysql_num_rows( $r);
04 for( $i=0; $i<$num; $i++)
```