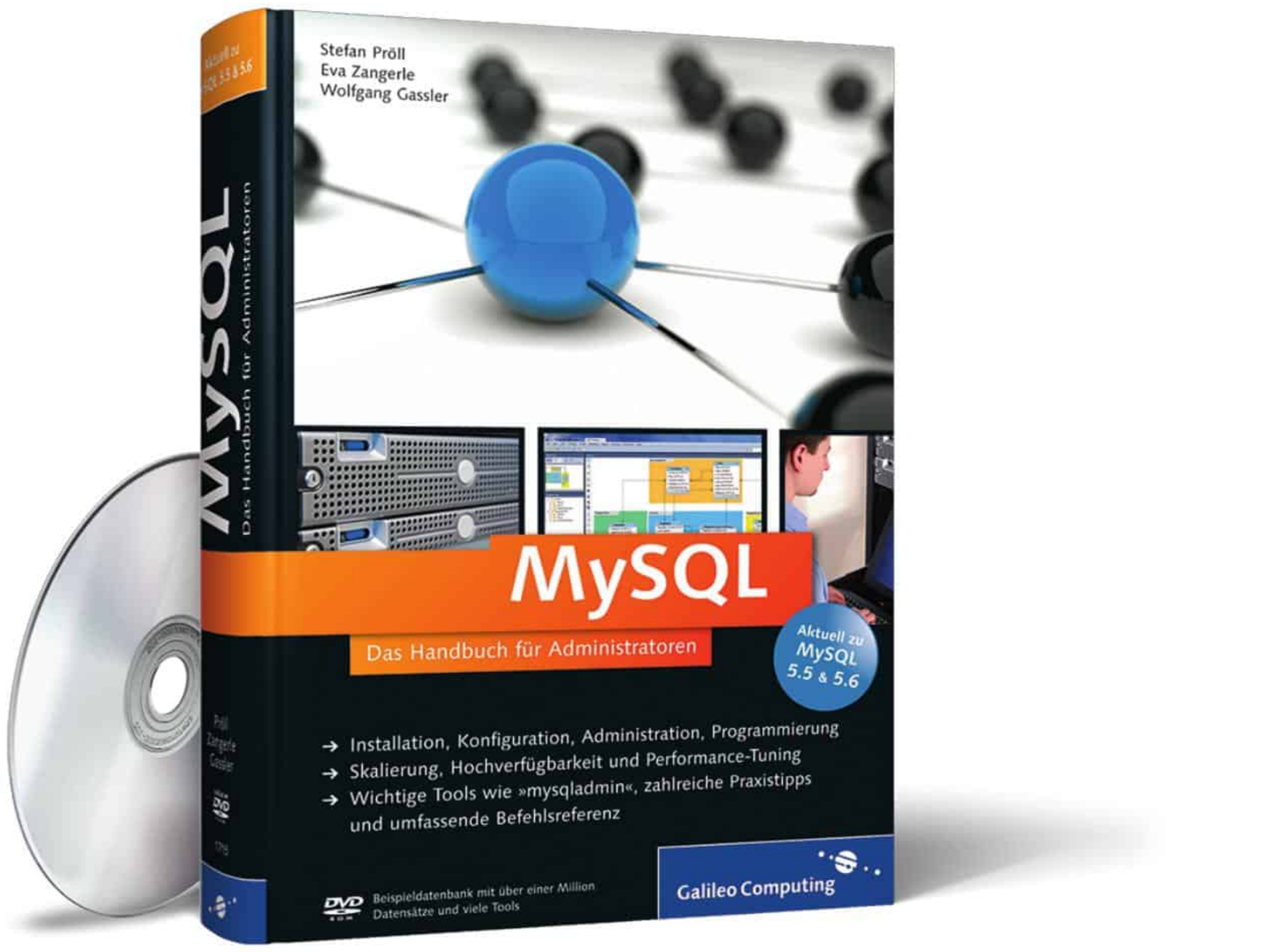
Stefan Pröll, Eva Zangerle, Wolfgang Gassler

MySQL

Das Handbuch für Administratoren



Galileo Press

Auf einen Blick

TEIL I Erste Schritte

1 Datenbanken – eine allgemeine Einleitung 23

2 Datenbankmodellierung 51

3 Schnellstart-Tutorial 93

4 Einführung in die Beispieldatenbank 129

TEIL II MySQL-Administration

5 Architektur 155

6 Manuelle Installation, Plugins und Upgrades 219

7 MySQL administrieren 263

8 Replikation und Hochverfügbarkeit 345

9 Anfrageoptimierung 359

10 Sicherheit 451

TEIL III Programmierung

11 Gespeicherte Programme und Events 513

12 Softwareentwicklung mit MySQL 625

TEIL IV Anhang

A Referenz 661

Inhalt

Vorwort 15
Für wen ist dieses Buch gedacht? 15
Konventionen 16
Der Aufbau dieses Buches 17
Die DVD zum Buch 18
Die Webseite zum Buch 19
Die Autoren 19

TEIL I Erste Schritte

1 Datenbanken – eine allgemeine Einleitung 23

1.1 Grundlagen von Datenbanken 23
1.1.1 Die Geschichte von Datenbanken 23
1.1.2 Moderne Datenbanken und Datenmodelle 25
1.1.3 Vorteile 28
1.1.4 Einsatzgebiete 31
1.1.5 Anforderungen an ein Datenbankmanagementsystem 34
1.2 MySQL 38
1.2.1 Einleitung und Vorstellung 39
1.2.2 Die Geschichte von MySQL 42
1.2.3 Open Source und Lizenzen 45
1.3 Zusammenfassung 48

2 Datenbankmodellierung 51

2.1 Einführung in den Datenbankentwurf 51
2.1.1 Anforderungsanalyse 52
2.1.2 Konzeptioneller Datenbankentwurf 53
2.1.3 Logischer Datenbankentwurf 54
2.1.4 Physischer Datenbankentwurf 54
2.1.5 Der Datenbankentwurf im Überblick 55
2.2 Ein konkretes Beispiel: Flughafenverwaltung 56
2.3 Das relationale Datenbankenmodell 57
2.3.1 Tabellen, Spalten und Zeilen 58
2.3.2 Schlüssel identifizieren Datensätze 59
2.3.3 Die relationale Algebra 60
2.3.4 Joins 66

2.4 Datenbankdesign 70
2.4.1 Das ER-Diagramm 70
2.4.2 Entitäten 71
2.4.3 Beziehungen 74
2.5 Normalformen 83
2.5.1 Erste Normalform 85
2.5.2 Die zweite Normalform 87
2.5.3 Die dritte Normalform 88
2.5.4 Weitere Normalformen 90
2.6 Zusammenfassung 91

3 Schnellstart-Tutorial 93

3.1 MySQL herunterladen 93
3.2 Minimale Systemanforderungen 94
3.3 MySQL installieren 95
3.3.1 Schnellinstallation unter Windows 96
3.3.2 Schnellinstallation unter Linux 103
3.3.3 Schnellinstallation unter Mac OS 110
3.3.4 Zusammenfassung der Installation 113
3.4 Endlich loslegen – MySQL ausprobieren 113
3.5 Übersicht der wichtigsten Befehle 116
3.5.1 Datenbanken anlegen und löschen 117
3.5.2 Tabellen anlegen, Daten einfügen und löschen 119
3.5.3 Datensätze anlegen, abfragen und löschen 124
3.5.4 Datensätze abfragen und anzeigen 126
3.5.5 Zusammenfassung 127

4 Einführung in die Beispieldatenbank 129

4.1 Die Datenbank für das Szenario anlegen 129
4.2 Die Beispieldatenbank auf der Buch-DVD 145
4.3 Hilfe bei der Arbeit mit MySQL 147
4.3.1 Fehlermeldungen und Warnungen 147
4.3.2 Online-Manual und weitere Anlaufstellen 149
4.4 Zusammenfassung 150

TEIL II MySQL-Administration

5 Architektur 155

5.1 Einführung 155

5.1.1 Teile einer Datenbank – das Fünf-Schichten-Modell 155
5.1.2 MySQL-Architektur 161
5.2 Abfrageverarbeitung 165
5.2.1 Der Query-Cache – das Kurzzeitgedächtnis von MySQL ... 165
5.2.2 Der Optimierer – die Intelligenz von MySQL 172
5.2.3 Execution Engine – der Arbeiter von MySQL 178
5.3 Transaktionen und Nebenläufigkeit 178
5.3.1 Nebenläufigkeit 179
5.3.2 Transaktionen 180
5.3.3 Lock-Management 192
5.4 Storage-Engines 195
5.4.1 Grundlagen des Storage-Engine-Plugin-Systems 196
5.4.2 InnoDB 199
5.4.3 MyISAM 205
5.4.4 MEMORY 212
5.4.5 Sonstige Engines 213
5.4.6 Fremdhersteller-Engines 216
5.4.7 Vergleichstabelle 217
5.5 Zusammenfassung 218

6 Manuelle Installation, Plugins und Upgrades 219

6.1 Von neuen Quellen – MySQL selbst kompilieren und installieren 220
6.1.1 Voraussetzungen 220
6.1.2 MySQL kompilieren und installieren 222
6.1.3 Die Installation grundlegend absichern 227
6.1.4 Anlegen der Konfigurationsdatei 232
6.1.5 Den MySQL-Dienst starten 233
6.1.6 Zusammenfassung der Installation 237
6.2 Usermanagement 237
6.3 Die zentrale Konfigurationsdatei »my.cnf« 243
6.3.1 Die Syntax der Konfigurationsdatei 245
6.3.2 Die grundlegenden Einstellungen 246
6.3.3 Kommandozeilenargumente 249
6.3.4 Server-Variablen setzen und ändern 251
6.4 MySQL aktualisieren und erweitern 254
6.4.1 Upgrades 254
6.4.2 Die Plugin-API 258
6.5 Zusammenfassung 260

7 MySQL administrieren 263

7.1 Der bordeigene Werkzeugkasten 263
7.2 Der Name ist Programm: mysqladmin 264
7.2.1 Datenbanken anlegen und löschen 264
7.2.2 Caches leeren 265
7.2.3 Das Root-Passwort ändern 266
7.2.4 Den Server überwachen 269
7.3 Daten importieren und exportieren 272
7.3.1 Daten aus Dateien importieren 272
7.3.2 Daten exportieren 278
7.4 Richtig sichern – Backups und Strategien 280
7.4.1 Die Basics 280
7.4.2 Physisches oder logisches Backup 282
7.4.3 Online- und Offline-Backups 284
7.4.4 Volle und inkrementelle Backups 285
7.4.5 Logische Backups und Wiederherstellung mit mysqldump 286
7.4.6 Physisches Offline-Backup – schnell, sicher und effizient 290
7.4.7 MySQL Hot Copy 293
7.4.8 InnoDB-Tabellen sichern mit Percona XtraBackup 294
7.4.9 Recovery von Bedienfehlern – zeitpunktbezogene Wiederherstellung 295
7.4.10 Backups mit Replikation 298
7.4.11 Backups automatisieren 299
7.5 Tabellenwartung mit mysqlcheck 299
7.5.1 MyISAM-Tabellen überprüfen und reparieren 300
7.5.2 Die Erholungsphase – Crash Recovery 302
7.6 Performance-Check mit mysqlslap 304
7.7 MySQL beobachten 305
7.7.1 Die INFORMATION_SCHEMA-Tabellen 305
7.7.2 Den Status abfragen 306
7.7.3 MySQL überwachen 307
7.8 Die grafischen Werkzeuge im Überblick 308
7.8.1 MySQL Workbench 308
7.8.2 MySQL Administrator 314
7.8.3 Weitere Tools 317
7.9 Optimierungstechniken 317
7.9.1 Leistungsschau 318
7.9.2 Profiling 320

8

7.9.3 Die wichtigsten Variablen der zentralen Konfigurationsdatei im Überblick 325
7.9.4 MySQL feineinstellen – Empfehlungen eines Skripts 330

8 Replikation und Hochverfügbarkeit 345

8.1 Grundprinzipien 346
8.1.1 Ablauf 347
8.1.2 Arten von Replikation 347
8.2 Topologien 348
8.2.1 Master – Slave 349
8.2.2 Master – Slave mit zwischengeschaltetem Slave 350
8.2.3 Master-Master 351
8.3 Vorarbeiten zur Replikation 352
8.4 Konfiguration einer Master-Slave-Replikation 354
8.5 Zusammenfassung 357

9 Anfrageoptimierung 359

9.1 Aufspüren von langsamen Anfragen 359
9.1.1 Langsame Anfrage protokollieren 359
9.1.2 Log-Dateien regelmäßig kontrollieren 362
9.2 Indexstrukturen 362
9.2.1 Die Grundidee von Indexstrukturen 363
9.2.2 Primärindex vs. Sekundärindex 364
9.2.3 Der limitierende Faktor – die Festplatte 365
9.2.4 Technische Umsetzung von Indexstrukturen – der B-Baum und MyISAM 366
9.2.5 Clustered Index 372
9.2.6 Non-Clustered Index 372
9.2.7 Mehrdimensionale Indexstrukturen – der R-Baum 373
9.2.8 GIS und raumbezogene Indizes 376
9.2.9 Volltextindex 385
9.2.10 Die großen Nachteile des Indexes 393
9.3 Optimierung von Anfragen 394
9.3.1 Die EXPLAIN-Anweisung 394
9.3.2 Auswertung der EXPLAIN-Informationen und Optimierungen 396
9.4 Partitionierung 427

10 Sicherheit 451

10.1 Grundlagen 451
10.1.1 Verschiedene Szenarien für das Rechtemanagement 451
10.1.2 Die zwei Phasen des Berechtigungssystems 453
10.2 Privilegiert – die Benutzerrechte im Detail 454
10.2.1 Die Ebenen der Benutzerberechtigungen 456
10.2.2 Berechtigungen zum Schutz der Daten 459
10.2.3 Berechtigungen zur Programmierung mit MySQL 465
10.2.4 Das Sicherheitskonzept von Programmen oder DEFINER vs. INVOKER 467
10.2.5 Berechtigungen zur Administration von MySQL 470
10.2.6 Die Befehle GRANT und REVOKE im Detail 476
10.2.7 Benutzerverwaltung ohne GRANT – die Datenbank »mysql« 480
10.3 Tipps und Tricks 486
10.3.1 Vorgehen bei der Erstellung von Benutzern 486
10.3.2 Tausende Benutzer und Datenbanken ohne Chaos verwalten 487
10.3.3 Benutzerverwaltungs-API 489
10.3.4 Benutzerverwaltung in der Applikation, nicht in der Datenbank? 490
10.3.5 Workaround für Berechtigungen auf Zeilenebene/ Kapselung 492
10.4 STOP! – MySQL absichern 495
10.4.1 Allgemeine Sicherheitshinweise – der gewissenhafte Administrator 495
10.4.2 Zugang beschränken oder verschlüsseln 498
10.4.3 MySQL im Web 503
10.5 Zusammenfassung 509

TEIL III Programmierung

11 Gespeicherte Programme und Events 513

11.1 Wozu gespeicherte Programme? 513
11.1.1 Arten von gespeicherten Programmen 515
11.1.2 MySQL Stored Program Language 516
11.1.3 Grundlegendes Arbeiten mit gespeicherten Programmen 517
11.1.4 Kommentare 520

11.1.5 Datentypen 521
11.1.6 Sicherung und Backup gespeicherter Programme und Sichten 521
11.1.7 Rechte von gespeicherten Programmen 523
11.1.8 Einschränkungen 523
11.2 Gespeicherte Prozeduren 524
11.2.1 Erzeugung gespeicherter Prozeduren 524
11.2.2 Erweiterte Einstellungen beim Erzeugen von gespeicherten Prozeduren 526
11.2.3 Verwendung gespeicherter Prozeduren 528
11.2.4 Löschen von gespeicherten Prozeduren 529
11.2.5 Bearbeiten gespeicherter Prozeduren 529
11.2.6 Parameter 530
11.2.7 Ergebnismengen 536
11.3 Gespeicherte Funktionen 536
11.3.1 Erzeugen von gespeicherten Funktionen 537
11.3.2 Erweiterte Optionen beim Erzeugen von gespeicherten Prozeduren 540
11.3.3 Aufrufen gespeicherter Funktionen 542
11.3.4 Bearbeiten gespeicherter Funktionen 543
11.3.5 Einschränkungen 544
11.4 Informationen zu bestehenden gespeicherten Programmen (Metadaten) 545
11.5 Programmieren in MySQL 549
11.5.1 BEGIN-/END-Blöcke 549
11.5.2 Variablen 550
11.5.3 Ablaufsteuerung und konditionale Anweisungen 557
11.5.4 Schleifen 564
11.5.5 Cursor 569
11.5.6 Handler 576
11.5.7 Selbst definierte Fehlermeldungen mit SIGNAL 585
11.5.8 Verschachtelte Cursor und deren Handler 590
11.5.9 Temporäre Tabellen 593
11.5.10 Rekursion 594
11.5.11 Transaktionen 598
11.6 Trigger 599
11.6.1 Trigger erstellen 599
11.6.2 Löschen und Bearbeiten von Triggern 604
11.6.3 Informationen über bestehende Trigger (Metadaten) 604
11.6.4 Einschränkungen 607

11.7 Events 608
11.7.1 Event-Scheduler 609
11.7.2 Erzeugung von Events 610
11.7.3 Löschen und Bearbeiten von Events 615
11.7.4 Informationen zu bestehenden Events (Metadaten) 615
11.7.5 Sicherheit, Rechte und Einschränkungen 617
11.8 Sichten (Views) 618
11.8.1 Erzeugung und Verwenden von Sichten 619
11.8.2 Updates auf Sichten 622
11.8.3 Löschen und Bearbeiten einer Sicht 623
11.8.4 Informationen zu bestehenden Sichten (Metadaten) 623
11.9 Zusammenfassung 624

12 Softwareentwicklung mit MySQL 625

12.1 Einleitung 625
12.2 Konnektoren/Programmiersprachen 627
12.2.1 Allgemeines 627
12.2.2 ODBC 632
12.2.3 Java, JDBC und Connector/J 636
12.2.4 PHP, mysqli, PDO_MYSQL 641
12.2.5 .NET 645
12.2.6 Perl, DBD:mysql 650
12.2.7 Ruby 653
12.2.8 Python 655

TEIL IV Anhang

A Referenz 661

A.1 Datentypen 662
A.2 Datendefinition, Datenmanipulation und Abfragen 664
A.3 Index 669
A.4 Boolesche Operatoren 670
A.5 Mathematische Funktionen 671
A.6 Zeit- und Datumsfunktionen 673
A.7 String-Funktionen 678
A.8 Transaktionen 686
A.9 Sperren 687
A.10 Sichten (Views) 688
A.11 MySQL-Client 689

A.12 Gespeicherte Prozeduren 691
A.13 Gespeicherte Funktionen 692
A.14 Triggers 694
A.15 Events 695
A.16 Kontrollfluss-Elemente 696
A.17 Variablen in Programmen 699
A.18 Cursors 699
A.19 Handlers 700
A.20 Volltextsuche 702
A.21 Replikation 703
A.22 GIS-Funktionen 705
A.23 Partitionierung 706
A.24 Administrationsaufgaben 710
A.25 Benutzerverwaltung und Sicherheit 714
A.26 Wartungsarbeiten 717
A.27 Backups, Wiederherstellung und Logfiles 719
A.28 Import und Export 721
A.29 Benchmarking, Monitoring und Profiling 722
A.30 Query-Cache 724
A.31 Plugins 724
A.32 Storage-Engines 725
A.33 Hilfe, Warnungen und Fehler 728
A.34 Fehlernummern und SQL-States 728
A.35 Escape-Sequenzen 730
A.36 Zeichensätze und Sortierfolgen (Character Sets und Collations) 731

Index 735

Vorwort

Da Sie dieses Buch aufgeschlagen haben, haben Sie mit großer Wahrscheinlichkeit bereits von MySQL gehört. In diesem Buch geht es um eines der erfolgreichsten und weit verbreitetesten Open-Source-Softwareprojekte der Welt. MySQL ist ein vollwertiges Datenbankmanagement-System, das nicht nur im Allgemeinen kostenlos für alle Betriebssysteme verfügbar ist, es ist zudem relativ einfach zu bedienen. Doch wozu dann ein eigenes Buch, wenn die Software doch so einfach zu bedienen ist? Der angemerkte einfache Umgang mit MySQL setzt voraus, dass Sie wissen, wo Sie welche Einstellungen vornehmen müssen. Nun könnten Sie sich durch die frei verfügbare englischsprachige Dokumentation im Umfang von mehreren tausend Seiten quälen. Oder Sie wählen die smarte Alternative: Sie haben sich richtig entschieden! Schließlich lesen Sie bereits dieses Buch, herzlichen Glückwunsch!

Damit Sie so schnell wie möglich sicher im Umgang mit MySQL werden, bietet dieses Buch ein umfangreiches und realistisches Praxisbeispiel, anhand dessen die wichtigsten Kniffe, Tipps und Tricks erklärt werden.

Für wen ist dieses Buch gedacht?

MySQL ist ein sehr weit verbreitetes Datenbanksystem, das in den verschiedensten Szenarien erfolgreich eingesetzt werden kann. Wie Sie das meiste aus MySQL herausholen können, lernen Sie in diesem Buch. Vorwiegend richtet sich das vorliegende Administrationshandbuch an Einsteiger und Fortgeschrittene. Aber gleichgültig, ob Sie Datenbanken-Einsteiger sind oder schon Erfahrungen mit MySQL gesammelt haben, ja sich vielleicht sogar schon zu den Profis zählen können: In diesem Buch werden Sie viel Neues lernen! Dasselbe gilt natürlich auch für Umsteiger von anderen Datenbankmanagementsystemen.

Das Besondere an diesem Buch ist das umfassende Praxisbeispiel, anhand dessen alle Kapitel und die darin enthaltenen Befehle genau erläutert werden. Wir sind der Meinung, dass ein interessantes und reales Praxisbeispiel besser ist als eine öde Schritt-für-Schritt-Anleitung. Das bedeutet jedoch auch, dass Sie an manchen Stellen selbst gefordert sind, da wir eine gewisse Eigeninitiative voraussetzen. Getreu dem Motto **learning by doing**, werden Sie MySQL immer besser kennen-

lernen, indem Sie die Entwicklung des vorgestellten Szenarios selbst nachvollziehen werden. Wir sind überzeugt, dass Sie so viel mehr lernen und sicherer im Umgang mit dem Datenbanksystem MySQL werden, als wenn Sie zu sehr an die Hand genommen werden. Doch keine Bange, auf der mitgelieferten DVD finden Sie die notwendigen Daten, um jederzeit neu anfangen zu können, falls Sie sich mal einen gröberen Schnitzer leisten.

Konventionen

In diesem Buch gibt es ein paar wenige Konventionen, welche die Lesbarkeit erhöhen. So werden sämtliche Befehle durch dieses `Format` hervorgehoben, damit Sie schneller erkennen können, dass es sich um eine Anweisung für den MySQL-Server bzw. für Ihr Betriebssystem handelt.

Eingaben, die für den MySQL-Server gedacht sind, tätigen Sie mit dem sogenannten MySQL-Client, einem Kommandozeilen-Programm, in das Sie die Befehle eintippen können, um mit dem Server zu interagieren. Alle kürzeren oder längeren Befehlssequenzen sind in Listings als Code-Auflistungen angegeben. Jene Befehle, die Sie innerhalb dieses Clients eingeben müssen, werden durch die Eingabeaufforderung (engl. *prompt*) `mysql>` hervorgehoben. Oftmals können die Eingaben für den Client aus Platzgründen nicht in einer Zeile eingegeben werden. Ein solcher Zeilenumbruch wird durch diesen Pfeil `->` dargestellt. Dadurch wird ausgedrückt, dass dieser Befehl noch nicht abgeschlossen ist und daher noch fortgesetzt wird. Diese Pfeile dürfen Sie nicht selbst eingeben. Der MySQL-Client schreibt diese Pfeile selbstständig nach einem Zeilenumbruch. In Listing 1 sehen Sie ein Beispiel für das Format, das in diesem Buch für MySQL-Befehle verwendet wird.

```
mysql> SELECT vorname, nachname FROM passagier
    -> WHERE passnummer = 'P137884';
```

Listing 1 Ein MySQL-Befehl

Die MySQL-Befehle selbst sind immer in `GROSSBUCHSTABEN` gehalten, damit Sie diese leichter von Benutzereingaben unterscheiden können. Jene Kommandos, die Sie in der Kommandozeilenumgebung eingeben müssen, werden durch den Linux-Prompt `bash#` eingeleitet. Dadurch sehen Sie auf den ersten Blick, für welche Umgebung der Befehl vorgesehen ist. Ein Beispiel für einen Konsolen-Befehl sehen Sie in Listing 2.

```
bash# mysqladmin ping
```

Listing 2 Ein Konsolen-Befehl

Besondere Formatierungsübereinkünfte gelten auch für die Referenz am Ende dieses Buches, in der Sie alle besprochenen Befehle und Kommandos vollständig aufgelistet finden.

Der Aufbau dieses Buches

Dieses Buch ist in drei Teile gegliedert: »Erste Schritte«, »Administration« und »Programmierung«. Der erste Teil enthält die Grundlagen zu Datenbanken im Allgemeinen und dem Datenbanksystem MySQL im Speziellen. Dabei lernen Sie nicht nur den Datenbankentwurf kennen, sondern erhalten auch eine Einführung in die Theorie von Datenbanken, die für das Verständnis der darauf aufbauenden Kapitel wichtig ist. Die Modellierungstechniken, die Sie hier kennenlernen werden, können Sie natürlich auch bei anderen Datenbankmanagementsystemen einsetzen. Durch diese Techniken lernen Sie eine einheitliche Sprache kennen, die für die Kommunikation mit allen an der Entwicklung eines Datenbank-Projekts beteiligten Personen wichtig ist. Nur wenn alle Beteiligten mit denselben Begriffen auch dieselben Sachverhalte beschreiben, kann ein Projekt erfolgreich umgesetzt werden. Dazu zählt jedoch nicht nur das notwendige Vokabular, sondern auch eine eigene grafische Darstellungsform (sogenannte ER-Diagramme), mit deren Hilfe auf den ersten Blick Zusammenhänge ersichtlich werden. Dieses Beschreibungswerkzeug wird auch von Hilfsprogrammen direkt angewendet, weshalb dies auch für die Arbeit mit MySQL notwendig ist.

Dieser Teil enthält ebenfalls ein Schnellstart-Tutorial, in dem Sie die wichtigsten Befehle kennenlernen, um mit MySQL grundlegend arbeiten zu können. Nach diesem Kapitel sind Sie in der Lage, die Datenbanken selbst samt ihren Tabellen, in denen letztendlich die Daten gespeichert werden, anzulegen und zu verwalten. Außerdem stellt dieser Teil das praxisnahe Flughafenszenario vor, das sich durch das gesamte Buch zieht und einen nachvollziehbaren Rahmen für die unterschiedlichsten Aufgaben bietet. Sie lernen in diesem Abschnitt, wie Sie die Anforderungen an ein Datenbanksystem herausarbeiten und umsetzen können. Auf der DVD zu diesem Buch finden Sie einen sehr umfangreichen Datensatz, mit dem Sie auf realistische Weise den Umgang mit MySQL testen können. Dieses Daten-Set für das Flughafenszenario enthält mehrere Millionen Datensätze und ist daher gut geeignet, um Tests und Optimierungsmaßnahmen durchzuführen.

Der zweite Teil, »Administration«, beschäftigt sich mit der Installation und der Wartung von MySQL. Hier lernen Sie nicht nur, wie Sie MySQL optimal an Ihre Anforderungen anpassen können, sondern auch wie Sie MySQL aktuell halten können. Des Weiteren lernen Sie mit der Replikation eine interessante Möglichkeit kennen, wie Sie MySQL im Hochverfügbarkeits-Bereich und somit auch im High-Performance-Bereich einsetzen können. Wie Sie Ihre Anfragen möglichst effizient gestalten können, lernen Sie ebenfalls in diesem Teil. Ein sehr wichtiger Aspekt dieses Teils ist auch die Absicherung der Datenbanken und der damit verbundenen Benutzerverwaltung.

Im dritten und letzten Teil, »Programmierung«, werden Sie lernen, mit MySQL zu programmieren, um so auch spezielle Anforderungen geschwindigkeitsoptimiert umzusetzen. Mit gespeicherten Programmen lassen sich viele Probleme sehr elegant lösen. Da MySQL in den meisten Fällen als Daten-Backend für Software-Programme dient, lernen Sie hier ebenfalls, wie Sie mit den unterschiedlichsten Programmiersprachen auf den Datenschatz einer MySQL-Installation zugreifen können.

Abgerundet wird dieses Buch durch eine umfangreiche Referenz, in der Sie alle im Buch besprochenen Befehle samt Syntax und Kurzbeschreibung vorfinden. Diese Befehle sind übersichtlich nach Themen geordnet, wodurch Sie sich schneller zurechtfinden werden. Natürlich enthält dieses Buch auch einen Index, in dem Sie alle wichtigen Begriffe und Befehle nachschlagen können und somit schnell finden werden.

Die DVD zum Buch

Auf der beliegenden DVD finden Sie nicht nur alle Versionen der aktuellen, stabilen Version 5.5 von MySQL für Windows, Mac und Linux, sondern auch im Buch vorgestellte Skripte und Tools, die Sie sofort einsetzen können.

Im Verzeichnis *FlughafenDB* finden Sie die Beispieldatenbank mit dem Fughafenszenario. Diese bildet das Herzstück dieses Buches, denn sie ermöglicht es Ihnen, alle Befehle, Tricks und Tools sofort anhand einer vollwertigen Datensammlung auszuprobieren. Die darin enthaltenen Flug-, Passagier- und Buchungsdaten sind möglichst realistisch gehalten. Sie finden zwei Versionen der Daten des Flughafenszenarios auf dieser DVD. Diese unterscheiden sich im Umfang der enthaltenen Daten. Während Sie die kleine Ausgabe sehr schnell importieren können, dient die große Version mit einigen Millionen Datensätzen besonders zum Testen von Optimierungen und Tuning-Maßnahmen. Wie Sie diese Daten importieren

können, erfahren Sie in Abschnitt 4.2, »Die Beispieldatenbank auf der Buch-DVD«.

Die Webseite zum Buch

Unter der Web-Adresse *www.mysqladmin.at* finden Sie aktuelle Informationen rund um dieses Buch, Kontaktmöglichkeiten und Neues rund um MySQL.

Die Autoren

Dipl.-Ing. Stefan Pröll, Jahrgang 1983, studierte in Innsbruck und Madrid Informatik und spezialisierte sich dabei im Masterstudiengang auf Datenbanken und Informationssysteme. Während seines Studiums hat er mehrere Praktika im In- und Ausland bei internationalen Organisationen und Unternehmen im Bereich Wissenschaftskommunikation, Webentwicklung und Datenbankadministration absolviert. Seit 2009 lebt er in Wien, wo er Wirtschaftsingenieurswesen an der Technischen Universität Wien als zweites Masterstudium studiert und zusätzlich im Bereich der Linux-Serveradministration bei privaten Unternehmen gearbeitet hat. Seit 2011 ist er außerdem wissenschaftlicher Mitarbeiter bei einem auf IT-Sicherheit spezialisierten Forschungszentrum. In seiner Freizeit treibt er gelegentlich Sport (Badminton, Jogging) hört leidenschaftlich gerne Soul-, Funk- und Jazzmusik, geht ins Theater und reist, wann immer die Zeit es zulässt.

Dipl.-Ing. Eva Zangerle studierte Informatik an der Universität Innsbruck in Österreich. Dabei spezialisierte sie sich während ihres Masterstudiums im Bereich von Datenbanken. In ihrer Masterarbeit entwickelte sie neue Techniken, um komplexe Datenstrukturen, wie graph-basierte Wissensdarstellungen, in Datenbanken optimiert zu verarbeiten. Seit 2007 arbeitet sie als wissenschaftliche Mitarbeiterin am Lehrstuhl für Datenbanken und Informationssysteme am Institut für Informatik der Universität Innsbruck. Dabei beschäftigt sich ihre Forschung und Lehre mit der Entwicklung von neuen Methoden zur Personalisierung – sogenannten Recommender-Systemen – und der damit verbundenen intelligenten Datenspeicherung. Die von ihr entwickelten Techniken und Verfahren wurden bereits auf zahlreichen internationalen Konferenzen veröffentlicht.

Neben der Forschung und Lehre von neuartigen Datenbanksystemen setzt Eva Zangerle aber bereits seit 2002 auf MySQL, mit dem sie zahlreiche erfolgreiche Projekte im universitären Umfeld realisierte.

Wolfgang Gassler (Dipl.-Ing. Informatik) absolvierte das Informatikstudium an der Universität Innsbruck in Österreich und ist seit 2007 wissenschaftlicher Mitarbeiter in der Forschungsgruppe Datenbanken und Informationssysteme an der Universität Innsbruck. Er beschäftigt sich in seiner Forschung mit neuen Datenbanktechnologien und neuartigen Verfahren zur optimierten Wissens-Speicherung und Analyse.

Seit 1999 arbeitet er mit dem Datenbanksystem MySQL – damals in der Version 3.21 – und der zu dieser Zeit in Tirol/Österreich beinahe unbekannten Programmiersprache PHP in der Version 3. Seit dieser Zeit, als einer der ersten PHP-Entwickler in Tirol, ist er in zahlreichen Softwareprojekten – vorrangig in den Bereichen Web & Mobile – bis heute als Entwicklungsleiter oder in beratender Funktion tätig.

Die Verbindung zwischen Forschung und Wirtschaft ist ihm ein großes Anliegen und so arbeitet er auch heute noch neben seiner wissenschaftlichen und lehrenden Tätigkeit an der Universität Innsbruck an Projekten in der Wirtschaft.

Tausende Benutzer, hunderte Datenbanken, Millionen von Datensätzen und trotzdem sicher? Der Leitfaden zur sicheren MySQL-Installation, damit Sie auch morgen noch beruhigt durchschlafen können.

10 Sicherheit

Der MySQL-Server kommt in zahlreichen Anwendungsbereichen zum Einsatz. Es kann sich um eine einfache Installation auf Ihrem lokalen Rechner, aber auch um eine Großinstallation über mehrere Server hinweg in einem Hochsicherheits-Rechenzentrum handeln. In beiden Fällen muss die Sicherheit Ihrer Daten in den Datenbanken gewährleistet sein, damit nur jene Personen Zugriff auf Ihre gespeicherten und unter Umständen sehr kritischen Informationen erhalten, die auch wirklich die Berechtigung dazu haben. MySQL bietet hierfür zahlreiche Funktionen und Sicherheitsmerkmale, die wir in den folgenden Abschnitten genauer behandeln werden.

10.1 Grundlagen

MySQL steuert die Sicherheit mit sogenannten *Berechtigungen* (oder auch *Rechte* genannt), die an Datenbankbenutzer vergeben werden können. Berechtigungen erlauben oder beschränken dabei das Ausführen von Aktionen in der Datenbank. So kann zum Beispiel eine Berechtigung `SELECT`-Abfragen auf einer bestimmten Tabelle erlauben. Ist keine Berechtigung vorhanden, wird die Durchführung des Befehls von MySQL aus Sicherheitsgründen verweigert.

10.1.1 Verschiedene Szenarien für das Rechtemanagement

Beginnen wir zunächst mit dem einfachsten Einsatzszenario, einer MySQL-Installation mit Ihnen als einzigem Benutzer auf Ihrem lokalen Rechner, auf den nur Sie Zugriff haben. In diesem Fall können Sie für erste Testzwecke die Anforderungen bezüglich Sicherheit und Rechtemanagement in den meisten Fällen auf ein Minimum reduzieren. Es ist ausreichend, wenn Sie ein Passwort für den bereits vorinstallierten Administratorbenutzer »root« setzen und sich aus Sicherheitsgründen ein zusätzliches Benutzerkonto mit weniger Berechtigungen für die eigentliche Arbeit mit MySQL anlegen.

Sobald mehrere Personen oder Applikationen auf die Datenbank zugreifen, sollten Sie sich auf jeden Fall mehr Gedanken über die Sicherheit machen und das sehr ausgereifte Sicherheitskonzept von MySQL nutzen. Denken Sie etwa an die Betreiberfirma unseres Flughafenbeispiels, die alle firmenrelevanten Informationen in der Datenbank ablegt. Dabei sollen natürlich nicht alle Mitarbeiter auf alle Daten zugreifen können. Gerade sehr sensible Informationen, wie zum Beispiel die Gehälter eines jeden Mitarbeiters, dürfen nicht für jede Person einsehbar sein. Es muss aber auch unterschieden werden, ob Daten nur gelesen oder auch geschrieben – also verändert – werden dürfen. In unserem Flughafenbeispiel etwa müssen zahlreiche Benutzer der Datenbank die Flugtabelle lesen können. Es soll jedoch nicht jeder beliebige Flughafenmitarbeiter Änderungen an der Flugtabelle vornehmen dürfen.

Wenn Sie nun die Rechteebene noch um eine Stufe verfeinern, befinden wir uns nicht mehr auf der Tabellenebene, sondern auf Spalten- oder sogar Zeilenebene. Betrachten Sie zum Beispiel die Mitarbeitertabelle, die neben Kontaktinformationen auch eine Spalte »gehalt« beinhaltet. Wie bereits erwähnt, darf die Höhe des Gehalts nur von einem sehr beschränkten Benutzerkreis – etwa der Managementebene – eingesehen werden. Die Kontaktdaten (Spalte der E-Mail-Adresse) hingegen sollen den meisten Mitarbeitern zugänglich gemacht werden. Wir benötigen daher eine genaue Beschränkung von Berechtigungen auf Spaltenebene. Dieselbe Einschränkung wäre auf Zeilenebene denkbar. Eventuell dürfen gewisse Daten der Mitarbeiter nur von Mitarbeitern aus der gleichen Abteilung eingesehen werden. Zum Beispiel könnte eine Abteilungsleiterin die Gehälter der Mitarbeiter ihrer Abteilung einsehen dürfen. Die anderen Zeilen (= Mitarbeiter anderer Abteilungen) müssten jedoch weiterhin geschützt bleiben und der Zugriff müsste verweigert werden. MySQL bietet auf Spalten- und Zeilenebene nur beschränkte Möglichkeiten, Berechtigungen und damit verbundene Beschränkungen zu definieren, jedoch gibt es auch dafür Tricks und Lösungswege, die wir ebenfalls in diesem Kapitel näher beschreiben.

In sehr umfangreichen Installationen mit vielen Datenbankbenutzern ist es üblich, dass gewisse Bereiche auch von geschulten Datenbankbenutzern selbstständig administriert werden. Sie kennen eine derartige Konfiguration vielleicht aus sogenannten Shared-Hosting-Angeboten im Internet, die Webspace inklusive Datenbank für den Internetauftritt vermieten. Dabei werden viele Kunden bzw. ihre Webseiten und Datenbanken auf einem Server betrieben. Zum Einsatz kommt jedoch nur eine einzelne MySQL-Installation. Trotzdem können die Administratoren der jeweiligen Webseiten ihre Tabellen oder sogar Datenbank selbst verwalten, anlegen, löschen und manipulieren. MySQL bietet hier sehr flexible Einstellungsvarianten im Rechtemanagement, die es ermöglichen, Berechti-

gungen auf bestimmte Datenbanknamenmuster zu setzen. So kann zum Beispiel realisiert werden, dass ein Administrator der Webseite *www.mein-kleiner-flughafen.at* alle Datenbanken, die mit »flughafen_« beginnen selbstständig administrieren kann. Er kann somit zum Beispiel die Datenbanken »flughafen_cms« und »flughafen_management« anlegen und darin auch alle Tabellen selbst verwalten. Erhält jeder Administrator einen eigenen zugeteilten Bereich, der sich in diesem Fall durch den Beginn des Datenbanknamens definiert, können so auch Hunderte Webseiten, deren Datenbanken und die Datenbankbenutzer flexibel mit einer Installation verwaltet werden. Die beschriebenen Szenarien sind natürlich nur ein Bruchteil der möglichen Realisierungen. Der MySQL-Server gibt Ihnen mit seinem Rechtemanagement ein sehr mächtiges, aber dennoch einfaches Werkzeug an die Hand, mit dem Sie zahlreiche Szenarien – von der kleinen Installation bis zur großen Installation mit tausenden Benutzern – sicher und verlässlich bewältigen können. Wie Sie mit diesem Werkzeug arbeiten können, erfahren Sie in den folgenden Abschnitten des Buches.

10.1.2 Die zwei Phasen des Berechtigungssystems

Das *Zugriffsberechtigungssystem* stellt das wichtigste Sicherheitsmerkmal von MySQL dar und ist dem eigentlichen Datenbanksystem vorgeschaltet. Es überwacht die gesamte Kommunikation zwischen Server und Client, überprüft dabei ständig die Berechtigungen und gewährleistet dadurch die notwendige Sicherheit. Dabei wird zwischen zwei *Zugriffsphasen* unterschieden. Die erste sicherheitsrelevante Phase, in der bereits Zugriffsrechte überprüft werden müssen, ist die Phase der Verbindung. In der *Verbindungsüberprüfung* wird kontrolliert, ob sich der Benutzer von seinem Computer (Host) aus zur Datenbank verbinden darf. Ist diese Überprüfung erfolgreich, wird vom Client das Passwort in Kombination mit dem Benutzernamen übertragen und vom Server überprüft. Sind alle drei Daten (Host, Benutzer, Passwort) korrekt, kann die Verbindung zur Datenbank ordnungsgemäß hergestellt werden.

Neben der üblichen Prüfung von Benutzername und Passwort bringt die Prüfung des Hosts eine zusätzliche Steigerung der Flexibilität und Sicherheit. So können Sie zum Beispiel gewissen Benutzern erlauben, sich nur von ihrem eigenen Rechner mit einer speziellen IP-Adresse zu verbinden. Üblich ist hier auch die Angabe eines ganzen IP-Bereichs, zum Beispiel der des internen Firmennetzwerks. So sind zum Beispiel auch Zugriffe von außen, bei denen eventuell das Passwort durch ein automatisches Hacker-Programm erraten werden könnte, nicht möglich und werden abgelehnt. Verwenden Sie dennoch »starke« Passwörter (mit Sonderzeichen und Zahlen) für Ihre Datenbankbenutzer, da auch Hostangaben unter Umständen gefälscht werden können. Die Host-Einschränkung ist daher

eine gute Zusatzabsicherung, kann jedoch den Passwortschutz durch ein »starkes« Passwort nicht ersetzen.

Wenn Sie nun die Verbindungsüberprüfung erfolgreich absolviert haben, werden Sie mit dem MySQL-Server verbunden und können nun Anfragen an den Server senden. Jede dieser Anfragen, wie zum Beispiel die Wahl einer Datenbank oder eine SQL-Abfrage, wird ebenfalls überprüft. Diese sogenannte *Anfrageüberprüfung* stellt die zweite Zugriffsphase dar. Dabei wird genau kontrolliert, ob Sie die ausreichenden Berechtigungen in der jeweiligen Datenbank und Tabelle besitzen, damit die Anfrage erfolgreich durchgeführt werden kann. Wie bereits oben erklärt, bietet MySQL für diese Überprüfung ein sehr umfangreiches *Rechtesystem*, das vom einfachen Lesen (`SELECT`) bis zur Administration einer Tabelle (`CREATE`, `ALTER` etc.) sehr genau für jeden Benutzer eingestellt werden kann. Im folgenden Abschnitt erklären wir Ihnen alle möglichen Rechteeinstellungen und die wichtigsten Anweisungen, um das Zugriffberechtigungssystem Ihrer MySQL-Installation zu verwalten.

10.2 Privilegiert – die Benutzerrechte im Detail

Wie im vorherigen Abschnitt erklärt, können für jeden Datenbankbenutzer zahlreiche Zugriffsbeschränkungen angelegt werden. In diesem Abschnitt erfahren Sie, welche Beschränkungen und Berechtigungen in MySQL möglich sind und wie diese über spezielle Anweisungen zugewiesen werden können.

Um einem Benutzer Berechtigungen zuweisen zu können, müssen Sie zunächst einen Benutzer anlegen. Verbinden Sie sich dazu als Benutzer »root« mit dem MySQL-Server, und erstellen Sie einen neuen Benutzer, wie in Listing 10.1 angegeben. Mit dieser Anweisung wird ein neuer Benutzer mit dem Namen »test« erstellt und das Passwort »testpasswort« gesetzt. Dieser Datenbankbenutzer verfügt durch die Anweisung `ALL PRIVILEGES` über alle verfügbaren Rechte in der MySQL-Installation. Anschließend legen Sie fest, auf welchen Datenbanken bzw. Tabellen die angegebenen Rechte gültig sind. In diesem Fall werden alle Berechtigungen (`ALL PRIVILEGES`) durch `ON *.*` auf allen verfügbaren Datenbanken und Tabellen gewährt.

Grundsätzlich wird vor dem Punkt der Datenbankbezeichner und nach dem Punkt ein Tabellenbezeichner einer Tabelle in der angegebenen Datenbank angezeigt, auf jene die Berechtigungen gewährt werden sollen. Das Zeichen * steht für einen beliebigen Namen bzw. Bezeichner und setzt die angegebene Berechtigung somit in diesem Beispiel auf allen Datenbanken bzw. Tabellen.

Durch die Bezeichnung eines Hosts nach dem @, in diesem Fall `localhost`, kann sich der Benutzer »test« nur vom lokalen Computer (IP-Adresse 127.0.0.1) aus zum MySQL-Server verbinden. Der Benutzer muss sich daher an demselben Computer wie der MySQL-Serverdienst befinden. Selbstverständlich können Sie auch durch % als Hostangabe spezifizieren, dass sich der Benutzer von einem beliebigen Host aus anmelden kann. Wird kein @ bzw. kein Host angegeben, wird ebenfalls die Anmeldung von jedem Host erlaubt. Die letzte Anweisung `WITH GRANT OPTION` erlaubt dem Benutzer, die `GRANT`-Anweisung auszuführen und damit seine Rechte auch an andere Benutzer weiterzugeben.

Bei der `GRANT`-Anweisung handelt es sich um den Befehl zum Setzen von gewissen Benutzerrechten. Existiert der angegebene Benutzer wie in unserem Beispiel jedoch noch nicht, legt die `GRANT`-Anweisung auch diesen automatisch an und setzt anschließend die Rechte:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'test'@'localhost'
    -> IDENTIFIED BY 'testpasswort' WITH GRANT OPTION;
```

Listing 10.1 Anlegen eines neuen Benutzers mit dem GRANT-Befehl

Neben der `GRANT`-Anweisung ist die Anweisung `REVOKE` sehr wichtig und wird dazu verwendet, bereits vergebene Rechte zu widerrufen bzw. aufzuheben. In Listing 10.2 sehen Sie eine Anweisung, die alle gesetzten Rechte auf allen Datenbanken und das `GRANT`-Recht unseres Testbenutzers wieder aufhebt.

```
mysql> REVOKE ALL PRIVILEGES, GRANT OPTION
    -> FROM 'test'@'localhost';
```

Listing 10.2 Löschen aller Rechte des angelegten Benutzers

Beachten Sie, dass der Benutzer zwar alle Rechte auf allen Datenbanken verliert, dass das Benutzerkonto jedoch weiterhin bestehen bleibt und der Benutzer sich daher auch weiterhin mit dem MySQL-Server verbinden kann. Die Anweisung `REVOKE` löscht also in keinem Fall einen Benutzer aus dem MySQL-System. Wollen Sie den Benutzer vollständig und unwiderruflich löschen, ist dies mit der Anweisung `DROP USER`, wie in Listing 10.3 dargestellt, möglich.

```
mysql> DROP USER 'test'@'localhost';
```

Listing 10.3 Löschen des angelegten Benutzers

Nun haben Sie die wichtigsten drei Grundanweisungen kennengelernt, mit denen Sie bereits neue Benutzer anlegen, Berechtigungen vergeben und widerrufen und selbstverständlich auch Benutzer wieder löschen können. Im Folgenden

gehen wir nun näher auf die verschiedenen Ebenen der Rechteverwaltung ein. Diese ermöglichen es Ihnen, Berechtigungen zum Beispiel nur auf einer gewissen Spalte oder aber auch datenbankweit zu setzen.

10.2.1 Die Ebenen der Benutzerberechtigungen

Je nach Berechtigung können die von MySQL zur Verfügung gestellten Rechte in unterschiedlichen Granularitäten vergeben werden. Dies hilft Ihnen dabei, mit dem Berechtigungssystem auf die Anforderungen in diversen Anwendungsszenarien möglichst flexibel zu reagieren. Dabei sind die folgenden Granularitäten oder Berechtigungsebenen in MySQL möglich:

- globale Ebene
- Datenbankebene
- Tabellenebene
- Spaltenebene
- Zeilenebene (nur begrenzt möglich)
- Objektebene

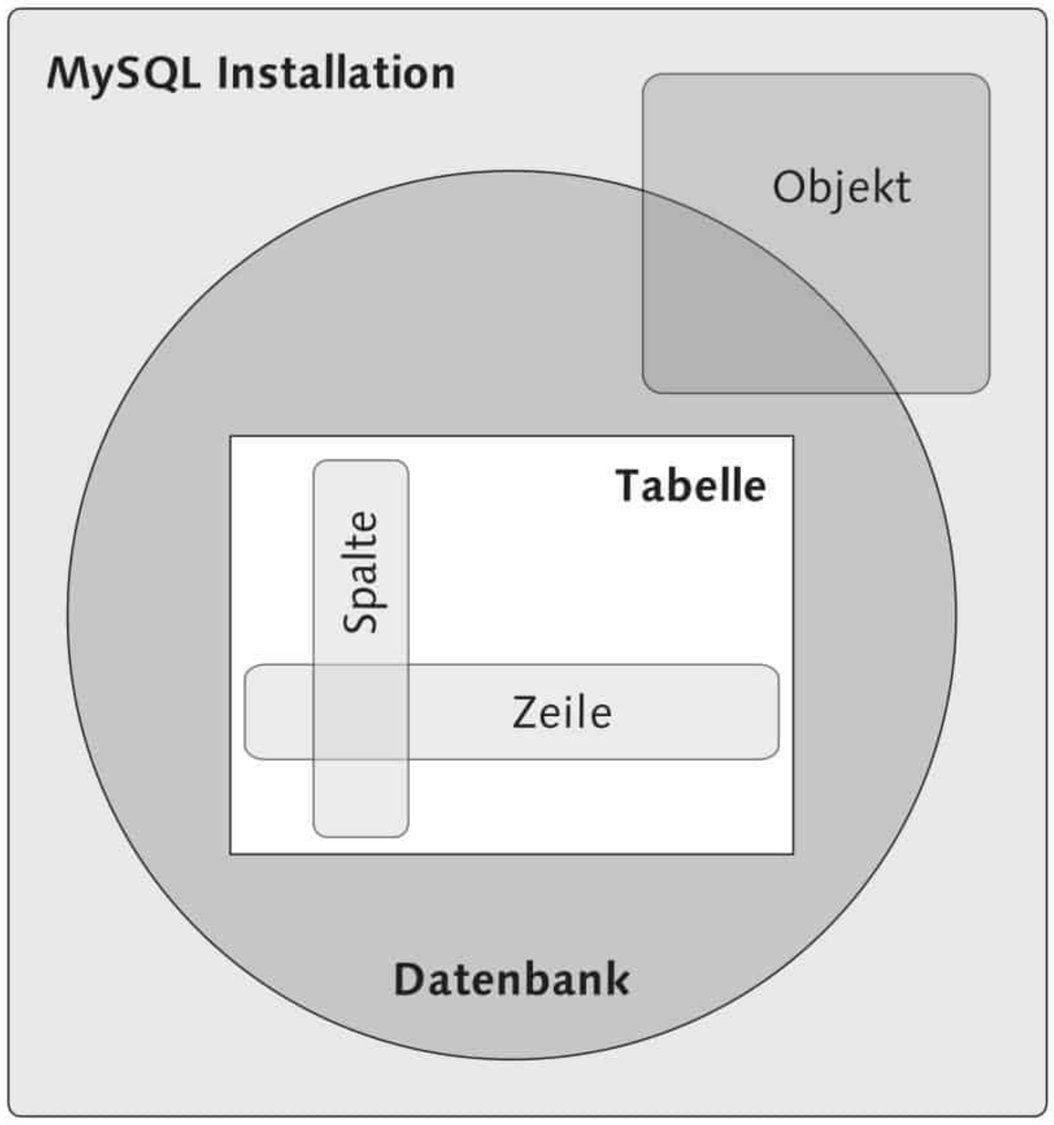


Abbildung 10.1 Berechtigungsebenen in MySQL

In Abbildung 10.1 sehen Sie die Berechtigungsebenen und wie diese zusammenwirken. Zum Beispiel schließt die Datenbankebene die Tabellenebene völlig mit ein. Das bedeutet, dass Berechtigungen auf Datenbankebene automatisch auch auf allen Tabellen in der jeweiligen Datenbank gültig sind. Andere Bereiche, wie zum Beispiel die Spalten- und Zeilenebene, schließen sich nicht vollständig ein, können sich aber bei gewissen Daten (Zellen) natürlich überschneiden. Auch Objekte sind unabhängig und können sich auf mehreren Ebenen befinden. Im Folgenden stellen wir Ihnen die verschiedenen Granularitäten bzw. Ebenen und deren Anwendung genauer vor.

Globale Ebene

Berechtigungen auf der *globalen Ebene* sind über die gesamte MySQL-Installation hinweg gültig. Eine dieser Berechtigungen, die `GRANT`-Berechtigung zur Vergabe von Berechtigungen, haben Sie bereits kennengelernt. Dieses Recht kann nur auf globaler Ebene vergeben werden, da es sich nicht auf eine spezielle Datenbank oder Tabelle bezieht. In diesem Kontext spricht man auch oft von *Administratorberechtigungen*, da globale Berechtigungen meist mit der Verwaltung der MySQL-Installation in Verbindung stehen. In Listing 10.4 sehen Sie eine Vergabe der Berechtigung `SELECT` auf der globalen Ebene an den Benutzer »admin«. Die globale Ebene wird dabei immer mit `*.*` gekennzeichnet, da dieses Recht auf globaler Ebene über alle Datenbanken und Tabellen hinweg gültig ist.

```
mysql> GRANT SELECT ON *.* TO 'admin'@'%';
```

Listing 10.4 GRANT-Befehl auf globaler Ebene

Datenbankebene

Berechtigungen auf *Datenbankebene* können Sie jeweils auf eine oder mehrere Datenbanken setzen. Gerade in großen MySQL-Installationen, in denen es oft zahlreiche Datenbankadministratoren gibt, ist eine Rechte-Granularität auf Datenbankebene vonnöten. So kann man zum Beispiel einem Web-Administrator des Flughafens erlauben, dass er die Tabellen in der Datenbank »cms« (Daten des Content-Management-Systems) selbst verwaltet. Er nimmt so dem Hauptadministrator viel Arbeit ab und kann in seiner Datenbank selbst Tabellen erzeugen (`CREATE`), editieren (`ALTER`) oder auch löschen (`DROP`). In Listing 10.5 wird die Berechtigung `DROP` auf Datenbankebene vergeben.

Die Datenbankebene wird mit dem Namen der Datenbank und einem darauffolgenden `.*` gekennzeichnet. Der `*` (Wildcard-Symbol) steht dabei für eine beliebige Tabelle in der angegebenen Datenbank. Der Benutzer »admin« besitzt damit in diesem Beispiel die `DROP`-Berechtigung auf allen Tabellen (`.*`) in der angegebenen Datenbank »FlughafenDB«.

```
mysql> GRANT DROP ON FlughafenDB.* TO 'admin'@'%';
```

Listing 10.5 GRANT-Befehl auf Datenbankebene

Tabellenebene

Selbstverständlich können Sie Berechtigungen auch auf *Tabellenebene* vergeben, um zum Beispiel gewissen Benutzern nur Leserechte (`SELECT`) auf einer Tabelle zu gewähren. Das Einfügen (`INSERT`) oder Aktualisieren (`UPDATE`) neuer Datensätze ist damit nicht möglich und wird vom MySQL-System verwehrt. In Listing 10.6 sehen Sie die Zuweisung der `SELECT`-Berechtigung auf die Tabelle »mitarbeiter«. Der Benutzer »admin« besitzt somit nur Leserechte auf der Tabelle »mitarbeiter«. Die Tabellenebene wird durch Angabe des Tabellennamens gekennzeichnet.

```
mysql> GRANT SELECT ON FlughafenDB.mitarbeiter TO 'admin'@'%';
```

Listing 10.6 GRANT-Befehl auf Tabellenebene

Spaltenebene

Die Lese- und Schreibrechte können Sie ebenfalls auf *Spaltenebene* setzen. So können Sie zum Beispiel einem Benutzer erlauben, die Spalte mit der E-Mail-Adresse der Mitarbeiter zu lesen. Sensible Daten in derselben Tabelle, wie zum Beispiel die Spalte mit der Höhe des Gehalts, können Sie für die Allgemeinheit sperren und zum Beispiel nur dem Personalmanager zugänglich machen. In Listing 10.7 wird die `UPDATE`-Berechtigung auf der Spalte »gehalt« dem Benutzer »admin« zugewiesen. Die Spalten werden dabei in einer Liste (durch Kommata getrennt) nach dem Bezeichner der Berechtigung (möglich sind `SELECT`, `INSERT`, `UPDATE`, `REFERENCES`) in runden Klammern angegeben.

```
mysql> GRANT UPDATE(gehalt) ON FlughafenDB.mitarbeiter
    -> TO 'admin'@'%';
```

Listing 10.7 GRANT-Befehl auf Spaltenebene

Zeilenebene

MySQL bietet derzeit keine direkte Unterstützung für die Einschränkung von Zugriffen auf *Zeilenebene*. Mit ein wenig Fingerfertigkeit kann aber auch dieses Problem gelöst werden. Nähere Details dazu finden Sie in Abschnitt 10.3.5, »Workaround für Berechtigungen auf Zeilenebene/Kapselung«.

Objektebene

Die bisher beschriebenen Ebenen betrachten Berechtigungen in Bezug auf den Zugriff auf die in MySQL gespeicherten Daten – also Tabellen gruppiert in Datenbanken. Neben den gespeicherten Daten existieren aber auch noch andere Konstrukte in MySQL, die unter dem Sammelbegriff »Objekte« zusammengefasst werden. Diese Objekte können Sichten (Views), gespeicherte Prozeduren (Stored Procedures), Ereignisse (Events) oder Indizes sein, die selbstverständlich auch durch ein Sicherheitskonzept bzw. durch Berechtigungen geschützt werden müssen. Rechte auf Objektebene werden durch eine zusätzliche Angabe des Objekttyps, wie in Listing 10.8 gezeigt, angegeben. In diesem Beispiel wird die `EXECUTE`-Berechtigung auf eine Prozedur (Schlüsselwort `PROCEDURE`) mit dem Namen `berechne_distanz` vergeben.

```
mysql> GRANT EXECUTE ON PROCEDURE FlughafenDB.berechne_distanz
    -> TO 'admin'@'%';
```

Listing 10.8 GRANT-Befehl auf Objektebene (PROCEDURE)

Die richtige Ebene für die Berechtigung

Die Berechtigungen in MySQL können meistens auf einer speziellen Ebene gesetzt werden. Gewisse Berechtigungen können auch auf mehreren Ebenen gesetzt werden. So können Sie je nach Bedarf eine einfache Leseberechtigung (`SELECT`) global, auf Datenbanken oder nur auf einzelne Tabellen setzen. Das Recht zur Erzeugung von neuen Benutzern (`CREATE USER`) hingegen können Sie nur auf globaler Ebene setzen. Auf welchen Ebenen die jeweiligen Rechte einsetzbar sind, erfahren Sie im nächsten Abschnitt zusammen mit den genauen Beschreibungen aller verfügbaren Berechtigungen.

10.2.2 Berechtigungen zum Schutz der Daten

In diesem Bereich gehen wir nun näher auf die einzelnen Rechte ein, die Ihnen MySQL zur Verfügung stellt. Beginnen wir zunächst mit dem wichtigsten Gut einer Datenbank – Ihren Daten.

In Tabelle 10.1 finden Sie eine Auflistung der wichtigsten Berechtigungen zur Verwaltung der Datenbanken, Tabellen und des Zugriffs auf die gespeicherten Daten. Die Spalte Ebene/Kontext beschreibt dabei, in welchem Kontext die Berechtigung verwendet wird bzw. auf welcher Granularitätsebene diese gesetzt werden kann. Im darauffolgenden Bereich finden Sie zu den in der Tabelle gelisteten Berechtigungen genauere Beschreibungen und praxisrelevante Beispiele.

Berechtigung	Ebene/Kontext
SELECT	Tabellen, Spalten
INSERT	Tabellen, Spalten
UPDATE	Tabellen, Spalten
DELETE	Tabellen
CREATE	Datenbanken, Tabellen, Indizes
CREATE TEMPORARY TABLES	Datenbanken
ALTER	Tabellen
DROP	Datenbanken, Tabellen, Sichten
INDEX	Tabellen
LOCK TABLES	Datenbanken
CREATE VIEW	Sichten
SHOW VIEW	Sichten

Tabelle 10.1 Wichtigste Berechtigungen bezüglich der Datenverwaltung

SELECT

Die Berechtigung SELECT können Sie auf Tabellen- und Spaltenebene setzen. Sie berechtigt einen Benutzer, lesend auf eine Tabelle bzw. Spalte zuzugreifen. Beachten Sie, dass bei UPDATE- und DELETE-Anweisungen stets eine Leseberechtigung vonnöten ist. So benötigt zum Beispiel die SQL-Anweisung in Listing 10.9 zur korrekten Durchführung auch Leserechte für die Spalte »flug_id«.

Betrachten Sie das Beispiel eines Webadministrators, der für den Webauftritt des Flughafens verantwortlich ist. Dieser soll auf allen Tabellen des Content-Management-Systems in der Datenbank »cms« uneingeschränkten Lesezugriff erhalten. In Listing 10.10 wird diese Berechtigung in der ersten Zeile für den Benutzer mit dem Namen »webadmin« gesetzt.

Gewisse SELECT-Anweisungen benötigen jedoch auch keine besonderen Berechtigungen. Wird zum Beispiel bei einer SELECT-Anweisung nicht auf Daten zugegriffen, so können diese Anweisungen auch ohne SELECT-Berechtigung, wie in Listing 10.11 angegeben, durchgeführt werden. Auch der Aufruf von Funktionen, wie im Beispiel der Cosinus-Funktion, ist ohne Berechtigung möglich, wenn in der Funktion auf keine Daten zugegriffen wird.

```
mysql> DELETE FROM flug WHERE flug_id < 5;
```

Listing 10.9 SQL-Anweisung, die auch Leserechte benötigt

```
mysql> GRANT SELECT ON `cms`.*  TO 'webadmin'@'%';
mysql> GRANT UPDATE ON `cms`.*  TO 'webadmin'@'%';
mysql> GRANT DELETE ON `cms`.*  TO 'webadmin'@'%';
```

Listing 10.10 Rechte eines Webadministrators in der Datenbank »cms«

```
mysql> SELECT 3*5;
+-----+
| 3*5 |
+-----+
|  15 |
+-----+
mysql> SELECT COS(90);
+---------------------+
| COS(90)             |
+---------------------+
| -0.4480736161291701 |
+---------------------+
```

Listing 10.11 SELECT-Anweisungen ohne Datenzugriff

INSERT und UPDATE

Die Berechtigungen `INSERT` und `UPDATE` können Sie in derselben Weise wie die Berechtigung `SELECT` auf Tabellen- und Spaltenebene vergeben. `INSERT` erlaubt dabei das Einfügen von neuen Datensätzen in der jeweiligen Tabelle. Beachten Sie, dass die Berechtigung `INSERT` auch für die korrekte Durchführung der Operationen `ANALYZE_TABLE`, `OPTIMIZE_TABLE` und `REPAIR_TABLE` benötigt wird. Das Recht `UPDATE` erlaubt einem Benutzer, den Inhalt einer Zeile zu aktualisieren, also zu verändern. Zusätzlich können Sie die `UPDATE`-Berechtigung auch auf Spaltenebene setzen, um die Aktualisierung einer Zeile auf bestimmte Spalten einzuschränken.

DELETE

Die Berechtigung `DELETE` erlaubt das Löschen von Zeilen in einer angegebenen Tabelle. Da mit der `DELETE`-Anweisung immer ganze Zeilen gelöscht werden, können Sie diese Berechtigung nicht auf Spaltenebene, sondern nur auf Tabellenebene setzen.

CREATE

Die Berechtigung `CREATE` erlaubt das Erstellen von Tabellen und Datenbanken. Für die Erstellung von Tabellen können Sie genau spezifizieren, in welchen Datenbanken es erlaubt ist, Tabellen anzulegen. Zusätzlich können Sie die

Namensgebung von Datenbanken und Tabellen beeinflussen. Mit der Anweisung in Listing 10.12 können wir unserem Webadministrator »webadmin« erlauben, nur Datenbanken anzulegen, die mit dem Präfix »cms_« beginnen. Beachten Sie, dass das Zeichen _ bereits als Sonderzeichen in MySQL vergeben ist und wir daher den normalen Unterstrich »escapen« und als _ angeben müssen. In den Datenbanken, die auf »cms_« lauten, darf der Benutzer durch die Tabellenmusterangabe .* Tabellen mit einem frei wählbaren Namen anlegen. Listing 10.13 zeigt die Fehlermeldung, falls der Benutzer »webadmin« eine Datenbank anlegt, die nicht dem angegebenen Muster entspricht.

Die möglichen Namenskonventionen beschränken sich nicht nur auf ein Präfix, Sie können selbstverständlich auch komplexere Muster für die Datenbank- und Tabellennamen angeben. Beachten Sie, dass die Namensschemas in sogenannten *Backticks* (`) (rückwärtsgerichtete Anführungszeichen) angegeben werden müssen. Diese Vorgehensweise wird oft verwendet, um verschiedenen Benutzern zu erlauben, selbst Datenbanken anzulegen. Damit man auch bei Hunderten von Datenbanken von den verschiedensten Benutzern noch den Überblick behalten kann, wird für jeden Benutzer ein eigenes Präfix vergeben. So kann man jede Datenbank anhand des Namens dem jeweiligen Benutzer zuordnen, und die Verwaltung wird vereinfacht. Weitere Tipps und Tricks zur Verwaltung von sehr großen Installationen mit vielen Benutzern erhalten Sie in Abschnitt 10.3.2.

```
mysql> GRANT CREATE ON `cms\_%`.* TO 'webadmin'@'%';
mysql> GRANT ALTER ON `cms\_%`.* TO 'webadmin'@'%';
mysql> GRANT DROP ON `cms\_%`.* TO 'webadmin'@'%';
mysql> GRANT INDEX ON `cms\_%`.* TO 'webadmin'@'%';
```

Listing 10.12 Festsetzen eines Namensschemas für Datenbanken

```
mysql> CREATE DATABASE cms_log;
Query OK, 1 row affected (0.00 sec)
mysql> CREATE DATABASE meine_tabelle;
ERROR 1044 (42000): Access denied for user 'webadmin'@'%' to
database 'meine_tabelle'
```

Listing 10.13 Anlegen von Datenbanken mit Benutzer »webadmin«

CREATE TEMPORARY TABLE

Die Berechtigung `CREATE TEMPORARY TABLE` erlaubt die Erzeugung von temporären Tabellen. Diese Tabellen können Sie mit dem erweiterten Befehl `CREATE TEMPORARY TABLE` zusammen mit den gewünschten Spalten wie gewohnt anlegen. Temporäre Tabellen sind dabei nur für die aktuelle Verbindung gültig und werden danach automatisch wieder gelöscht. Die jeweilige temporäre Tabelle ist

auch nur für den erstellenden Benutzer bzw. innerhalb der aktuellen Verbindung sichtbar, wobei temporäre Tabellen bei `SHOW TABLES` nicht mit angezeigt werden. Andere Benutzer sehen die temporäre Tabelle nicht und können diese auch nicht ansprechen.

ALTER

Die Berechtigung `ALTER` erlaubt das Ausführen der Anweisung `ALTER TABLE` zur Modifizierung von Tabellen. Das Recht vergeben Sie daher auf der Granularitätsebene von Tabellen. Beachten Sie, dass Sie für die Veränderung von Tabellen auch die Berechtigungen `INSERT` und `CREATE` zusätzlich vergeben müssen. In Listing 10.12 erhält der Benutzer »webadmin« die Berechtigung `ALTER` auf allen Tabellen in den Datenbanken mit dem Präfix »web_«. Damit ist der Benutzer nun auch in der Lage, alle Tabellen in diesen Datenbanken zu modifizieren (`INSERT` und `CREATE` werden hier als bereits vergeben vorausgesetzt).

DROP

Das Recht `DROP` erlaubt dem Benutzer das Löschen von Datenbanken und Tabellen. In Listing 10.12 wird in der dritten Zeile dem Benutzer »webadmin« erlaubt, auch alle Datenbanken mit dem Präfix »cms_« bzw. die darin befindlichen Tabellen und Sichten (Views) zu löschen. Der Benutzer »webadmin« hat mit diesen Berechtigungen aus Listing 10.12 nun im Subbereich »cms_« alle Möglichkeiten und kann dort beliebige Datenbanken mit dem Präfix »cms_« und Tabellen anlegen, verwalten und auch wieder löschen.

Beachten Sie, dass die Berechtigung `DROP` auch für die Anweisung `TRUNCATE TABLE` benötigt wird, da diese Anweisung eine angegebene Tabelle löscht (`DROP`) und anschließend wieder anlegt (`CREATE`) und damit ein sehr schnelles Leeren von sehr speicherintensiven Tabellen realisiert.

Auch Sichten (siehe Abschnitt 11.8, »Sichten (Views)«) werden mit dieser tabellenbezogenen Berechtigung verwaltet, da sie als »virtuelle« Tabellen angesehen werden.

INDEX

Mit der Berechtigung `INDEX`, die auf Tabellenebene spezifiziert wird, kann der jeweilige Benutzer Indizes auf den angegebenen Tabellen anlegen und auch wieder löschen. Beachten Sie, dass auch ein Benutzer mit der Berechtigung `CREATE` Indizes direkt bei der Erzeugung von Tabellen (ohne vorhandene `INDEX`-Berechtigung) anlegen kann. Soll ein Index nachträglich angelegt oder gelöscht werden, wird jedoch das `INDEX`-Recht benötigt. Damit der Beispielbenutzer »webadmin« in sei-

nen Datenbanken bzw. Tabellen auch Indizes verwalten kann, wird die `INDEX`-Berechtigung in Listing 10.12 analog zu den anderen Berechtigungen gesetzt.

LOCK TABLES

Die Berechtigung `LOCK TABLES` ist auf Datenbankebene gültig und erlaubt das explizite Setzen von Sperren auf den Tabellen in der jeweiligen Datenbank. Die zugehörigen Befehle `LOCK TABLES` bzw. `UNLOCK TABLES` ermöglichen mit der Berechtigung `LOCK TABLES` das Setzen und Aufheben von Schreib- oder Lesesperren, wie in Listing 10.14 angegeben. Beachten Sie, dass dieses Recht keinen Einfluss auf das automatische Setzen von Sperren durch den MySQL-Server hat. MySQL setzt zum Beispiel auch unabhängig von Berechtigungen selbstständig Sperren zur Gewährleistung der Konsistenz im laufenden Betrieb. Weitere Informationen zu Sperren finden Sie in Abschnitt 5.3.3, »Lock-Management«.

```
mysql> LOCK TABLES FlughafenDB.mitarbeiter READ;
mysql> LOCK TABLES FlughafenDB.mitarbeiter WRITE;
mysql> UNLOCK TABLES;
```

Listing 10.14 Verwenden von LOCK und UNLOCK

CREATE VIEW

Die Berechtigung `CREATE VIEW` erlaubt das Erstellen von neuen Sichten (Views) mit der Anweisung `CREATE VIEW` und kann in derselben Weise wie die Berechtigung `CREATE` verwendet werden. Es ist ebenfalls möglich, die zu erstellenden Sichten auf ein Namensmuster, wie zum Beispiel ein Präfix in Listing 10.12, einzuschränken. Zusätzlich zu diesem Recht setzen Sichten ein weiteres Sicherheitskonzept ein, das während ihrer eigentlichen Ausführung ebenfalls zusätzliche Berechtigungen überprüft. Nähere Informationen zu diesem Sicherheitskonzept finden Sie im Abschnitt »DEFINER vs. INVOKER« in Abschnitt 10.2.4.

SHOW VIEW

Die Berechtigung `SHOW VIEW` erlaubt die Verwendung der Anweisung `SHOW CREATE VIEW`, die den Quellcode der Sicht (View) anzeigt. Im ersten Moment erscheint die Erstellung einer eigenen Berechtigung für die Anzeige von Quellcodes etwas überspitzt. Sichten werden jedoch oft als Sicherungsschicht eingesetzt, die gewisse Details und Daten von Tabellen verstecken. Dadurch kann alleine die Ansicht des Quellcodes einer Sicht bereits ein Sicherheitsrisiko darstellen. Ein Angreifer könnte so über den Quellcode erfahren, welche Teile einer Tabelle versteckt werden und weiterführende Angriffe durchführen. Aus diesem Grund wurde diese Anweisung auch mit einem eigenen Recht versehen.

10.2.3 Berechtigungen zur Programmierung mit MySQL

Neben den Berechtigungen zur Verwaltung und zum Zugriff auf Daten werden auch die Erstellung und Verwaltung von Funktionen, Prozeduren, Triggern und Events, die in Kapitel 11, »Gespeicherte Programme und Events«, im Detail beschrieben werden, ebenfalls über eigene Berechtigungen kontrolliert. In Tabelle 10.2 sehen Sie eine Auflistung der Berechtigungen in Bezug auf die Programmierung mit MySQL, die im Folgenden näher beschrieben werden.

Berechtigung	Ebene/Kontext
`EVENT`	Datenbanken
`CREATE ROUTINE`	gespeicherte Prozeduren
`ALTER ROUTINE`	gespeicherte Prozeduren
`EXECUTE`	gespeicherte Prozeduren
`TRIGGER`	Tabellen

Tabelle 10.2 Berechtigungen im Bereich der Programmierung

EVENT

Die Berechtigung `EVENT` setzen Sie auf Datenbankebene. Sie erlaubt dem Benutzer das Anlegen (`CREATE EVENT`) und Verwalten (`ALTER EVENT`, `DROP EVENT`) von Events, die auf den angegebenen Datenbanken operieren. Vielleicht fragen Sie sich, warum für dieses mächtige Konstrukt der Events nur eine Berechtigung existiert und diese nicht weiter unterteilt wird. Bei Events gibt es jedoch noch weitere Sicherheitsmaßnahmen, die mit dem Benutzer, der den Event erstellt bzw. startet, gekoppelt sind. Näheres zu diesem allgemeinen Sicherheitskonzept, das beim Aufruf von Programmcode (Events, Triggern, Prozeduren) verwendet wird, finden Sie im Abschnitt »DEFINER vs. INVOKER« in Abschnitt 10.2.4.

CREATE ROUTINE

Die Berechtigung `CREATE ROUTINE` operiert auf Datenbankebene und erlaubt dem jeweiligen Benutzer, gespeicherte Prozeduren und Funktionen in der angegebenen Datenbank zu erstellen. Beachten Sie, dass es nicht möglich ist, diese Berechtigung auf spezielle Namen oder Namensmuster für Prozeduren einzuschränken. Die Berechtigung gilt immer uneingeschränkt in der angegebenen Datenbank oder auf eine angegebene Prozedur. Das Beispiel in Listing 10.15 weist dem Benutzer »admin« das Recht zur Erstellung von gespeicherten Prozeduren in der Datenbank »FlughafenDB« zu:

```
mysql> GRANT CREATE ROUTINE ON FlughafenDB.* TO 'admin';
mysql> GRANT ALTER ROUTINE ON PROCEDURE
```

```
    -> FlughafenDB.berechne_distanz TO 'admin';
mysql> GRANT EXECUTE ON FlughafenDB.* TO 'admin';
```

Listing 10.15 Berechtigung zur Erzeugung und Verwaltung von Prozeduren

ALTER ROUTINE

Die Berechtigung `ALTER ROUTINE` erlaubt einem Benutzer die Modifizierung von Prozeduren. In der zweiten Zeile in Listing 10.15 finden Sie ein Beispiel, in dem der Benutzer »admin« die Berechtigung zur Modifizierung der Prozeduren `berechne_distanz` in der Datenbank »FlughafenDB« erhält. Beachten Sie, dass Sie bei Prozeduren nur die Metadaten modifizieren können. Für die Veränderung des Quellcodes einer Prozedur müssen Sie die Prozedur löschen und wieder erneut erstellen. Daher erlaubt diese Berechtigung neben der Modifizierung auch die Löschung (über den Befehl `DROP PROCEDURE`) der jeweiligen Prozedur bzw. Funktion. Beachten Sie auch, dass Benutzer beim Erzeugen einer neuen Routine das zugehörige Recht `ALTER ROUTINE` für die erstellte Routine automatisch erhalten. Dieses Verhalten können Sie mithilfe der Variablen `automatic_sp_privileges` verändern und durch den Wert 0 deaktivieren.

EXECUTE

Die Berechtigung `EXECUTE` benötigen Sie, um eine gespeicherte Prozedur auszuführen. Diese Berechtigung wenden Sie in derselben Weise wie die oben angeführte Berechtigung `ALTER ROUTINE` an, und zwar auf einzelne oder mehrere Prozeduren. Listing 10.15 zeigt ein Beispiel, in dem der Benutzer »admin« die Berechtigung erhält, alle Prozeduren in der Datenbank »FlughafenDB« auszuführen. Wie auch das Recht `ALTER ROUTINE`, wird die `EXECUTE`-Erlaubnis dem Ersteller einer Prozedur automatisch zugewiesen. Dieses automatische Verhalten können Sie ebenfalls über die Variable `automatic_sp_privileges` deaktivieren (Wert 0).

Beachten Sie aber, dass das Recht `EXECUTE` allein nicht die ordnungsgemäße Ausführung einer Prozedur garantiert. Während der Ausführung können noch weitere Rechte benötigt werden, die in Zusammenhang mit den in der Prozedur verwendeten Befehlen stehen. Nähere Informationen zu diesem erweiterten Sicherheitskonzept von Prozeduren finden Sie im Abschnitt »DEFINER vs. INVOKER« in Abschnitt 10.2.4.

TRIGGER

Trigger beziehen sich immer auf Tabellen. Die zugehörige Berechtigung `TRIGGER` vergeben Sie daher auf Tabellenebene. Sie erlaubt das Anlegen, Löschen und Ausführen der Trigger auf der jeweiligen Tabelle. Die eigentliche Rechteüberprüfung des Triggers wird über das DEFINER- bzw. INVOKER-Konzept realisiert.

10.2.4 Das Sicherheitskonzept von Programmen oder DEFINER vs. INVOKER

Im vorherigen Abschnitt haben Sie die grundlegenden Benutzerberechtigungen zur Programmierung in MySQL kennengelernt. Wie bereits erwähnt, beschränken sich diese Berechtigungen nur auf die grundlegende Absicherung – zum Beispiel ob Prozeduren ausgeführt oder erstellt werden dürfen. Das Sicherheitskonzept in MySQL sieht bei der Programmierung neben diesen Grundberechtigungen eine zweite Sicherheitsüberprüfung vor. Da es sich bei Programmen meist um eine Aneinanderreihung von SQL-Anweisungen handelt, müssen natürlich die Berechtigungen für alle einzelnen SQL-Anweisungen überprüft werden.

Betrachten Sie zum Beispiel die sehr einfache Prozedur `flug_loeschen()` in Listing 10.16. Der Benutzer, der diese Prozedur startet (`CALL flug_loeschen();`), muss selbstverständlich über die `DELETE`-Berechtigung auf der Tabelle »flug« verfügen, um diese Prozedur korrekt ausführen zu können. Welcher Benutzer für die Rechteüberprüfung von MySQL herangezogen wird und wie dies von Ihnen beeinflusst werden kann, erklären wir im folgenden Abschnitt.

DEFINER vs. INVOKER

Das Sicherheitskonzept von `DEFINER` (erstellender Benutzer) und `INVOKER` (aufrufender Benutzer) wird bei allen gespeicherten Programmen in MySQL verwendet. Dazu zählen die Konstrukte der gespeicherten Prozeduren, Funktionen, Trigger, Sichten (Views) und Ereignisse (Events), die in Kapitel 11, »Gespeicherte Programme und Events«, näher beschrieben werden. Betrachten wir zunächst diese zwei Begrifflichkeiten. Der `DEFINER` eines beliebigen Programms ist jener Benutzer, der das Programm erstellt hat. Der `INVOKER` ist der Benutzer, der das Programm aufruft (zum Beispiel `CALL flug_loeschen();` bei einer gespeicherten Prozedur) und damit auch das Programm ausführt. Der `INVOKER` kann sich bei jedem Aufruf ändern, da Programme von verschiedenen Benutzern aufgerufen werden können. Der `DEFINER` hingegen ist im Normalfall durch den erstellenden Benutzer festgelegt und ändert sich nicht mehr.

Das Sicherheitskonzept von MySQL bei Programmen sieht zwei Einstellungen bezüglich des Benutzers vor. Sie können ein Programm mit den Berechtigungen eines vorher manuell festgelegten Benutzers oder des gerade aktiven Benutzers ausführen lassen. Diese Einstellung legen Sie mithilfe der Anweisung `SQL SECURITY` fest, und geben Sie entweder mit `INVOKER` (aufrufender Benutzer) oder `DEFINER` (erstellender Benutzer) an.

In Listing 10.16 sehen Sie ein Beispiel, bei dem in der zweiten Zeile mit `SQL SECURITY INVOKER` angegeben wird, dass das Programm unter dem aktiven aufru-

fenden Benutzer ausgeführt werden soll. Dies hat zur Folge, dass der Benutzer beim Start der Prozedur `flug_loeschen()` über die notwendigen Berechtigungen – in diesem Fall die `DELETE`-Berechtigungen auf der Tabelle »flug« – verfügen muss.

In Listing 10.17 sehen Sie ein Beispiel unseres Benutzers »webadmin«, der auf der Tabelle »flug« keine Berechtigungen besitzt und dennoch versucht, die Prozedur `flug_loeschen()` auszuführen. In der resultierenden Fehlermeldung wird auch angegeben, bei welcher Anweisung (im Beispiel `DELETE`) ein Berechtigungsproblem aufgetreten ist.

```
DELIMITER $$
CREATE PROCEDURE flug_loeschen()
SQL SECURITY INVOKER
BEGIN
  DELETE FROM FlughafenDB.flug;
END$$
DELIMITER ;
```

Listing 10.16 Beispiel für das Sicherheitskonzept von Programmen

```
mysql> CALL FlughafenDB.flug_loeschen();
ERROR 1142 (42000): DELETE command denied to user
'webadmin'@'localhost' for table 'flug'
```

Listing 10.17 Aufruf einer Prozedur ohne ausreichende Berechtigungen

Die zweite Möglichkeit ist die Einstellung des Sicherheitskonzepts auf die Ausführung unter dem Ersteller des Programms. Diese Option setzen Sie mit der Anweisung `SQL SECURITY DEFINER`, wie in Listing 10.18 in der zweiten Zeile angegeben. Ihnen ist bestimmt schon aufgefallen, dass wir im Beispiel in Listing 10.18 mit der Anweisung `DEFINER` einen konkreten Benutzer angeben. Wie oben beschrieben, ist der `DEFINER` standardmäßig jener Benutzer, der das betreffende Programm erstellt hat. Dies ist der Fall, wenn, wie in Listing 10.16 gezeigt, kein expliziter `DEFINER` angegeben wird. Der `DEFINER` wird darum automatisch von MySQL auf den aktuellen Benutzer, der das Programm erstellt, gesetzt.

Es besteht jedoch die Möglichkeit, den `DEFINER` eines Programms manuell festzulegen und damit die Standardeinstellung (der gerade aktive Benutzer) zu überschreiben. Beachten Sie, dass Sie die globale Berechtigung `SUPER` (siehe Abschnitt 10.2.5, »Berechtigungen zur Administration von MySQL«) benötigen, um den `DEFINER` auf einen anderen Benutzer zu setzen. Andernfalls wäre es möglich, beliebige Anweisungen unter einem beliebigen Benutzer (auch »root« mit allen Rechten) auszuführen.

```
DELIMITER $$
CREATE DEFINER = 'admin' PROCEDURE flug_loeschen()
SQL SECURITY DEFINER
BEGIN
  DELETE FROM FlughafenDB.flug;
END$$
DELIMITER ;
```

Listing 10.18 Erstellen einer Prozedur mit DEFINER

Wenn das Sicherheitskonzept eines Programms wie in Listing 10.16 auf `INVOKER` gesetzt wird, ist der `DEFINER` nicht relevant. Wird jedoch wie in Listing 10.18 die Sicherheitsrichtlinie eines Programms auf `DEFINER` gesetzt, wird das Programm bei jedem Aufruf unter dem gesetzten `DEFINER`-Benutzer ausgeführt. Beachten Sie, dass in diesem Fall bei einer Ausführung immer nur die Berechtigungen von `DEFINER` überprüft werden. Der Benutzer, der das Programm startet, benötigt nur das jeweilige Recht zur Ausführung, zum Beispiel `EXECUTE` bei einer Prozedur. Alle weiteren Rechteüberprüfungen der einzelnen Anweisungen innerhalb eines Programms erfolgen auf den Berechtigungen des `DEFINER`-Benutzers.

Diese Vorgehensweise wird oft zur sogenannten *Kapselung* verwendet. Dabei ist der direkte Zugriff auf die eigentliche Tabelle und auf die Daten nicht möglich. Eine Sicht (View) oder Prozedur kann aber bestimmten Benutzern einzelne Aspekte aus einer Tabelle zur Verfügung stellen.

Betrachten Sie das Beispiel in Listing 10.19. Es geht um einen Benutzer, der aus Sicherheitsgründen die Tabelle »mitarbeiter« bzw. die Spalte »gehalt« nicht lesen darf, aber dennoch Zugriff auf das durchschnittliche Gehalt aller Mitarbeiter im Betrieb erhalten soll. Dies realisieren Sie über die Prozedur `durchschnitt_gehalt()`, die das aktuelle durchschnittliche Gehalt berechnet. Diese Prozedur wird dabei unter dem Benutzer »admin« ausgeführt, der wiederum Lesezugriffe auf die Tabelle »mitarbeiter« besitzt. Sie sehen, dass hier mit sehr einfachen Bordmitteln von MySQL gezielt ausgewählte Daten an Benutzer weitergereicht werden können, ohne dabei dem Benutzer gleich den Zugriff auf den gesamten Datenbestand (in diesem Fall die Mitarbeitertabelle) zu gewähren.

Ein weiteres Beispiel für eine derartige Kapselung mit Sichten zur Realisierung von Berechtigungen auf Zeilenebene finden Sie in Abschnitt 10.3.5, »Workaround für Berechtigungen auf Zeilenebene/Kapselung«.

```
DELIMITER $$
CREATE DEFINER = 'admin' PROCEDURE durchschnitt_gehalt()
SQL SECURITY DEFINER
BEGIN
```

```
  SELECT AVG(gehalt) FROM FlughafenDB.mitarbeiter;
END$$
DELIMITER ;
```

Listing 10.19 Kapselung von Daten

Tipps und Besonderheiten bei der Verwendung von INVOKER und DEFINER

- Bevorzugen Sie, wenn möglich, die Option `SQL SECURITY INVOKER`, da dadurch immer die Rechte des ausführenden Benutzers überprüft werden und nicht erlaubte Zugriffe leichter verhindert werden können.
- Trigger werden immer im Kontext von `DEFINER` aufgerufen. Geben Sie daher, wenn möglich, bei der Erzeugung von Triggern immer einen `DEFINER` an, und achten Sie darauf, dass dieser Benutzer möglichst eingeschränkte Rechte besitzt.
- Geben Sie bei der Verwendung von `SQL SECURITY DEFINER` immer einen `DEFINER` an, der möglichst eingeschränkte Berechtigungen besitzt.

Sie haben in den letzten Abschnitten Absicherungsmaßnahmen für Daten und die darauf operierenden Programme kennengelernt. Im folgenden Abschnitt widmen wir uns nun den Berechtigungen zur Administration und Benutzerverwaltung in MySQL.

10.2.5 Berechtigungen zur Administration von MySQL

Die Berechtigungen zur Verwaltung von Benutzern und sonstiger administrativer Tätigkeiten reichen vom Anlegen von Benutzern über den Zugriff von Dateien im File-System des Servers bis hin zu Replikationsprozessen des MySQL-Servers. In Tabelle 10.3 finden Sie alle Berechtigungen, die im Administrationskontext von MySQL angeboten werden. Im darauffolgenden Abschnitt beschreiben wir alle Berechtigungen im Detail.

Berechtigung	Ebene/Kontext
`GRANT OPTION`	Datenbanken, Tabellen, Prozeduren
`CREATE USER`	Administration/global
`ALL`	Administration/global
`SUPER`	Administration/global
`FILE`	Dateizugriff/Dateisystem
`CREATE TABLESPACE`	Administration/global

Tabelle 10.3 Administrative Berechtigungen

Berechtigung	Ebene/Kontext
USAGE	leeres Recht
PROCESS	Administration/global
RELOAD	Administration/global
REPLICATION_CLIENT	Administration/global
REPLICATION_SLAVE	Administration/global
SHOW DATABASES	Administration/global
SHUTDOWN	Administration/global

Tabelle 10.3 Administrative Berechtigungen (Forts.)

GRANT OPTION

Die Berechtigung GRANT OPTION können Sie auf globaler, Datenbank- oder Tabellenebene setzen. Sie erlaubt dem Benutzer die Verwendung des GRANT-Befehls und damit die Weitergabe oder das Widerrufen von Berechtigungen anderer Benutzer. Beachten Sie, dass Sie immer nur jene Berechtigungen weitergeben oder widerrufen können, die der Benutzer auch selbst besitzt. In Listing 10.20 wird zum Beispiel dem Benutzer »webadmin« die DELETE-Berechtigungen auf allen Tabellen in der Datenbank »FlughafenDB« vergeben. Der Zusatz am Ende des GRANT-Befehls, WITH GRANT OPTION, erlaubt dem Benutzer dabei, diese Berechtigung auch an andere Benutzer weiterzugeben bzw. zu entziehen.

```
mysql> GRANT DELETE ON FlughafenDB.*
    -> TO 'admin' WITH GRANT OPTION;
```

Listing 10.20 Zuweisen einer Berechtigung mit GRANT OPTION

Beachten Sie, dass GRANT OPTION immer über alle Berechtigungen einer Ebene gültig ist. Betrachten Sie das Beispiel in Listing 10.21, in dem zuerst die INSERT-Berechtigung ohne GRANT-Option vergeben wird. In der zweiten Anweisung wird nun die DELETE-Berechtigung mit der GRANT-Option vergeben. Nachdem die GRANT-Option immer auf der gesamten angesprochenen Ebene (in diesem Fall »FlughafenDB.*«) gültig ist, erhält der Benutzer das Recht zur Weitergabe von Berechtigungen nicht nur auf die DELETE-Berechtigung, sondern im Nachhinein auch auf die INSERT-Berechtigung.

```
mysql> GRANT INSERT ON FlughafenDB.*
    -> TO 'admin';
mysql> GRANT DELETE ON FlughafenDB.*
    -> TO 'admin' WITH GRANT OPTION;
```

Listing 10.21 Gültigkeit der GRANT-Option auf ganzer Ebene

Um solche Probleme zu vermeiden, empfehlen wir Ihnen, die Berechtigung `GRANT OPTION` gesondert zu vergeben. Wie in Listing 10.22 angegeben, werden so unerwartete Berechtigungen vermieden, da die Folgen einer gesonderten Operation klarer ersichtlich sind. Zusätzlich sehen Sie in Listing 10.22, wie sie einem Benutzer die Berechtigung `GRANT OPTION` auf der jeweiligen Ebene mit dem Befehl `REVOKE` wieder entziehen können.

```
mysql> GRANT GRANT OPTION ON FlughafenDB.* TO 'admin';
mysql> REVOKE GRANT OPTION ON FlughafenDB.* FROM 'admin';
```

Listing 10.22 Vergabe und Entzug der Berechtigung GRANT OPTION

CREATE USER

Neben der impliziten Erzeugung von Benutzern durch die Anweisung `GRANT` (siehe Listing 10.1), können Sie Benutzer auch dezidiert mit dem Befehl `CREATE USER` anlegen. Die dazu nötige Berechtigung `CREATE USER` wird auf globaler Ebene gesetzt und erlaubt zusätzlich auch noch die Anweisungen `DROP USER`, `RENAME USER`, `REVOKE ALL PRIVILEGES`.

In Listing 10.23 sehen Sie alle Befehle, die mit dieser Berechtigung möglich sind. Die erste Anweisung erstellt einen neuen Benutzer mit dem Namen »admin« und weist ihm das Passwort »testpasswort« zu. Im nächsten Schritt wird der Benutzer in »admin2« umbenannt. Dem umbenannten Benutzer werden anschließend alle Rechte entzogen. Der Benutzer bleibt aber auch ohne Rechte bestehen und könnte sich noch weiterhin mit dem MySQL-Server vom Host `localhost` aus verbinden, jedoch keine Abfragen mehr ausführen. Mit dem letzten Befehl wird der Benutzer nun endgültig und unwiderruflich gelöscht.

Bitte vergeben Sie die Berechtigung `CREATE USER` nur mit Vorsicht, da sie zwar nicht die Vergabe von Berechtigungen, jedoch das unwiderrufliche Löschen aller Benutzer erlaubt. Ein Benutzer könnte daher mit dieser Berechtigung eine MySQL-Installation völlig lahmlegen.

```
mysql> CREATE USER 'admin'@'localhost'
    -> IDENTIFIED BY 'testpasswort';
mysql> RENAME USER 'admin'@'localhost' TO 'admin2'@'localhost';
mysql> REVOKE ALL PRIVILEGES, GRANT OPTION
    -> FROM 'admin2'@'localhost';
mysql> DROP USER 'admin2'@'localhost';
```

Listing 10.23 Mögliche Befehle mit der Berechtigung CREATE USER

ALL

Die Berechtigung `ALL` (Kurzform von `ALL PRIVILEGES`) ist keine eigene Berechtigung, sondern steht für alle Berechtigungen außer `GRANT OPTION`. Setzen Sie zum Beispiel die Berechtigung `ALL` auf eine Datenbank, erhalten Sie in dieser Datenbank alle verfügbaren Berechtigungen auf Datenbankebene. Die Verwendung von `ALL` auf globaler Ebene durch `*.*` (Listing 10.1) bewirkt zum Beispiel das Setzen aller Berechtigungen auf globaler Ebene. Dies beinhaltet auch sehr kritische Berechtigungen, wie zum Beispiel `SUPER`, `ALTER ROUTINE` oder `CREATE USER`. Gehen Sie daher vor allem auf der globalen Ebene (`*.*`) mit der Verwendung der Anweisung `ALL` sehr gewissenhaft um. Wir empfehlen, den `ALL`-Befehl generell zu vermeiden. Der sicherere Weg besteht in der Erstellung eines Benutzers ohne jegliche Rechte und der anschließenden schrittweisen Zuordnung weiterer Berechtigungen.

SUPER

Die globale Berechtigung `SUPER` erlaubt diverse Administrationsanweisungen, die sich auf den gesamten MySQL-Server auswirken können. So ist zum Beispiel das Abbrechen fremder Verbindungen bzw. laufender Befehle mit dem Kommando `KILL` möglich. Zusätzlich können Sie globale Variablen (auch bei aktivierter `read_only`-Option von Systemvariablen) mit dem Befehl `SET GLOBAL` verändern. Die Berechtigung erlaubt auch die Verwendung der Anweisungen `CHANGE MASTER TO` (Ändern der Replikationseinstellungen), `PURGE BINARY LOGS` (Löschen von Log-Dateien) und die Angabe eines beliebigen `DEFINER`-Benutzers bei der Programmierung mit MySQL (siehe Abschnitt 10.2.4, »Das Sicherheitskonzept von Programmen oder DEFINER vs. INVOKER«).

Sollte Ihr MySQL-Server einmal die maximale Anzahl an Client-Verbindungen (siehe Abschnitt »Threads – gleichzeitige Verbindungen« auf Seite 333) erreicht haben, wird den Benutzern mit der Berechtigung `SUPER` trotzdem eine Verbindung zum weiteren administrativen Vorgehen ermöglicht. Diese Berechtigung `SUPER` sollte daher nur an die höchsten Administratoren vergeben werden, da Sie über Umwege die vollständige Kontrolle über eine MySQL-Installation ermöglicht.

FILE

Die globale Berechtigung `FILE` ermöglicht in erster Linie den Umgang mit den Anweisungen `LOAD DATA INFILE`, `SELECT ... INTO OUTFILE` und `LOAD_FILE()`, die zum Import und Export von Daten aus bzw. in das Dateisystem ermöglichen. Die Berechtigung erlaubt dabei das Lesen und Schreiben von allen Dateien im Dateisystem, auf die der MySQL-Server zugreifen darf. Welche Dateien durch den MySQL-Server lesbar oder schreibbar sind, hängt von den Rechteeinstellungen des Dateisystems bzw. des Betriebssystems ab.

Beachten Sie aber, dass die Berechtigung `FILE` auch missbräuchlich verwendet werden kann. Die vom MySQL-Server lesbaren Dateien beinhalten diverse Log-Dateien von MySQL oder auch die Datenverzeichnisse selbst, in denen die eigentlichen Daten der Tabellen liegen. Vergeben Sie daher auch diese Berechtigung nur mit Vorsicht, da die erwähnten Dateien gelesen werden könnten. Beachten Sie auch, dass der MySQL-Server unabhängig von Berechtigungen aus Sicherheitsgründen nicht erlaubt, bestehende Dateien im Dateisystem zu überschreiben.

CREATE TABLESPACE

Die Berechtigung `CREATE TABLESPACE` erlaubt die Verwendung von `CREATE`, `ALTER` und `DROP` in Verbindung mit `TABLESPACE` und `LOGFILE GROUP`. Diese dienen zur Verwaltung der eigentlichen Dateien im Dateisystem, in denen die Daten des MySQL-Servers gespeichert werden. Diese Anweisungen betreffen derzeit jedoch nur den Cluster-Betrieb mit der Storage-Engine NDB.

USAGE

Die Berechtigung `USAGE` dient als Platzhalter und spezifiziert keine bestimmte Berechtigung. `USAGE` wird verwendet, wenn mit dem `GRANT`-Befehl keine Berechtigungen, sondern Optionen, wie zum Beispiel SSL-Einstellungen (siehe Abschnitt 10.4.2, »Zugang beschränken oder verschlüsseln«), am jeweiligen Benutzer vorgenommen werden sollen. In diesem Fall wird das »leere« Recht USAGE verwendet, da der `GRANT`-Befehl immer die Angabe einer Berechtigung benötigt. Weitere Details zum `GRANT`-Befehl finden Sie in Abschnitt 10.2.6, »Die Befehle GRANT und REVOKE im Detail«.

PROCESS

Die Berechtigung `PROCESS` erlaubt Ihnen, alle aktuellen Prozesse am MySQL-Server anzuzeigen. Dies inkludiert auch die laufenden Prozesse anderer Benutzer und deren aktuell ausgeführte SQL-Anweisungen. Beachten Sie, dass das Lesen von fremden SQL-Anweisungen bereits ein großes Sicherheitsrisiko darstellen kann, da diese Anweisungen oft auch sensible Informationen beinhaltet können. Denken Sie etwa an eine SQL-Abfrage, die mit der WHERE-Klausel `WHERE passwort='geheim'` ein Passwort überprüft. Wird diese Abfrage in der Prozessliste angezeigt, kann ein Angreifer das gerade eingegebene Passwort einsehen.

Weitere Informationen zu diesem Sicherheitsproblem finden Sie auch in Abschnitt 10.4, »STOP! – MySQL absichern«. Die Anzeige der Prozesse erfolgt mit der Anweisung `SHOW PROCESSLIST`.

RELOAD

Die Berechtigung `RELOAD` ermöglicht die Verwendung des Befehls `FLUSH`, mit dem Sie diverse Caches in MySQL (siehe Abschnitt 7.2.2) leeren oder optimieren können. So können Sie zum Beispiel mit der Anweisung `FLUSH PRIVILEGES` alle Benutzerberechtigungen neu einlesen. Dieses Kommando wird bei der Ausführung von `GRANT`-Anweisungen automatisch durchgeführt. Andere Anwendungsbereiche sind Logs (`FLUSH LOGS`), diverse Caches im Bereich der Replikation oder auch der Query Cache (`FLUSH QUERY CACHE`) zur Beschleunigung von SQL-Abfragen. Beachten Sie, dass manche `FLUSH`-Anweisungen zusätzliche Berechtigungen benötigen.

REPLICATION CLIENT

Die Berechtigung `REPLICATION CLIENT` verwenden Sie, wie der Name schon sagt, in Replikationsumgebungen. Sie erlaubt Ihnen das Ausführen der Kommandos `SHOW MASTER STATUS` und `SHOW SLAVE STATUS`. Diese Anweisungen zeigen genauere Informationen zum aktuellen Replikationsstatus am Master bzw. am Slave an. Nähere Informationen erhalten Sie in Kapitel 8, »Replikation und Hochverfügbarkeit«.

REPLICATION SLAVE

Die Berechtigung `REPLICATION SLAVE` benötigen Sie zur Replikation selbst. Ein Slave-Server in einer Replikationsumgebung verbindet sich über einen bestimmten Benutzer mit dem Master-Server. Dieser Benutzer muss am Master-Server die Berechtigung `REPLICATION SLAVE` besitzen, damit dieser die nötigen Informationen zur Replikation erhält.

SHOW DATABASES

Die Berechtigung `SHOW DATABASES` erlaubt das Ausführen der gleichnamigen Anweisung `SHOW DATABASES` auf allen Datenbanken. Der Befehl an sich kann grundsätzlich von jedem Benutzer ausgeführt werden, zeigt jedoch nur jene Datenbanken an, in denen der Benutzer eine Berechtigung besitzt. Erhält ein Benutzer das Recht `SHOW DATABASES`, kann er sich alle Datenbanken anzeigen lassen.

SHUTDOWN

Die Berechtigung `SHUTDOWN` erlaubt das Herunterfahren des MySQL-Servers. Diese Aktion können Sie nicht direkt in der MySQL-Kommandozeile ausführen, sondern müssen Sie mit dem Hilfsprogramm *mysqladmin* von außen mit dem Befehl `mysqladmin shutdown` durchführen. Weitere Informationen zum Hilfsprogramm *mysqladmin* finden Sie in Abschnitt 7.2.

10.2.6 Die Befehle GRANT und REVOKE im Detail

Sie haben den Befehl `GRANT` zur Vergabe von Berechtigungen bereits in den vorangegangen Abschnitten kennengelernt. In Listing 10.24 sehen Sie die ausführliche Syntax des `GRANT`-Befehls mit allen Optionen. Die Grundstruktur besteht dabei immer aus einer Berechtigung (siehe Abschnitte 10.2.2, 10.2.3 und 10.2.5), die einem Benutzer auf ein bestimmtes Objekt/eine bestimmte Ebene (siehe Abschnitt 10.2.1, »Die Ebenen der Benutzerberechtigungen«) gewährt wird.

Natürlich ist es auch möglich, mit einem Befehl gleich mehrere Berechtigungen zu vergeben. Diese Berechtigungen können Sie auch gleichzeitig auf mehrere Objekte vergeben und an mehr als einen Benutzer zuweisen. Diese Aufzählungen werden dabei jeweils durch ein Komma getrennt.

Zu den bisher behandelten Optionen des Befehls sind die Klauseln `REQUIRE` und `WITH`, die weitere zusätzliche Optionen zur Verfügung stellen, hinzugekommen. Eine Option der `WITH`-Klausel, die `GRANT OPTION` zur Weitergabe von Berechtigungen, haben Sie bereits im letzten Abschnitt kennengelernt.

Zusätzlich können Sie über die `WITH` Klausel dem Benutzer weitere Einschränkungen auferlegen. Sie können dabei die Anzahl der SQL-Anweisungen (`MAX_QUERIES_PER_HOUR`), `UPDATE`-Anweisungen (`MAX_UPDATES_PER_HOUR`) und Verbindungen (`MAX_CONNECTIONS_PER_HOUR`) pro Stunde begrenzen. Da vor allem die Verwaltung von Verbindungen für den MySQL-Server sehr ressourcenaufwendig ist, können Sie auch die gleichzeitigen Verbindungen pro Benutzer mithilfe der Option `MAX_USER_CONNECTIONS` zusätzlich einschränken.

```
GRANT
    berechtigung [(spaltenliste)]
      [, berechtigung [(spaltenliste)]] ...
    ON [objekttyp] berechtigungsebene
    TO benutzerangabe [,benutzerangabe] ...
    [REQUIRE {NONE | SSLoption [[AND] SSLoption] ...}]
    [WITH zusatzoption ...]

objekttyp:
    TABLE
  | FUNCTION
  | PROCEDURE

berechtigungsebene:
    *
  | *.*
  | bezeichner_datenbank.*
  | bezeichner_datenbank.bezeichner_tabelle
```

```
  | bezeichner_tabelle
  | bezeichner_datenbank. bezeichner_tabelle

benutzerangabe:
  'benutzername'[@'hostname']
  [IDENTIFIED BY [PASSWORD] 'passwort']

SSLoption:
    SSL
  | X509
  | CIPHER 'cipher_text'
  | ISSUER 'issuer_text'
  | SUBJECT 'subject_text'

zusatzoption:
    GRANT OPTION
  | MAX_QUERIES_PER_HOUR numerischer_wert
  | MAX_UPDATES_PER_HOUR numerischer_wert
  | MAX_CONNECTIONS_PER_HOUR numerischer_wert
  | MAX_USER_CONNECTIONS numerischer_wert
```

Listing 10.24 Ausführliche Syntax des GRANT-Befehls

In der Standardeinstellung eines MySQL-Servers ist eine einfache unverschlüsselte Verbindung vom Client zum Server möglich. Dieses Verhalten (`REQUIRE NONE`) können Sie über die Klausel `REQUIRE` ändern. So können Sie zum Beispiel durch die Anweisung `REQUIRE SSL` erzwingen, dass sich der jeweilige Benutzer nur über eine SSL-verschlüsselte Verbindung mit dem Server verbinden kann. Die Option `X509` erzwingt die Benutzer-Authentifizierung über den x509-Standard. Die Angaben von `CIPHER`, `ISSUER` und `SUBJECT` beziehen sich auf weitere Einschränkungen bzw. Einstellungen der jeweils angegebenen Verschlüsselung. Nähere Informationen zur verschlüsselten Kommunikation finden Sie in Abschnitt 10.4.2, »Zugang beschränken oder verschlüsseln«.

Das Gegenstück zum `GRANT`-Befehl stellt der `REVOKE`-Befehl dar, der Ihnen ermöglicht, Berechtigungen wieder zu entziehen. Der `REVOKE`-Befehl unterscheidet sich vom `GRANT`-Befehl nur geringfügig. Seine ausführliche Syntax finden Sie in Listing 10.25, die sich partiell auch auf Teile der `GRANT`-Syntax (Listing 10.24) bezieht.

`REVOKE`

```
berechtigung [(spaltenliste)]
  [,berechtigung [(spaltenliste)]] ...
ON [objekttyp] berechtigungsebene
FROM benutzerangabe [,benutzerangabe] ...
```

```
REVOKE ALL PRIVILEGES, GRANT OPTION
    FROM benutzerangabe [,benutzerangabe] ...
```

Listing 10.25 Ausführliche Syntax des REVOKE-Befehls

In Listing 10.26 finden Sie ein Beispiel beider Befehle. Zuerst wird mit dem GRANT-Befehl ein Benutzer »webadmin« mit dem Passwort geheim angelegt und die Berechtigungen SELECT und UPDATE auf allen Datenbanken, die mit »cms_« beginnen, vergeben. Der zweite Befehl, die REVOKE-Anweisung, entzieht demselben Benutzer die Berechtigungen wieder. Sie können erkennen, dass der Umgang mit REVOKE und GRANT sehr ähnlich ist und Sie diese Befehle beinahe in gleicher Form verwenden können.

```
mysql> GRANT SELECT, UPDATE ON `cms\_%`.* TO 'webadmin'@'%'
    -> IDENTIFIED BY 'geheim';
mysql> REVOKE SELECT, UPDATE ON `cms\_%`.* FROM 'webadmin'@'%';
```

Listing 10.26 Beispiel von GRANT und REVOKE

Der Umgang mit diesen Befehlen führt aber vor allem bei unerfahrenen Administratoren oft zu Verwirrung. Betrachten Sie das Beispiel in Listing 10.27, in dem wiederum die zwei Berechtigungen SELECT und UPDATE auf der Datenbankebene vergeben werden. In der zweiten Zeile wird mit einer REVOKE-Anweisung versucht, die zwei Berechtigungen auf der Ebene *.* zu entziehen.

Nun würde man intuitiv erwarten, dass der Benutzer »webadmin« anschließend keine Berechtigungen mehr besitzt, da eine beispielhafte Datenbank »cms« eigentlich auch in *.* enthalten sein müsste. In MySQL ist dies jedoch nicht der Fall. Berechtigungen werden strikt nach Berechtigungsebenen getrennt.

Durch die Angabe von *.* werden nur Berechtigungen auf globaler Ebene angesprochen. Dies bedeutet, dass ein REVOKE-Befehl auf globaler Ebene nicht überprüft, ob etwaige Berechtigungen auf einer darunterliegenden Ebene existieren. Dem Benutzer werden dementsprechend eventuell gesetzte SELECT- und UPDATE-Berechtigungen auf globaler Ebene entzogen, die Berechtigungen auf der Datenbankebene (Datenbank »cms«) bleiben dem Benutzer aber erhalten.

```
mysql> GRANT SELECT, UPDATE ON cms.* TO 'webadmin'@'%';
mysql> REVOKE SELECT,UPDATE ON *.* FROM 'webadmin'@'%';
```

Listing 10.27 Irreführende Anwendung von REVOKE

Beachten Sie also bei REVOKE immer, dass die Berechtigungsebene genau der Ebene im ursprünglichen GRANT-Befehl entspricht. Stimmen die Ebenen überein, können Sie auch einzelne Berechtigungen entziehen. Listing 10.28 zeigt ein Bei-

spiel, in dem nur das UPDATE-Recht zurückgenommen wird. Diese Anweisung funktioniert, da GRANT und REVOKE jeweils auf der gleichen Berechtigungsebene und auf demselben Benutzer »webadmin@%« operieren.

```
mysql> GRANT SELECT, UPDATE ON cms.* TO 'webadmin'@'%';
mysql> REVOKE UPDATE ON cms.* FROM 'webadmin'@'%';
```

Listing 10.28 GRANT und REVOKE auf derselben Berechtigungsebene

Neben der Berechtigungsebene muss also auch der Benutzer übereinstimmen. Diese Übereinstimmung muss jedoch nicht nur bei dem Benutzernamen gewährleistet sein, sondern auch bei der Hostangabe.

Betrachten Sie das Beispiel in Listing 10.29, in dem zwei Fehler eingebaut sind. Der GRANT-Befehl wird dabei auf der Datenbank »cms« ausgeführt. Der zugehörige Benutzer wird durch den Namen »webadmin« und einen beliebigen Host % angegeben. Der REVOKE-Befehl versucht nun, die UPDATE-Berechtigung auf der Tabelle »cms.user« zu entziehen, wenn sich der Benutzer »webadmin« vom Host localhost aus verbindet. Dies ist aber nur dann möglich, wenn auch bereits einige Rechte genau auf der Tabellenebene »cms.user« zusammen mit dem Benutzer »webadmin« und dem Host localhost gewährt wurden. In diesem Beispiel ist dies jedoch nicht der Fall und der Benutzer hat weiterhin durch die Datenbankebene auf allen Tabellen in der Datenbank »cms« die Berechtigungen SELECT und UPDATE.

Die Realisierung eines Berechtigungskonstrukts, wie zum Beispiel »Berechtigung auf allen Tabellen in der Datenbank ›cms‹ außer der Tabelle ›user‹«, ist nur möglich, wenn alle Berechtigungen einzeln für die Tabellen auf der Tabellenebene vergeben werden.

Beachten Sie daher, dass REVOKE immer nur genau diese Berechtigungen entziehen kann, die auch im Vorhinein vergeben wurde. REVOKE kann keine »intelligenten« Berechtigungen durchführen, wie Sie eventuell von der Anweisung in Listing 10.29 erwarten könnten.

```
mysql> GRANT SELECT, UPDATE ON cms.* TO 'webadmin'@'%';
mysql> REVOKE UPDATE ON cms.user FROM 'webadmin'@'localhost';
ERROR 1141 (42000): There is no such grant defined for user
'webadmin' on host 'localhost'
```

Listing 10.29 Falsche Berechtigungsebene und falscher Benutzer

Gerade bei Benutzern, die man vor längerer Zeit erstellt hat, kann man schon einmal die genauen Berechtigungen (GRANT-Anweisungen, Berechtigungsebenen etc.) vergessen haben. Eine Abänderung der Berechtigungen wird dann schwieri-

ger und führt sehr schnell zu unerwartetem Verhalten. Zu diesem Zweck können Sie sich die Berechtigungen (GRANT-Befehle) eines bestehenden Benutzers anzeigen lassen. In Listing 10.30 sehen Sie ein Beispiel der Berechtigungen des Benutzers »webadmin« von einem beliebigen Host. Die erste Anweisung (GRANT USAGE) dient dabei nur der Zuordnung des Passwortes, da die Berechtigung USAGE (leere Berechtigung) nur als Platzhalter verwendet wird.

```
mysql> SHOW GRANTS FOR 'webadmin'@'%';
+-----------------------------------------------------------------+
| Grants for webadmin@%                                           |
+-----------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'webadmin'@'%' IDENTIFIED BY              |
|   PASSWORD '*94BDCEBE19083CE2A1F959FD02F964C7AF4CFC29'          |
| GRANT SELECT, UPDATE ON `cms\_%`.* TO 'webadmin'@'%'            |
| GRANT EXECUTE, ALTER ROUTINE ON PROCEDURE                       |
|   `FlughafenDB`.`berechne_distanz` TO 'webadmin'@'%'            |
+-----------------------------------------------------------------+
```

Listing 10.30 Beispiel zum Aufruf von SHOW GRANTS

Die restlichen GRANT-Anweisungen entsprechen genau den Berechtigungen, die Sie dem Benutzer zugewiesen haben. Diese Auflistung erleichtert vor allem den Überblick über die verschiedenen Berechtigungsebenen, auf denen der Benutzer Berechtigungen besitzt. Davon ausgehend können Sie auch sehr einfach weitere GRANT- oder REVOKE-Befehle zur Verfeinerung der Berechtigungen des jeweiligen Benutzers ausführen. Sie verlieren so nie den Überblick und sind über die Berechtigungen eines jeden Benutzers bestens informiert.

10.2.7 Benutzerverwaltung ohne GRANT – die Datenbank »mysql«

In den bisherigen Abschnitten haben Sie alle Berechtigungen und die wichtigsten Befehle zur Benutzerverwaltung kennengelernt. Die Anweisungen zur Benutzerverwaltung, wie zum Beispiel der GRANT-Befehl, stellen jedoch nur eine vereinfachte Möglichkeit zur Verwaltung der in MySQL abgespeicherten Benutzerberechtigungen dar. Diese Benutzerberechtigungen speichert MySQL – wie viele andere Datenbanksysteme auch – in speziellen Tabellen. Das Datenbankmanagementsystem kann daher alle Vorteile der ausgereiften relationalen Speicherung auch für »hausinterne Angelegenheiten« nutzen. In MySQL liegen alle benutzerrelevanten Daten in der Datenbank mit dem Namen »mysql«. In diesem Abschnitt gehen wir nun näher auf die Einzelheiten dieser direkten Handhabung der Berechtigungen ein.

In der Systemdatenbank »mysql« befinden sich zahlreiche Systemtabellen, wobei uns hier nur die relevanten Tabellen in Bezug auf die Benutzerverwaltung interessieren. Diese relevanten Tabellen finden Sie im verkürzten Listing 10.31.

```
mysql> USE mysql;
mysql> SHOW TABLES;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| procs_priv                |
| tables_priv               |
| user                      |
+---------------------------+
```

Listing 10.31 Tabellen in der Systemdatenbank »mysql« (Auszug)

Die Tabellen entsprechen dabei den jeweiligen Berechtigungsebenen, wie in Abschnitt 10.2.1, »Die Ebenen der Benutzerberechtigungen«, näher beschrieben. Die Tabelle »user« beinhaltet die Benutzerdaten selbst und die Berechtigungen auf globaler Ebene. Die Tabelle »db« bestimmt die jeweiligen Berechtigungen auf Datenbankebene. Dementsprechend sind die Berechtigungen der Tabellen- und Spaltenebene in den Tabellen »tables_priv« und »columns_priv« gespeichert. Werden auf bestimmte gespeicherte Prozeduren oder Funktionen Berechtigungen gesetzt, werden diese in der Tabelle »procs_priv« abgespeichert. Das »priv« steht dabei für »privileges« (dt. Berechtigungen).

Sie fragen sich vielleicht, warum die Tabellen »user« und »db« kein »priv« beinhalten. Die zwei Tabellen waren im Gegensatz zu den priv-Tabellen jedoch schon in sehr frühen Versionen von MySQL verfügbar und wurden nicht mehr umbenannt. Alle später eingeführten Tabellen, die Berechtigungen beinhalten, wurden mit dem Suffix »priv« versehen.

Werfen wir nun aber einen Blick auf die Struktur dieser Tabellen, die jeweils sehr ähnlich ist. Wie auch beim `GRANT`-Befehl beinhaltet jede Zeile in einer Berechtigungstabelle den Benutzer, das Objekt, für welches die Berechtigungen gelten, und natürlich die gesetzten Berechtigungen selbst. In Listing 10.32 sehen Sie die globale Berechtigungsebene durch eine verkürzte Ausgabe der Tabellenstruktur der Tabelle »user«.

Wie Sie bereits wissen, wird der Benutzer durch die drei Werte User, Host und Passwort definiert. Jede Zeile in der angegebenen Tabelle enthält dabei einen Benutzer, den zugehörigen Host und alle Berechtigungen. User und Host werden

direkt als String-Wert angegeben. Auch die Platzhalter `%` können selbstverständlich direkt verwendet werden. Beachten Sie, dass das Passwort nicht in Klartext abgespeichert wird. Der Wert in der Passwortspalte entspricht einem Hashwert des eigentlichen Passwortes und wird mit der in MySQL vorhandenen Funktion `PASSWORD('geheim')` erzeugt. Ein Beispiel zur manuellen Erstellung eines Benutzers finden Sie auch in Listing 10.39 in Abschnitt 10.3.3, »Benutzerverwaltungs-API«. Beachten Sie, dass der `GRANT`-Befehl diese Funktion intern selbst aufruft. Sie können daher in einer `GRANT`-Anweisung das Passwort in Klartext angeben.

Die Berechtigungen sind in jenen Spalten gespeichert, die mit dem Text »priv« enden. Wird die betreffende Spalte in einer Zeile auf `Y` (steht für »yes«) gesetzt, besitzt der in der Zeile angegebene Benutzer das jeweilige Recht. Der andere mögliche Wert `N` (steht für »no«) gibt an, dass die jeweilige Berechtigung nicht vorhanden ist. Die Namen der Spalten entsprechen dabei der in den Abschnitten 10.2.2, 10.2.3 und 10.2.5 beschriebenen Berechtigungen und können leicht zugeordnet werden.

```
mysql> DESCRIBE user;
+------------------------+---------------+------+-----+---------+
| Field                  | Type          | Null | Key | Default |
+------------------------+---------------+------+-----+---------+
| Host                   | char(60)      | NO   | PRI |         |
| User                   | char(16)      | NO   | PRI |         |
| Password               | char(41)      | NO   |     |         |
| Select_priv            | enum('N','Y') | NO   |     | N       |
| Insert_priv            | enum('N','Y') | NO   |     | N       |
| Update_priv            | enum('N','Y') | NO   |     | N       |
| Delete_priv            | enum('N','Y') | NO   |     | N       |
| Create_priv            | enum('N','Y') | NO   |     | N       |
| Drop_priv              | enum('N','Y') | NO   |     | N       |
| Reload_priv            | enum('N','Y') | NO   |     | N       |
| Super_priv             | enum('N','Y') | NO   |     | N       |
| Trigger_priv           | enum('N','Y') | NO   |     | N       |
| Create_tablespace_priv | enum('N','Y') | NO   |     | N       |
+------------------------+---------------+------+-----+---------+
```

Listing 10.32 Verkürzte Ausgabe der Tabellenbeschreibung der Tabelle »user«

Auf diesen Tabellen können Sie selbstverständlich wie auf jeder gewöhnlichen Tabelle in der üblichen Weise operieren. So kann auch jede `GRANT`-Anweisung durch eine oder mehrere SQL-Anweisungen auf die entsprechenden Tabellen ersetzt werden. Beachten Sie aber, dass bei einer direkten Manipulation der Tabellendaten im Gegensatz zur Verwendung von `GRANT` keine Überprüfungen stattfinden. Sie müssen daher genau wissen, welche Anweisungen und Änderun-

gen Sie durchführen. Andernfalls können ungültige Einträge in den Tabellen entstehen, und die Berechtigungen könnten von MySQL falsch interpretiert werden. Wir empfehlen daher, immer den `GRANT`-Befehl zu verwenden und nur, wenn es nicht anders möglich ist, auf den direkten Umgang mit den Berechtigungstabellen zurückzugreifen.

Ein Beispiel für die Umsetzung einer `GRANT`-Anweisung durch SQL-Anweisungen finden Sie in Listing 10.33. Dabei wird in der ersten Zeile durch die `GRANT`-Anweisung die Vergabe der Berechtigungen `INSERT` und `DELETE` auf globaler Ebene an den Benutzer »admin« vergeben. Wird kein zusätzlicher Host spezifiziert, kann sich der Benutzer von jedem Host aus (`%`) zu Ihrem MySQL-Server verbinden. In der zweiten und dritten Zeile sehen Sie die SQL-Anweisungen zur Vergabe derselben Berechtigungen an den Benutzer. Wir gehen in diesem Beispiel davon aus, dass der Benutzer »admin« bereits erstellt wurde und daher in der Tabelle »user« in der Datenbank »mysql« eine Zeile für den Benutzer »admin« vorhanden ist. Diese Zeile wird nun mit dem `UPDATE`-Befehl aktualisiert und die zugehörigen Spalten auf `Y` gesetzt. Sollten mehrere Zeilen zu diesem Benutzer vorliegen – zum Beispiel für den Host `localhost` und den Host `chef-pc.mein-kleiner-flughafen.at` –, werden die Änderungen in allen Zeilen durchgeführt.

Die Berechtigungen sind anschließend korrekt in der Tabelle gesetzt, jedoch noch nicht aktiv. MySQL lädt aus Geschwindigkeitsgründen beim Start alle Berechtigungen in den Speicher, um diese bei jeder Anfrage schnell überprüfen zu können. Damit MySQL die Berechtigungen nun aktualisiert und die zugehörigen Tabellen aus der Datenbank »mysql« neu einliest, müssen Sie den Befehl `FLUSH PRIVILEGES` aufrufen (Listing 10.33). Sie benötigen dazu die Berechtigung `FLUSH` und weisen MySQL damit an, den Cache aller Berechtigungen zu leeren und neu einzulesen. Auch dieser Schritt entfällt bei der Verwendung von `GRANT`, da der `GRANT`-Befehl diese Aktualisierung des Berechtigungs-Caches automatisch durchführt.

```
mysql> GRANT INSERT, DELETE ON *.* TO 'admin';
mysql> UPDATE mysql.user SET Insert_priv = 'Y', Delete_priv = 'Y'
    -> WHERE User = 'admin';
mysql> FLUSH PRIVILEGES;
```

Listing 10.33 Umsetzung einer GRANT-Anweisung durch SQL-Anweisungen

Neben den globalen Berechtigungen in der Tabelle »user« werden die Rechte auf Datenbankebene in der Tabelle »db« gespeichert. Während in der Tabelle »user« die Zeile durch die Spalten »user« und »host« identifiziert werden (siehe Primary Key in Listing 10.32), wird eine Zeile in der Tabelle »db« durch »user«, »host« und »db« eindeutig identifiziert. Die Spalte »db« beinhaltet dabei den Namen der

Datenbank, auf der die Rechte gesetzt sind. Selbstverständlich können Sie in diesem Feld auch ein Muster von Datenbanknamen, wie in Listing 10.12 gezeigt, angeben. Den resultierenden Eintrag in der Tabelle »db« sehen Sie in gekürzter Form in Listing 10.34.

```
mysql> SELECT * FROM mysql.db \G
*************************** 1. row ***************************
                 Host: %
                   Db: cms\_%
                 User: webadmin
          Select_priv: N
          Insert_priv: N
          Update_priv: N
          Delete_priv: N
          Create_priv: Y
            Drop_priv: Y
           Grant_priv: N
      References_priv: N
```

Listing 10.34 Ausgabe der Tabelle »mysql.db« (Auszug)

Wie bereits erwähnt, haben alle Rechte-Tabellen eine ähnliche Struktur und entsprechen weitestgehend der Struktur in Listing 10.32. Neben der Abbildung von Rechten als Spalten mit den Ausprägungen `Y` oder `N` können mehrere Rechte aber auch in einer Spalte gespeichert werden. Diese Vorgehensweise wird bei den Tabellen »tables_priv«, »columns_priv« und »procs_priv« angewendet. In Listing 10.35 sehen Sie die Struktur der Tabelle »tables_priv«. Die Berechtigungen zu einer Tabelle sind dabei in einer Zeile in der Spalte »Table_priv« gespeichert. Diese Spalte vom Typ `set` kann mehrere durch Kommata getrennte String-Werte aufnehmen.

Vielleicht haben Sie sich schon gefragt, warum auf der Tabellenebene (Tabelle »tables_priv«) eine Spalte »Column_priv« existiert. Diese Spalte dient der Beschleunigung des Rechtesystems und gibt an, ob es noch feinere Berechtigungen auf der darunterliegenden Spaltenebene gibt. Beinhaltet die Spalte »Column_priv« Werte, prüft MySQL auch die Einträge in der Tabelle »columns_priv«. Andernfalls kann MySQL diese Prüfung einsparen, und die Rechteüberprüfung wird somit beschleunigt.

Einen Beispieleintrag der Tabelle »tables_priv« finden Sie in Listing 10.36, in dem der Benutzer »webadmin« die Berechtigungen `CREATE` und `DELETE` auf die Tabelle »cms_log.test« zugewiesen bekommen hat. Beachten Sie, dass Namensmuster mit dem Wildcard-Zeichen `%` nur in der Spalte »db.db« und den Spalten »host« erlaubt sind und in den beschriebenen priv-Tabellen nicht funktionieren.

```
mysql> DESCRIBE mysql.tables_priv;
+-------------+-----------------------------------+------+-----+
| Field       | Type                              | Null | Key |
+-------------+-----------------------------------+------+-----+
| Host        | char(60)                          | NO   | PRI |
| Db          | char(64)                          | NO   | PRI |
| User        | char(16)                          | NO   | PRI |
| Table_name  | char(64)                          | NO   | PRI |
| Grantor     | char(77)                          | NO   | MUL |
| Timestamp   | timestamp                         | NO   |     |
| Table_priv  | set('Select','Insert','Update',   |      |     |
                    'Delete','Create','Drop',     |      |     |
                    'Grant','References','Index', |      |     |
                    'Alter','Create View',        |      |     |
                    'Show view','Trigger')        | NO   |     |
| Column_priv | set('Select','Insert',            |      |     |
                    'Update','References')        | NO   |     |
+-------------+-----------------------------------+------+-----+
8 rows in set (0.00 sec)
```

Listing 10.35 Struktur der Tabelle »tables.priv«

```
mysql> SELECT * FROM mysql.tables_priv \G
*************************** 1. row ***************************
       Host: %
         Db: cms_log
       User: webadmin
 Table_name: test
    Grantor: root@127.0.0.1
  Timestamp: 2011-03-18 14:55:11
 Table_priv: Delete,Create
Column_priv:
```

Listing 10.36 Beispieleintrag der Tabelle »tables_priv«

Die Tabellen »columns_priv« und »procs_priv« sind ähnlich aufgebaut, und Sie können sie daher analog zur Tabelle »tables_priv« verwenden. Eine Übersicht aller Berechtigungstabellen, die MySQL verwendet, finden Sie in Tabelle 10.4

Tabellenname	Inhalt/Berechtigungen	Rechtestruktur
»user«	Benutzerdaten und globale Berechtigungen	ein Recht pro Spalte
»db«	Rechte Datenbankebene	ein Recht pro Spalte

Tabelle 10.4 Übersicht der Berechtigungstabellen in der Datenbank »mysql«

Tabellenname	Inhalt/Berechtigungen	Rechtestruktur
»tables_priv«	Rechte Tabellenebene	SET-Spalte (»Table_priv«)
»columns_priv«	Rechte Spaltenebene	SET-Spalte (»Column_priv«)

Tabelle 10.4 Übersicht der Berechtigungstabellen in der Datenbank »mysql« (Forts.)

Sie haben nun die direkte Möglichkeit kennengelernt, Berechtigungen zu vergeben bzw. zu verwalten. Ob Sie nun den `GRANT`-Befehl verwenden oder lieber direkt auf den jeweiligen Berechtigungstabellen operieren, bleibt natürlich Ihnen überlassen. Der `GRANT`-Befehl führt zwar zur Vermeidung von Fehlern einige Überprüfungen durch, kann diese aber trotzdem nicht ausschließen. Wir empfehlen daher also auch bei Verwendung des `GRANT`-Befehls – vor allem bei komplexeren Rechtekonstrukten – auch die jeweiligen Einträge in den Rechtetabellen zu überprüfen, um eventuell fehlerhafte Konfigurationen ausschließen zu können.

Sie können auch über die Datenbank »INFORMATION_SCHEMA« auf die Rechte zugreifen, die in den jeweiligen Tabellen (zum Beispiel »TABLE_PRIVILEGES«) verfügbar sind. Die Metadaten werden im Hintergrund aber wiederum aus den Systemtabellen der Datenbank »mysql« geladen. Der Vorteil dieser Zugriffsvariante liegt im einheitlichen Zugriff über die Datenbank »INFORMATION_SCHEMA«. Weitere Praxistipps zur Verwaltung von Berechtigungen lernen Sie im nächsten Abschnitt kennen.

10.3 Tipps und Tricks

In diesem Abschnitt gehen wir auf Best-Practice-Vorgehensweisen im Sicherheitsbereich ein, die das Leben eines Administrators auf längere Sicht stark vereinfachen, Fehler vermeiden helfen und die Sicherheit des Systems erhöhen können. Zusätzlich zeigen wir Ihnen mögliche Lösungen für einige herausfordernde Szenarien, damit Sie ein Gefühl dafür entwickeln, mit welchen Tricks man in MySQL arbeiten kann. Eventuell treffen auch Sie auf derartige Szenarien oder können die beschriebenen Lösungswege für Ihre Herausforderungen adaptieren.

10.3.1 Vorgehen bei der Erstellung von Benutzern

Wir haben Ihnen in den vorangegangenen Abschnitten ausführlich die Berechtigungen und die zugehörigen Befehle zur Verwaltung gezeigt. Im ersten Kontakt mit MySQL reicht es natürlich völlig aus, dass Sie Benutzer einfach erstellen und mit den Berechtigungen experimentieren. In Produktivumgebungen ist ein »Experimentieren« jedoch nicht mehr anzuraten. Allgemein sollten Sie einige Punkte bei

der Erstellung von neuen Benutzern in produktiven Umgebungen beachten, um die Sicherheit in Ihrer MySQL-Installation ständig gewährleisten zu können:

- Erstellen Sie einen Benutzer zu Beginn immer ohne jegliche Berechtigungen, wie in Listing 10.37 angegeben.
- Erlauben Sie, wenn dies möglich ist, nur die lokale Verbindung mit dem MySQL-Server (Host `localhost`), wie in Listing 10.37 angegeben.
- Wenn der Zugriff über das externe Netzwerk gestattet wird, grenzen Sie den `Host` ein, und erlauben Sie nicht automatisch alle Hostnamen (`%`).
- Vergeben Sie schrittweise so lange Berechtigungen, bis diese für den Benutzer ausreichend sind.
- Vermeiden Sie die Berechtigung `ALL`, und geben Sie alle Berechtigungen manuell an.
- Versuchen Sie, Berechtigungen möglichst auf Datenbank- oder sogar Tabellenebene zu vergeben, und vermeiden Sie die Vergabe von Rechten auf globaler Ebene.

```
mysql> CREATE USER 'benutzername'@'localhost';
```

Listing 10.37 Benutzer ohne Berechtigungen erstellen

Wenn Sie diese Regeln befolgen, werden Ihre Benutzer immer die minimalen Berechtigungen erhalten und Sie somit die größte mögliche Sicherheit. Natürlich ist diese Art der Rechtevergabe wesentlich aufwendiger als eine einfache Zuweisung der `ALL`-Berechtigung auf globaler Ebene. Eine Sicherheitslücke durch falsche Berechtigungen bedeutet für Sie als Administrator jedoch meist einen wesentlich höheren Zeitaufwand und mehr Arbeit. Wir empfehlen daher, bei der Sicherheit nicht zu sparen und ausreichend Zeit dafür zu investieren. Wie Sie dagegen Zeit einsparen und auch in sehr großen Installationen bei Tausenden Benutzern noch den Überblick bewahren können, zeigen wir Ihnen im folgenden Abschnitt.

10.3.2 Tausende Benutzer und Datenbanken ohne Chaos verwalten

Je größer MySQL-Installationen werden, desto mehr Benutzer müssen Sie als Administrator auch verwalten. Oft beinhalten MySQL-Installationen Hunderte von Benutzern und Tausende Tabellen. Denken Sie an einen Anbieter von Webspace im Internet, der auf einem Datenbankserver oft Tausende Internetdomains und ihre dazugehörigen Datenbanken verwaltet. Hier verliert man als Administrator sehr schnell den Überblick über alle Benutzer und Datenbanken. Leider verfügt MySQL derzeit über keine Gruppierungsmöglichkeiten bei Datenbanken

oder Benutzern. Es liegt also bei Ihnen, die Verwaltung von Benutzern und Datenbanken sinnvoll und übersichtlich zu gestalten.

Wir empfehlen Ihnen daher, sich zuerst ein *Sicherheits- und Benutzerkonzept* zurechtzulegen. Dieses Konzept sollte vor allem Namenskonventionen für Benutzer und Datenbanken beinhalten. Natürlich können wir hier keine Musterlösung für jedes mögliche Szenario anbieten, jedoch können Sie im folgenden Beispiel sehen, wie ein mögliches Verwaltungskonzept aussehen könnte. Basierend auf diesen Informationen können Sie dann ein Konzept für Ihren speziellen Anwendungsfall – maßgeschneidert auf Ihre Bedürfnisse als Administrator – entwickeln.

Gehen wir beispielhaft von dem Szenario eines großen Web-Hosters aus, der seinen Kunden Webspace für eine Internetseite bzw. Domain und eine dazugehörige MySQL-Datenbank anbietet und der auch die Website *www.meinkleinerflughafen.de* hostet. Jede Domain erhält dabei einen MySQL-Benutzer, der auch Datenbanken selbst anlegen darf. Da auf einem Datenbanksystem meist auch andere Benutzer existieren, zum Beispiel für interne Verwaltungsprogramme, sollten die Kunden durch ein Präfix gekennzeichnet werden. Zum Beispiel könnten Benutzerkonten für Kunden immer nach dem Schema www_kundenummer oder www_domainname benannt werden. Beachten Sie, dass die Länge von MySQL-Benutzernamen auf 16 Zeichen beschränkt ist. In diesem Fall wäre eine Kürzung des Domainnamens auf 8 Zeichen sinnvoll. An diesen Namen wird dann noch eine dreistellige fortlaufende Nummer gehängt. Der Benutzer der Domain *www.meinkleinerflughafen.de* würde damit www_meinklei001 lauten, und Sie hätten ausreichend Benutzernamen für alle benötigten Kunden.

Wie in den Anforderungen beschrieben, sollen die Benutzer selbstständig Datenbanken anlegen und verwalten können. Damit es zu keinen Überschneidungen oder Namenskonflikten kommt, erhält jeder Benutzer einen eigenen Namensraum zur Verfügung gestellt. Dies kann mit einem einfachen Präfix für die Datenbanken des jeweiligen Benutzers erreicht werden. Dieses Präfix kann nun wiederum dem Benutzernamen entsprechen, um eine direkt ersichtliche Verbindung zwischen jeder Datenbank und dem zugehörigen Benutzer herzustellen. In Listing 10.38 sehen Sie, wie Sie dem Benutzer »www_meinklei001« alle Berechtigungen für seinen Namensraum »meinklei001_« zuweisen. Da der _ in MySQL als Platzhalter für ein beliebiges Zeichen steht, müssen wir den Unterstrich »escaped« (maskiert) als _ angeben.

```
mysql> GRANT ALL PRIVILEGES ON `meinklei001\_%`.*
    -> TO 'www_meinklei001';
```

Listing 10.38 Zuweisung des Namenraums für einen Benutzer

Der Benutzer »www_meinklei001« kann nun nur Datenbanken, die mit dem Text »meinklei001_« beginnen, erstellen und verwalten. Datenbanknamen können in MySQL bis zu 64 Zeichen lang sein und schränken den jeweiligen Benutzer daher auch mit einem derartigen Präfix in der Namensgebung kaum ein. Jeder Benutzer arbeitet so in seinem eigenen Unterbereich der Datenbank, der durch das Präfix definiert ist. Sie können derartige Präfixe selbstverständlich auch für andere Zwecke, wie etwa das Einteilen in Gruppen oder Abteilungen, nutzen und so die fehlende Unterstützung in diesem Bereich durch MySQL einfach und übersichtlich kompensieren. Durch die Verwendung des Benutzernamens als Präfix ist zusätzlich die Verbindung zwischen den jeweiligen Datenbanken und den zugehörigen Benutzern sofort erkennbar. Sie sehen also, dass sich die Planung und Erstellung eines Sicherheitskonzepts bzw. einer Namenskonvention gerade bei großen Installationen bewährt. Die investierte Zeit für das Konzept macht sich durch die stark vereinfachte Verwaltung im laufenden Betrieb schnell bezahlt.

10.3.3 Benutzerverwaltungs-API

In sehr großen Installationen mit sehr vielen Benutzern, wie im letzten Abschnitt erwähnt, werden oft sogar externe Programme zur Benutzerverwaltung eingesetzt. Diese externen Programme erstellen und verwalten die Benutzer und zugehörigen Gruppen selbst, müssen aber selbstverständlich auf das Berechtigungssystem in MySQL zugreifen. In der Softwareentwicklung spricht man hierbei oft von sogenannten APIs (*Application Programming Interfaces*), Schnittstellen, die einen gekapselten Zugriff auf Daten oder Funktionen in Programmen erlauben. MySQL stellt für externe Programme keine eigene Berechtigungs-API zur Verfügung. Dies hat jedoch einen sehr guten Grund, denn MySQL bietet eine viel einfachere Möglichkeit, auf Berechtigungen zuzugreifen. Sie können für diesen Zweck den klassischen Datenbankzugriff auf SQL-Ebene, der beinahe mit jeder Programmiersprache möglich ist, verwenden.

Wie Sie bereits in Abschnitt 10.2.7, »Benutzerverwaltung ohne GRANT – die Datenbank ›mysql‹«, gelernt haben, speichert MySQL alle Berechtigungen in Tabellen, auf die auch mit üblichen SQL-Anweisungen zugegriffen werden kann. Ein externes Programm kann also auf den Berechtigungstabellen wie auf normalen Datentabellen operieren und die Berechtigungen verwalten und verändern. Um einen Benutzer zu erstellen, muss ein externes Verwaltungsprogramm also nur die in Listing 10.39 angeführten Anweisungen ausführen:

```
mysql> INSERT INTO mysql.user (User,Host,Password)
    -> VALUES ('admin', '%', password('geheim'));
mysql> FLUSH PRIVILEGES;
```

Listing 10.39 Manuelles Erstellen eines Benutzers

Das Erlernen von und Programmieren mit einer neuen API entfällt also vollständig, und Sie können mit bestehenden Mitteln auch in großen Installationen direkt auf Ihre Benutzer und Berechtigungen zugreifen und diese verwalten. Beachten Sie aber, dass Sie den zusätzlichen Schutz der `GRANT`-Anweisung verlieren, da Ihre SQL-Anweisungen nicht überprüft werden. Verwenden Sie diese Methode also nur, wenn die automatische Erzeugung von `GRANT`-Befehlen aus technischen Gründen nicht möglich ist. Nähere Details zur Rechteorganisation und Struktur dieser Berechtigungstabellen finden Sie in Abschnitt 10.2.7.

10.3.4 Benutzerverwaltung in der Applikation, nicht in der Datenbank?

Die meisten modernen Applikationen bringen heutzutage ihr eigenes Berechtigungssystem mit. Dies bedeutet, dass zwar die Daten in der Datenbank gespeichert werden, die Sicherheitsüberprüfungen aber in der Applikation ablaufen. Die Applikation verbindet sich daher immer mit demselben Benutzer zur Datenbank und legt in der Applikation selbst eine zusätzliche Sicherheitsschicht über die Datenbank. In dieser Sicherheitsschicht wird dann überprüft, welcher Benutzer welche Daten einsehen, ändern oder löschen darf. Die eigentlichen Endbenutzer sind daher dem Datenbanksystem gar nicht mehr bekannt und werden von der Applikation vor dem Datenbanksystem »versteckt«.

Der Grund für dieses Vorgehen ist sehr einfach zu erklären. Jede Applikation hat eigene Anforderungen an eine Benutzerverwaltung und die Granularität an Berechtigungen. Oft sind diese Anforderungen nur schwer oder sogar gar nicht in Datenbanken umsetzbar. Denken Sie an eine Buchung in unserer Flughafendatenbank. Nehmen Sie an, ein Mitarbeiter einer Fluggesellschaft darf die Personendaten der Buchungen seiner Flughafengesellschaft einsehen. Selbstverständlich soll dieser Mitarbeiter aber nicht alle Personendaten im gesamten System einsehen können. Die Berechtigung auf der Tabelle »passagier« wäre daher pro Zeile abhängig von einer eventuell bestehenden Buchung eines Flugs der Fluglinie des Mitarbeiters. Eine derartige Berechtigungskonstruktion innerhalb der Datenbank wäre nur sehr schwer zu realisieren. In der Applikation können derartige Konstrukte im Normalfall wesentlich leichter umgesetzt werden.

Zusätzlich müssen moderne Applikationen mit möglichst vielen Datenbanken der unterschiedlichsten Hersteller zusammenarbeiten können. Die meisten Datenbanksysteme »sprechen« zwar SQL, doch unterscheiden sie sich gerade beim Berechtigungssystem in sehr großen Teilen. Es ist daher einfacher, die Sicherheitsüberprüfungen in der Applikation zu programmieren und nur die Benutzerdaten bzw. Applikationsberechtigungen in der Datenbank in Tabellen abzuspeichern.

Diese Vorgehensweise zahlreicher Applikationen sorgt immer wieder für zündenden Diskussionsstoff innerhalb der Community. Wir werden uns aber an dieser Diskussion nicht beteiligen und möchten Ihnen nur einen wichtigen Tipp für den Einsatz der oben beschriebenen Applikationen geben.

Wie erwähnt verwendet die Applikation nur einen Datenbankbenutzer, über den alle Datenmanipulationen abgehandelt werden. Oftmals wird hier ein hochprivilegierter Benutzer oder sogar der Benutzer »root« verwendet. Gerechtfertigt wird diese Vorgehensweise damit, dass die Applikation die Berechtigungen überprüft und keine unerlaubten Zugriffe gestattet. Doch kennen Sie eine Software, die keine Fehler hat und unhackbar ist? Gerade bei Web-Applikationen wird hier oft fahrlässig gehandelt. Wird eine Web-Applikation von außen geknackt, kann der Angreifer über den hochprivilegierten Benutzer auf die Datenbank zugreifen und großen Schaden anrichten.

Eine Gegenmaßnahme ist bereits im Vorfeld ohne großen Aufwand möglich. Vergeben Sie jeder Applikation, die nur einen Datenbankbenutzer verwendet, nur die allernötigsten Berechtigungen. Wägen Sie auch ab, ob der jeweilige Benutzer für die Applikation überall Schreibzugriffe benötigt. Oftmals reichen auch Leserechte für bestimmte Bereiche aus.

Überdenken Sie auch andere Berechtigungen, wie `CREATE` oder `DROP`. Diese Berechtigungen sind meist nur für die Installation einer Applikation vonnöten. Im laufenden Betrieb erzeugen die meisten Applikationen eher selten neue Tabellen. Viele Applikationen bieten hierfür auch die Möglichkeit zur Benennung eines speziellen Benutzers für die Installationsphase und eines Benutzers für den laufenden Betrieb. Machen Sie von solchen Angeboten unbedingt Gebrauch, da diese bereits große Sicherheitslücken stopfen können. Wird zum Beispiel eine Web-Applikation angegriffen und die Kontrolle übernommen, sperrt die Sicherungsschicht auf Datenbankebene zumindest andere Bereiche der Datenbank und grenzt so den Schadensbereich auf ein Minimum ein.

Beachten Sie daher, dass eine Berechtigungskontrolle auf Applikationsebene die Sicherheitsmaßnahmen auf Datenbankebene nicht ersetzen kann und die Grundregel »nur die nötigsten Berechtigungen« in diesem Szenario weiterhin ihre Gültigkeit bewahrt.

Wie Sie sehr komplexe und verstrickte Sicherungskonzepte, wie Sie oft in Applikationen realisiert werden, auch auf der Datenbankebene umsetzen können, erfahren Sie im nächsten Abschnitt.

10.3.5 Workaround für Berechtigungen auf Zeilenebene/Kapselung

Wie in Abschnitt 10.2.1, »Die Ebenen der Benutzerberechtigungen«, erwähnt, verfügt MySQL derzeit über keine Möglichkeit, Berechtigungen auf Zeilenebene zu setzen.

Betrachten Sie das Beispiel der Mitarbeiter und deren Gehälter. Ein Abteilungsleiter darf unter Umständen die Gehälter seiner Abteilung einsehen, jedoch auf keinen Fall auf die Gehälter anderer Mitarbeiter zugreifen. Dieses Szenario lässt sich im ersten Moment mit den klassischen Berechtigungen und Sicherheitsmechanismen in MySQL nicht umsetzen, da hier Berechtigungen auf Zeilenebene benötigt werden und diese von MySQL nicht direkt angeboten werden. Es gibt jedoch auch Lösungen bzw. Workarounds, um sogar diese Herausforderung zu bewältigen. Alle benötigten Techniken haben Sie bereits kennengelernt, und Sie müssen sie nur noch in der richtigen Weise kombinieren.

Für dieses spezielle Szenario möchten wir Ihnen zwei mögliche Lösungswege vorstellen, die alle auf dem Kapselungsprinzip basieren. Von einer Kapselung spricht man, wenn der Zugriff auf die eigentlichen Daten durch eine Kapselungsschicht in einer vereinfachten Form erfolgt. Der Benutzer erhält dadurch eine vereinfachte Sicht bzw. Zugriffsmöglichkeit auf die Daten. Dies können Sie durch Sichten oder gespeicherte Prozeduren realisieren, die in Kapitel 11, »Gespeicherte Programme und Events«, vorgestellt werden.

Realisierung mit gespeicherten Prozeduren

Wie Sie bereits in Abschnitt 10.2.4, »Das Sicherheitskonzept von Programmen oder DEFINER vs. INVOKER«, kennengelernt haben, eignen sich Prozeduren sehr gut für die *Kapselung* bzw. das *Information Hiding*. Der Zugriff auf die Daten bzw. eine Tabelle ist dabei nicht direkt möglich. Der jeweilige Benutzer kann nur über vordefinierte Prozeduren auf die Daten in eingeschränkter Form zugreifen. Dieses Vorgehen lässt sich natürlich auch für dieses Szenario gut nutzen. Aufgabe der Prozedur ist es dabei, die jeweiligen Zeilen bzw. Mitarbeiter anderer Abteilungen zu filtern und dem jeweiligen Benutzer nur die erlaubten Daten auszuliefern.

Eine einfache Umsetzung zeigt Listing 10.40, in dem die Prozedur `proc_mitarbeiter_marketing` durch die angegebene SQL-Anweisung nur die Mitarbeiterdaten der Marketingabteilung zurückliefert. Durch die Anweisung `SQL SECURITY DEFINER` wird die Prozedur immer unter dem Benutzer »admin« ausgeführt, der die Zugriffsrechte auf die eigentliche Tabelle »mitarbeiter« benötigt. Der Abteilungsleiter der Abteilung Marketing benötigt daher nur die Ausführungsrechte auf die Prozedur, jedoch nicht auf die Tabelle »mitarbeiter«.

```
mysql> CREATE DEFINER = 'admin'
    -> PROCEDURE proc_mitarbeiter_marketing()
    -> SQL SECURITY DEFINER
    -> SELECT * FROM mitarbeiter WHERE abteilung = 'Marketing';
```

Listing 10.40 Prozedur zur Kapselung des Zugriffs auf Mitarbeiter

Der Nachteil dieser Umsetzung ist die Beschränkung auf das Lesen der Daten. Mit der in Listing 10.40 angegebenen Prozedur ist es dem Abteilungsleiter nicht möglich, die Daten seiner Mitarbeiter zu ändern. Sie müssten daher eine weitere Prozedur erstellen, die als Parameter die jeweiligen Daten zur Änderung übernimmt und dann die Änderungen auf der eigentlichen Tabelle durchführt. So können sehr schnell sehr viele Prozeduren entstehen, und der Verwaltungsaufwand wird dadurch stark erhöht. Eine flexiblere Lösung, basierend auf Sichten, stellen wir Ihnen im Folgenden vor.

Realisierung mit Sichten (Views)

Wie der Name dieser Vorgehensweise beschreibt, wird in diesem Lösungsansatz eine spezielle Sicht auf die Daten erstellt. Diese Sicht versteckt alle Datensätze bzw. in diesem Fall Gehälter aus anderen Abteilungen, die aufgrund sicherheitstechnischer Gründe nicht zugänglich sein sollen. Der Benutzer hat auf der einen Seite keine Zugriffsrechte auf die Mitarbeiter-Tabelle, erhält aber auf der anderen Seite die Zugriffsrechte auf die erstellte Sicht. Diese *Sicht*, oder auch *View* genannt, kann wie eine übliche Tabelle angesprochen werden. Die Tabelle enthält selbst aber keine Daten, sondern greift im Hintergrund auf die Originaltabelle – in diesem Fall »mitarbeiter« – zu und leitet Anfragen an diese Tabelle weiter. Darum spricht man auch oft von »virtuellen« Tabellen. Weitere Details zu Sichten finden Sie in Abschnitt 11.8.

In Listing 10.41 sehen Sie ein Lösungsbeispiel für unser Szenario mit einer Sicht, die nur die Mitarbeiter der Marketingabteilung zur Verfügung stellt. Eine derartige Lösung mit Sichten benötigt aber auch eine sehr genaue Analyse der Berechtigungen. Besonders hier können sich sehr leicht Fehler einschleichen. Beachten Sie daher, dass der `DEFINER` (im Beispiel admin) die ausreichenden Berechtigungen für die Originaltabelle – in diesem Fall »mitarbeiter« – besitzt und Sie die Sicherheitseinstellung `SQL SECURITY DEFINER` angeben. Der eigentliche Zugriff auf die Originaltabelle wird dadurch mit dem angegebenen `DEFINER`-Benutzer durchgeführt. In der `WHERE`-Klausel wird spezifiziert, welche Daten in der Sicht sichtbar sein sollen. In diesem Beispiel sind dies alle Zeilen, die die Ausprägung *Marketing* in der Spalte »abteilung« aufweisen.

```
mysql> CREATE DEFINER = 'admin' SQL SECURITY DEFINER
    -> VIEW mitarbeiter_marketing AS
    -> SELECT * FROM mitarbeiter WHERE abteilung = 'Marketing'
    -> WITH CHECK OPTION;
```

Listing 10.41 Erzeugen der Sicht zur Filterung der Marketingmitarbeiter

Beachten Sie auch unbedingt die unerlässliche Angabe `WITH CHECK OPTION`. Diese Option garantiert auch bei sogenannten *aktualisierbaren Sichten* das sichere Arbeiten. Aktualisierbare Sichten erlauben `UPDATE`-, `DELETE`- und `INSERT`-Operationen und leiten diese an die darunterliegende Tabelle weiter. Aus diesem Grund müssen Sie unbedingt sicherstellen, dass nur Daten gelöscht, geändert oder hinzugefügt werden, die auch der Sicht bzw. deren `WHERE`-Klausel entsprechen. In diesem Fall können also nur Mitarbeiter der Abteilung Marketing aktualisiert oder eingefügt werden.

In Listing 10.42 sehen Sie einen Update-Versuch auf einen Mitarbeiter mit der `mitarbeiter_id=2`, der zwar in der Originaltabelle »mitarbeiter« vorhanden ist, jedoch nicht in der Abteilung Marketing arbeitet. Wie Sie sehen, werden keine Zeilen (`Changed: 0`) aktualisiert, da dieser Mitarbeiter von der Sicht auch bei einer `UPDATE`-Anweisung durch die Option `WITH CHECK` ordnungsgemäß »versteckt« wird.

```
mysql> UPDATE mitarbeiter_marketing SET gehalt = 0
    -> WHERE mitarbeiter_id = 2;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

Listing 10.42 Update-Versuch auf eine Zeile, die nicht in der Sicht vorhanden ist

Versuchen wir nun, einen neuen Mitarbeiter in der Abteilung Buchhaltung über die Sicht einzufügen (Listing 10.43), erhalten wir die Fehlermeldung, dass `CHECK OPTION` – also die Bedingung in der `WHERE`-Klausel – nicht erfüllt ist und es daher nur möglich ist, Mitarbeiter aus der Abteilung Marketing einzufügen.

```
mysql> INSERT INTO mitarbeiter_marketing
    -> (mitarbeiter_id,abteilung) VALUES (100,'Buchhaltung');
ERROR 1369 (HY000): CHECK OPTION failed
'FlughafenDB.mitarbeiter_marketing'
```

Listing 10.43 Einfügen von ungültigen Werten in eine Sicht

Wichtig ist auch, dass der Benutzer, der auf dieser Sicht arbeitet, selbstverständlich keine Berechtigungen auf der Originaltabelle haben soll. Nur dann funktioniert das ordnungsgemäße Verstecken (Kapseln) von Daten. Sie sehen, dass diese

Variante sehr elegant ist und trotzdem alle Möglichkeiten – sogar `UPDATE`- und `INSERT`-Operationen – auf die eingeschränkte Sicht erlaubt.

Sie haben aber gelernt, dass Sie auf zahlreiche Kleinigkeiten achten müssen, bei denen sich oft Fehler einschleichen, die dann unter Umständen zu großen Sicherheitslücken führen. Entwerfen Sie Ihr Sicherheitskonzept daher mit möglichst viel Sorgfalt, und überprüfen Sie es auch mit diversen Abfragen und von Ihnen angelegten Testbenutzern. Nutzen Sie für diese Tests sowohl Abfragen, die korrekt durchgeführt werden sollen, wie auch Abfragen, die nicht erlaubt sind und mit einer Fehlermeldung abgebrochen werden sollen. Nur durch beide Arten von Abfragen können Sie Ihr Rechtesystem ausreichend testen.

10.4 STOP! – MySQL absichern

Im vorangegangen Abschnitt haben Sie das Sicherheitskonzept und die zugehörigen Berechtigungen innerhalb des MySQL-Servers kennengelernt. In diesem Abschnitt beschäftigen wir uns mit der Sicherheit rund um den Server, der Verbindung nach außen über das Netzwerk, allgemeinen Sicherheitsmaßnahmen und dem sehr häufigen Anwendungsfall »MySQL im Web«.

10.4.1 Allgemeine Sicherheitshinweise – der gewissenhafte Administrator

Wie Sie bereits gesehen haben, bietet MySQL zahlreiche Funktionen, um die Sicherheit in Ihrer MySQL-Installation zu erhöhen. Zahlreiche andere Faktoren – vor allem der Faktor Mensch – beeinflussen aber auch die Sicherheit Ihres Systems. Sie sollten daher auch die Sicherheit auf anderen Ebenen analysieren und Sicherheitslücken gegebenenfalls beseitigen oder schon im Vorfeld verhindern. Im Folgenden zeigen wir Ihnen einige Best-Practice-Vorgehensweisen, die ihren MySQL-Server zusätzlich absichern.

Sicherheitslücke Administrator

Beginnen wir mit dem Faktor Mensch der Administratorenseite. Sie verwalten als Administrator zahlreiche Server und sichern diese mit höchster Sorgfalt ab. Doch wie oft aktualisieren, analysieren und überprüfen Sie Ihre Benutzerkonten, Gruppen und Sicherheitseinstellungen? Haben Sie die Benutzerkonten der ehemaligen Mitarbeiter alle gelöscht oder deaktiviert? Hat ein Mitarbeiter vielleicht einmal die Abteilung gewechselt und hat nun Zugriff auf die alte und neue Abteilung? Dies sind nur einige Beispiele, die Ihnen als Administrator eines MySQL-Servers tagtäglich begegnen können und die Sie vermeiden sollten.

In Kapitel 11, »Gespeicherte Programme und Events«, lernen Sie Trigger kennen, die im Datenbanksystem automatisch auf Aktionen, zum Beispiel das Einfügen einer neuen Zeile, reagieren können. In der Realität müssen Sie als Administrator auf Aktionen selbst reagieren und bestimmte Aufgaben durchführen. Kleine Hilfsmittel, die Sie vielleicht auch bereits benutzen, können das Leben eines Administrators aber erleichtern und Datenbank-Trigger in der Realität ermöglichen.

Eine sehr einfache, aber umso effektivere Methode ist die Definition von sogenannten *Workflows*, die einen Ablauf – angestoßen durch eine reale Aktion – beschreiben. Ein Ablauf könnte zum Beispiel der Austritt eines Mitarbeiters sein. Solche Workflows sollen für Sie eine Art Checkliste sein, die Sie zum Beispiel beim Austritt eines Mitarbeiters Schritt für Schritt abarbeiten können. Ein mögliches Beispiel dazu sehen Sie in Tabelle 10.5. Derartige einfache Auflistungen in beliebiger Form erleichtern Ihnen regelmäßige Tätigkeiten und verhindern das Entstehen von »Systemleichen« oder anderen ungültigen Altbeständen. Für solche Workflows oder Aktionsbeschreibungen eignen sich firmeninterne Wiki-Systeme sehr gut, da diese dann auch im Team verwaltet werden können und Sie somit automatisch über eine gewisse Dokumentation verfügen. Selbstverständlich sind die beste Dokumentation und die Vordefinition von Abläufen nur dann wirksam, wenn Sie diese Hilfestellungen auch bei Bedarf zu Rate ziehen.

Ein anderes häufiges Problem ist die Informationspolitik in Unternehmen. Wenn Sie als Administrator nicht über allgemeine Veränderungen, wie etwa den Austritt eines Mitarbeiters, informiert werden, können Sie auch nicht darauf reagieren. Versuchen Sie daher auch diese Informationskanäle zu nutzen bzw. gegebenenfalls zu optimieren, da ein daraus resultierendes Sicherheitsproblem immer auf den Administrator zurückfällt.

Kontext	Aktion	Kommentar
Datenbank	Benutzer löschen	Berechtigungen überprüfen
Datenbank	Kontaktdaten archivieren	Backup der Daten für Rückfragen an den Mitarbeiter
CMS	Kontaktdaten auf Homepage entfernen	Online-Inhalte aktualisieren

Tabelle 10.5 Workflow der Aktionen beim Austritt eines Mitarbeiters

Sie sehen, dass ein Administrator neben den Kernaufgaben, der Verwaltung der Serversysteme, auch das Umfeld gut im Auge behalten muss, um die größtmögliche Sicherheit gewährleisten zu können.

Sicherheitslücke »Benutzer und ihre Anwendungen«

Sie haben als Administrator tagtäglich mit Benutzern und deren Anwendungen zu »kämpfen«, durch die viele Sicherheitslücken entstehen. Einige grundlegende Tipps zu diesem Thema möchten wir im Folgenden besprechen.

Selbstverständlich müssen wir Ihnen als Administrator nicht erklären, dass Passwörter nicht nur aus Buchstaben, sondern auch aus Sonderzeichen und Zahlen bestehen und in keinem Wörterbuch gefunden werden sollen. Auch die Sicherheitslücke »PostIt« – das Notieren von Passwörtern auf Klebezetteln, die anschließend auf den Monitor geklebt werden – ist Ihnen vermutlich hinreichend bekannt. Diese Probleme sind vielen Anwendern jedoch nicht bewusst. Wir empfehlen Ihnen daher, Ihre Benutzer über diese Problematik regelmäßig zu informieren, sie aufzuklären und zu sensibilisieren. Leider zeigen viele reale Fälle, dass der Faktor Mensch noch immer eine der größten Sicherheitslücken darstellt.

Aber auch die »fremden« Anwendungen Ihrer Benutzer, die Ihre Datenbank benutzen, stellen natürlich ein Sicherheitsrisiko dar. Gleichzeitig kann die unzureichende Absicherung der Datenbank ein Risiko für die Daten der Anwendung darstellen. Um eine möglichst hohe Sicherheit zu erreichen, sollten Sie die folgenden Punkte beachten:

Anwendungen mit einer eigenen Benutzerverwaltung sollten die Passwörter der Benutzer nie im Klartext abspeichern. Wir empfehlen zur Speicherung der Passwörter die Verwendung einer Hashfunktion (zum Beispiel `md5('geheim')`), die statt des Passwortes den Hashwert in der Datenbank abspeichert (Hashing siehe auch Abschnitt 9.4, »Partitionierung«). Bei der Überprüfung wird wiederum der Hashwert des angegebenen Passwortes erzeugt und mit dem gesicherten Hashwert in der Datenbank verglichen. Diese Vorgehensweise hat zwei Vorteile: Der erste Vorteil ist, dass die Passwörter nicht rückführbar sind, wodurch auch ein eventuell erfolgreicher Angriff auf die Datenbank nicht automatisch alle Passwörter der Benutzer offenlegen würde. Der Angreifer hätte dann »nur« die Hashwerte aller Passwörter erbeutet, mit denen er bloß sehr schwer auf die Passwörter rückschließen kann.

```
mysql> SELECT * FROM user WHERE user='test'
    -> AND password = MD5('geheim');
mysql> SELECT * FROM user WHERE user='test'
    -> AND password = 'f14a298bc87fff2cd757f71054fdd94d';
```

Listing 10.44 md5-Passwort-Hashing

In Listing 10.44 sehen Sie zwei Möglichkeiten zur Verwendung des md5-Hashes. In der ersten Zeile wird dabei die von MySQL angebotene md5-Hashfunktion ver-

wendet. Die zweite Variante verwendet bereits den aus dem Passwort berechneten Hashwert. Dieser kann zum Beispiel in den meisten Programmiersprachen über bereits vorhandene Funktionen (zum Beispiel `md5()` in PHP) berechnet werden.

Die Variante der Berechnung des Hashwerts in der Applikation ist dabei zu bevorzugen, da diese den zweiten großen Vorteil der Verwendung von Passwort-Hashing mit sich bringt. Wie Sie sehen, ist in der zweiten Anweisung in Listing 10.44 das eigentliche Passwort nicht mehr zu sehen. Dies bedeutet, dass auch ein eventueller Abhörversuch der zum MySQL-Server übermittelten Anweisungen keinen Zugriff auf die Klartext-Passwörter ermöglicht. Im Gegenzug wird in der ersten Variante in Listing 10.44 das Passwort im Klartext übermittelt und erst auf der MySQL-Serverseite in den Hash umgewandelt. Ein Abhörversuch würde somit Zugriff auf den Benutzernamen und das zugehörige richtige Passwort erhalten.

Sie sehen, dass eine kleine Maßnahme, wie das Client-seitige »Verschlüsseln« (bzw. Hashen) des Passwortes, bereits Abhörversuchen einen Riegel vorschiebt. Bedenken Sie daher immer, welche Daten Sie wirklich im Klartext über die Datenleitung zum Server übertragen müssen.

Natürlich lässt es sich oft nicht verhindern, dass zwischen Client und Server auch sehr kritische Daten im Klartext ausgetauscht werden. Diese sensiblen Daten können aber nicht nur durch Abhören der Datenleitung den Weg zu einem Angreifer finden. SQL-Anweisungen werden auch in vielen Fällen geloggt und in Dateien am Server abgelegt. Denken Sie an die Slow-Query-Log-Funktion, die zu langsame SQL-Anweisungen protokolliert. Sorgen Sie daher immer dafür, dass die Log-Dateien von MySQL durch weitere Schutzmaßnahmen des Dateisystems (zum Beispiel Lese-/Schreibrechte nur für den Systembenutzer, der auch zur Ausführung von MySQL verwendet wird) abgesichert sind. Auch Backups sollten in der gleichen Form vor fremdem Zugriff geschützt werden. Andernfalls können andere Systembenutzer über die Log-Dateien oder Backup-Dateien von MySQL sehr schnell und einfach an sehr sensible Daten gelangen.

Wie Sie Ihre Datenleitung zu Ihrem MySQL-Server zusätzlich durch Verschlüsselung absichern können und wie Sie Ihren MySQL-Server vor Zugriffen von außen über das Netzwerk schützen, erfahren Sie im folgenden Abschnitt.

10.4.2 Zugang beschränken oder verschlüsseln

Als Administrator sind Sie es gewohnt, Ihre Systeme nach außen zum Netzwerk hin abzusichern und die Kommunikation der einzelnen Dienste Ihres Servers zu beschränken. Wir empfehlen Ihnen, diese Vorgehensweise auch bei Ihrem MySQL-Server anzuwenden. In vielen Anwendungsfällen können Sie auf die Kommunikation nach außen in das Netzwerk oder Internet vollständig verzich-

ten. Ist dies der Fall, können Sie den standardmäßigen Serverport 3306 von MySQL mit Ihrer Firewall sperren. Es gibt somit keine direkten Angriffsmöglichkeiten mehr. Die Applikationen greifen dann entweder direkt lokal auf Ihren MySQL-Server zu oder über ein internes Netzwerk, indem die Kommunikation über den Port 3306 ermöglicht wird.

Benötigen Sie jedoch einen Zugriff von außen (Internet oder Netzwerk), bieten sich mehrere Möglichkeiten an, diesen zu schützen:

- Beschränkung der Benutzer
- indirekter Zugriff über SSH
- verschlüsselte Kommunikation (SSL)

Die erste, sehr einfach zu realisierende Möglichkeit zur Erhöhung der Sicherheit ist die *Einschränkung des Benutzerkreises*, der sich von außen mit dem MySQL-Server verbinden darf. Dabei reicht es aus, wenn Sie bei jedem Benutzer kurz über die Möglichkeit nachdenken, die Host-Angabe nicht automatisch mit `%` anzugeben. Nicht jeder Benutzer muss sich von außen verbinden können. Vor allem hochprivilegierte Benutzer mit vielen Berechtigungen sollten sich nur in Ausnahmefällen von außen verbinden können. Wir empfehlen Ihnen daher möglichst oft den Host mit `localhost` anzugeben und daher den Benutzer auf die lokale Kommunikation mit dem Server zu beschränken. Trotzdem ist natürlich der Port nach außen weiterhin geöffnet und bietet Möglichkeiten für Angriffe und Abhörversuche.

In vielen Szenarien ist die Kommunikation nach außen überwiegend für die Administration, Wartung oder andere seltenere Tätigkeiten geöffnet. In diesem Fall lässt sich die Sicherheit drastisch erhöhen, wenn Sie die Kommunikation nach außen vollständig per Firewall sperren. Die Wartungstätigkeiten können Sie dann *indirekt per SSH* (Unix-Systeme) oder mit *Fernwartungssoftware* (Windows) durchführen. Die Verbindung zum Server ist somit abgesichert, und die eigentliche Kommunikation mit dem MySQL-Server erfolgt lokal.

Da diese Variante nicht für alle Szenarien möglich ist und oft auch eine Kommunikation nach außen unabdingbar ist, bietet es sich an, die Kommunikation mit dem MySQL-Server zu verschlüsseln. Die *Verschlüsselung der MySQL-Verbindungen per SSL* stellt dabei die letzte und komplexeste Möglichkeit zum Schutz vor Angriffen dar. Diesen von MySQL selbst angebotenen Schutzmechanismus und alle notwendigen zugehörigen Einstellungen lernen Sie im folgenden Abschnitt im Detail kennen.

MySQL-Verbindungen per SSL

In MySQL besteht die Möglichkeit, verschlüsselte Verbindungen zwischen Client und Server zu nutzen. Dabei wird der gesamte Datenverkehr verschlüsselt und

das Abhören des Datenverkehrs verhindert. Wir empfehlen daher – wenn möglich – in öffentlichen Netzwerken (zum Beispiel Internet) eine SSL-Verschlüsselung (Secure Socket Layer) zu nutzen.

Um eine SSL-Verbindung aufbauen zu können, muss Ihre MySQL-Version mit der zugehörigen SSL-Option kompiliert worden sein. Ob das der Fall ist, können Sie mit der Abfrage der Systemvariablen `have_ssl`, wie in Listing 10.45 gezeigt, herausfinden.

```
mysql> SHOW VARIABLES LIKE 'have_ssl';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_ssl      | YES   |
+---------------+-------+
```

Listing 10.45 Abfrage, ob SSL in MySQL verfügbar ist

Ist der Wert `NO`, wurde MySQL nicht mit der SSL-Option kompiliert. Weitere Informationen zur Kompilierung von MySQL finden Sie auch in Kapitel 6, »Manuelle Installation, Plugins und Upgrades«. `DISABLED` bedeutet, dass MySQL SSL unterstützt, die nötigen Komponenten aber nicht beim Start aktiviert wurden. Für eine Aktivierung müssen Sie, wie in Listing 10.46 dargestellt, die nötigen Zertifikate beim Start von MySQL angeben. Sie können die Optionen natürlich auch in der Datei *my.cnf* angeben, um diese Werte nicht immer bei jedem Start schreiben zu müssen. Wurden beim Start die SSL-Optionen gesetzt, werden die notwendigen Komponenten zur Verschlüsselung geladen und Ihnen die Aktivierung von SSL über die Systemvariable `have_ssl` mit dem Wert `YES` angezeigt.

```
bash# mysqld --ssl-ca=ca.crt --ssl-cert=mysql.crt \
       --ssl-key=mysql.key
```

Listing 10.46 Aktivierung der SSL-Verschlüsselung beim Start des Servers

Um eine SSL-Verbindung zu aktivieren, müssen Sie die in Listing 10.46 angegebenen Optionen nicht nur beim Start des MySQL-Servers, sondern auch beim Start des MySQL-Clients (mysql) angeben. Die SSL-Parameternamen der Optionen sind bei Client und Server dieselben, wobei die Angabe des Zertifikats und des Schlüssels auf Client-Seite, wie in Listing 10.47 angegeben, ausreichend ist.

```
bash# mysql -u admin -p --ssl-cert=client.crt \
       -ssl-key=client.key
```

Listing 10.47 Aktivierung der SSL-Verschlüsselung beim Start des Clients

Bei den Angaben handelt es sich um Zertifikate, die zur Verschlüsselung benötigt werden. Normalerweise werden Zertifikate von einer offiziellen Zertifizierungsstelle (Certificate Authority, kurz CA) signiert. Für erste Testzwecke oder lokale Netze reichen in den meisten Fällen aber auch selbst signierte Zertifikate aus. Eine Schnellanleitung zur Erstellung eines selbst signierten Zertifikats und weitere Informationen zu Zertifikaten finden Sie in der grauen Box weiter unten.

Über den Parameter `ssl-ca` können Sie das Zertifikat der Zertifizierungsstelle angeben. Ihr eigenes Zertifikat geben Sie über `ssl-cert` an, und `ssl-key` spezifiziert Ihren *privaten Schlüssel*.

Wenn Sie nun, wie in Listing 10.47 gezeigt, eine erfolgreiche verschlüsselte Verbindung aufgebaut haben, können Sie die korrekte Verbindung über die Abfrage des sogenannten *Cipher* (Verschlüsselungsalgorithmus) überprüfen. Wird Ihnen nach Eingabe des Befehls aus Listing 10.48 ein Cipher angezeigt, ist die aktuelle Verbindung sicher und verschlüsselt.

```
mysql> SHOW STATUS LIKE 'ssl_cipher';
+---------------+--------------------+
| Variable_name | Value              |
+---------------+--------------------+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+---------------+--------------------+
1 row in set (0.00 sec)
```

Listing 10.48 Anzeige des Ciphers der aktuellen Verbindung

Erstellen eines Zertifikats mit OpenSSL

Mit OpenSSL, das auf den meisten Linux-Distributionen bereits standardmäßig verfügbar sein sollte, können Sie sich schnell und einfach ein selbst signiertes Zertifikat erstellen.

(1) Im ersten Schritt erstellen Sie sich mit dem nachfolgenden Befehl zuerst einen sogenannten privaten Schlüssel, der mit 2.048 Bit verschlüsselt wird und in die Datei *ca.key* gespeichert wird. Der zweite Befehl erstellt Ihnen das zugehörige Zertifikat mit dem Namen ca.crt. Sie werden während der Schritte des Öfteren nach Dateneingaben (wie zum Beispiel Ländercodes, Adresse usw.) gefragt. Diese können Sie für erste Testzwecke ignorieren und einfach bestätigen.

```
bash# openssl genrsa -out ca.key 2048
bash# openssl req -new -x509 -key ca.key -out ca.crt
```

(2) Im zweiten Schritt erstellen Sie nun eine Signierungsanfrage (Certificate Signing Request, kurz CSR) und einen zweiten privaten Schlüssel für Ihren Server. Die in diesem Schritt abgefragten Daten (Adresse usw.) werden anschließend fest mit Ihrem Zertifikat verknüpft.

```
bash# openssl req -newkey rsa:2048 -nodes \
      -keyout mysql.key -out mysql.csr
```

Diese Datei *mysql.csr* senden Sie im Normalfall an eine offizielle Zertifizierungsstelle, wie A-Trust in Österreich oder D-Trust in Deutschland. Diese Stelle stellt gegen eine Bearbeitungsgebühr anschließend ein Zertifikat aus.

(3) Mit dem folgenden Kommando erstellen Sie ein selbst signiertes und kostenloses Zertifikat, das aber für erste Testzwecke völlig ausreichend ist.

```
bash# openssl x509 -req -days 365 -in mysql.csr \
      -CA ca.crt -CAkey ca.key -out mysql.crt -set_serial 01
```

Die resultierende Datei *mysql.crt* stellt dabei das Zertifikat dar, das Sie nun für Ihre SSL-Verbindung, wie in Listing 10.46 angegeben, nutzen können.

(Client) Für den Client können Sie sich ein weiteres Zertifikat durch Wiederholung der zwei letzten Schritte (2) und (3) erstellen. Wählen Sie dabei statt des Dateinamens *mysql.Endung* (zum Beispiel *mysql.crt*) eine andere Bezeichnung, wie zum Beispiel *client.Endung* (zum Beispiel *client.crt*).

Beachten Sie, dass ein MySQL-Server, bei dem die SSL-Verschlüsselung aktiviert wurde, auch weiterhin unverschlüsselte Verbindungen akzeptiert. Wenn Sie Benutzer zu einer verschlüsselten Verbindung verpflichten wollen, müssen Sie das bei der Erstellung der Benutzer angeben. Zum Beispiel können Sie einem Benutzer die lokale (`localhost`) Verbindung unverschlüsselt und anderen Hosts nur SSL-Verbindungen erlauben. In Listing 10.49 sehen Sie ein Beispiel, wie Sie dem Benutzer »admin« nur mehr SSL-verschlüsselte Verbindungen erlauben können.

```
mysql> GRANT USAGE ON *.* TO 'admin'@'%' REQUIRE SSL;
```

Listing 10.49 Benutzer auf SSL-Verbindungen beschränken

Selbstverständlich gibt es noch weitere Einschränkungen, die Sie bei der Verschlüsselung vornehmen können. X509 setzt zum Beispiel ein gültiges Zertifikat zur Authentifizierung mit dem X509-Standard voraus. Dieses muss – wie in Listing 10.47 angegeben – beim Start des Clients benannt werden. Durch die Angabe von einem oder mehreren Ciphern können Sie für die Verbindungen des jeweiligen Benutzers einen bestimmten Verschlüsselungsalgorithmus (zum Beispiel DHE-RSA-AES256-SHA) voraussetzen. Über die Schlüsselwörter `ISSUER` und `SUBJECT` können Sie einschränken, von welchen Quellen Sie das Zertifikat (X509) des Clients akzeptieren wollen. In Listing 10.50 sehen Sie ein komplexes Beispiel, bei dem ein Cipher und beispielhafte X509-Optionen durch `Issuer` und `Subject` angegeben werden.

```
mysql> GRANT USAGE ON *.* TO 'admin'@'%'
    -> REQUIRE CIPHER 'DHE-RSA-AES256-SHA' AND
    -> ISSUER '/C=GB/ST=Berkshire/L=Newbury/O=My Company Ltd' AND
    -> SUBJECT '/C=GB/ST=Berkshire/L=Newbury/O=My Company Ltd';
```

Listing 10.50 Genaue Angabe der SSL-Optionen

Wie Sie gesehen haben, bietet MySQL also auch zur Verschlüsselung der Verbindung zahlreiche Optionen an. Administratoren sind vom Umgang mit SSL des Öfteren abgeschreckt und auch die Kosten der offiziellen Signierungsstelle werden gerne als Argument gegen den Einsatz von SSL verwendet. Wir können Ihnen aber nur stark dazu raten, SSL einzusetzen, da die gesamte Verbindung verschlüsselt wird und dies daher einen sehr großen Gewinn für die Sicherheit Ihres Systems darstellt. In vielen Fällen ist auch bereits ein selbst signiertes Zertifikat ausreichend und offiziell signierte Zertifikate lassen sich meist auch für andere Dienste am Server (zum Beispiel Webserver) wiederverwenden. Vor allem wenn Sie MySQL im Internet einsetzen und der Datenverkehr durch das Internet geleitet wird, ist eine verschlüsselte Verbindung beinahe unumgänglich. Weitere Punkte, die Sie bei einer Installation von MySQL im Web beachten sollten, besprechen wir im folgenden Abschnitt.

10.4.3 MySQL im Web

Der große Erfolg von MySQL ist unter anderem auf die Verwendung im Web zurückzuführen. Gebündelt mit PHP war MySQL bereits vor vielen Jahren der Standard für die Programmierung von Web-Applikationen.

Wenn Sie MySQL selbst im Web einsetzen wollen, gibt es nur wenige Unterschiede zu einer klassischen Installation im lokalen Netzwerk. In einem abgeschlossenen lokalen Netzwerk ist die Gefahr von Angriffen jedoch meist geringer einzustufen. Im Internet ist Ihr Server für jedermann zugänglich. Durch diese Tatsache wird Ihre Installation weder sicherer noch unsicherer. Sie können aber davon ausgehen, dass Sicherheitslücken im Internet schneller eine Gefahr darstellen und ausgenutzt werden.

Die zwei größten Gefahren liegen dabei im Zugang zu Ihrer Datenbank, der auf zwei verschiedene Arten durchgeführt wird. Einerseits können sich Benutzer klassisch über einen Datenbank-Client mit Ihrem Server verbinden. Die Absicherungsmöglichkeiten dieser Schnittstelle durch eine vollständige Sperre oder die Verwendung einer Verschlüsselung haben Sie bereits im letzten Abschnitt kennengelernt.

Die zweite Verbindungsmöglichkeit besteht über die Web-Applikation (zum Beispiel Ihr Content-Management-System) selbst, die im Internet frei zugänglich ist und sich wiederum mit Ihrer Datenbank verbindet. Natürlich haben Sie auf diese Verbindungsvariante als Administrator nur beschränkte Eingriffsmöglichkeiten. Wie Sie Ihre MySQL-Installation trotzdem vor dieser potenziellen Gefahr schützen können und welche Gefahren diese Schnittstelle birgt, behandeln wir im Detail im folgenden Abschnitt.

SQL-Injections

Die *SQL-Injections*, oder auf Deutsch auch *SQL-Einschleusungen* genannt, sind kein direktes Sicherheitsproblem des MySQL-Servers, sondern gehen vom Benutzer bzw. der Applikation aus. Sie betreffen daher auch nicht nur das MySQL-System, sondern können in jedem SQL-basierten Datenbanksystem auftreten und stellen ein sehr großes Sicherheitsrisiko dar.

Aufgrund der großen Verbreitung der Sicherheitslücke ist diese Thematik auch für Sie als Administrator relevant, obwohl die Problemlösung nur auf Client- bzw. Applikationsseite getroffen werden kann. Das große Problem dieser Sicherheitslücke ist aber nicht die Schwierigkeit, diese zu beseitigen, sondern die Unwissenheit vieler Programmierer über diese Problematik. Sie können daher auch als Administrator helfen, diesen Wissensmissstand zu beseitigen.

Das Problem beruht auf der Tatsache, dass Applikationen meist mit dem Benutzer interagieren und diesem erlauben, diverse Eingaben zu tätigen. Meist muss der Benutzer bereits vor der Verwendung der eigentlichen Applikation einen Benutzernamen und ein Passwort eingeben. In der Welt der Softwareentwicklung gibt es die Grundregel »Traue nie den Eingaben des Benutzers«, und so muss die Applikation jede Benutzereingabe überprüfen. Oft wird dies gerade bei banalen Feldern, wie dem Benutzer- oder Passwortfeld bei der Anmeldung, vergessen.

Betrachten wir das Beispiel einer Internetapplikation, bei der sich Mitarbeiter unseres Flughafens anmelden können, um interne Informationen (zum Beispiel Dienstpläne) zu erhalten. Die Applikation ist selbstverständlich nur nach der erfolgreichen Anmeldung über den Benutzer und das korrekte Passwort zugänglich. Die Anmeldeseite selbst muss aber natürlich im Internet verfügbar sein. In Listing 10.51 sehen Sie eine Abfrage, die während des Anmeldeprozesses an den Datenbankserver gestellt wird, wenn ein Benutzer die Kombination aus dem Benutzernamen admin und dem Passwort geheim eingibt.

```
SELECT * FROM mitarbeiter WHERE benutzername = 'admin'
AND passwort = md5('geheim');
```

Listing 10.51 Abfrage während des Web-Anmelde-Prozesses

Bereits dieser Vorgang könnte ein Sicherheitsrisiko darstellen, wenn die Eingaben des Benutzers nicht ordnungsgemäß überprüft werden. Betrachten Sie das Beispiel in Abbildung 10.2, bei dem ein böswilliger Benutzer statt des eigentlichen Benutzernamens eine SQL-Anweisung `';DELETE FROM mitarbeiter;` in das Feld eingibt.

In Listing 10.52 sehen Sie die resultierende SQL-Anweisung. Die Zeichenfolge `';` ist dabei ausschlaggebend, da diese die umschließende SQL-Anweisung verfrüht abbricht. Für den Datenbankserver handelt es sich nun nicht mehr um eine SQL-Anweisung, sondern um drei aneinandergereihte Anweisungen, die in Listing 10.53 angeführt sind. Die ersten zwei Anweisungen werden dabei im Normalfall korrekt ausgeführt, und die dritte Anweisung wird mit einer Fehlermeldung (falsche Syntax) abgebrochen. Das große Problem ist hierbei die Ausführung der zweiten SQL-Anweisung, die alle Benutzer in der Tabelle »mitarbeiter« löscht.

```
SELECT * FROM mitarbeiter WHERE
benutzername = '';DELETE FROM mitarbeiter;'
AND passwort = md5('geheim');
```

Listing 10.52 Resultierende Anweisung durch SQL-Injection

```
SELECT * FROM mitarbeiter WHERE benutzername = '';
DELETE FROM mitarbeiter;
' AND passwort = md5('geheim');
```

Listing 10.53 Drei resultierende Anweisungen durch SQL-Injection



Abbildung 10.2 Beispiel einer SQL-Injection

Das bedeutet, dass eine SQL-Anweisung in eine bestehende Abfrage eingeschleust wurde und fälschlicherweise vom Datenbankserver ausgeführt wird. Diese Einschleusung ist also besagte SQL-Injection. Gerade bei Web-Applikationen wird meist, wie in Abschnitt 10.3.4, »Benutzerverwaltung in der Applikation, nicht in der Datenbank?«, erwähnt, nur ein Datenbankbenutzer für die gesamte Web-Applikation verwendet. Das eigentliche Rechtesystem wird von

der Applikationsebene übernommen. Das bedeutet aber auch, dass der Datenbankbenutzer meist auch kritische `DELETE`-Befehle, wie in Listing 10.53 angegeben, ausführen kann. Gerade durch die stark wachsende Zahl an Web-Applikationen wurde die SQL-Injection-Sicherheitslücke so zu einem Massenproblem im Internet.

Eine derartige Sicherheitslücke kann großen Schaden anrichten, lässt sich aber sehr einfach durch striktes Überprüfen bzw. Maskieren der Benutzereingaben vermeiden. Unter *Escapen* (dt. Maskieren) versteht man die Kennzeichnung von bestimmten Steuersymbolen als »normale« Eingaben.

Betrachten Sie das Beispiel in Listing 10.54, in dem die Benutzereingabe aus Abbildung 10.2 korrekt maskiert wurde. Das kritische Hochkomma-Symbol wird dabei mit einem \ (Escape-Zeichen) maskiert und daher vom MySQL-Server nicht mehr als Steuersymbol, sondern als normales Zeichen interpretiert. Die Abfrage sucht so nach einem Benutzer mit dem Namen `';DELETE FROM mitarbeiter;`. Dieser Benutzer wird nicht gefunden, und die Web-Applikation kann die übliche Fehlermeldung bei einem unbekannten Benutzer anzeigen. Wichtig dabei ist, dass der Versuch, eine SQL-Anweisung einzuschleusen, nicht erfolgreich war. Sie sehen also, dass bereits dieses eine Escape-Zeichen über die Sicherheit einer Applikation entscheiden kann.

```
SELECT * FROM mitarbeiter WHERE
benutzername = '\';DELETE FROM mitarbeiter;'
AND passwort = md5('geheim');
```

Listing 10.54 Maskierte Benutzereingabe

Doch wie können Sie sich vor diesen SQL-Injections am besten schützen? Leider kann man auf der MySQL-Serverseite nicht entscheiden, ob eine SQL-Anweisung einen Angriff beinhaltet oder nicht. Die Absicherung kann daher nur auf der Applikationsseite erfolgen. Hierzu gibt es einige einfache Möglichkeiten, diese SQL-Injections zu verhindern. Die Gegenmaßnahmen können grob in die folgenden drei Ansätze unterteilt werden:

- Überprüfung der Benutzereingaben
- Maskieren von Benutzereingaben
- Prepared Statements

Bei der *Überprüfung der Benutzereingaben* wird von der Applikation zuerst geprüft, ob die Eingaben eventuelle Angriffe beinhalten. Bei unserem Beispiel des Benutzernamens ist dies sehr einfach möglich. Benutzernamen dürfen im Normalfall nur aus Buchstaben und Zahlen bestehen. Eine Prüfung kann daher

alle Benutzernamen, die beliebige Sonderzeichen beinhalten (zum Beispiel das Hochkomma), sofort verwerfen. Bei komplexen Benutzereingaben kann eine derartige Prüfung jedoch sehr schnell an die Grenzen stoßen, da die Programmierung zum Abfangen aller Möglichkeiten zu komplex wird.

Die einfachere Möglichkeit ist daher die *Maskierung der Benutzereingaben*. Dabei werden alle »gefährlichen« Symbole maskiert, damit diese nicht von der Datenbank als Steuerzeichen interpretiert werden. Einige Programmiersprachen bieten hierfür bereits vordefinierte Funktionen für diesen Zweck an. In PHP können Sie zum Beispiel die Funktion `mysql_real_escape_string('eingegebener Text')`[1] verwenden, die Ihnen einen »sicheren« String zurückliefert. Dies erspart Ihnen, eine derartige Maskierungsfunktion selbst zu programmieren und zu testen. Den zurückgelieferten String können Sie anschließend ohne Bedenken in einer SQL-Anweisung verwenden.

Die letzte Möglichkeit zur Vermeidung von SQL-Injections besteht in der Verwendung von sogenannten *Prepared Statements* oder *parametrisierten SQL-Anweisungen*. Diese Variante ist ebenfalls in sehr vielen Programmiersprachen verfügbar. Dabei wird im ersten Schritt eine SQL-Anweisung mit sogenannten Parametern versehen. Diese können Sie wie Variablen oder Platzhalter sehen, die Sie zu einem späteren Zeitpunkt mit richtigen Daten füllen.

Listing 10.55 zeigt die Anmelde-Abfrage unseres Beispiels als Prepared Statement. Das `?` steht dabei als Platzhalter für einen später gesetzten Wert. Dieser vorbereitende Schritt hat zusätzlich den Vorteil, dass MySQL bereits bei der Erstellung eines Prepared Statements die Abfrage optimiert und kompiliert. Wird die Anfrage also öfter mit unterschiedlichen Parametern (Werte der `?`) ausgeführt, muss die Abfrage nicht jedes Mal neu optimiert und kompiliert werden. Es können also Berechnungen wiederverwendet werden, was die Abfragegeschwindigkeit bei wiederholter Ausführung erhöht. Nachdem auch bereits im Vorfeld die genaue Struktur der SQL-Anweisung festgelegt wird, erwartet sich MySQL auch bei den Parametern nur mehr Werte, wie zum Beispiel Zahlen, Datumsangaben und Strings, jedoch keine neuen SQL-Anweisungen. Wird also als Parameter eine neue Anweisung eingeschleust, wird diese einfach als Wert interpretiert und nicht ausgeführt.

```
SELECT * FROM mitarbeiter WHERE
benutzername = ? AND passwort = md5(?)
```

Listing 10.55 Beispiel eines Prepared Statements

1 *http://php.net/manual/de/function.mysql-real-escape-string.php*

In Listing 10.56 sehen Sie die Erzeugung und den Aufruf eines Prepared Statements in der MySQL-Konsole. In der Programmiersprache Ihrer Wahl gibt es für Prepared Statements meist eigene Funktionen, die Sie in dem jeweiligen Manual der Programmiersprache oder in Kapitel 12, »Softwareentwicklung mit MySQL«, finden.

```
mysql> PREPARE abfrage1 FROM
    -> "SELECT * FROM mitarbeiter WHERE
    -> benutzername = ? AND passwort = md5(?)";
Query OK, 0 rows affected (0.00 sec)
Statement prepared
mysql> SET @benutzer = 'meinBenutzer';
mysql> SET @passwort = 'geheim';
mysql> EXECUTE abfrage1 USING @benutzer,@passwort;
Empty set (0.00 sec)
mysql> DEALLOCATE PREPARE abfrage1;
```

Listing 10.56 Verwendung von Prepared Statements in der MySQL-Konsole

Der allgemeine Ablauf bei der Verwendung von Prepared Statements ist dabei immer derselbe (siehe Listing 10.56). Zuerst wird das Prepared Statement erzeugt und die Platzhalterpositionen werden mit dem `?` markiert. Beim Aufruf durch `EXECUTE` werden die Werte bzw. in diesem Fall Variablen, die an die jeweilige Platzhalterposition geschrieben werden sollen, mit `USING` übergeben. Der Aufruf resultiert, wie gewöhnlich, beim direkten Aufruf einer Abfrage in einer Ergebnismenge. Wird das Prepared Statement nicht mehr benötigt, sollten Sie es mit `DEALLOCATE PREPARE` löschen. Dieses Kommando entfernt auch alle damit verbundenen Vorberechnungen zur Geschwindigkeitsoptimierung der Abfrage.

Alle drei Gegenmaßnahmen vermeiden SQL-Injections. Die erste Variante der Überprüfung der Benutzereingaben kann aber sehr schnell komplex werden und bietet damit wiederum Potenzial für neue Sicherheitslücken. Wir empfehlen Ihnen daher als einfachste und sicherste Methode, alle Benutzereingaben zu maskieren. Gerade in Programmiersprachen, die bereits Funktionen für diesen Maskierungszweck zur Verfügung stellen, wie zum Beispiel PHP, ist diese Vorgehensweise sehr einfach zu handhaben.

Das Wichtigste ist jedoch, dass Sie die Thematik der leicht entstehenden Sicherheitslücke der SQL-Injections im Kopf behalten. Vielleicht programmieren Sie selbst Software und können eine der besprochenen Gegenmaßnahmen nutzen oder können die Programmierer der eingesetzten Software darüber informieren.

10.5 Zusammenfassung

In diesem Kapitel haben Sie alle wichtigen Aspekte zur Absicherung von MySQL kennengelernt. Die interne Benutzerverwaltung mit all ihren Berechtigungen auf den verschiedenen Ebenen stellt dabei das Kernstück der Sicherheit in MySQL dar. Sie können damit jedem Benutzer Berechtigungen auf globaler, Datenbank-, Tabellen- oder sogar Spalten- und Zeilenebene vergeben. Wir haben Ihnen auch die Berechtigungen von Objekten in MySQL, wie zum Beispiel Events oder gespeicherte Prozeduren, vorgestellt. Sie haben Tipps und Tricks kennengelernt, wie Sie herausfordernde Berechtigungsaufgaben lösen und direkt auf den Berechtigungstabellen von MySQL operieren können.

Der letzte Teil des Sicherheitskapitels hat sich mit dem Schutz rund um MySQL beschäftigt. Dabei haben wir Ihnen die Möglichkeit zur Verschlüsselung von Datenbankverbindungen vorgestellt. Aber auch zahlreiche Aspekte in Bezug auf Ihre Datenbankbenutzer und deren Anwendungen haben Sie kennengelernt. SQL-Injections, ausgehend von »unachtsamen« Anwendungen, sind dabei nur ein Sicherheitsrisiko des MySQL-Administrators.

Die Tipps zur Erstellung von Sicherheitskonzepten oder die Erstellung von regelmäßigen Abläufen (Workflows) sollen Ihnen im Administratorenalltag helfen und die Arbeit erleichtern.

Wenn Sie die innere Sicherheit – die Berechtigungen von MySQL – und die äußeren Einflüsse auf MySQL im Auge behalten, analysieren und gegebenenfalls reagieren, werden Sie mit einem sehr sicheren und stabilen Datenbanksystem belohnt werden, das auch Ihre Benutzer zu schätzen wissen.

Index

@-Zeichen 277
[mysqld] 247
1NF 85
2NF 88
32 Bit *96*
3NF 88
64 Bit *96*

A

Abbruchbedingung 567
Abfrageerweiterung 391
Abfrageverarbeitung 163, 165
Abhörversuch 498
Ablaufsteuerung 557
Aborted_connects 306
Abstraktion 35
  Modellierung 51
Absturz 295, 300
ACID-Eigenschaften 180, 191
ACID-konform 199, 337
ACTION_CONDITION *607*
ACTION_ORDER *607*
ACTION_ORIENTATION *607*
ACTION_REFERENCE_NEW_ROW *607*
ACTION_REFERENCE_NEW_TABLE *607*
ACTION_REFERENCE_OLD_ROW *607*
ACTION_REFERENCE_OLD_TABLE *607*
ACTION_STATEMENT *607*
ACTION_TIMING *607*
ADD PARTITION 443, 444
Administrationsaufgaben 263
Administratorberechtigungen *97*, 457
Administratorkonto 232
AFTER *600*
AFTER DELETE *603*
AFTER INSERT *603*
AFTER UPDATE *603*
Aktualisierbare Sichten *622*
Aktualisierung → Update
Alias *67*, 602
ALL 473
ALL PRIVILEGES 454
ALTER 463, 544
ALTER EVENT 615, *695*
ALTER FUNCTION 543, *693*
ALTER PROCEDURE 529, *692*
ALTER ROUTINE *466*
ALTER TABLE 134
ALTER TABLE ADD FOREIGN KEY *669*
ALTER TABLE ADD PARTITION *707*
ALTER TABLE ANALYZE PARTITION *708*
ALTER TABLE CHECK PARTITION *708*
ALTER TABLE COALESCE PARTITION *707*
ALTER TABLE DROP FOREIGN KEY *669*
ALTER TABLE DROP PARTITION *707*
ALTER TABLE EXCHANGE PARTITION *709*
ALTER TABLE OPTIMIZE PARTITION *708*
ALTER TABLE REBUILD PARTITION *708*
ALTER TABLE REMOVE PARTITIONING *708*
ALTER TABLE REORGANIZE PARTITION *709*
ALTER TABLE REPAIR PARTITION *709*
ALTER TABLE TRUNCATE PARTITION *709*
ALTER VIEW 523, *688*
ALTER_ROUTINE 523
ANALYZE PARTITION 445
ANALYZE_TABLE 461
AND 559
Änderungsanomalie 84
AND-Verknüpfung *670*
Anfrageüberprüfung 454
Anmeldung *689*
Anmeldung → Login
ANSI SQL
  2003 SQL/PSM 516
Antelope 203
API 258
Application Programming Interface → API
Applikationsberechtigungen 490
apt 109
ARCHIVE 213
ASCII 240, *678*
ASTEXT *705*
ASTEXT() *379*

Atomarität 180
Attribut 58, 70
Aufwärmen 319
Ausfallsicherheit 345
Ausführer 178
Ausführung 528
Ausführungsplan 157, 173
Ausgabe
vertikal 126
Ausgabeparameter 531
Authentifizierung 239
Autocommit-Modus 184
Auto-Increment 351
auto-increment-increment 352
auto-increment-offset 352
automysqlbackup 299
Axmark, David 42

B

Backticks 462
Backup 38, 279, 280, 346, 521
inkrementelles 285
logisches 282, 286
physisches 282
volles 285
Barracuda 203
base_dir 251
bashrc 236
Bash-Shell 220, 234
Bedienfehler 295
Bedingte Anweisungen 558
BEFORE 600
BEFORE DELETE 603
BEFORE INSERT 603
BEFORE UPDATE 603
BEGIN 184, *686*
BEGIN WORK 184
BEGIN/END-Blöcke *697*
Begrenzungszeichen 517
BENCHMARK 319, *722*
Benchmark 304, 319
Benutzerdefinierte MySQL-Variablen 552
Benutzergruppe 223, 227, 241
Benutzerkonto
Betriebssystemebene 223
Benutzeroberfläche 264
Benutzerrechte 237
Owner 228
Benutzerzugang 237
Berechtigungen 239, 451
auf Zeilenebene 492
Berechtigungsebenen 456
Berechtigungstabelle 265
Bereichsanfragen 212, 372
Berkeley DB 44
Bezeichner 54, 119
Bezeichner- oder Namensschema 551
Beziehung 70, 74
1:1-Beziehung 77
1:n-Beziehung 79
Arten 75
identifizierende 78
n:m-Beziehung 79
symmetrische 79
BIN *679*
Binärdatei 40
Binärdistribution 263
Binäres Logging 523
Binaries 94
Binär-Log 249, 296
bind_address 251
binlog_format 249
BIT_LENGTH *679*
BLACKHOLE 216
Blattknoten 367
BLOB 339
Block 366, 526, 549
Blog 33
Boolesche Volltextsuche → Volltextsuche
Buffer-Management 160
Bug 47
Build-Tools 222
Business Intelligence 32

C

CA 501
Cache 168, 265
CALL 528, *691*
CallableStatements 641
Cardinality 398
CASE 562, *697*
Catch-All-Partition 433
Certificate Authority 501
Change History 255
CHANGE MASTER 356
CHANGE MASTER TO *703*

CHAR *679*
CHAR_LENGTH *679*
CHARACTER_LENGTH *679*
CHARACTER_MAXIMUM_LENGTH 546
CHARACTER_OCTET_LENGTH 546
CHARACTER_SET_CLIENT 547, 605, 607, 617
CHARACTER_SET_NAME 547
CHECK 300
CHECK PARTITION 445
CHECK TABLE 544, *717*
Chen-Notation 75
Cipher 501
Clearing 617
Client 95
Client-/Server-System 40
CLOSE 571
CLOSE CURSOR *700*
cmake 222
COALESCE 444
Codd, Edgar 34, 85
Cold Copy → Offline-Backup
COLLATION_CONNECTION 547, 605, 607, 617
COLLATION_NAME 547
COLUMNS 431
columns_priv 481
COLUMNS-Partitionierung 436
COMMENT 528
COMMIT 183, 544, 598, *686*
Community 39
Compiler 220
CONCAT *679*
CONCAT_WS *679*
Condition 580
Connector 627
Connector/J *636*
Connector/NET 645
Constraints 135, 273
CONTAINS SQL 527, 540
Content
  dynamisch generiert 32
Content-Management-System 33
Continue-Handler 576, 581
Crash Recovery 295, 302
CREATE 461, 544
Create An Anonymous Account 102
CREATE DATABASE *664*, *710*
CREATE EVENT 610, *695*
CREATE FUNCTION 537, *692*
CREATE INDEX *669*
CREATE PROCEDURE 524, 525, *691*
CREATE ROUTINE 465
CREATE SERVER *726*
CREATE TABLE 120, *665*
  mit Partitionen *706*
  Storage-Engine-Einstellungen *726*
CREATE TABLESPACE 474
CREATE TEMPORARY TABLE 462
CREATE TRIGGER *694*
CREATE USER 472, *714*
CREATE VIEW 464, *688*
CREATE_ROUTINE 523
CREATE_TEMPORARY_TABLES 594
CREATED 547, 605, 607
Cronjobs 516, 608
CRUD 130
CSV 215, 272
CSV-Storage-Engine 278
CURRENT_TIMESTAMP 612
Cursor 569, 571
  öffnen 571
  verschachtelter 590

D

Daemon → Dämon
Dämon 40, 229, 243
Data Definition Language → DDL
Data Dictionary 36
DATA DIRECTORY 434
DATA_TYPE 546
Database Collation 605, 617
DATABASE_COLLATION 547, 607
Dateiberechtigung 228, 232
Daten 23
Datenabstraktion 35
Datenbank 23, 24
  aktivieren 239
  deduktive 28
  mobile 28
  NoSQL-Datenbank 27
  objektorientierte 27
  XML-Datenbank 27
Datenbankabstraktionsschicht 628
Datenbankapplikation 157
Datenbankdesign 36, 70
Datenbankebene 457

Datenbankentwurf
Anforderungsanalyse 52
konzeptioneller 53
logischer 54
physischer 54
Datenbankindex 33
INDEX 138
Datenbankmanagementsystem 25, 237
relationales 26
Datenbankmodell
hierarchisches 25
Netzwerkdatenbankmodell 25
objektorientiertes 27
objektrelationales 27
relationales 26, 57, 155
Datenbankmodellierung 51
Datenbankserver 35
Datenbankverwaltung 157
Datendurchsatz 319
Datenintegrität 36
Datenkommunikation 514
Datensatz 25, 59
Datensicherheit 37
Datenstruktur 26
Datentypen 521
Datenverzeichnis 232
Datum *663*
Datum, aktuelles *676*
Datumsangaben, Formatierung *678*
Datumsarithmetik *675*
Dauerhaftigkeit 181
Db *616*
DBD *650*
mysql *650*
DBI *650*
DBMS → Datenbankmanagementsystem
DDL 54, 544
Deadlock 192
DEALLOCATE PREPARE 544, *717*
Debian 220
Debugging 534, 566
DECLARE 551, *699*
DECLARE CONDITION *701*
DECLARE CURSOR 570, *699*
DECLARE HANDLER 577, *700*
default-character-set 279
Default-Wert 554
DEFINER 467, 547, *607*
Definer 538, 601, 605, *616*
Deklarieren 551
Delayed_errors 210
delayed_insert_limit 211
delayed_insert_timeout 211
delayed_queue_size 211
Delayed_writes 210
DELETE 242, 461, *668*
Delimiter 517
DESCRIBE *689*
DETERMINISTIC 527, 540
Deterministisch 527
DHE-RSA-AES256-SHA 502
Dienst *99*
Dirty Reads 187
Disjunktion *66*
Distance() 380
Distributed Transaction Processing
The XA Specification 192
DNS *29*
DO 614
Dokumentation 521
Downloadlink *93*
DROP 463, 544
DROP DATABASE *664*, *710*
DROP EVENT 615, *696*
DROP FUNCTION *693*
DROP INDEX *670*
DROP PARTITION 444
DROP PREPARE *717*
DROP PROCEDURE 529, *691*
DROP SERVER *726*
DROP TABLE *667*
DROP TRIGGER 604, *694*
DROP USER 243, 455, *715*
DROP VIEW *689*
DTD_IDENTIFIER 547
Durchschnitt *66*
-DWITH_PARTITION_STORAGE_-
ENGINE=1 430

E

Editor
nano 245
Einfügeanomalie 84
Eingabeparameter 531
Einschränkungen 450
Eintritts- und Austrittspunkt 539
Einwegfunktion 241

ELSEIF 561
ELT *679*
Enable root access from remote machines 102
Ende der Ergebnismenge 575
Ends 616
End-User License Agreement → EULA
ENGINE=InnoDB 198
Entität 70, 71
  existenzabhängige 78
Entity-Relationship-Modell → ER-Modell
enumeration 141
ENVELOPE 382, *706*
Ergebnismengen 536
ER-Modell 53, 70
error code 147
error.log → Error-Log
error_log 249
Error-Log 290, 293, 300
Erzeugen von Cursors 570
Erzeugung einer partitionierten Tabelle 430
Erzeugung gespeicherter Prozeduren 524
Erzeugung von Benutzern 472
Erzeugung von Datums- und Zeitangaben *676*
Erzeugung von Geometrieobjekten 378
Escapen 506
Escape-Sequenz *730*
Escape-Sequenz → Maskierungszeichen
Escape-Zeichen 506
EULA 97
EVENT 465, 618
EVENT_MANIPULATION 606
EVENT_OBJECT _TABLE 606
EVENT_OBJECT_CATALOG 606
EVENT_OBJECT_SCHEMA 606
event_scheduler 609
Events 515, 516, 522, 605, 608
  bearbeiten 615
Event-Scheduler 609
EVERY 613
EXECUTE 250, 466, 523, 544, *689*, *717*
Execute At 616
Execution-Engine 163, 178
Exit-Handler 576, 582
EXPLAIN 394, 544
EXPLAIN PARTITIONS 446, *710*
Export 278
  Linux-Befehl 235
EXPORT_SET *680*
EXTENDED STATUS *711*
EXTERNAL_LANGUAGE 547
EXTERNAL_NAME 547

F

Faktor Mensch 495
Fallunterscheidung 557, 558, 561
false 558
FEDERATED 214
Fehler 576
Fehlermeldung 577
Fehlernummer 576, 578, *728*
Fernwartungssoftware 499
FETCH 572
FETCH INTO *700*
FIELD *680*
FILE 473
  Privileg 278
FIND_IN_SET *680*
Firewall 499
FIRST 213
FLOOR 383, 527
FLUSH 265, 544
FLUSH HOSTS *710*
FLUSH LOGS 296
FLUSH PRIVILEGES 483, *716*
FLUSH QUERY CACHE *724*
FLUSH TABLES WITH READ LOCK 355
flush-hosts 265, 266
flush-logs 265, 266
flush-privileges 265
flush-status 265
flush-tables 265
flush-threads 265
FORMAT *680*
Format
  dynamisches 207
  statisches 206
Forum 33
Forward-Engineering 83, 311
Free Software Foundation → FSF
Fremdschlüsselbeziehungen 274
frm-Datei 199
FROM_BASE64 *680*
FSF 46

ft_boolean_syntax 391
ft_max_word_len 390
ft_min_word_len 390
ft_query_expansion_limit 391, 392
ft_stopword_file 390
Full-Table-Scan 363
FULLTEXT 385
FUNCTION 538
Fünf-Schichten-Modell 155
Funktion
statistische 32
Funktionale Abhängigkeit 61
Funktionalität 70

G

Geoinformationssystem → GIS
Geo-Koordinaten 34
GEOMETRY 377
GEOMETRYCOLLECTION 377
GeomFromText *705*
Geplante Tasks 608
Gespeicherte Funktionen 515, 536, 537
Gespeicherte Programme 290, 384, 513, 515
Gespeicherte Prozeduren 515, 524
bearbeiten 529
GIS 377
GLOBAL 252, 254
Globale Ebene 457
Gnome 103
GNU General Public License → GPL
GNU/Linux 40, 103
GPL 46, 97
GRANT 242, 455, 476, *715*
GRANT OPTION 471
Groß- und Kleinschreibung 386
Grundinstallation 95
Gültigkeitsbereich 555
einer Variable 555
eines Handlers 589
Gültigkeitsbereich → Scope

H

Handler 575, 576
Hardwareanforderungen 94
HASH 245, 431
Hashausdruck 438
Hashfunktion 240
Hashindex 366
Hashing 438
HASH-Partitionierung 438
Hashtabelle 366
Hauptverzeichnis 292
have_query_cache 169
HELP *728*
HEX *681*
Hinzufügen von Partitionen 442
Hochverfügbarkeit 41
Homeverzeichnis 244
Host *266*
Hostname 239
Hot Copy → Online-Backup

I

ib_logfile0 202
ib_logfile1 202
ibdata1 201
IDEF1X 75
Identifier
vollständig qualifizierter 119
Identität
Benutzeridentifizierung 239
IF-THEN-ELSE 558
IF-THEN-ELSEIF-ELSE 561, *697*
Import 272, *720*
Import komprimiert *720*
Include Bin Directory in Windows PATH 101
INDEX 138, 463
Index 34
Adaptive Hashindex 200
B-Baum 200, 206, 367
Clustered 200, 372
gruppierter 372
Hashindex 200, 212
nicht-gruppierter 372
Non-Clustered 206, 372
Primärindex 364
räumlicher 378, 379
R-Baum 205, 374, 379
Sekundärindex 365, 372
Spatial-Index 379
INDEX DIRECTORY 434
Indexstruktur 379
Information Hiding 514

INFORMATION_SCHEMA 259, 274, 305, 545
INFORMATION_SCHEMA.PARTITIONS 449
Informationssystem 33
init.d 234
Initialized 617
Inkonsistenz 28, 36
InnoDB 181, 199, 330
reparieren 295
innodb_buffer_pool_size 201, 330, 336
innodb_file_format 203
innodb_file_per_table 201, 202
innodb_flush_log_at_trx_commit 337
innodb_force_recovery 303
innodb_log_file_size 201
INOUT 608
INOUT-Parameter 534
IN-Parameter 531
INSERT 242, 461, *681*
INSERT DELAYED 210, 524
INSERT INTO *668*
INSTALL PLUGIN *725*
Instanz 64, 99, 244
Integrität
Datenintegrität 36
referentielle 133
INTERVAL 612
Interval Field 616
Interval Value 616
INVOKER 467
IP-Adresse 239
IS_DETERMINISTIC 547
ISAM 371
Isolation 181, 186
Isolationslevel 186
ITERATE 565, *698*
Iteratives Vorgehensmodell 55

J

Java 636
Java Database Connectivity → JDBC
JDBC 636
Join → Verbund
Join-Reihenfolge 177

K

Kapselung 469, 492, 513
Kapselungsschicht 492
Kardinalität
Relation 74
Kartografiesystem 34
Katalog 157
KEY 431
key_buffer 343
key_len 396
Key-Cache 338
KEY-Partitionierung 439
KILL 271
KILL CONNECTION *712*
KILL QUERY *711*
Kindknoten 367
Knoten 345
Kommandozeile 107, 113
Kommandozeilenparameter
dynamisch 251
statisch 251
Kommentar 54, 245, 520, 528
einzeiliger 520
mehrzeiliger 520
Kompatibilität 552
Kompilieren 94, 219
Kompilierung 220
Komprimierte Tabellen 203
Konfigurationsdatei 219, 232, 243
global 244
my.cnf 232
sichern 289
Konkurrierende Schreibvorgänge 37
Konnektor 163, 627, 629
nativer 629
Konsistenz 36, 181, 576, 599
Konsole 250
Kontext 252
Kontrollfluss 539
Konzeptionelles Schema 52
Korrektheit 576
Korrelierte Subquery 406
Korruption 299
Krähenfuß-Notation 75
Kreuzprodukt 67
Künstlicher Schlüssel 401
Kurzschreibweise 250

L

LANGUAGE 526
LANGUAGE SQL 540
Larsson, Allan 42
LAST 213
LAST_ALTERED 547
Lastverteilung 345
Latenz 318
Laufzeit 251
LCASE *685*
LEAVE 565, *697*
LEFT *681*
LENGTH *681*
LIKE 385, 414, *681*
LINE 377
Lineare KEY-Partitionierung 440
Linearer Zweierpotenz-Algorithmus 439
Lineares Hashing 439
LINEARRING 377
LINESTRING 377, 378
Linux-User 293
LIST 430
LIST COLUMNS 436
LIST-Paritionierung 435
Lizenz
  Open-Source-Lizenz 45
Lizenzmodell 39
  duales 47
Lizenznehmer 46
LOAD
  Systemwerte 281
LOAD DATA 272, 523, *721*
LOAD TABLE 523
LOAD_FILE *681*
local 277
LOCATE *682*
Lochkarte 24
LOCK 544
LOCK TABLES 194, 464, 523, *687*
Lock-Management 192
Locks 192
  Sperren 37
lock-tables 288
log_bin 354
log_bin_basename 249
log_bin_trust_function_creators 523, 541
Logarithmen *672*
Logging
  logisches 347
Logik
  ternär 122
Login 238
Logischer Operator 559
Log-Tabelle *602*
Lokalität 365
LOOP 565, *698*
Löschanomalie 84
Löschen von gespeicherten Prozeduren 529
Lost Update 179
LOWER *685*
LPAD *682*
LTRIM *682*

M

Maatkit 398
  mk-query-profiler 723
make 225
  install 257
MAKE_SET *682*
Maskieren 506
  der Benutzereingaben 507
Maskierungszeichen 246
Master-Master-Replikation → Replikation
Master-Slave-Replikation → Replikation
max_allowed_packet 249, 275
max_binlog_size *296*
max_connect_errors 266
max_connections 333
MAX_CONNECTIONS_PER_HOUR 476
max_insert_delayed_threads 211
MAX_QUERIES_PER_HOUR 476
max_sp_recursion_depth 595
MAX_UPDATES_PER_HOUR 476
MAX_USER_CONNECTIONS 476
MAXVALUE 433
MBRContains 383, *706*
md5-Hash 497
Mehrbenutzersystem 37
Mehrfaches Fetchen 574
Mehrfach-Zuweisung 553
memcached 216
MEMORY 212
Memory-Tabelle 595
Mengentheorie 57

MERGE 213
Metadaten 25, 306
Minimale Systemanforderungen 94
Modellentwurf 52
MODIFIES SQL DATA 527, 540
mSQL 42
Multidimensionale Eigenschaft 379
MULTILINESTRING 377
MULTIPOINT 377
Multipoint 382
MULTIPOLYGON 377
Multiversion Concurrency Control 199
Munin 281
my.cnf 232, 243, 244
  Block 245
  Optionen 246
MYD-Datei 208
MYI-Datei 208
MyISAM 205, 371
  räumlicher Index 378
myisam_sort_buffer_size 329
myisamcheck 302
myisamchk 208, *718*
myisamchk (Komprimierung) *725*
myisampack (Komprimierung) *725*
mysql 263
MySQL AB 39, 43, 44
MySQL Administrator 263, 314
MySQL Cluster 48
MySQL Community Server 39, 219
MySQL Enterprise Edition 48
MySQL Performance Tuning Script 330
MySQL Proxy 349
MySQL Query Browser 263
MySQL Stored Program Language 514, 516
MySQL Workbench 71, 263, 308, 517, 519
mysql_install_db 229
mysql_secure_installation 230
mysql_upgrade 257
mysqladmin 263, 264
mysqlbinlog 296, *720*
mysqlcheck 263, 301, 302
mysqld 244, 247, 249
MySQL-Daemon 105
MySQLdb 655, *656*
mysqldump 263, 283, 286, 356, 522, *719*
mysqlhotcopy 293, *720*
mysqli 641
mysqlimport 263, 275
MySQL-Konsole 518
MySQL-python 656
mysql-server 103
mysqlslap 263, 304, *723*
MySQLStartupItem.pkg 112
mytop 307, *723*

N

Nagios 269
Name 616
Namensschema 551
Natürlichsprachliche Suche 387
Nebenläufigkeit 178, 598
.NET Framework 94, 645
Netzwerk absichern 498
NEW 602
Newline 273
NO SQL 527, 540
Non_unique 397
Non-repeatable Read 188
Normalform 83
  erste 85
  zweite 87, 88
  dritte 88
  weitere 90
NOT 559
NOT DETERMINISTIC 527
NOT FOUND 580
NOT LIKE *681*
NOT NULL 123
NOT REGEXP *683*
Not_flushed_delayed_rows 210
NOT-Verknüpfung *670*
NOW() 348
NUMERIC_PRECISION 546
NUMERIC_SCALE 546
Numerische Datentypen (Ganzzahlen) *662*
Numerische Datentypen (reelle Zahlen) *662*
Nutzungsrechte 45

O

Object-Relational-Mapping 627
Objektebene 459

OCTET_LENGTH *681*
ODBC 632
ODBC-Konnektor 632
OFF 254
Offline-Backup 284
OLAP 31
OLD 602
OLTP 31
ON 254
ON COMPLETION PRESERVE 614
ON DELETE CASCADE 135
ON SCHEDULE 614
Online Analytical Processing → OLAP
Online Transaction Processing → OLTP
Online-Backup 284
OPEN 571
OPEN CURSOR *700*
Open Database Connectivity → ODBC
Open Geospatial Consortium 377
Open Group 192
Open Source 45, 317
Open Source Initiative 46
OpenGIS Simple Features Specification for SQL 377
openSUSE 103, 105
Operatorgraph 157
Optimierer 157, 163, 172
Optimierung 41
Kompilierung 220
OPTIMIZE *718*
OPTIMIZE PARTITION 445
OPTIMIZE TABLE 207, 302
OPTIMIZE_TABLE 461
optimizer_prune_level 174
optimizer_search_depth 174
Optionen
Wert 250
Zuweisung 250
Optionsdatei → Konfigurationsdatei
Optionsname 246
OR 559
Oracle 39, 44
Originator 617
OR-Verknüpfung *671*
OS X 10.2.x 110
OUT 608
OUTFILE 278
OUT-Parameter 533

P

PACK_KEYS 206
Paging 330
Paketgröße 275
Paketmanager 256
apt 222
Parameter 246, 530
PARAMETER_STYLE 547
Parametrisierte Anweisungen → Prepared Statements
Parser 163
partition 429
PARTITION BY RANGE() 432
Partition Pruning 445
Partition Selection 446
Partitionieren 427
Partitionierung
basierend auf mehreren Spalten 436
horizontale 428
vertikale 428
Partitionierungsschlüssel und UNIQUE-Schlüssel 431
Partitionsfunktion 430
Partitionsschlüssel 430
Partitions-Unterstützung 429
PASSWORD 240, 440, 482, *711*
Passwort
neu setzen 267
root 231, 266
Sicherheit 231
Passwort-Hashing 497
PATH 113, 234
PDO 641
Percona XtraBackup 294
Peripherie-Speicherverwaltung 161
Perl 293, 650
Perl's Database Interface → DBI
perl-DBD-MySQL 650
Pfad → Umgebungsvariable
Phantom-Reads 189
PHP 641
PHP Data Objects → PDO
PING 269, *711*
Pipe 104
Plugin 259, 429
Plugin-API 258
Pluginstatus 259
POINT 377, 378, *705*

Point-in-Time Recovery 298
POLYGON 377
Port 246, 247
Portabilität 578
POSITION *682*
POSIX-Threads 94
possible_keys 395
POW 383
PREPARE 544, *717*
Prepared Statements 507, 544, 630
Primärindex → Index
PRIMARY KEY 401
Privater Schlüssel 501
Privilegientabellen 231
priv-Tabellen 481
proc 522
PROCESS 474
PROCESSLIST 271, *711*
procs_priv 481
Profildatei 235
Profildatei → bashrc
Programmiersprache 41
  prozedurale 516
Projektion 62
Prozedurkopf 525
Prozess → Dämon
Prozessliste 223
Pufferspeicher 265
Python 655

Q

QCache_free_blocks 171
QCache_free_memory 171
Qcache_hits 171
QCache_inserts 171
Qcache_lowmem_prunes 171
Qcache_not_cached 171
Qcache_queries_in_cache 171
QCache_total_blocks 171
Quellcode 46, 94, 220, 237
Quelldistribution 222
query_cache_size 169, 329
query_cache_type 168, 169, 170
query_cache_wlock_invalidate 170
Query-Cache 163, 165, 166, 340
QUOTE *682*

R

RANGE 430
RANGE COLUMNS 436
RANGE-Partitionierung 431
Raumbezogene Erweiterung 377
Raymond, Eric 46
RDBMS 26
READ 193
READ COMMITTED 186, 188
READ UNCOMMITTED 186, 187
read_buffer_size 329
read_rnd_buffer_size 329
READS SQL DATA 527, 540
REBUILD PARTITION 445
Rechte 451, 523
Rechtemanagement 451
Recovery 38
Redo 303
Redundanz 28, 37
Referentielle Integrität 134, 199
REFRESH *710*
REGEXP 385, *683*
Reguläre Ausdrücke 385
Reihen sperren 195
Rekursion 595
Rekursionstiefe 595
Rekursiv 544
Relation 26, 57, 61
Relationale Algebra 57, 61, 157
Relationale-Algebra-Verwaltung 158
Relationenverwaltung 159
Relationship 70
Relay-Log 347
RELEASE SAVEPOINT 185, *687*
RELOAD 266, 475
REORGANIZE PARTITION 443
REPAIR 301, *718*
REPAIR PARTITION 445
REPAIR_TABLE 461
REPEAT *683*
REPEAT UNTIL 565, 567, *698*
REPEATABLE READ 186, 189
REPLACE *683*
REPLICATION CLIENT 475
REPLICATION SLAVE 353, 475
Replikation 41, 345
  asynchrone 347
  Ausfallsicherheit 345

Replikation (Forts.)
Backup 298
Binär-Logging am Master-Knoten 352
CHANGE MASTER 356
Lastverteilung 345
logische 347
Master 346
Master – Slave 349
Master – Slave mit zwischengeschaltetem Slave 350
Master-Master 351
mehrstufige Master-Slave-Replikation 350
Slave 346
Statement-basierte 347
Topologie 346, 348
zeilenweise 347
Repräsentation der Daten 35
REQUIRE 476
RESET QUERY CACHE *724*
RESIGNAL 588, *702*
Restrukturierung 29
RETURN *693*
RETURNS 538
REVERSE *683*
Reverse Engineering 83, 310
Dekompilieren 46
REVOKE 455, 477, *716*
RIGHT *681*
RLIKE *683*
ROLLBACK 183, 544, *686*
Rollback 184
ROLLBACK TO SAVEPOINT 185, *687*
ROUTINE_BODY 547
ROUTINE_CATALOG 546
ROUTINE_COMMENT 547
ROUTINE_DEFINITION 547
ROUTINE_NAME 546
ROUTINE_SCHEMA 546
ROUTINE_TYPE 546
ROUTINES 545
ROUTINES-Tabelle 545
RPAD *682*
RPM 110
RTRIM *682*
Ruby 653
Ruby on Rails 653
ruby-mysql 653
Rückgabe 525
Rückgabeparameter 533
Rückgabewert 538
atomarer 537
Rumpf 525, 538

S

Sakila 39
SAVEPOINT *687*
Savepoints 184
Schleifen 557, 564
Schlüssel 60, 61
Fremdschlüssel 70, 79
künstliche 60
künstlicher Primärschlüssel 62, 120
Primärschlüssel 62, 72
Schlüsselkandidat 62
Superschlüssel 61
zusammengesetzter 87
Schnellstart-Tutorial 219
Schnittstelle 41, 626
Schreibsperre
extern 248
Schwachstellen 254
Scope 251, 555
SCP 292
SECURITY_TYPE 547
Segmentverwaltung 160
Sekundärindex → Index
SELECT 126, 460, 544, *667*
SELECT INTO 544, 553, *699*
SELECT INTO OUTFILE *722*
SELECT VERSION 257, *690*
select_type 395
Selection Push-Down 175
Selektion 62
Semikolon 275
Seq_in_index 397
SERIALIZABLE 191
Server 95, 249
Server-Daemon → Server-Dämon
Server-Dämon 263
Serverhardware 293
Server-ID 354
Serverport 499
Servervariable
anzeigen 270
SESSION 252, 254
Session 251

Sessionkontext 254
Sessionvariable → Sitzungsvariable
SET 252, 552, *690*, *699*
SET AUTOCOMMIT 184, 544, *686*
SET GLOBAL TRANSACTION ISOLATION LEVEL 186
SET SESSION TRANSACTION ISOLATION LEVEL 186
SET TRANSACTION ISOLATION LEVEL *687*
Shell
History 238
Kommandozeileninterpreter 220
SHOW 544, *712*
SHOW CHARACTER SET *731*
SHOW COLLATION *732*
SHOW CREATE 530
SHOW CREATE EVENT 617, *696*
SHOW CREATE FUNCTION *694*
SHOW CREATE PROCEDURE 548, *692*
SHOW CREATE TRIGGER *607*
SHOW CREATE VIEW 464
SHOW DATABASES 117, 475, *664*
SHOW ENGINES *725*
SHOW ERRORS *728*
SHOW EVENTS 615, *696*
SHOW FUNCTION STATUS 545, 548, *694*
SHOW GRANTS 480, *716*
SHOW MASTER STATUS 355, *705*
SHOW OPEN TABLES *690*
SHOW PLUGINS *724*
SHOW PROCEDURE STATUS 545, 548, *692*
SHOW PROFILE *723*
SHOW SLAVE STATUS *705*
SHOW STATUS 306
SHOW TABLES 594, *667*
SHOW TRIGGERS 604, *694*
SHOW VARIABLES 252, *690*
SHOW VIEW 464
SHOW WARNINGS 126, *728*
SHUTDOWN 475
Sicherheit 451, 513
Sicherheits- und Benutzerkonzept 488
Sicherheitsschicht 490
Sicherheits-Updates 254
Sicherung 521
Sicht 493, 522, 618
aktualisierbare 494
eingeschränkte 619
SIGNAL 586, *701*
single-transaction 288
Sitzungsvariable 251, 253
skip_external_locking 247
Skript
Absicherung 229
Initialisierung 229
Startskript 233
Socket 246, 247
Software
Free Software 46
proprietäre 45
quelloffene 45
Solaris 43
sort_buffer_size 254, 329, 341
Sort-Cache 340
SOUNDEX *683*
SOUNDS LIKE *683*
SOURCE 518, *722*
Sourceforge *299*
SPACE *684*
Spaltenebene 458
Spatial extension 377
SPECIFIC_NAME 546
Speichermedium
extern 292
Speicherort 434
Speicherschema *27*
Sperren 192
Spion 238
Sprache 54
Sprechende Namen 54
Sprechende Variablennamen 552
SQL *27*
SQL SECURITY 467, 527
SQL SECURITY DEFINER 527
SQL SECURITY INVOKER 527
SQL_CACHE 169, *724*
SQL_DATA_ACCESS 547
SQL_MODE 547, 605, *607*
SQL_NO_CACHE 168, *724*
SQL_PATH 547
SQL-Einschleusungen 504
SQLEXCEPTION 579
SQL-Injections 504
SQLSTATE 578

SQL-States *728*
SQLWARNING 581
SQRT 383
SSH 499
SSL 499
ssl-ca 501
ssl-cert 501
ssl-key 501
Stallman, Richard 46
Standardfunktionen *672*
Standardkonfiguration 220
Standardwert 554
START SLAVE *704*
START TRANSACTION 184, 544, 598, *686*
Starten
  automatisch 233
  manuell 233
Starts 616
Statement 117, 605
STATUS 269, 616, *690*, *711*
STOP SLAVE *704*
Storage-Engine 131, 164, 195
  BLACKHOLE 260
  Speichermanagement 40
Stored Functions 515, 536
Stored Procedures 290, 515, 524
STRCMP *685*
String-Datentypen *663*
Strings 385
SUBPARTITION BY KEY() 441
Subquerys 404
SUBSTR *684*
SUBSTRING *684*
SUBSTRING_INDEX *684*
Suche
  natürlichsprachliche 139
Sun Microsystems 43
SUPER 252, 266, 473, 523
Surrogatschlüssel → Künstlicher Schlüssel
Swapping 330
Synaptic 103
Syntax-Fehler 526
Syntax-Highlighting 519
System R 155
Systemdienst 229, 233
Systemleichen 496

T

Tabelle 57, 58
  reparieren 301
  sperren 194
  temporäre 593, 594
Tabelle optimieren → OPTIMIZE TABLE
Tabellen-Cache 266
Tabellendump 286, 287
Tabellenebene 458
Tabellenname 58
Tabellenoptimierung 265
Tabellenschema 27
Table 605
table_open_cache 329
tables_priv 481
Tablespace 203
tail 291
tar 221, 292
TCP/IP 95
TEMPORARY 594
Terminal 113
Terminalfenster 223
Ternäre Logik 558
Text-Datentypen *663*
Texteditor 518
Textsammlung 385
thread_cache_size 329
Threads 269, 333
Time zone 616
Timing 605
TO_BASE64 *684*
TO_DAYS(datum) 434
Tools 263
Transaktion 37, 178, 180, 598
Transaktionsanweisungen 544
Transaktions-Befehle 608
Transitive Abhängigkeit 88
Transparenz 604
Treiber 628
TRIGGER 466, 605
Trigger 515, 522, 599, 605
TRIGGER_CATALOG 606
TRIGGER_NAME 606
TRIGGER_SCHEMA 606
TRIM *684*
true 558
TrueWORM 285
TRUNCATE *669*

TRUNCATE TABLE 463
Trunkierung 415
Tuning 319
Tupel 59
Tupelverwaltung 159
Type 616

U

Überladen 556
Ubuntu 103
Ubuntu Linux 103
UCASE *685*
Umgebungsvariable 113, 234, 264
  Pfadangaben 234
  Pfadeinstellungen 235
Umkreissuche 377, 380
UML 75
Umwandlungen von Datums- und Zeitangaben *676*
Unbounded Selects 536
Undo 295, 303
Undo-Handler 576
Ungebundene Selects 536, 608
Ungebundene SELECT-Statements 593
UNHEX *685*
UNINSTALL PLUGIN *725*
Union 66
UNIREG 42
UNIX_TIMESTAMP(datum) 434
UNLOCK TABLES 194, 523, 544, *688*
Unterpartition 440
UNTIL 567
UPDATE 242, 461, *668*
  Statement 267
Update 254
Update-fähige Sichten 622
UPGRADE *719*
Upgrade 255, 258
UPPER *685*
Uptime 269
Urheberrecht 45
USAGE 474
USE *665*
user 481
Usermanagement 237, 242

V

VALUES 124
VALUES IN() 435
VALUES LESS THAN 431
Variable 277, 550
  globale 251
  lokale 251
Variable tx_isolation 186
Verbindung
  aktiv 251
  verschlüsselte 499
Verbindung → Session
Verbindungsüberprüfung 453
Verbund 66, 67
Vereinigung 66
Verifizierung 239
Vermittlungsschicht 632
Verschachtelte Anfragen → Subquerys
Verschlüsseln 498
Verschlüsselungsalgorithmus 501
Versionierung
  Versionsnummer 45
Verwendung von Cursors 571
Verzahnungen von Operationen 179
View → Sicht
Virtuelle Tabellen 618
Virtueller Einbenutzerbetrieb 181
Volltextindex 139, 385
Volltextsuche 385, 388, *702*
  Abfrageerweiterung 391
  boolesche Operatoren 388
  IN BOOLEAN MODE 388
  Länge der Wörter 390
  MATCH() ... AGAINST() 386
  natürlichsprachliche Abfrage 386
  Probleme und Einschränkungen 393
  Relevanzwert 387
  Stoppwörter 389
  Stoppwörter-Liste 390

W

wait_timeout 252
Waiting for next activation 617
Waiting for scheduler to stop 617
Waiting on empty queue 617
Warm Copy 285
Warnung 576

Wartungsarbeiten 516
Web 2.0 33
Web-Programmierung 641
WEIGHT_STRING *685*
Well-Known Binary → WKB
Well-Known Text → WKT
WHILE 565, 568, *698*
Widenius, Michael 42
Wiederherstellung 281
Wiederherstellungsmethode 289
Wiederkehrende Aufgaben 514
Wiederverwendung von Code 513
Wildcard 126
Wildcard-Character 415
Winkelfunktionen *671*
WITH 476
WITH GRANT OPTION 455
--with-partition 430
WKB 378
WKT 378
Workflows 496
WRITE 193
Wurzelknoten 367

X

X509 502
XA-Transaktionen 191
XML 27
XOR 559
XOR-Verknüpfung *671*
Xtrabackup *720*
xtrabackup *295*

Y

YaST 105
YEAR(datum) 434

Z

Zahlensysteme *672*
Zeichenketten 385
Zeiger 569, 571
Zeilenebene 458, 492
Zeit *663*
Zeit, aktuelle *676*
Zeitangaben, Formatierung *678*
Zeitarithmetik *675*
Zeiteinheiten *673*
Zeitformate *674*
Zeitstempel 522, 611
Zertifikate 500
Zugriff
  gekapselter 489
Zugriffsberechtigungssystem 453
Zugriffsphase 453
Zusammengesetztes Partitionieren 440
Zustand
  konsistenter 295
Zuweisung 552
Zuweisungszeichen 249
Zwischentabelle 79
zypper 108