Developer’s Library

ESSENTIAL REFERENCES FOR PROGRAMMING PROFESSIONALS

Developer’s Library books are designed to provide practicing programmers with unique, high-quality references and tutorials on the programming languages and technologies they use in their daily work.

All books in the Developer’s Library are written by expert technology practitioners who are especially skilled at organizing and presenting information in a way that's useful for other programmers.

Key titles include some of the best, most widely acclaimed books within their topic areas:

- **PHP & MySQL Web Development**
  - Luke Welling & Laura Thomson

- **MySQL**
  - Paul DuBois

- **Linux Kernel Development**
  - Robert Love

- **Python Essential Reference**
  - David Beazley

- **PostgreSQL**
  - Korry Douglas

- **C++ Primer Plus**
  - Stephen Prata

Developer’s Library books are available in print and in electronic formats at most retail and online bookstores, as well as by subscription from Safari Books Online at [safari.informit.com](http://safari.informit.com)
Contents at a Glance

Introduction  1

Part I: General MySQL Use
  1  Getting Started with MySQL  11
  2  Using SQL to Manage Data  95
  3  Data Types  179
  4  Views and Stored Programs  261
  5  Query Optimization  277

Part II: Using MySQL Programming Interfaces
  6  Introduction to MySQL Programming  307
  7  Writing MySQL Programs Using C  319
  8  Writing MySQL Programs Using Perl DBI  395
  9  Writing MySQL Programs Using PHP  485

Part III: MySQL Administration
  10  Introduction to MySQL Administration  537
  11  The MySQL Data Directory  543
  12  General MySQL Administration  563
  13  Security and Access Control  645
  14  Database Maintenance, Backups, and Replication  699

Part IV: Appendixes
  A  Software Required to Use This Book  735
  B  Data Type Reference  747
  C  Operator and Function Reference  763
  D  System, Status, and User Variable Reference  835
  E  SQL Syntax Reference  897
  F  MySQL Program Reference  999
  Index  1073

Note: Appendixes G, H, and I are located online and are accessible either by registering this book at informit.com/register or by visiting www.kitebird.com/mysql-book.

G  C API Reference  Web: 1073
H  Perl DBI API Reference  Web: 1129
I  PHP API Reference  Web: 1157
Table of Contents

Introduction 1
  Why Choose MySQL? 2
  What You Can Expect from This Book 4
  Road Map to This Book 4
    Part I: General MySQL Use 4
    Part II: Using MySQL Programming Interfaces 5
    Part III: MySQL Administration 5
    Part IV: Appendixes 6
  How to Read This Book 6
  Versions of Software Covered in This Book 7
  Conventions Used in This Book 9
  Additional Resources 9

Part I: General MySQL Use

1 Getting Started with MySQL 11
  1.1 How MySQL Can Help You 11
  1.2 A Sample Database 14
    1.2.1 The U.S. Historical League Project 15
    1.2.2 The Grade-Keeping Project 17
    1.2.3 How the Sample Database Applies to You 17
  1.3 Basic Database Terminology 18
    1.3.1 Structural Terminology 18
    1.3.2 Query Language Terminology 20
    1.3.3 MySQL Architectural Terminology 21
  1.4 A MySQL Tutorial 22
    1.4.1 Obtaining the Sample Database Distribution 23
    1.4.2 Preliminary Requirements 23
    1.4.3 Establishing and Terminating Connections to the MySQL Server 25
    1.4.4 Executing SQL Statements 27
    1.4.5 Creating a Database 30
    1.4.6 Creating Tables 31
    1.4.7 Adding New Rows 49
    1.4.8 Resetting the sampdb Database to a Known State 53
    1.4.9 Retrieving Information 54
1.4.10 Deleting or Updating Existing Rows  85
1.5  Tips for Interacting with mysql  87
1.5.1  Simplifying the Connection Process  87
1.5.2  Issuing Statements with Less Typing  90
1.6  Where to Now?  94

2  Using SQL to Manage Data  95
2.1  The Server SQL Mode  96
2.2  MySQL Identifier Syntax and Naming Rules  97
2.3  Case Sensitivity in SQL Statements  99
2.4  Character Set Support  101
  2.4.1  Specifying Character Sets  102
  2.4.2  Determining Character Set Availability and Current Settings  103
  2.4.3  Unicode Support  104
2.5  Selecting, Creating, Dropping, and Altering Databases  105
  2.5.1  Selecting Databases  105
  2.5.2  Creating Databases  106
  2.5.3  Dropping Databases  107
  2.5.4  Altering Databases  107
2.6  Creating, Dropping, Indexing, and Altering Tables  107
  2.6.1  Storage Engine Characteristics  108
  2.6.2  Creating Tables  113
  2.6.3  Dropping Tables  121
  2.6.4  Indexing Tables  122
  2.6.5  Altering Table Structure  127
2.7  Obtaining Database Metadata  130
  2.7.1  Obtaining Metadata with SHOW  130
  2.7.2  Obtaining Metadata with INFORMATION_SCHEMA  132
  2.7.3  Obtaining Metadata from the Command Line  135
2.8  Performing Multiple-Table Retrievals with Joins  136
  2.8.1  Inner Joins  137
  2.8.2  Qualifying References to Columns from Joined Tables  139
  2.8.3  Left and Right (Outer) Joins  139
2.9  Performing Multiple-Table Retrievals with Subqueries  143
  2.9.1  Subqueries with Relative Comparison Operators  144
  2.9.2  IN and NOT IN Subqueries  145
  2.9.3  ALL, ANY, and SOME Subqueries  146
## Contents

2.9.4  **EXISTS and NOT EXISTS Subqueries**  147  
2.9.5  **Correlated Subqueries**  148  
2.9.6  **Subqueries in the FROM Clause**  149  
2.9.7  **Rewriting Subqueries as Joins**  149  
2.10  **Performing Multiple-Table Retrievals with UNION**  151  
2.11  **Multiple-Table Deletes and Updates**  154  
2.12  **Performing Transactions**  156  
2.12.1  **Using Transactions to Ensure Safe Statement Execution**  157  
2.12.2  **Using Transaction Savepoints**  161  
2.12.3  **Transaction Isolation**  162  
2.13  **Foreign Keys and Referential Integrity**  164  
2.14  **Using FULLTEXT Searches**  170  
2.14.1  **Natural Language FULLTEXT Searches**  172  
2.14.2  **Boolean Mode FULLTEXT Searches**  174  
2.14.3  **Query Expansion FULLTEXT Searches**  175  
2.14.4  **Configuring the FULLTEXT Search Engine**  176  

### 3  Data Types  179

3.1  **Data Value Categories**  181  
3.1.1  **Numeric Values**  181  
3.1.2  **String Values**  182  
3.1.3  **Temporal (Date and Time) Values**  191  
3.1.4  **Spatial Values**  191  
3.1.5  **Boolean Values**  192  
3.1.6  **The NULL Value**  192  
3.2  **MySQL Data Types**  192  
3.2.1  **Data Type Overview**  193  
3.2.2  **Specifying Column Types in Table Definitions**  194  
3.2.3  **Specifying Column Default Values**  196  
3.2.4  **Numeric Data Types**  196  
3.2.5  **String Data Types**  204  
3.2.6  **Temporal (Date and Time) Data Types**  218  
3.3  **How MySQL Handles Invalid Data Values**  228  
3.4  **Working with Sequences**  230  
3.4.1  **General AUTO_INCREMENT Properties**  230  
3.4.2  **Storage Engine-Specific AUTO_INCREMENT Properties**  232  
3.4.3  **Issues to Consider with AUTO_INCREMENT Columns**  235
3.4.4 Tips for Working with AUTO_INCREMENT Columns 235
3.4.5 Generating Sequences Without AUTO_INCREMENT 237
3.5 Expression Evaluation and Type Conversion 239
  3.5.1 Writing Expressions 240
  3.5.2 Type Conversion 247
3.6 Choosing Data Types 255
  3.6.1 What Kind of Values Will the Column Hold? 257
  3.6.2 Do Your Values Lie Within Some Particular Range? 259

4 Views and Stored Programs 261
  4.1 Using Views 262
  4.2 Using Stored Programs 265
    4.2.1 Compound Statements and Statement Delimiters 266
    4.2.2 Stored Functions and Procedures 268
    4.2.3 Triggers 272
    4.2.4 Events 274
  4.3 Security for Views and Stored Programs 275

5 Query Optimization 277
  5.1 Using Indexing 277
    5.1.1 Benefits of Indexing 278
    5.1.2 Costs of Indexing 281
    5.1.3 Choosing Indexes 281
  5.2 The MySQL Query Optimizer 285
    5.2.1 How the Optimizer Works 286
    5.2.2 Using EXPLAIN to Check Optimizer Operation 290
  5.3 Choosing Data Types for Efficient Queries 296
  5.4 Choosing Table Storage Formats for Efficient Queries 299
  5.5 Loading Data Efficiently 300
  5.6 Scheduling, Locking, and Concurrency 303

Part II: Using MySQL Programming Interfaces

6 Introduction to MySQL Programming 307
  6.1 Why Write Your Own MySQL Programs? 307
  6.2 APIs Available for MySQL 310
    6.2.1 The C API 311
    6.2.2 The Perl DBI API 311
    6.2.3 The PHP API 313
6.3 Choosing an API  314
  6.3.1 Execution Environment  314
  6.3.2 Performance  315
  6.3.3 Development Time  316
  6.3.4 Portability  317

7 Writing MySQL Programs Using C  319
  7.1 Compiling and Linking Client Programs  320
  7.2 Connecting to the Server  323
  7.3 Handling Errors and Processing Command Options  327
    7.3.1 Checking for Errors  327
    7.3.2 Getting Connection Parameters at Runtime  330
    7.3.3 Incorporating Option Processing into a Client Program  344
  7.4 Processing SQL Statements  348
    7.4.1 Handling Statements That Modify Rows  350
    7.4.2 Handling Statements That Return a Result Set  351
    7.4.3 A General-Purpose Statement Handler  354
    7.4.4 Alternative Approaches to Statement Processing  356
    7.4.5 mysql_store_result() Versus mysql_use_result()  357
    7.4.6 Using Result Set Metadata  359
    7.4.7 Encoding Special Characters and Binary Data  364
  7.5 An Interactive Statement-Execution Program  368
  7.6 Writing Clients That Include SSL Support  370
  7.7 Using Multiple-Statement Execution  375
  7.8 Using Server-Side Prepared Statements  377
  7.9 Using Prepared CALL Support  389

8 Writing MySQL Programs Using Perl DBI  395
  8.1 Perl Script Characteristics  396
  8.2 Perl DBI Overview  396
    8.2.1 DBI Data Types  396
    8.2.2 A Simple DBI Script  397
    8.2.3 Handling Errors  402
    8.2.4 Handling Statements That Modify Rows  406
    8.2.5 Handling Statements That Return a Result Set  407
    8.2.6 Quoting Special Characters in Statement Strings  416
8.2.7 Placeholders and Prepared Statements 419
8.2.8 Binding Query Results to Script Variables 422
8.2.9 Specifying Connection Parameters 423
8.2.10 Debugging 426
8.2.11 Using Result Set Metadata 430
8.2.12 Performing Transactions 434

8.3 Putting DBI to Work 436
8.3.1 Generating the Historical League Directory 436
8.3.2 Sending Membership Renewal Notices 442
8.3.3 Historical League Member Entry Editing 448
8.3.4 Finding Historical League Members with Common Interests 454
8.3.5 Putting the Historical League Directory Online 455

8.4 Using DBI in Web Applications 459
8.4.1 Setting Up Apache for CGI Scripts 460
8.4.2 A Brief CGI.pm Primer 461
8.4.3 Connecting to the MySQL Server from Web Scripts 468
8.4.4 A Web-Based Database Browser 471
8.4.5 A Grade-Keeping Project Score Browser 475
8.4.6 Historical League Common-Interest Searching 479

9 Writing MySQL Programs Using PHP 485
9.1 PHP Overview 487
9.1.1 A Simple PHP Script 489
9.1.2 Using PHP Library Files for Code Encapsulation 492
9.1.3 A Simple Data-Retrieval Page 497
9.1.4 Processing Statement Results 500
9.1.5 Testing for NULL Values in Query Results 504
9.1.6 Using Prepared Statements 505
9.1.7 Using Placeholders to Handle Data Quoting Issues 505
9.1.8 Handling Errors 507

9.2 Putting PHP to Work 509
9.2.1 An Online Score-Entry Application 510
9.2.2 Creating an Interactive Online Quiz 522
9.2.3 Historical League Online Member Entry Editing 528
Part III: MySQL Administration

10 Introduction to MySQL Administration  537
   10.1 MySQL Components  538
   10.2 General MySQL Administration  539
   10.3 Access Control and Security  540
   10.4 Database Maintenance, Backups, and Replication  540

11 The MySQL Data Directory  543
   11.1 The Data Directory Location  544
   11.2 Structure of the Data Directory  545
      11.2.1 How the MySQL Server Provides Access to Data  546
      11.2.2 Representation of Databases in the Filesystem  547
      11.2.3 Representation of Tables in the Filesystem  548
      11.2.4 Representation of Views and Triggers in the Filesystem  549
      11.2.5 How SQL Statements Map onto Table File Operations  549
      11.2.6 Operating System Constraints on Database Object Names  550
      11.2.7 Factors That Affect Maximum Table Size  551
      11.2.8 Implications of Data Directory Structure for System Performance  553
      11.2.9 MySQL Status and Log Files  554
   11.3 Relocating Data Directory Contents  556
      11.3.1 Relocation Methods  557
      11.3.2 Relocation Precautions  558
      11.3.3 Assessing the Effect of Relocation  558
      11.3.4 Relocating the Entire Data Directory  559
      11.3.5 Relocating Individual Databases  559
      11.3.6 Relocating Individual Tables  560
      11.3.7 Relocating the InnoDB System Tablespace  561
      11.3.8 Relocating Status and Log Files  561

12 General MySQL Administration  563
   12.1 Securing a New MySQL Installation  564
      12.1.1 Establishing Passwords for the Initial MySQL Accounts  564
      12.1.2 Setting Up Passwords for Additional Servers  569
   12.2 Arranging for MySQL Server Startup and Shutdown  570
      12.2.1 Running the MySQL Server On Unix  570
      12.2.2 Running the MySQL Server On Windows  575
      12.2.3 Specifying Server Startup Options  577
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.2.4 Controlling How the Server Listens for Connections</td>
<td>579</td>
</tr>
<tr>
<td>12.2.5 Stopping the Server</td>
<td>580</td>
</tr>
<tr>
<td>12.2.6 Regaining Control of the Server When You Cannot Connect to It</td>
<td>581</td>
</tr>
<tr>
<td>12.3 Using System and Status Variables</td>
<td>583</td>
</tr>
<tr>
<td>12.3.1 Checking and Setting System Variable Values</td>
<td>584</td>
</tr>
<tr>
<td>12.3.2 Checking Status Variable Values</td>
<td>588</td>
</tr>
<tr>
<td>12.4 The Plugin Interface</td>
<td>589</td>
</tr>
<tr>
<td>12.5 Storage Engine Configuration</td>
<td>593</td>
</tr>
<tr>
<td>12.5.1 Selecting Storage Engines</td>
<td>593</td>
</tr>
<tr>
<td>12.5.2 Selecting a Default Storage Engine</td>
<td>594</td>
</tr>
<tr>
<td>12.5.3 Configuring the InnoDB Storage Engine</td>
<td>594</td>
</tr>
<tr>
<td>12.6 Globalization Issues</td>
<td>601</td>
</tr>
<tr>
<td>12.6.1 Configuring Time Zone Support</td>
<td>601</td>
</tr>
<tr>
<td>12.6.2 Selecting the Default Character Set and Collation</td>
<td>603</td>
</tr>
<tr>
<td>12.6.3 Selecting the Language for Error Messages</td>
<td>604</td>
</tr>
<tr>
<td>12.6.4 Selecting the Locale</td>
<td>604</td>
</tr>
<tr>
<td>12.7 Server Tuning</td>
<td>605</td>
</tr>
<tr>
<td>12.7.1 General-Purpose System Variables for Server Tuning</td>
<td>606</td>
</tr>
<tr>
<td>12.7.2 Storage Engine Tuning</td>
<td>609</td>
</tr>
<tr>
<td>12.7.3 Using the Query Cache</td>
<td>614</td>
</tr>
<tr>
<td>12.7.4 Hardware Optimizations</td>
<td>616</td>
</tr>
<tr>
<td>12.8 Server Logs</td>
<td>617</td>
</tr>
<tr>
<td>12.8.1 The Error Log</td>
<td>620</td>
</tr>
<tr>
<td>12.8.2 The General Query Log</td>
<td>621</td>
</tr>
<tr>
<td>12.8.3 The Slow Query Log</td>
<td>621</td>
</tr>
<tr>
<td>12.8.4 The Binary Log</td>
<td>622</td>
</tr>
<tr>
<td>12.8.5 The Relay Log</td>
<td>624</td>
</tr>
<tr>
<td>12.8.6 Using Log Tables</td>
<td>624</td>
</tr>
<tr>
<td>12.8.7 Log Management</td>
<td>625</td>
</tr>
<tr>
<td>12.9 Running Multiple Servers</td>
<td>632</td>
</tr>
<tr>
<td>12.9.1 General Multiple Server Issues</td>
<td>632</td>
</tr>
<tr>
<td>12.9.2 Configuring and Compiling Different Servers</td>
<td>635</td>
</tr>
<tr>
<td>12.9.3 Strategies for Specifying Startup Options</td>
<td>636</td>
</tr>
<tr>
<td>12.9.4 Using <code>mysqld_multi</code> for Server Management</td>
<td>637</td>
</tr>
<tr>
<td>12.9.5 Running Multiple Servers on Windows</td>
<td>639</td>
</tr>
<tr>
<td>12.9.6 Running Clients of Multiple Servers</td>
<td>641</td>
</tr>
<tr>
<td>12.10 Updating MySQL</td>
<td>642</td>
</tr>
</tbody>
</table>
13 Security and Access Control 645
  13.1 Securing Filesystem Access to MySQL 646
    13.1.1 How to Steal Data 647
    13.1.2 Securing Your MySQL Installation 648
  13.2 Managing MySQL User Accounts 654
    13.2.1 High-Level MySQL Account Management 655
    13.2.2 Granting Privileges 660
    13.2.3 Displaying Account Privileges 671
    13.2.4 Revoking Privileges 671
    13.2.5 Changing Passwords or Resetting Lost Passwords 672
    13.2.6 Avoiding Access-Control Risks 673
    13.2.7 Pluggable Authentication and Proxy Users 676
  13.3 Grant Table Structure and Contents 679
    13.3.1 Grant Table Scope-of-Access Columns 683
    13.3.2 Grant Table Privilege Columns 683
    13.3.3 Grant Table Authentication Columns 684
    13.3.4 Grant Table SSL-Related Columns 685
    13.3.5 Grant Table Resource Management Columns 685
  13.4 How the Server Controls Client Access 686
    13.4.1 Scope Column Contents 687
    13.4.2 Statement Access Verification 689
    13.4.3 Scope Column Matching Order 690
    13.4.4 A Privilege Puzzle 691
  13.5 Setting Up Secure Connections Using SSL 694

14 Database Maintenance, Backups, and Replication 699
  14.1 Principles of Preventive Maintenance 699
  14.2 Performing Database Maintenance with the Server Running 701
    14.2.1 Locking Individual Tables for Read-Only or Read/Write Access 702
    14.2.2 Locking All Databases for Read-Only Access 705
  14.3 General Preventive Maintenance 705
    14.3.1 Using the Server’s Auto-Recovery Capabilities 706
    14.3.2 Scheduling Preventive Maintenance 706
  14.4 Making Database Backups 707
    14.4.1 Storage Engine Portability Characteristics 709
    14.4.2 Making Text Backups with mysqldump 711
    14.4.3 Making Binary Database Backups 714
    14.4.4 Backing Up InnoDB Tables 715
B.2 String Types 753
  B.2.1 Binary String Types 755
  B.2.2 Nonbinary String Types 756
  B.2.3 ENUM and SET Types 758
B.3 Temporal (Date and Time) Types 759

C Operator and Function Reference 763
C.1 Operators 764
  C.1.1 Operator Precedence 764
  C.1.2 Grouping Operators 765
  C.1.3 Arithmetic Operators 766
  C.1.4 Comparison Operators 768
  C.1.5 Bit Operators 773
  C.1.6 Logical Operators 774
  C.1.7 Cast Operators 775
  C.1.8 Pattern-Matching Operators 776
C.2 Functions 780
  C.2.1 Comparison Functions 781
  C.2.2 Cast Functions 783
  C.2.3 Numeric Functions 784
  C.2.4 String Functions 789
  C.2.5 Date and Time Functions 802
  C.2.6 Summary Functions 817
  C.2.7 Security and Compression Functions 821
  C.2.8 Advisory Locking Functions 824
  C.2.9 IP Address Functions 826
  C.2.10 XML Functions 828
  C.2.11 Spatial Functions 828
  C.2.12 Miscellaneous Functions 829

D System, Status, and User Variable Reference 835
D.1 System Variables 835
  D.1.1 InnoDB System Variables 870
D.2 Status Variables 881
  D.2.1 InnoDB Status Variables 888
  D.2.2 Query Cache Status Variables 891
  D.2.3 SSL Status Variables 892
D.3 User-Defined Variables 894
E  SQL Syntax Reference  897
   E.1 SQL Statement Syntax (Noncompound Statements)  898
   E.2 SQL Statement Syntax (Compound Statements)  987
       E.2.1 Control Structure Statements  987
       E.2.2 Declaration Statements  989
       E.2.3 Cursor Statements  991
       E.2.4 Condition-Handling Statements  992
   E.3 Comment Syntax  996

F  MySQL Program Reference  999
   F.1 Displaying a Program’s Help Message  1000
   F.2 Specifying Program Options  1001
       F.2.1 Standard MySQL Program Options  1003
       F.2.2 Option Files  1007
       F.2.3 Environment Variables  1011
   F.3 myisamchk  1013
       F.3.1 Standard Options Supported by myisamchk  1014
       F.3.2 Options Specific to myisamchk  1015
       F.3.3 Variables for myisamchk  1018
   F.4 mysql  1019
       F.4.1 Standard Options Supported by mysql  1021
       F.4.2 Options Specific to mysql  1021
       F.4.3 Variables for mysql  1025
       F.4.4 mysql Commands  1026
       F.4.5 mysql Prompt Definition Sequences  1028
   F.5 mysql.server  1030
       F.5.1 Options Supported by mysql.server  1030
   F.6 mysql_config  1030
       F.6.1 Options Specific to mysql_config  1031
   F.7 mysql_install_db  1031
       F.7.1 Standard Options Supported by mysql_install_db  1032
       F.7.2 Options Specific to mysql_install_db  1032
   F.8 mysql_upgrade  1033
       F.8.1 Standard Options Supported by mysql_upgrade  1033
       F.8.2 Options Specific to mysql_upgrade  1033
   F.9 mysqladmin  1034
       F.9.1 Standard Options Supported by mysqladmin  1034
F.9.2 Options Specific to mysqladmin 1034
F.9.3 Variables for mysqladmin 1035
F.9.4 mysqladmin Commands 1035
F.10 mysqlbinlog 1038
   F.10.1 Standard Options Supported by mysqlbinlog 1038
   F.10.2 Options Specific to mysqlbinlog 1038
   F.10.3 Variables for mysqlbinlog 1041
F.11 mysqlcheck 1041
   F.11.1 Standard Options Supported by mysqlcheck 1042
   F.11.2 Options Specific to mysqlcheck 1042
F.12 mysqld 1045
   F.12.1 Standard Options Supported by mysqld 1046
   F.12.2 Options Specific to mysqld 1046
   F.12.3 Variables for mysqld 1056
F.13 mysqld_multi 1056
   F.13.1 Standard Options Supported by mysqld_multi 1057
   F.13.2 Options Specific to mysqld_multi 1057
F.14 mysqld_safe 1058
   F.14.1 Standard Options Supported by mysqld_safe 1058
   F.14.2 Options Specific to mysqld_safe 1058
F.15 mysqldump 1060
   F.15.1 Standard Options Supported by mysqldump 1060
   F.15.2 Options Specific to mysqldump 1061
   F.15.3 Data Format Options for mysqldump 1067
   F.15.4 Variables for mysqldump 1068
F.16 mysqlimport 1068
   F.16.1 Standard Options Supported by mysqlimport 1068
   F.16.2 Options Specific to mysqlimport 1069
   F.16.3 Data Format Options for mysqlimport 1070
F.17 mysqlshow 1070
   F.17.1 Standard Options Supported by mysqlshow 1071
   F.17.2 Options Specific to mysqlshow 1071
F.18 perror 1072
   F.18.1 Standard Options Supported by perror 1072

Note: Appendixes G, H, and I are located online and are accessible either by registering this book at informit.com/register or by visiting www.kitebird.com/mysql-book.
G   C API Reference   1073
   G.1 Compiling and Linking   1074
   G.2 C API Data Structures   1075
      G.2.1 Scalar Data Types   1075
      G.2.2 Nonscalar Data Structures   1076
      G.2.3 Accessor Macros   1087
   G.3 C API Functions   1088
      G.3.1 Client Library Initialization and Termination Routines   1088
      G.3.2 Connection Management Routines   1089
      G.3.3 Error-Reporting Routines   1101
      G.3.4 Statement Construction and Execution Routines   1102
      G.3.5 Result Set Processing Routines   1104
      G.3.6 Multiple Result Set Routines   1113
      G.3.7 Information Routines   1113
      G.3.8 Transaction Control Routines   1116
      G.3.9 Prepared Statement Routines   1116
      G.3.10 Administrative Routines   1125
      G.3.11 Threaded Client Routines   1126
      G.3.12 Debugging Routines   1127

H   Perl DBI API Reference   1129
   H.1 Writing Scripts   1130
   H.2 DBI Methods   1130
      H.2.1 DBI Class Methods   1132
      H.2.2 Database-Handle Methods   1137
      H.2.3 Statement-Handle Methods   1142
      H.2.4 General Handle Methods   1146
      H.2.5 MySQL-Specific Administrative Methods   1147
   H.3 DBI Utility Functions   1148
   H.4 DBI Attributes   1149
      H.4.1 Database-Handle Attributes   1149
      H.4.2 General Handle Attributes   1149
      H.4.3 MySQL-Specific Database-Handle Attributes   1150
      H.4.4 Statement-Handle Attributes   1152
H.4.5 MySQL-Specific Statement-Handle Attributes 1154
H.4.6 Dynamic Attributes 1155
H.5 DBI Environment Variables 1156

I PHP API Reference 1157
I.1 Writing PHP Scripts 1157
I.2 PDO Classes 1158
I.3 PDO Methods 1159
  I.3.1 PDO Class Methods 1159
  I.3.2 PDOStatement Object Methods 1166
  I.3.3 PDOException Object Methods 1172
  I.3.4 PDO Constants 1173

Index 1175
About the Author

Paul DuBois is a writer, database administrator, and leader in the open source and MySQL communities. He has contributed to the online documentation for MySQL and is the author of MySQL and Perl for the Web (New Riders), MySQL Cookbook, Using csh and tcsh, and Software Portability with imake (O'Reilly). He is currently a technical writer with the MySQL documentation team at Oracle Corporation.

Acknowledgments

My technical reviewer, Stephen Frein, provided good insights and suggestions for improvement. In addition, because this edition would not have been possible without the previous ones, my continued thanks go to everyone listed in those editions who served as technical reviewer or who patiently answered my questions.

The staff at Pearson responsible for this edition were Mark Taber, acquisitions editor; Tonya Simpson, project editor; Sarah Kearns, copy editor; Kim Scott, compositor; Jess DeGabriele, proofreader; Heather McNeill, indexer; and Chuti Prasertsith, cover designer. My thanks to each of them.

Thanks to my wife Karen for her support and encouragement throughout the production of this edition.
We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we’re doing right, what we could do better, what areas you’d like to see us publish in, and any other words of wisdom you’re willing to pass our way.

You can email or write directly to let us know what you did or didn’t like about this book—as well as what we can do to make our books stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book’s title and author, as well as your name and phone or email address.

Email: feedback@developers-library.info
Mail: Reader Feedback
Addison-Wesley Developer’s Library
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at www.informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

Your purchase of this book includes access to a free online edition for 45 days through the Safari Books Online subscription service. Details are on the last page of this book.
Introduction

A relational database management system (RDBMS) is an essential tool in many environments, from uses in business, research, and educational contexts, to content delivery on the Internet. However, despite the importance of a good database system for managing and accessing information resources, many organizations have found them to be out of reach of their financial resources. Historically, database systems have been an expensive proposition, with vendors charging healthy fees both for software and for support. Also, because database engines often had substantial hardware requirements to run with any reasonable performance, the cost was even greater.

Times have changed, on both the hardware and software sides of the picture. Small desktop systems and servers are inexpensive but powerful, and there is a thriving movement devoted to writing high-performance operating systems for them. These operating systems are available free over the Internet or at the cost of an inexpensive CD. They include several BSD Unix derivatives and several distributions of Linux.

Production of free operating systems has proceeded in concert with—and to a large extent has been made possible by—the development of freely available Open Source tools like the gcc GNU C compiler; Apache, the most widely used Web server on the Internet; and well-established general-purpose scripting languages such as Perl, PHP, Python, and Ruby. These all stand in contrast to proprietary solutions that lock you into high-priced products from vendors that don’t even provide source code.

Database software has become more accessible, too, and Open Source database systems are freely available. One of the most important is MySQL, a SQL client/server relational database management system originating from Scandinavia. MySQL includes an SQL server, client programs for accessing the server, administrative tools, and a programming interface for writing your own programs.

MySQL’s roots begin in 1979, with the UNIREG database tool created by Michael “Monty” Widenius for the Swedish company TcX. In 1994, TcX began searching for an RDBMS with an SQL interface for use in developing Web applications. Commercial servers tested were all found too slow for TcX’s large tables, and the freely available mSQL lacked features that TcX required. Consequently, Monty began developing a new server.
In 1995, David Axmark of Detron HB began to push for TcX to release MySQL on the Internet. David also worked on the documentation and on getting MySQL to build with the GNU configuration auto-tools. MySQL 3.11.1 was unleashed on the world in 1996 in the form of binary distributions for Linux and Solaris. The company MySQL AB was formed to provide distributions of MySQL and to offer commercial services. In 2008, Sun Microsystems acquired MySQL AB, and in 2010, Oracle acquired Sun. Today, MySQL is available in both binary and source form and works on many more platforms.

Initially, MySQL became widely popular because of its speed and simplicity. But there was criticism, too: It lacked features such as transactions and foreign key support. MySQL continued to develop, adding not only those features but others such as replication, subqueries, stored routines, triggers, and views.

These capabilities take MySQL into the realm of enterprise applications. As a result, people who once would have considered only “big iron” database systems for their applications now give serious consideration to MySQL, which runs on anything from modest hardware all the way up to enterprise servers. Its performance rivals any database system you care to put up against it, and it can handle large databases with billions of rows. In the business world, MySQL’s presence continues to increase as companies discover it capable of handling their database needs, with the cost for commercial licensing and support a fraction of what they are used to paying.

MySQL lies squarely within the picture that unfolds before us: freely available operating systems running on powerful but inexpensive hardware, putting substantial processing power and capabilities in the hands of businesses and individuals on a wider variety of systems than ever before. This lowering of the economic barriers to computing puts the power of a high-performance RDBMS to work for more organizations than at any time in the past, for very little cost. This is true for individuals as well. For example, I use MySQL with Perl, PHP, and Apache on my Apple laptop running Mac OS X. This enables me to carry my work with me anywhere. Total cost: the cost of the laptop.

**Why Choose MySQL?**

Several free or low-cost database management systems are available from which to choose, such as MySQL, PostgreSQL, or SQLite. When you compare MySQL with other database systems, think about what’s most important to you. Performance, features (such as SQL conformance or extensions), support, licensing conditions, and price all are factors to take into account. Given these considerations, MySQL has many attractive qualities:

- **Speed.** MySQL is fast, and getting faster; see [http://www.mysql.com/why-mysql/benchmarks](http://www.mysql.com/why-mysql/benchmarks). There have been many improvements recently, particularly within InnoDB (which is now the default storage engine) and the query optimizer.

- **Ease of use.** MySQL is a high-performance but relatively simple database system and is much less complex to set up and administer than larger systems.

- **Query language support.** MySQL understands SQL (Structured Query Language), the standard language of choice for all modern database systems.
---

**Capability.** The MySQL server is multi-threaded, so many clients can connect to it at the same time. Each client can use multiple databases simultaneously. You can access MySQL interactively using several interfaces that let you enter queries and view the results: command-line clients, Web browsers, or GUI clients. In addition, programming interfaces are available for many languages, such as C, Perl, Java, PHP, Python, and Ruby. You can also access MySQL using applications that support ODBC and .NET (protocols developed by Microsoft). This gives you the choice of using prepackaged client software or writing your own for custom applications.

**Connectivity and security.** MySQL is fully networked, and databases can be accessed from anywhere on the Internet, so you can share your data with anyone, anywhere. But MySQL has access control so that one person who shouldn’t see another’s data cannot. To provide additional security, MySQL supports encrypted connections using the Secure Sockets Layer (SSL) protocol.

**Portability.** MySQL runs on many varieties of Unix and Linux, as well as on other systems such as Windows. MySQL runs on hardware from small devices such as routers and personal computers up to high-end servers with many CPUs and huge amounts of memory.

**Availability and cost.** MySQL is an Open Source project available under multiple licensing terms. First, it is available under the terms of the GNU General Public License (GPL). This means that MySQL is available without cost for most in-house uses. Second, for organizations that prefer or require formal arrangements or that do not want to be bound by the conditions of the GPL, commercial licenses are available.

**Open distribution and source code.** MySQL is easy to obtain; just use your Web browser. If you don’t understand how something works, are curious about an algorithm, or want to perform a security audit, you can get the source code and examine it. If you think you’ve found a bug, please report it; the developers want to know.

What about support? Good question; a database system isn’t much use if you can’t get help for it. This book is one form of assistance, and I like to think that it’s useful in that regard. (That the book has reached its fifth edition suggests that it accomplishes that goal.) There are other resources open to you as well, and you’ll find that MySQL has good support:

- The MySQL Reference Manual is included in MySQL distributions, and is easily accessible online. The Reference Manual regularly receives good marks in the MySQL user community. This is important; the value of a good product is diminished if no one can figure out how to use it.

- Technical support contracts and educational resources such as training classes are available from Oracle.

- MySQL mailing lists and forums are invaluable support resources that anyone may access. These have many helpful participants, including several MySQL developers.
The MySQL community, developers and nondevelopers alike, is very responsive. Answers to questions on the mailing lists often arrive within minutes. When bugs are reported, the developers generally fix them quickly, and new releases appear regularly.

If you are in the database-selection process, MySQL is an ideal candidate for evaluation. You can try it with no risk or financial commitment. Time for installation and setup is less than for many other systems. If you get stuck, you can use the mailing lists to get help.

Perhaps you’re currently running another database system but feel constrained by it: Performance of your current system is a concern; it’s proprietary and you don’t like being locked into it; you’d like to run on hardware that’s not supported by your current system; your software is provided in binary-only format but you want to have the source available; or maybe it just costs too much! All of these are reasons to look into MySQL. Use this book to familiarize yourself with MySQL’s capabilities, contact the MySQL sales crew, ask questions on the mailing lists, and you’ll find the answers you need to make a decision.

What You Can Expect from This Book

You’ll learn how to use MySQL effectively so that you can get your work done more productively. You’ll be able to figure out how to get your information into a database, and you’ll learn how to get it back out by formulating queries that answer the questions you want to ask of that data.

You need not be a programmer to understand or use SQL. This book shows you how it works. But there’s more to understanding how to use a database system properly than knowing SQL syntax. This book emphasizes MySQL’s unique capabilities and shows how to use them.

You’ll also see how MySQL integrates with other tools. The book shows how to write your own programs that access MySQL databases, and you’ll learn to use MySQL with Perl and PHP to generate dynamic Web pages created from the result of database queries.

If you’ll be responsible for administering a MySQL installation, this book will tell you what your duties are and how to carry them out. You’ll learn how to create user accounts, perform database backups, set up replication, and make sure your site is secure.

Road Map to This Book

This book has four parts. The first concentrates on general concepts of database use. The second focuses on writing your own programs that use MySQL. The third is for readers who have administrative duties. The fourth provides a set of reference appendixes.

Part I: General MySQL Use

- Chapter 1, “Getting Started with MySQL.” Discusses how MySQL can be useful to you, provides a tutorial that introduces the interactive `mysql` client program, covers the basics of SQL, and demonstrates MySQL’s general capabilities.
■ Chapter 2, “Using SQL to Manage Data.” Every major RDBMS now available understands SQL, but every database engine implements a slightly different SQL dialect. This chapter discusses SQL with particular emphasis on those features that make MySQL distinctive.

■ Chapter 3, “Data Types.” Discusses the data types that MySQL provides for storing your information, properties and limitations of each type, when and how to use them, expression evaluation, and type conversion.

■ Chapter 4, “Views and Stored Programs.” How to write and use SQL objects that are stored on the server side. These include views (virtual tables) and stored programs (functions and procedures, triggers, and events).

■ Chapter 5, “Query Optimization.” How to make your queries run faster.

Part II: Using MySQL Programming Interfaces

■ Chapter 6, “Introduction to MySQL Programming.” Discusses some of the application programming interfaces (APIs) for MySQL and provides a general comparison of the APIs that the book covers in detail.

■ Chapter 7, “Writing MySQL Programs Using C.” How to write C programs using the API provided by the MySQL C client library.

■ Chapter 8, “Writing MySQL Programs Using Perl DBI.” How to write Perl scripts using the DBI module. Covers standalone command-line scripts and scripts for Web site programming.

■ Chapter 9, “Writing MySQL Programs Using PHP.” How to use the PHP scripting language and the PHP Data Objects (PDO) database-access extension to write dynamic Web pages that access MySQL databases.

Part III: MySQL Administration

■ Chapter 10, “Introduction to MySQL Administration.” An overview of the database administrator’s duties and what you should know to run a MySQL site successfully.

■ Chapter 11, “The MySQL Data Directory.” An in-depth look at the organization and contents of the data directory, the area under which MySQL stores databases, logs, and status files.

■ Chapter 12, “General MySQL Administration.” How to make sure your operating system starts and stops the MySQL server properly when your system comes up and shuts down. Also discusses configuring storage engines, tuning the server, log maintenance, and running multiple servers.

■ Chapter 13, “Security and Access Control.” What you need to know to make your MySQL installation safe from intrusion, both from other users on the server host and from clients connecting over the network. Discusses how to set up MySQL user accounts, explains the structure of the grant tables that control client access to the MySQL server, and describes how to set up your server to support secure connections over SSL.
Chapter 14, “Database Maintenance, Backups, and Replication.” Discusses how to reduce the likelihood of disaster through preventive maintenance, how to back up your databases, how to perform crash recovery if disaster strikes in spite of your preventive measures, and how to set up replication servers.

Part IV: Appendixes
- Appendix A, “Software Required to Use This Book.” Where to get the major tools and sample database files described in the book.
- Appendix B, “Data Type Reference.” The characteristics of MySQL’s data types.
- Appendix C, “Operator and Function Reference.” The operators and functions that are used to write expressions in SQL statements.
- Appendix D, “System, Status, and User Variable Reference.” Describes each variable maintained by the MySQL server, and how to use your own variables in SQL statements.
- Appendix E, “SQL Syntax Reference.” Describes each SQL statement supported by MySQL.
- Appendix F, “MySQL Program Reference.” The programs provided in MySQL distributions.
- Appendix H, “Perl DBI API Reference.” The methods and attributes provided by the Perl DBI module.
- Appendix I, “PHP API Reference.” The methods provided for MySQL support in PHP by the PDO extension.

How to Read This Book
Whatever part of the book you happen to be reading, it’s best to try the examples as you go along. That means you should do two things:
- If MySQL isn’t installed on your system, install it or ask someone to do so for you.
- Get the files needed to set up the sampdb sample database used throughout the book.

Appendix A, “Software Required to Use This Book,” indicates where to obtain all the necessary components.

If you’re new to MySQL or SQL, begin with Chapter 1, “Getting Started with MySQL.” It provides you with a tutorial introduction that grounds you in basic MySQL and SQL concepts and brings you up to speed for the rest of the book. Then proceed to Chapter 2, “Using SQL to Manage Data,” Chapter 3, “Data Types,” and Chapter 4, “Views and Stored Programs,” to find
out how to describe and manipulate your own data so that you can exploit MySQL’s capabili-
ties for your own applications.

If you already know some SQL, you should still read Chapter 2, “Using SQL to Manage Data,” and Chapter 3, “Data Types.” SQL implementations vary, and you’ll want to find out what makes MySQL’s implementation distinctive in comparison to others with which you may be familiar. If you have experience with MySQL but need more background on performing particular tasks, use the book as a reference, looking up topics on a need-to-know basis. You’ll find the reference appendixes especially useful.

If you’re interested in writing your own programs to access MySQL databases, read the API chapters, beginning with Chapter 6, “Introduction to MySQL Programming.” To produce a Web-based front end to your databases for easier access to them, or, conversely, to provide a database back end for your Web site to enhance your site with dynamic content, check out Chapter 8, “Writing MySQL Programs Using Perl DBI,” and Chapter 9, “Writing MySQL Programs Using PHP.”

If your responsibilities include administering a MySQL installation, read the chapters beginning with Chapter 10, “Introduction to MySQL Administration.”

If you’re evaluating MySQL to find out how it compares to your current RDBMS, several parts of the book are useful. Read the SQL syntax and data type chapters in Part I to compare MySQL to the version of SQL that you’re used to, the programming chapters in Part II if you need to write custom applications, and the administrative chapters in Part III to assess the level of administrative support a MySQL installation requires. This information is also useful if you’re not currently using a database but are performing a comparative analysis of MySQL along with other database systems for the purpose of choosing one of them.

Versions of Software Covered in This Book

The first edition of this book covered MySQL 3.22 and the beginnings of MySQL 3.23. The second edition expanded that range to include MySQL 4.0 and the first release of MySQL 4.1. The third edition covered MySQL 4.1 and the initial releases of MySQL 5.0. The fourth edition covered MySQL 5.0 and the initial releases of MySQL 5.1.

For this fifth edition, the baseline is MySQL 5.5. That is, the book covers MySQL 5.5 and the early releases of MySQL 5.6. Most of this book still applies if you have a version older than 5.5, but differences specific to older versions usually are not explicitly noted.

The MySQL 5.5 series has reached General Availability (GA) status, which means that it is suitable for production environments. There have been many changes compared to earlier pre-production 5.5 releases, so use the most recent version if possible (5.5.30 as I write). The MySQL 5.6 series currently is a development series (not intended for production use yet) but will reach GA status soon, and may have done so by the time you read this.

For information about older versions, check the MySQL Web site at http://dev.mysql.com/doc, where you can access the Reference Manual for each version.
When updating each edition with new material, it’s always a challenge to keep the length down. In the interest of space, I have removed some information present in previous editions. The most pervasive change is that InnoDB is now the default storage engine (not MyISAM), so in keeping with the greater emphasis on InnoDB, there is less on MyISAM. Other more minor storage engines such as FEDERATED and BLACKHOLE are mentioned only in passing. I have removed information about libmysqld (the embedded server), mysqlhotcopy, myisampack, spatial data types and functions, and replaced detailed installation material with more general instructions. For information about any of those topics, I recommend the MySQL Reference Manual.

I also draw your attention to some other topics not covered in this book:

■ The MySQL Connectors, which provide client access for Java, ODBC, and .NET programs.
■ The NDB storage engine and MySQL Cluster, which provide in-memory storage, high availability, and redundancy. See the MySQL Reference Manual for details.
■ The graphical user interface (GUI) tool, MySQL Workbench, which helps you use MySQL in a windowing environment.
■ MySQL Enterprise, the commercial version of MySQL that includes features such as MySQL Enterprise Monitor that provides server monitoring and diagnostic capabilities, and MySQL Enterprise Backup for hot backups.

To acquire any of these products or see their documentation, visit http://www.mysql.com/products or http://dev.mysql.com/doc.

For the other major software packages discussed in the book, any recent versions should be sufficient for the examples shown. The following table shows the current versions at the time of writing.

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl DBI module</td>
<td>1.623</td>
</tr>
<tr>
<td>Perl DBD::mysql module</td>
<td>4.020</td>
</tr>
<tr>
<td>PHP</td>
<td>5.4.10</td>
</tr>
<tr>
<td>Apache</td>
<td>2.4.3</td>
</tr>
<tr>
<td>CGI.pm</td>
<td>3.63</td>
</tr>
</tbody>
</table>

All software discussed in this book is available on the Internet. Appendix A, “Software Required to Use This Book,” provides assistance for getting MySQL, Perl DBI, PHP and PDO, Apache, and CGI.pm onto your system. The appendix also contains instructions for obtaining the sampdb sample database that is used in examples throughout the book and contains the programs developed in the programming chapters.
If you are using Windows, I assume that you have Windows 2000 or newer. Some features covered in this book, such as named pipes and Windows services, are not available in older versions.

### Conventions Used in This Book

This book uses the following typographical conventions:

- **Monospaced font** indicates hostnames, filenames, directory names, commands, options, and Web sites.
- **Bold monospaced font** is used in command examples to indicate input that you type.
- **Italic monospaced font** is used in commands to indicate where you should substitute a value of your own choosing.

Interactive examples assume that you enter commands by typing them into a terminal window or console window. To provide context, the prompt in command examples indicates the program from which you run the command. For example, SQL statements that are issued from within the `mysql` client program are shown preceded by the `mysql>` prompt. For commands that you issue from your command interpreter, the `%` prompt usually is used. In general, this prompt indicates commands that can be run either on Unix or Windows, although the particular prompt you see will depend on your command interpreter. (The command interpreter is your login shell on Unix, or `cmd.exe` on Windows.) More specialized command-line prompts are `#`, which indicates a command run on Unix as the `root` user with `su` or `sudo`, and `C:\>` to indicate a command intended specifically for Windows.

The following example shows a command that should be entered from your command interpreter. The `%` indicates the prompt (which you do not type). To issue the command, you'd enter the boldface characters as shown, and substitute your own username for the italic word:

```
% mysql --user=user_name sampdb
```

In SQL statements, SQL keywords and function names are written in uppercase. Database, table, and column names generally appear in lowercase.

In syntax descriptions, square brackets ([]) indicate optional information. In lists of alternatives, vertical bar (|) is used as a separator between items. A list enclosed within [ ] is optional and indicates that an item may be chosen from the list. A list enclosed within { } is mandatory and indicates that an item must be chosen from the list.

### Additional Resources

If you have a question that this book doesn't answer, where should you turn? Useful documentation resources include the Web sites for the software you need help with, shown in the following table.
Those sites provide information such as reference manuals, frequently asked-question (FAQ) lists, and mailing lists:

- **Reference manuals.** The primary documentation included with MySQL itself is the Reference Manual. It’s available in several formats, including online and downloadable versions.

- **Manual pages.** Documentation for the DBI module and its MySQL-specific driver, DBD::mysql, can be read from the command line with the `perldoc` command. Try `perldoc DBI` and `perldoc DBD::mysql`. The DBI document provides general concepts, and the MySQL driver document discusses capabilities specific to MySQL.

- **FAQs.** There are frequently-asked-question lists for DBI, PHP, and Apache.

- **Mailing lists.** Several mailing lists centering around the software discussed in this book are available. It’s a good idea to subscribe to the ones that deal with the tools you want to use. It’s also a good idea to use the archives for those lists that have them. When you’re new to a tool, you will have many of the same questions that have been asked (and answered) many times, and there is no reason to ask again when you can find the answer with a quick search of the archives.

Instructions for subscribing to the mailing lists vary. The following table indicates where you can find the necessary information.

<table>
<thead>
<tr>
<th>Package</th>
<th>Mailing List Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td><a href="http://lists.mysql.com">http://lists.mysql.com</a></td>
</tr>
<tr>
<td>Perl DBI</td>
<td><a href="http://dbi.perl.org/support">http://dbi.perl.org/support</a></td>
</tr>
<tr>
<td>PHP</td>
<td><a href="http://www.php.net/mailing-lists.php">http://www.php.net/mailing-lists.php</a></td>
</tr>
<tr>
<td>Apache</td>
<td><a href="http://httpd.apache.org/lists.html">http://httpd.apache.org/lists.html</a></td>
</tr>
</tbody>
</table>

- **Ancillary Web sites.** Besides the official Web sites, some of the tools discussed here have ancillary sites that provide more information, such as sample source code or topical articles. Check for a “Links” area on the official site you’re visiting.
Using SQL to Manage Data

The MySQL server understands Structured Query Language (SQL). Therefore, SQL is the means by which you tell the server how to perform data management operations, and fluency with it is necessary for effective communication. When you use a program such as the `mysql` client, it functions primarily as a way for you to send SQL statements to the server to be executed. If you write programs in a language that has a MySQL interface, such as the Perl DBI module or PHP PDO extension, these interfaces enable you to communicate with the server by issuing SQL statements.

Chapter 1, “Getting Started with MySQL,” presented a tutorial introducing many of MySQL’s capabilities, including some basic use of SQL. We’ll build on that material here to go into more detail on several topics:

- Changing the SQL mode to affect server behavior
- Referring to elements of databases
- Using multiple character sets
- Creating and destroying databases, tables, and indexes
- Obtaining information about databases and their contents
- Retrieving data using joins, subqueries, and unions
- Using multiple-table deletes and updates
- Performing transactions that enable statements to be grouped or canceled
- Setting up foreign key relationships
- Using the `FULLTEXT` search engine

The items just listed cover a broad range of topics of what you can do with SQL. Other chapters provide additional SQL-related information:

- Chapter 4, “Views and Stored Programs,” discusses how to create and use views (virtual tables that provide alternative ways of looking at data) and stored programs (functions and procedures, triggers, and events).
Chapter 12, “General MySQL Administration,” describes how to use administrative statements such as `GRANT` and `REVOKE` to manage user accounts. It also discusses the privilege system that controls what operations accounts are permitted to perform.

Appendix E, “SQL Syntax Reference,” shows the syntax for SQL statements implemented by MySQL and the privileges required to use them. It also covers the syntax for using comments in your SQL statements.

See also the MySQL Reference Manual, especially for changes made in recent versions of MySQL.

### 2.1 The Server SQL Mode

The MySQL SQL mode affects several aspects of SQL statement execution, and the server has a system variable named `sql_mode` that enables you to configure this mode. The variable can be set globally to affect all clients, and each individual client can change the mode to affect its own session with (connection to) the server. This means that any client can change how the server treats it without impact on other clients.

The SQL mode affects behaviors such as handling of invalid values during data entry and identifier quoting. The following list describes a few of the possible mode values:

- `STRICT_ALL_TABLES` and `STRICT_TRANS_TABLES` enable “strict” mode. In strict mode, the server is more restrictive about accepting bad data values. (Specifically, it rejects bad values rather than changing them to the closest legal value.)

- `TRADITIONAL` is a composite mode. It is like strict mode, but enables other modes that impose additional constraints for even stricter data checking. Traditional mode causes the server to behave like more traditional SQL servers with regard to how it handles bad data values.

- `ANSI_QUOTES` tells the server to recognize double quote as an identifier quoting character.

- `PIVES_AS_CONCAT` causes `||` to be treated as the standard SQL string concatenation operator rather than as a synonym for the `OR` operator.

- `ANSI` is another composite mode. It turns on `ANSI_QUOTES`, `PIVES_AS_CONCAT`, and several other mode values that cause the server to conform more closely to standard SQL.

When you set the SQL mode, specify a value consisting of one or more mode values separated by commas, or an empty string to clear the value. Mode values are not case sensitive.

To set the SQL mode when you start the server, set the `sql_mode` system variable on the `mysqld` command line or in an option file. On the command line, you might use a setting like one of these:

```
--sql_mode="TRADITIONAL"
--sql_mode="ANSI_QUOTES,PIVES_AS_CONCAT"
```
To change the SQL mode at runtime, set the `sql_mode` system variable with a `SET` statement. Any client can set its own session-specific SQL mode:

```
SET sql_mode = 'TRADITIONAL';
```

To set the SQL mode globally, add the `GLOBAL` keyword:

```
SET GLOBAL sql_mode = 'TRADITIONAL';
```

Setting the global variable requires the `SUPER` administrative privilege. The global value becomes the default SQL mode for clients that connect afterward.

To determine the current value of the session or global SQL mode, use these statements:

```
SELECT @@SESSION.sql_mode;
SELECT @@GLOBAL.sql_mode;
```

The value returned consists of a comma-separated list of enabled modes, or an empty value if no modes are enabled.

Section 3.3, “How MySQL Handles Invalid Data Values,” discusses the SQL mode values that affect handling of erroneous or missing values during data entry. Appendix D, “System, Status, and User Variable Reference,” describes the full set of permitted mode values for the `sql_mode` variable. For additional information about using system variables, see Section 12.3.1, “Checking and Setting System Variable Values.”

### 2.2 MySQL Identifier Syntax and Naming Rules

Almost every SQL statement uses identifiers in some way to refer to a database or its constituent elements such as tables, views, columns, indexes, stored routines, triggers, or events. When you refer to elements of databases, identifiers must conform to the following rules.

**Legal characters in identifiers.** Unquoted identifiers may consist of latin letters `a-z` in any lettercase, digits `0-9`, dollar, underscore, and Unicode extended characters in the range `U+0080` to `U+FFFF`. Identifiers can start with any character that is legal in an identifier, including a digit. However, an unquoted identifier cannot consist entirely of digits because that would make it indistinguishable from a number. MySQL’s support for identifiers that begin with a number is somewhat unusual among database systems. If you use such an identifier, take particular care if it contains an ‘E’ or ‘e’ because those characters can lead to ambiguous expressions. For example, the expression `23e + 14` (with spaces surrounding the ‘+’ sign) means column `23e` plus the number `14`, but what about `23e+14`? Does it mean the same thing, or is it a number in scientific notation?

Identifiers can be quoted (delimited) within backtick characters (`'`), which permits use of any character except a NUL byte or Unicode supplementary characters (`U+10000` and up):

```
CREATE TABLE `my table` (`my-int-column` INT);
```
Quoting is useful when an identifier is an SQL reserved word or contains spaces or other special characters. Quoting an identifier also enables it to be entirely numeric, something not true of unquoted identifiers. To include an identifier quote character within a quoted identifier, double it.

Your operating system might impose additional constraints on database and table identifiers. See Section 11.2.6, “Operating System Constraints on Database Object Names.”

Aliases for column and table names can be fairly arbitrary. You should quote an alias within identifier quoting characters if it is an SQL reserved word, is entirely numeric, or contains spaces or other special characters. Column aliases also can be quoted with single quotes or double quotes.

**Server SQL mode.** If the ANSI_QUOTES SQL mode is enabled, you can quote identifiers with double quotes (although backticks still are permitted).

```sql
CREATE TABLE "my table" ("my-int-column" INT);
```

Enabling ANSI_QUOTES has the additional effect that string literals must be written using single quotes. If you use double quotes, the server interprets the value as an identifier, not as a string.

Names of built-in functions normally are not reserved and can be used as identifiers without quotes. However, if the IGNORE_SPACE SQL mode is enabled, function names become reserved and must be quoted if used as identifiers.

For instructions on setting the SQL mode, see Section 2.1, “The Server SQL Mode.”

**Identifier length.** Most identifiers have a maximum length of 64 characters. The maximum length for aliases is 256 characters.

**Identifier qualifications.** Depending on context, an identifier might need to be qualified to clarify what it refers to. To refer to a database, just specify its name:

```sql
USE db_name;
SHOW TABLES FROM db_name;
```

To refer to a table, you have two choices:

- A fully qualified table name consists of a database identifier and a table identifier:
  ```sql
  SHOW COLUMNS FROM db_name.tbl_name;
  SELECT * FROM db_name.tbl_name;
  ```

- A table identifier by itself refers to a table in the default (current) database. If sampdb is the default database, the following statements are equivalent:
  ```sql
  SELECT * FROM member;
  SELECT * FROM sampdb.member;
  ```

If no database is selected, it is an error to refer to a table without a database qualifier because the database to which the table belongs is unknown.

The same considerations about qualifying table names apply to names of views (which are “virtual” tables) and stored programs.
To refer to a table column, you have three choices:

- A name written as `db_name.tbl_name.col_name` is fully qualified.
- A partially qualified name written as `tbl_name.col_name` refers to a column in the named table in the default database.
- An unqualified name written simply as `col_name` refers to whatever table the surrounding context indicates. The following two queries use the same column names, but the context supplied by the `FROM` clause of each statement indicates the table from which to select the columns:
  ```sql
  SELECT last_name, first_name FROM president;
  SELECT last_name, first_name FROM member;
  ```

Usually, it’s unnecessary to supply fully qualified names, although it’s always legal to do so. If you select a database with a `USE` statement, it becomes the default database for subsequent statements and is implicit in every unqualified table reference. If you write a `SELECT` statement that refers to only one table, that table is implicit for every column reference in the statement. It’s necessary to qualify identifiers only when a table or database cannot be determined from context. For example, if a statement refers to tables from multiple databases, you must reference any table not in the default database using `db_name.tbl_name` syntax to let MySQL know which database contains the table. Similarly, if a query uses multiple tables and refers to a column name that is used in more than one table, qualify the column identifier with a table identifier to make it clear which column you mean.

If you use quotes when referring to a qualified name, quote individual identifiers within the name separately. For example:

```sql
SELECT * FROM `sampdb`.`member` WHERE `sampdb`.`member`.`member_id` > 100;
```

Do not quote the name as a whole. This statement is incorrect:

```sql
SELECT * FROM `sampdb.member` WHERE `sampdb.member.member_id` > 100;
```

The requirement that a reserved word be quoted if used as an identifier is waived if the word follows a qualifier period because context then dictates that the reserved word is an identifier.

## 2.3 Case Sensitivity in SQL Statements

Case sensitivity rules in SQL statements vary for different statement elements, and also depend on what you are referring to and the operating system of the machine on which the server is running.

**SQL keywords and function names.** Keywords and function names are not case sensitive. They can be given in any lettercase. The following statements are equivalent:

```sql
SELECT NOW();
SELECT NOW();
SELECT NOW();
SELECT NOW();
```
Database, table, and view names. MySQL represents databases and tables using directories and files in the underlying filesystem on the server host. As a result, the default case sensitivity of database and table names depends on how the operating system on that host treats filenames. Windows filenames are not case sensitive, so a MySQL server running on Windows does not treat database and table names as case sensitive. Servers running on Unix usually treat database and table names as case sensitive because Unix filenames are case sensitive. An exception is that names in Mac OS X Extended filesystems can be case insensitive.

MySQL represents each view using a file, so the preceding remarks about tables also apply to views.

Stored program names. Stored function and procedure names and event names are not case sensitive. Trigger names are case sensitive, which differs from standard SQL.

Column and index names. Column and index names are not case sensitive in MySQL. The following statements are equivalent:

```
SELECT name FROM student;
SELECT NAME FROM student;
SELECT nAmE FROM student;
```

Alias names. By default, table aliases are case sensitive. You can specify an alias in any lettercase (upper, lower, or mixed), but if you use it multiple times in a statement, you must use the same lettercase each time. If the `lower_case_table_names` system variable is nonzero, table aliases are not case sensitive.

String values. Case sensitivity of a string value depends on whether it is a binary or nonbinary string, and, for a nonbinary string, on the collation of its character set. This is true for literal strings and the contents of string columns. For further information, see Section 3.1.2, “String Values.”

You should consider lettercase issues when you create databases and tables on a machine with case sensitive filenames if you might someday move them to a machine where filenames are not case sensitive. Suppose that you create two tables named `abc` and `ABC` on a Unix server where those names are considered distinct. You would have problems moving the tables to a Windows machine: `abc` and `ABC` are not distinguishable because names are not case sensitive. You would also have trouble replicating the tables from a Unix master server to a Windows slave server.

To avoid having case sensitivity become an issue, pick a given lettercase and always create databases and tables using names in that lettercase. Then case of names won’t be a problem if you move a database to a different server. I recommend lowercase, particularly if you are using InnoDB tables, because InnoDB stores database and table names internally in lowercase.

To force creation of databases and tables with lowercase names even if not specified that way in `CREATE` statements, configure the server by setting the `lower_case_table_names` system variable. For more information, see Section 11.2.6, “Operating System Constraints on Database Object Names.”
Regardless of whether a database or table name is case sensitive on your system, you must refer to it using the same lettercase throughout a given query. That is not true for SQL keywords, function names, or column and index names, all of which may be referred to in varying letter-case style throughout a query.

2.4 Character Set Support

MySQL supports multiple character sets, and character sets can be specified independently at the server, database, table, column, or string constant level. For example, if you want a table’s columns to use \texttt{latin1} by default, but also to include a Hebrew column and a Greek column, you can do that. In addition, you can explicitly specify collations (sorting orders). It is possible to find out what character sets and collations are available, and to convert data from one character set to another.

This section provides general background on using character set support in MySQL. Chapter 3, “Data Types,” provides more specific discussion of character sets, collations, binary versus nonbinary strings, and how to define and work with character-based table columns.

MySQL provides the following character set features:

- The server supports simultaneous use of multiple character sets.
- A given character set can have one or more collations. You can choose the collation most appropriate for your applications.
- Unicode support is provided by the \texttt{utf8} and \texttt{ucs2} character sets, which include Basic Multilingual Plane (BMP) characters, and the \texttt{utf16}, \texttt{utf32}, and \texttt{utf8mb4} character sets, which include BMP and supplementary characters. MySQL 5.6.1 adds \texttt{utf16le}, which is like \texttt{utf16} but uses little-endian rather than big-endian encoding.
- You can specify character sets at the server, database, table, column, and string constant level:
  - The server has a default character set.
  - \texttt{CREATE DATABASE} enables you to assign the database character set, and \texttt{ALTER DATABASE} enables you to change it.
  - \texttt{CREATE TABLE} and \texttt{ALTER TABLE} have clauses for table- and column-level character set assignment.
  - The character set for string constants is determined by context or can be specified explicitly.

- Several functions and operators are available for converting individual values from one character set to another, and the \texttt{CHARSET()} function returns the character set of a value. Similarly, the \texttt{COLLATE} operator can be used to alter the collation of a string and the \texttt{COLLATION()} function returns the collation of a string.
SHOW statements and INFORMATION_SCHEMA tables provide information about the available character sets and collations.

The server automatically reorders indexes when you change the collation of an indexed character column.

You cannot mix character sets within a string, or use different character sets for different rows of a given column. However, you can implement multi-lingual support by using a Unicode character set (which represents characters for many languages within a single encoding).

### 2.4.1 Specifying Character Sets

Character set and collation assignments can be made at several levels, from the default used by the server to the character set used for individual strings.

The server’s default character set and collation are built in at compile time. You can override them at server startup or at runtime by setting the `character_set_server` and `collation_server` system variables, as described in Section 12.6.2, “Selecting the Default Character Set and Collation.” If you specify only the character set, its default collation becomes the server’s default collation. If you specify a collation, it must be compatible with the character set. A collation is compatible with a character set if its name begins with the character set name. For example, `utf8_danish_ci` is compatible with `utf8` but not with `latin1`.

In SQL statements that create databases and tables, two clauses specify database, table, and column character set and collation values:

```
CHARACTER SET charset
COLLATE collation
```

`CHARSET` can be used as a synonym for `CHARACTER SET`. `charset` is the name of a character set supported by the server, and `collation` is the name of one of that character set’s collations. These clauses can be specified together or separately. If both are given, the collation name must be compatible with the character set. If only `CHARACTER SET` is given, its default collation is used. If only `COLLATE` is given, the character set is implicit in the first part of the character set name. These rules apply at several levels:

- To specify a default character set and collation for a database when you create it, use this statement:
  ```
  CREATE DATABASE db_name CHARACTER SET charset COLLATE collation;
  ```
  If no character set or collation is given, the database uses the server defaults.

- To specify a default character set and collation for a table, use `CHARACTER SET` and `COLLATE` table options at table creation time:
  ```
  CREATE TABLE tbl_name (...) CHARACTER SET charset COLLATE collation;
  ```
  If no character set or collation is given, the table uses the database defaults.
■ Columns in a table can be assigned a character set and collation explicitly with
CHARACTER SET and COLLATE attributes. For example:

```
c CHAR(10) CHARACTER SET charset COLLATE collation
```

If no character set or collation is given, the column uses the table defaults. These
attributes apply to the CHAR, VARCHAR, TEXT, ENUM, and SET data types.

It’s also possible to sort string values according to a specific collation by using the COLLATE
operator. For example, if `c` is a latin1 column that has a collation of latin1_swedish_ci, but
you want to order it using Spanish sorting rules, do this:

```
SELECT c FROM t ORDER BY c COLLATE latin1_spanish_ci;
```

### 2.4.2 Determining Character Set Availability and Current Settings

To find out which character sets and collations are available, use these statements:

```
SHOW CHARACTER SET;
SHOW COLLATION;
```

Each statement supports a LIKE clause that narrows the results to those character set or colla-
tion names matching a pattern. For example, the following statements list the Latin-based char-
acter sets and the collations available for the utf8 character set:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | cp1252 West European        | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+

mysql> SHOW COLLATION LIKE 'utf8%';
+-----------------------+---------+-----+---------+----------+---------+
| Collation             | Charset | Id  | Default | Compiled | Sortlen |
+-----------------------+---------+-----+---------+----------+---------+
| utf8_general_ci       | utf8    |  33 | Yes     | Yes      |       1 |
| utf8_bin              | utf8    |  83 |         | Yes      |       1 |
| utf8_unicode_ci       | utf8    | 192 |         | Yes      |       8 |
| utf8_icelandic_ci     | utf8    | 193 |         | Yes      |       8 |
| utf8_latvian_ci       | utf8    | 194 |         | Yes      |       8 |
| utf8_romanian_ci      | utf8    | 195 |         | Yes      |       8 |
| utf8_slovenian_ci     | utf8    | 196 |         | Yes      |       8 |
+-----------------------+---------+-----+---------+----------+---------+
```

Collation names always begin with the character set name. Each character set has at least one
collation, and one of them is its default collation.
Information about the available character sets or collations can also be obtained from the `CHARACTER_SETS` or `COLLATIONS` table in the `INFORMATION_SCHEMA` database (see Section 2.7, “Obtaining Database Metadata”).

To display the server’s current character set and collation settings, use `SHOW VARIABLES`:

```sql
mysql> SHOW VARIABLES LIKE 'character\_set\_\%';
+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| character_set_client     | utf8   |
| character_set_connection | utf8   |
| character_set_database   | latin1 |
| character_set_filesystem | binary |
| character_set_results    | utf8   |
| character_set_server     | latin1 |
| character_set_system     | utf8   |
+--------------------------+--------+
mysql>
```

```sql
mysql> SHOW VARIABLES LIKE 'collation\_\%';
+----------------------+-------------------+
| Variable_name        | Value             |
+----------------------+-------------------+
| collation_connection | utf8_general_ci   |
| collation_database   | latin1_swedish_ci |
| collation_server     | latin1_swedish_ci |
+----------------------+-------------------+
```

Several of these system variables affect how a client communicates with the server after establishing a connection. For details, refer to Section 3.1.2.2, “Character Set-Related System Variables.”

### 2.4.3 Unicode Support

One of the reasons there are so many character sets is that different character encodings have been developed for different languages. This presents several problems. For example, a given character that is common to several languages might be represented by different numeric values in different encodings. Also, different languages require different numbers of bytes to represent characters. The `latin1` character set is small enough that every character fits in a single byte, but languages such as those used in Japan and China contain so many characters that they require multiple bytes per character.

Unicode deals with these issues by providing a unified character-encoding system within which character sets for all languages can be represented in a consistent manner.

The `utf8` and `ucs2` Unicode character sets include only characters in the Basic Multilingual Plane (BMP), which is limited to 65,536 characters. They do not support supplementary characters outside the BMP.
The `ucs2` character set corresponds to the Unicode UCS-2 encoding. It represents each character using 2 bytes, most significant byte first. UCS is an abbreviation for Universal Character Set.

The `utf8` character set has a variable-length format that represents characters using from 1 to 3 bytes. It corresponds to the Unicode UTF-8 encoding. UTF is an abbreviation for Unicode Transformation Format.

Beginning with MySQL 5.5.3, other Unicode character sets are available that include supplementary characters in addition to BMP characters.

- The `utf16` and `utf32` character sets are like `ucs2` but with supplementary characters added. For `utf16`, BMP characters take 2 bytes (as for `ucs2`) and supplementary characters take 4 bytes. For `utf32`, all characters take 4 bytes.
- The `utf8mb4` character set contains all the `utf8` characters (which take 1 to 3 bytes each), but also supplementary characters that take 4 bytes each.

MySQL 5.6.1 adds `utf16le`, which is like `utf16` but uses little-endian rather than big-endian encoding.

## 2.5 Selecting, Creating, Dropping, and Altering Databases

MySQL provides several database-level statements: `USE` for selecting a default database, `CREATE DATABASE` for creating databases, `DROP DATABASE` for removing them, and `ALTER DATABASE` for modifying global database characteristics.

The keyword `SCHEMA` is a synonym for `DATABASE` in any statement where the latter occurs.

### 2.5.1 Selecting Databases

The `USE` statement selects a database to make it the default (current) database for a given session with the server:

```
USE db_name;
```

You must have some access privilege for the database or an error occurs.

It is not strictly necessary to select a database explicitly. You can refer to tables in a database without selecting it first by using qualified names that identify both the database and the table. For example, to retrieve the contents of the `president` table in the `sampdb` database without making it the default database, write the query like this:

```
SELECT * FROM sampdb.president;
```

Selecting a database doesn’t mean that it must be the default for the duration of the session. You can issue `USE` statements as necessary to switch between databases. Nor does selecting a
database limit you to using tables only from that database. While one database is the default, you can refer to tables in other databases by qualifying their names with the appropriate database identifier.

When you disconnect from the server, any notion by the server of which database was the default for the session disappears. If you connect to the server again, it doesn’t remember what database you had selected previously.

### 2.5.2 Creating Databases

To create a database, use a `CREATE DATABASE` statement:

```
CREATE DATABASE db_name;
```

The database must not already exist, and you must have the `CREATE` privilege for it.

`CREATE DATABASE` supports several optional clauses. The full syntax is as follows:

```
CREATE DATABASE [IF NOT EXISTS] db_name
    [CHARACTER SET charset] [COLLATE collation];
```

By default, an error occurs if you try to create a database that already exists. To suppress this error and create a database only if it does not already exist, add an `IF NOT EXISTS` clause:

```
CREATE DATABASE IF NOT EXISTS db_name;
```

By default, the server character set and collation become the database default character set and collation. To set these database attributes explicitly, use the `CHARACTER SET` and `COLLATE` clauses. For example:

```
CREATE DATABASE mydb CHARACTER SET utf8 COLLATE utf8_icelandic_ci;
```

If `CHARACTER SET` is given without `COLLATE`, the character set default collation is used. If `COLLATE` is given without `CHARACTER SET`, the first part of the collation name determines the character set.

The character set must be one of those supported by the server, such as `latin1` or `sjis`. The collation should be a legal collation for the character set. For further discussion of character sets and collations, see Section 2.4, “Character Set Support.”

When you create a database, the MySQL server creates a directory under its data directory that has the same name as the database. The new directory is called the database directory. The server also creates a `db.opt` file in the database directory for storing attributes such as the database character set and collation. When you create a table in the database later, the database defaults become the table defaults if the table definition does not specify its own default character set and collation.

To see the definition for an existing database, use a `SHOW CREATE DATABASE` statement:
2.5.3 Dropping Databases

Dropping a database is as easy as creating one, assuming that you have the DROP privilege for it:

```
DROP DATABASE db_name;
```

The `DROP DATABASE` statement is not something to use with wild abandon. It removes the database and all its contents (tables, stored routines, and so forth), which are therefore gone forever unless you have been making regular backups.

A database is represented by a directory under the data directory, and the directory is intended for storage of objects such as tables, views, and triggers. If a `DROP DATABASE` statement fails, the reason most likely is that the database directory contains files not associated with database objects. `DROP DATABASE` will not delete such files, and as a result will not delete the directory, either. This means that the database directory continues to exist and will show up if you issue a `SHOW DATABASES` statement. To really drop the database if this occurs, manually remove any extraneous files and subdirectories from the database directory, then issue the `DROP DATABASE` statement again.

2.5.4 Altering Databases

The `ALTER DATABASE` statement changes a database's global attributes, if you have the ALTER privilege for it. Currently, the only such attributes are the default character set and collation:

```
ALTER DATABASE [db_name] [CHARACTER SET charset] [COLLATE collation];
```

The earlier discussion for `CREATE DATABASE` describes the effect of the `CHARACTER SET` and `COLLATE` clauses, at least one of which must be given.

If you omit the database name, `ALTER DATABASE` applies to the default database.

2.6 Creating, Dropping, Indexing, and Altering Tables

MySQL enables you to create tables, drop (remove) them, and change their structure with the `CREATE TABLE`, `DROP TABLE`, and `ALTER TABLE` statements. The `CREATE INDEX` and `DROP INDEX` statements enable you to add or remove indexes on existing tables. The following sections provide the details for these statements, but first it's necessary to discuss the storage engines that MySQL supports for managing different types of tables.
2.6.1 Storage Engine Characteristics

MySQL supports multiple storage engines (or “table handlers” as they used to be known). Each storage engine implements tables that have a specific set of properties or characteristics. Table 2.1 briefly describes these storage engines, and later discussion provides more detail about some of them (primarily InnoDB and MyISAM). Others are either less commonly used or, in the case of NDB, require extensive discussion beyond what can be given here. Consequently, the remainder of this book says little about them.

Table 2.1 MySQL Storage Engines

<table>
<thead>
<tr>
<th>Storage Engine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCHIVE</td>
<td>Archival storage (no modification of rows after insertion)</td>
</tr>
<tr>
<td>BLACKHOLE</td>
<td>Engine that discards writes and returns empty reads</td>
</tr>
<tr>
<td>CSV</td>
<td>Storage in comma-separated values format</td>
</tr>
<tr>
<td>FEDERATED</td>
<td>Engine for accessing remote tables</td>
</tr>
<tr>
<td>InnoDB</td>
<td>Transactional engine with foreign keys</td>
</tr>
<tr>
<td>MEMORY</td>
<td>In-memory tables</td>
</tr>
<tr>
<td>MERGE</td>
<td>Manages collections of MyISAM tables</td>
</tr>
<tr>
<td>MyISAM</td>
<td>The main nontransactional storage engine</td>
</tr>
<tr>
<td>NDB</td>
<td>The engine for MySQL Cluster</td>
</tr>
</tbody>
</table>

Some of the engine names have synonyms. MRG_MyISAM and NDBCLUSTER are synonyms for MERGE and NDB, respectively. The MEMORY and InnoDB storage engines originally were known as HEAP and Innobase, respectively. The latter names are still recognized but deprecated.

Originally, the MySQL server was built such that all storage engines to be made available were compiled in. Now the server uses a “pluggable” architecture that enables plugins to be loaded selectively, and many storage engines are built as plugins. This permits the DBA to treat those engines as optional and load only those needed. The plugin interface also permits storage engines from third-party developers to be integrated into the server. For information about this interface, see Section 12.4, “The Plugin Interface.”

2.6.1.1 Checking Which Storage Engines Are Available

The engines actually available for a given server depend on your version of MySQL, how the server was configured at build time, and the startup options you use. For information about selecting storage engines, see Section 12.5, “Storage Engine Configuration.”

To see which storage engines the server knows about, use the `SHOW ENGINES` statement:
2.6 Creating, Dropping, Indexing, and Altering Tables

mysql> SHOW ENGINES\G
*************************** 1. row ***************************
  Engine: InnoDB
  Support: DEFAULT
  Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
  XA: YES
  Savepoints: YES
...
*************************** 8. row ***************************
  Engine: MyISAM
  Support: YES
  Comment: MyISAM storage engine
Transactions: NO
  XA: NO
  Savepoints: NO
...

The Support column value is YES or NO to indicate that the engine is or is not available, DISABLED if the engine is present but turned off, or DEFAULT for the storage engine that the server uses by default. The engine designated as DEFAULT should be considered available. The Transactions column indicates whether an engine supports transactions. XA and Savepoints indicate whether an engine supports distributed transactions (not covered in this book) and partial transaction rollback.

The ENGINES table in the INFORMATION_SCHEMA database provides the same information as SHOW ENGINES, but since you access it with SELECT, you can apply query conditions to select only the information in which you’re interested. For example, this query uses the ENGINES table to check for available engines that support transactions:

mysql> SELECT ENGINE FROM INFORMATION_SCHEMA.ENGINES
  -> WHERE TRANSACTIONS = 'YES';

+--------+
| ENGINE |
+--------+
| InnoDB |
+--------+

2.6.1.2 Table Representation on Disk

Each time you create a table, MySQL creates a disk file that contains the table’s format (that is, its definition). The format file has a basename that is the same as the table name and an .frm extension. For a table named t, the format file is named t.frm. The server creates the file in the database directory for the database that the table belongs to. The .frm file is an invariant because there is one for every table, no matter which storage engine manages the table. The name of a table as used in SQL statements might differ from the table-name part of the associated .frm file if the name contains characters that are problematic in filenames. See
Section 11.2.6, “Operating System Constraints on Database Object Names,” for a description of the rules for mapping from SQL names to filenames.

Individual storage engines may also create other files that are unique to the table, to be used for storing the table’s content. For a given table, any files specific to it are located in the database directory for the database that contains the table. Table 2.2 shows the filename extensions for table-specific files created by certain storage engines.

### Table 2.2 Table Files Created by Storage Engines

<table>
<thead>
<tr>
<th>Storage Engine</th>
<th>Files on Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>InnoDB</td>
<td>.ibd (data and indexes)</td>
</tr>
<tr>
<td>MyISAM</td>
<td>.MYD (data), .MYI (indexes)</td>
</tr>
<tr>
<td>CSV</td>
<td>.CSV (data), .CSM (metadata)</td>
</tr>
</tbody>
</table>

For some storage engines, the format file is the only file specifically associated with a particular table. Other engines may store table content elsewhere than on disk, or may use one or more tables (storage areas shared by multiple tables):

- The MEMORY storage engine stores table contents in memory, not on disk.
- By default, InnoDB stores table data and indexes in its system tablespace. That is, all InnoDB table contents are managed within a shared storage area, not within files specific to a particular table. Alternatively, InnoDB creates .ibd files if you configure it to use individual per-table tablespaces.

The following sections characterize the features and behavior of selected MySQL storage engines. For additional information about how engines represent tables physically, see Section 11.2.3, “Representation of Tables in the Filesystem.”

### 2.6.1.3 The InnoDB Storage Engine

The InnoDB storage engine is the default engine in MySQL, unless you have configured your server otherwise. The following list describes some of its features:

- Transaction-safe tables with commit and rollback. Savepoints can be created to enable partial rollback.
- Automatic recovery after a crash.
- Foreign key and referential integrity support, including cascaded delete and update.
- Row-level locking and multi-versioning for good concurrency performance under query mix conditions that include both retrievals and updates.
- As of MySQL 5.6, InnoDB supports full-text searches and FULLTEXT indexes.
By default, InnoDB manages tables within a single system tablespace, rather than by using table-specific files like most other storage engines. The tablespace consists of one or more files and can include raw partitions. The InnoDB storage engine, in effect, treats the tablespace as a virtual filesystem within which it manages the contents of all InnoDB tables. Tables thus can exceed the size permitted by the filesystem for individual files. You can also configure InnoDB to use a separate tablespace file for each table. In this case, each table has an `.ibd` file in its database directory.

To configure individual tablespaces, enable the `innodb_file_per_table` system variable, either at server startup or at runtime. Enabling this variable also enables other InnoDB features, such as fast table truncation and row storage formats that offer more efficient table processing for some kinds of data. For more information, see Section 12.5.3.1.4, “Using Individual (Per-Table) InnoDB Tablespaces.”

### 2.6.1.4 The MyISAM Storage Engine

The MyISAM storage engine offers these features:

- Key compression when storing runs of successive similar string index values. MyISAM also can compress runs of similar numeric index values because numeric values are stored with the high byte first. (Index values tend to vary faster in the low-order bytes, so high-order bytes are more subject to compression.) To enable numeric compression, use the `PACK_KEYS=1` option when creating a MyISAM table.

- More features for `AUTO_INCREMENT` columns than provided by other storage engines. For more information, see Section 3.4, “Working with Sequences.”

- Each MyISAM table has a flag that is set when a table-check operation is performed. MyISAM tables also have a flag indicating whether a table was closed properly when last used. If the server shuts down abnormally or the machine crashes, the flags can be used to detect tables that need to be checked. To do this automatically, start the server with the `myisam_recover_options` system variable set to a value that includes the `FORCE` option. This causes the server to check the table flags whenever it opens a MyISAM table and perform a table repair if necessary. See Section 14.3.1, “Using the Server’s Auto-Recovery Capabilities.”

- Full-text searches and `FULLTEXT` indexes.

- Spatial data types and `SPATIAL` indexes.

### 2.6.1.5 The MEMORY Storage Engine

The MEMORY storage engine uses tables that are stored in memory and that have fixed-length rows, two properties that make them very fast.

MEMORY tables are temporary in the sense that their contents disappear when the server terminates. That is, a MEMORY table still exists when the server restarts, but will be empty. However, in contrast to temporary tables created with `CREATE TEMPORARY TABLE`, MEMORY tables are visible to other clients.
MEMORY tables have characteristics that enable them to be handled more simply, and thus more quickly:

- By default, MEMORY tables use hashed indexes, which are very fast for equality comparisons but slow for range comparisons. Consequently, hashed indexes are used only for comparisons performed with the `=` and `<=>` equality operators, but not for comparison operators such as `<` or `>`. Hashed indexes also are not used in `ORDER BY` clauses for this reason.

- Rows are stored in MEMORY tables using fixed-length format for easier processing. A consequence is that you cannot use the `BLOB` and `TEXT` variable-length data types. `VARCHAR` is a variable-length type, but is permitted because it is treated internally as `CHAR`, a fixed-length type.

To use a MEMORY table for comparisons that look for a range of values using operators such as `<`, `>`, or `BETWEEN`, you can use `BTREE` indexes instead of hashed indexes. See Section 2.6.4.2, “Creating Indexes,” and Section 5.1.3, “Choosing Indexes.”

### 2.6.1.6 The NDB Storage Engine

NDB is MySQL’s cluster storage engine. For this storage engine, the MySQL server actually acts as a client to a cluster of other processes that provide access to the NDB tables. Cluster node processes communicate with each other to manage tables in memory. The tables are replicated among cluster processes for redundancy. Memory storage provides high performance, and the cluster provides high availability because it survives failure of any given node.

NDB configuration and use is beyond the scope of this book and is not covered further here. See the MySQL Reference Manual for details.

### 2.6.1.7 Other Storage Engines

MySQL has several other storage engines that I group here under the “miscellaneous” category:

- The ARCHIVE engine provides archival storage. It’s intended for storage of large numbers of rows that are written once and never modified thereafter. For this reason, it supports only a limited number of statements. `INSERT` and `SELECT` work, but `REPLACE` always acts like `INSERT`, and you cannot use `DELETE` or `UPDATE`. Rows are compressed during storage and decompressed during retrieval to save space. An ARCHIVE table can include an indexed `AUTO_INCREMENT` column; other columns cannot be indexed.

- The BLACKHOLE engine creates tables for which writes are ignored and reads return nothing. It is the database equivalent of the Unix `/dev/null` device.

- The CSV engine stores data in comma-separated values format. For each table, it creates a `.CSV` file in the database directory. This is a plain text file in which each table row appears as a single line. The CSV engine does not support indexing.
The FEDERATED engine provides access to tables that are managed by other MySQL servers. In other words, the contents of a FEDERATED table really are located remotely. For a FEDERATED table, you specify the host where the other server is running and provide the username and password of an account on that server. When you access the FEDERATED table, the local server connects to the remote server using this account.

The MERGE engine provides a means of grouping a set of MyISAM tables into a single logical unit. Querying a MERGE table in effect queries all the constituent tables. One advantage of this is that you can exceed the maximum table size permitted by the filesystem for individual MyISAM tables. Partitioned tables provide an alternative to MERGE tables and are not limited to MyISAM tables. See Section 2.6.2.5, “Using Partitioned Tables.”

2.6.2 Creating Tables

To create a table, use a CREATE TABLE statement. You must have the CREATE privilege for the table. The full syntax for this statement is complex because there are so many optional clauses, but it’s usually fairly simple to use in practice. For example, most of the CREATE TABLE statements that we used in Chapter 1, “Getting Started with MySQL,” are reasonably uncomplicated. If you start with the more basic forms and work up, you shouldn’t have much trouble.

A CREATE TABLE statement specifies, at a minimum, the table name and a list of the columns in it. For example:

```sql
CREATE TABLE mytbl
{
    name CHAR(20),
    birth DATE NOT NULL,
    weight INT,
    sex ENUM('F','M')
};
```

In addition to the column definitions, you can specify how the table should be indexed when you create it. Another option is to leave the table unindexed when you create it and add the indexes later. For MyISAM tables, that’s a good strategy if you plan to populate the table with a lot of data before you begin using it for queries. Updating indexes as you insert each row is much slower than loading the data into an unindexed MyISAM table and creating the indexes afterward.

We have already covered the basic syntax for CREATE TABLE in Chapter 1, “Getting Started with MySQL.” Details on how to write column definitions are given in Chapter 3, “Data Types.” Here, we deal more generally with some important extensions to CREATE TABLE that give you a lot of flexibility in how you construct tables:

- Table options that modify storage characteristics
- Creating a table only if it doesn’t already exist
- Temporary tables that the server drops automatically when the client session ends
Chapter 2 Using SQL to Manage Data

- Creating a table from another table or from the result of a `SELECT` query
- Using partitioned tables

### 2.6.2.1 Table Options

To modify a table's storage characteristics, add one or more table options following the closing parenthesis in the `CREATE TABLE` statement. For a complete list of options, see the description for `CREATE TABLE` in Appendix E, “SQL Syntax Reference.”

One table option is `ENGINE = engine_name`, which specifies the storage engine to use for the table. For example, to create a MEMORY or MyISAM table, write the statement like this:

```sql
CREATE TABLE mytbl ( ... ) ENGINE=MEMORY;
CREATE TABLE mytbl ( ... ) ENGINE=MyISAM;
```

The engine name is not case sensitive. With no `ENGINE` option, the server creates the table using the default storage engine. The built-in default is InnoDB, but you can tell the server to use a different default using the instructions in Section 12.5.2, “Selecting a Default Storage Engine.”

If you name a storage engine that is not enabled, two warnings occur:

```sql
mysql> CREATE TABLE t (i INT) ENGINE=ARCHIVE;
Query OK, 0 rows affected, 2 warnings (0.01 sec)
mysql> SHOW WARNINGS;
+---------+------+-------------------------------------------+
| Level   | Code | Message                                   |
+---------+------+-------------------------------------------+
| Warning | 1286 | Unknown storage engine 'ARCHIVE'          |
| Warning | 1266 | Using storage engine InnoDB for table 't'  |
+---------+------+-------------------------------------------+
```

To make sure that a table uses a particular storage engine, be sure to include the `ENGINE` table option. Because the default engine can be changed, you might not get the default you expect if you omit `ENGINE`. In addition, verify that the `CREATE TABLE` statement produces no warnings, which often indicate that the specified engine was not available and that the default engine was used instead.

To tell MySQL to issue an error if the engine you specify is not available, (instead of substituting the default storage engine), enable the `NO_ENGINE_SUBSTITUTION` SQL mode.

To determine which storage engine a table uses, issue a `SHOW CREATE TABLE` statement and look for the `ENGINE` option in the output:

```sql
mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
Table: t
Create Table: CREATE TABLE `t` (  
`i` int(11) DEFAULT NULL  ) ENGINE=MyISAM DEFAULT CHARSET=latin1
```
The storage engine is also available in the output from the `SHOW TABLE STATUS` statement or the `INFORMATION_SCHEMA.TABLES` table.

The `MAX_ROWS` and `AVG_ROW_LENGTH` options can help you size a MyISAM table. By default, MyISAM creates tables with an internal row pointer size that permits table files to grow up to 256TB. If you specify the `MAX_ROWS` and `AVG_ROW_LENGTH` options, that gives MyISAM information that it should use a pointer size for a table that can hold at least `MAX_ROWS` rows.

To modify the storage characteristics of an existing table, table options can be used with an `ALTER TABLE` statement. For example, to change `mytbl` from its current storage engine to InnoDB, do this:

```sql
ALTER TABLE mytbl ENGINE=InnoDB;
```

For more information about changing storage engines, see Section 2.6.5, “Altering Table Structure.”

### 2.6.2.2 Provisional Table Creation

To create a table only if it doesn’t already exist, use `CREATE TABLE IF NOT EXISTS`. You can use this statement for an application that makes no assumptions about whether a table that it needs has been set up in advance. The application can go ahead and attempt to create the table as a matter of course. The `IF NOT EXISTS` modifier is particularly useful for scripts that you run as batch jobs with `mysql`. In this context, a regular `CREATE TABLE` statement doesn’t work very well. The first time the job runs, it creates the table, but the second time, an error occurs because the table already exists. If you use `IF NOT EXISTS`, there is no problem. The first time the job runs, it creates the table, as before. For second and subsequent times, table creation attempts are silently ignored without error. This enables the job to continue processing as if the attempt had succeeded.

If you use `IF NOT EXISTS`, be aware that MySQL does not compare the table structure in the `CREATE TABLE` statement with the existing table. If a table exists with the given name but has a different structure, the statement does not fail. If that is a risk you wish not to take, it is better instead to precede your `CREATE TABLE` statement by `DROP TABLE IF EXISTS`.

### 2.6.2.3 TEMPORARY Tables

Adding the `TEMPORARY` keyword to a table-creation statement causes the server to create a temporary table that disappears automatically when your session with the server terminates:

```sql
CREATE TEMPORARY TABLE tbl_name ... ;
```

This is handy because you need not issue a `DROP TABLE` statement to get rid of the table, and the table doesn’t persist if your session terminates abnormally. For example, if you have a complex query stored in a batch file that you run with `mysql` and you decide not to wait for it to finish, you can kill the script with impunity and the server will remove any `TEMPORARY` tables created by the script.
To create a temporary table using a particular storage engine, add an `ENGINE` table option to the `CREATE TEMPORARY TABLE` statement.

Although the server drops a `TEMPORARY` table automatically when your client session ends, you can drop it explicitly as soon as you’re done with it to enable the server to free any resources associated with it. This is a good idea if your session with the server will not end for a while, particularly for temporary MEMORY tables.

A `TEMPORARY` table is visible only to the client that creates the table. Different clients can each create a `TEMPORARY` table with the same name and without conflict because each client sees only the table that it created.

The name of a `TEMPORARY` table can be the same as an existing permanent table. This is not an error, nor does the existing permanent table get clobbered. Instead, the permanent table becomes hidden (inaccessible) to the client that creates the `TEMPORARY` table while the `TEMPORARY` table exists. Suppose that you create a `TEMPORARY` table named `member` in the `sampdb` database. The original `member` table becomes hidden, and references to `member` refer to the `TEMPORARY` table. If you issue a `DROP TABLE member` statement, the `TEMPORARY` table is removed and the original member table “reappears.” If you disconnect from the server without dropping the `TEMPORARY` table, the server automatically drops it for you. The next time you connect, the original `member` table is visible again. (The original table also reappears if you rename a `TEMPORARY` table that hides it to have a different name.)

The name-hiding mechanism works only to one level. That is, you cannot create two `TEMPORARY` tables with the same name.

Keep in mind the following caveats when considering whether to use a `TEMPORARY` table:

- If your client program automatically reconnects to the server if the connection is lost, any `TEMPORARY` tables will be gone when you reconnect. If you were using the `TEMPORARY` table to “hide” a permanent table with the same name, the permanent table now becomes the table that you use. For example, a `DROP TABLE` after an undetected reconnect will drop the permanent table. To avoid this problem, use `DROP TEMPORARY TABLE` instead.

- Because `TEMPORARY` tables are visible only within the session that created them, they are not useful with connection pooling mechanisms that do not guarantee the same connection for each statement that you issue.

- With connection pooling or persistent connections, your connection to the MySQL server will not necessarily close when your application terminates. Those mechanisms might hold the connection open for use by other clients, which means that you cannot assume that `TEMPORARY` tables will disappear automatically when your application terminates.

### 2.6.2.4 Creating Tables from Other Tables or Query Results

It’s sometimes useful to create a copy of a table. For example, you might have a data file that you want to load into a table using `LOAD DATA`, but you’re not quite sure about the options for
specifying the data format. You can end up with malformed rows in the original table if you don’t get the options right the first time. Using an empty copy of the original table enables you to experiment with the `LOAD DATA` options for specifying column and line delimiters until you’re satisfied your input rows are being interpreted properly. Then you can load the file into the original table by rerunning the `LOAD DATA` statement with the original table name.

It’s also sometimes desirable to save the result of a query into a table rather than displaying it on your screen. By saving the result, you can refer to it later without rerunning the original query, perhaps to perform further analysis on it.

MySQL provides two statements for creating new tables from other tables or from query results. These statements have differing advantages and disadvantages:

- **CREATE TABLE ... LIKE** creates a new table as an empty copy of the original one. It copies the original table structure exactly, so that each column is preserved with all of its attributes. The index structure is copied as well. However, the new table is empty, so to populate it a second statement is needed (such as `INSERT INTO ... SELECT`). Also, `CREATE TABLE ... LIKE` cannot create a new table from a subset of the original table’s columns, and it cannot use columns from any other table but the original one.

- **CREATE TABLE ... SELECT** creates a new table from the result of an arbitrary `SELECT` statement. By default, this statement does not copy all column attributes such as `AUTO_INCREMENT`. Nor does creating a table by selecting data into it automatically copy any indexes from the original table, because result sets are not themselves indexed. On the other hand, `CREATE TABLE ... SELECT` can both create and populate the new table in a single statement. It also can create a new table using a subset of the original table and include columns from other tables or columns created as the result of expressions.

To use `CREATE TABLE ... LIKE` for creating an empty copy of an existing table, write a statement like this:

```
CREATE TABLE new_tbl_name LIKE tbl_name;
```

To create an empty copy of a table and then populate it from the original table, use `CREATE TABLE ... LIKE` followed by `INSERT INTO ... SELECT`:

```
CREATE TABLE new_tbl_name LIKE tbl_name;
INSERT INTO new_tbl_name SELECT * FROM tbl_name;
```

To create a table as a temporary copy of itself, include the `TEMPORARY` keyword:

```
CREATE TEMPORARY TABLE tbl_name LIKE tbl_name;
INSERT INTO tbl_name SELECT * FROM tbl_name;
```

Using a `TEMPORARY` table with the same name as the original can be useful when you want to try some statements that modify the contents of a table, without changing the original table. To use prewritten scripts that use the original table name, you need not edit them to refer to a different table. Just add the `CREATE TEMPORARY TABLE` and `INSERT` statements to the beginning of the script. The script will create a temporary copy and operate on the copy, which the server deletes when the script finishes. (However, bear in mind the auto-reconnect caveat noted in Section 2.6.2.3, “TEMPORARY Tables.”)
To insert into the new table only some of the rows from the original table, add a *WHERE*
clause that identifies which rows to select. The following statements create a new table named
*student_f* that contains only the rows for female students from the *student* table:

```
CREATE TABLE student_f LIKE student;
INSERT INTO student_f SELECT * FROM student WHERE sex = 'f';
```

If you don’t care about retaining the exact column definitions from the original table, *CREATE
TABLE ... SELECT* sometimes is easier to use than *CREATE TABLE ... LIKE* because it can create
and populate the new table in a single statement:

```
CREATE TABLE student_f SELECT * FROM student WHERE sex = 'f';
```

*CREATE TABLE ... SELECT* also can create new tables that don’t contain exactly the same set of
columns in an existing table. You can use it to cause a new table to spring into existence on the
fly to hold the result of an arbitrary *SELECT* query. This makes it exceptionally easy to create a
table fully populated with the data in which you’re interested, ready to be used in further state-
ments. However, the new table can contain strange column names if you’re not careful. When
you create a table by selecting data into it, the column names are taken from the columns
that you are selecting. If a column is calculated as the result of an expression, the name of the
column is the text of the expression, which creates a table with an unusual column name:

```
mysql> CREATE TABLE mytbl SELECT PI() * 2;
```

```
+----------+
| PI() * 2 |
+----------+
| 6.283185 |
```

That’s unfortunate, because the column name can be referred to directly only as a quoted
identifier:

```
mysql> SELECT `PI() * 2` FROM mytbl;
```

```
+----------+
| PI() * 2 |
+----------+
| 6.283185 |
```

To avoid this problem, use a column alias to provide a name that is easier to work with:

```
mysql> DROP TABLE mytbl;
mysql> CREATE TABLE mytbl SELECT PI() * 2 AS mycol;
mysql> SELECT mycol FROM mytbl;
```

```
+----------+
| mycol    |
+----------+
| 6.283185 |
```

```
A related difficulty occurs if you select from different tables columns that have the same name. Suppose that tables t1 and t2 both have a column c and you want to create a table from all combinations of rows in both tables. The following statement fails because it attempts to create a table with two columns named c:

```
mysql> CREATE TABLE t3 SELECT * FROM t1 INNER JOIN t2;
ERROR 1060 (42S21): Duplicate column name 'c'
```

To solve this problem, provide aliases as necessary to give each column a unique name in the new table:

```
mysql> CREATE TABLE t3 SELECT t1.c, t2.c AS c2
    -> FROM t1 INNER JOIN t2;
```

As mentioned previously, a shortcoming of `CREATE TABLE ... SELECT` is that it does not incorporate all characteristics of the original data into the structure of the new table. For example, creating a table by selecting data into it does not copy indexes from the original table, and it can lose column attributes. The retained attributes include whether the column is `NULL` or `NOT NULL`, the character set and collation, the default value, and the column comment.

In some cases, you can force specific attributes to be used in the new table by invoking the `CAST()` function in the `SELECT` part of the statement. The following `CREATE TABLE ... SELECT` statement forces the columns produced by the `SELECT` to be treated as `INT UNSIGNED`, `TIME`, and `DECIMAL(10,5)`, as you can verify with `DESCRIBE`:

```
mysql> CREATE TABLE mytbl SELECT
    -> CAST(1 AS UNSIGNED) AS i,
    -> CAST(CURTIME() AS TIME) AS t,
    -> CAST(PI() AS DECIMAL(10,5)) AS d;
```

```
mysql> DESCRIBE mytbl;
+-------+-----------------+------+-----+---------+-------+
| Field | Type            | Null | Key | Default | Extra |
|-------+-----------------+------+-----+---------+-------+
| i     | int(1) unsigned | NO   |     | 0       |       |
| t     | time            | YES  | NULL|         |       |
| d     | decimal(10,5)   | NO   |     | 0.00000 |       |
+-------+-----------------+------+-----+---------+-------+
```

The permitted cast types are `BINARY`, `CHAR`, `DATE`, `DATETIME`, `TIME`, `SIGNED`, `UNSIGNED`, `UNSIGNED INTEGER`, and `DECIMAL`.

It is also possible to provide explicit column definitions in the `CREATE TABLE` part, to be used for the columns retrieved by the `SELECT` part. Columns in the two parts are matched by name (not position), so provide aliases in the `SELECT` part as necessary to cause them to match properly:

```
mysql> CREATE TABLE mytbl (i INT UNSIGNED, t TIME, d DECIMAL(10,5))
    -> SELECT
    -> 1 AS i,
    -> CAST(CURTIME() AS TIME) AS t;
```
CAST(PI() AS DECIMAL(10,5)) AS d;
mysql> DESCRIBE mytbl;
+---------------------+--------------+----------+---------+-----------+--------+
| Field   | Type         | Null    | Key    | Default  | Extra  |
+---------------------+--------------+----------+---------+-----------+--------+
| i        | int(10) unsigned | YES     |         | NULL     |        |
| t        | time         | YES     |         | NULL     |        |
| d        | decimal(10,5) | YES     |         | NULL     |        |
+---------------------+--------------+----------+---------+-----------+--------+

The technique of providing explicit definitions enables you to create numeric columns with specified precision and scale, character columns that have a different width than the longest value in the result set, and so forth. Also note that the Null and Default attributes for some of the columns differ in this example from those in the previous one. You can provide explicit definitions for those attributes in the CREATE TABLE part if necessary.

2.6.2.5 Using Partitioned Tables

MySQL supports table partitioning, which enables division of table contents into different physical storage locations. By sectioning table storage, partitioned tables offer benefits such as these:

- Table storage can be distributed over multiple devices, which may improve access time by virtue of I/O parallelism.
- The optimizer may be able to localize searches to specific partitions, or to search partitions in parallel.

To create a partitioned table, supply the list of columns and indexes in the CREATE TABLE statement, as usual. In addition, specify a PARTITION BY clause that defines a partitioning function to be used to assign rows to partitions, and possibly other partition-related options. A partitioning function assigns rows based on ranges or lists of values or hash values:

- Use range partitioning when rows contain a domain of values such as dates, income level, or weight that can be divided into discrete ranges.
- Use list partitioning when it makes sense to specify an explicit list of values for each partition, such as sets of postal codes, phone number prefixes, or IDs for entities that you group by geographical region.
- Use hash partitioning to distribute the rows among partitions according to hash values computed from row keys. You can either supply the hash function yourself or tell MySQL which columns to use and it computes values based on those columns using a built-in hash function.

The partitioning function must be deterministic so that the same input values consistently result in row assignment to the same partition. This rules out functions such as RAND() or NOW().
Suppose that you want to create a table for storing simple log entries consisting of a date and a descriptive string, and that you already have several years’ worth of entries to be loaded into the table. For data entries that each contain a date, range partitioning is most natural. To assign rows for each year to a given partition, use the year part of the date value:

```
CREATE TABLE log_partition
(
    dt    DATETIME NOT NULL,
    info  VARCHAR(100) NOT NULL,
    INDEX (dt)
)
PARTITION BY RANGE(YEAR(dt))
(
    PARTITION p0 VALUES LESS THAN (2010),
    PARTITION p1 VALUES LESS THAN (2011),
    PARTITION p2 VALUES LESS THAN (2012),
    PARTITION p3 VALUES LESS THAN (2013),
    PARTITION pmax VALUES LESS THAN MAXVALUE
);
```

The `MAXVALUE` partition is assigned all rows that have dates from the year 2014 or later. When the year 2014 arrives, you can split that partition so that all year 2014 rows get their own partition and rows for 2015 and later go into the `MAXVALUE` partition:

```
ALTER TABLE log_partition REORGANIZE PARTITION pmax
INTO (
    PARTITION p4 VALUES LESS THAN (2014),
    PARTITION pmax VALUES LESS THAN MAXVALUE
)
```

By default, MySQL stores partitions under the directory for the database to which the partitioned table belongs. To distribute storage to other locations (for example, to place them on different physical devices), use the `DATA_DIRECTORY` and `INDEX_DIRECTORY` partition options. For more information about the syntax for these and other partitioning options, see the description for `CREATE TABLE` in Appendix E, “SQL Syntax Reference.”

### 2.6.3 Dropping Tables

Dropping a table is much easier than creating it because you need not specify anything about the format of its contents. You just have to name it, assuming that you have the `DROP` privilege for it:

```
DROP TABLE tbl_name;
```

In MySQL, the `DROP TABLE` statement has several useful extensions. To drop multiple tables, specify them all in the same statement:

```
DROP TABLE tbl_name1, tbl_name2, ... ;
```
By default, an error occurs if you try to drop a table that does not exist. To suppress this error and generate a warning instead for nonexistent tables, include `IF EXISTS` in the statement:

```
DROP TABLE IF EXISTS tbl_name;
```

If the statement generates warnings, you can view them with `SHOW WARNINGS`.

`IF EXISTS` is particularly useful in scripts that you use with the `mysql` client. By default, `mysql` exits when an error occurs, and it is an error to try to remove a table that doesn't exist. For example, you might have a setup script that creates tables used as the basis for further processing in other scripts. In this situation, you want to make sure the setup script has a clean slate when it begins. If you use a regular `DROP TABLE` at the beginning of the script, it fails the first time because the tables have never been created. Using `IF EXISTS` makes the problem go away. If the tables exist, they are dropped. If they do not exist, no error occurs and the script continues to execute.

To drop a table only if it is a temporary table, include the `TEMPORARY` keyword:

```
DROP TEMPORARY TABLE tbl_name;
```

### 2.6.4 Indexing Tables

Indexes are the primary means of speeding up access to the contents of your tables, particularly for queries that involve joins on multiple tables. This is an important enough topic that most of an entire chapter discusses why you use indexes, how they work, and how best to take advantage of them to optimize your queries (see Chapter 5, “Query Optimization”). This section covers the characteristics of indexes for the various table types and the syntax for creating and dropping indexes.

#### 2.6.4.1 Storage Engine Index Characteristics

MySQL provides quite a bit of flexibility for index construction:

- You can index single columns or multiple columns. Multiple-column indexes are also known as composite indexes.
- An index can be constrained to contain only unique values or permitted to contain duplicate values.
- You can have more than one index on a table to help optimize different types of queries on the table.
- For string data types other than `ENUM` or `SET`, you can elect to index a prefix of a column; that is, only the leftmost \( n \) characters, or \( n \) bytes for binary string types. (For `BLOB` and `TEXT` columns, you can set up an index only if you specify a prefix length.) If the column is mostly unique within the prefix length, you usually won’t sacrifice performance, and may well improve it: Indexing a column prefix rather than the entire column can make an index much smaller and faster to access.
Not all storage engines offer all indexing features. Table 2.3 summarizes the index properties for some of MySQL’s storage engines. The table does not include the MERGE storage engine, because MERGE tables are created from MyISAM tables and have similar index characteristics. Nor does it include the ARCHIVE, BLACKHOLE, or CSV engines, which support indexing either not at all or only in limited fashion.

### Table 2.3  Storage Engine Index Characteristics

<table>
<thead>
<tr>
<th>Index Characteristic</th>
<th>InnoDB</th>
<th>MyISAM</th>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL values permitted</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Columns per index</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Indexes per table</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Maximum index row size (bytes)</td>
<td>3072</td>
<td>1000</td>
<td>3072</td>
</tr>
<tr>
<td>Index column prefixes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Maximum prefix size (bytes)</td>
<td>767</td>
<td>1000</td>
<td>3072</td>
</tr>
<tr>
<td>BLOB/TEXT indexes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>FULLTEXT indexes</td>
<td>As of 5.6.4</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SPATIAL indexes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>HASH indexes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

One implication of the variations in index characteristics for different storage engines is that if you require an index to have certain properties, you may not be able to use certain types of tables. For example, to use a HASH index, you must use a MEMORY table. To index a TEXT column, you must use InnoDB or MyISAM.

To convert an existing table to use a different storage engine that has more suitable index characteristics, use `ALTER TABLE`. Suppose that you have an InnoDB table in MySQL 5.5 but need to perform searches using a FULLTEXT index. In MySQL 5.5, this is supported only by MyISAM. Convert the table using this statement:

```
ALTER TABLE tbl_name ENGINE=MyISAM;
```

### 2.6.4.2  Creating Indexes

MySQL can create several types of indexes:

- A unique index. This prohibits duplicate values for a single-column index, and duplicate combinations of values for a multiple-column (composite) index.
- A regular (nonunique) index. This gives you indexing benefits but permits duplicates.
A **FULLTEXT** index, used for performing full-text searches. This index type is supported only for MyISAM tables (or, as of MySQL 5.6.4, InnoDB). For more information, see Section 2.14, “Using FULLTEXT Searches.”

A **SPATIAL** index. These can be used only with MyISAM tables containing spatial values, which are described briefly in Section 3.1.4, “Spatial Values.”

A **HASH** index. This is the default index type for **MEMORY** tables, although you can override the default to create **BTREE** indexes instead.

You can include index definitions for a new table when you use `CREATE TABLE`. For examples, see Section 1.4.6, “Creating Tables.” To add indexes to existing tables, use `ALTER TABLE` or `CREATE INDEX`. (MySQL maps `CREATE INDEX` statements onto `ALTER TABLE` operations internally.)

`ALTER TABLE` is the more versatile than `CREATE INDEX` because it can create any kind of index supported by MySQL. For example:

```
ALTER TABLE tbl_name ADD INDEX index_name (index_columns);
ALTER TABLE tbl_name ADD UNIQUE index_name (index_columns);
ALTER TABLE tbl_name ADD PRIMARY KEY (index_columns);
ALTER TABLE tbl_name ADD FULLTEXT index_name (index_columns);
ALTER TABLE tbl_name ADD SPATIAL index_name (index_columns);
```

`tbl_name` is the name of the table to which the index should be added, and `index_columns` names the column or columns to index, separated by commas. The index name `index_name` is optional. If you leave it out, MySQL picks a name based on the name of the first indexed column.

An indexed column must be **NOT NULL** if indexed using a **PRIMARY KEY** or **SPATIAL** index. Other indexes permit indexed columns to contain **NULL** values.

A single `ALTER TABLE` statement can include multiple table alterations if you separate them by commas. This enables you to create several indexes at the same time, which is faster than adding them one at a time with individual `ALTER TABLE` statements.

To constrain an index to contain only unique values, create the index as a **PRIMARY KEY** or as a **UNIQUE** index. The two types of index are very similar, but have two differences:

- A table can contain only one **PRIMARY KEY**. This is because the name of a **PRIMARY KEY** is always **PRIMARY** and a table cannot have two indexes with the same name. You can place multiple **UNIQUE** indexes on a table.
- A **PRIMARY KEY** cannot contain **NULL** values. A **UNIQUE** index can. If a **UNIQUE** index can contain **NULL** values, it can contain multiple **NULL** values. (A **NULL** is not considered equal to any other value, even another **NULL**.)

`CREATE INDEX` can add most types of indexes, with the exception of **PRIMARY KEY**:

```
CREATE INDEX index_name ON tbl_name (index_columns);
CREATE UNIQUE INDEX index_name ON tbl_name (index_columns);
```
CREATE FULLTEXT INDEX index_name ON tbl_name (index_columns);
CREATE SPATIAL INDEX index_name ON tbl_name (index_columns);

tbl_name, index_name, and index_columns have the same meaning as for ALTER TABLE. Unlike ALTER TABLE, the index name is not optional with CREATE INDEX, and you cannot create multiple indexes with a single statement.

To create indexes for a new table with a CREATE TABLE statement, the syntax is similar to that used for ALTER TABLE, but you specify the index-creation clauses in addition to the column definitions:

CREATE TABLE tbl_name
{
    ... column definitions ...
    INDEX index_name (index_columns),
    UNIQUE index_name (index_columns),
    PRIMARY KEY (index_columns),
    FULLTEXT index_name (index_columns),
    SPATIAL index_name (index_columns),
    ...
};

As with ALTER TABLE, index_name is optional. MySQL picks an index name if you leave it out.

As a special case, you can create a single-column PRIMARY KEY or UNIQUE index by adding a PRIMARY KEY or UNIQUE clause to the end of a column definition. For example, the following CREATE TABLE statements are equivalent:

CREATE TABLE mytbl
{
    i INT NOT NULL PRIMARY KEY,
    j CHAR(10) NOT NULL UNIQUE
};

CREATE TABLE mytbl
{
    i INT NOT NULL,
    j CHAR(10) NOT NULL,
    PRIMARY KEY (i),
    UNIQUE (j)
};

The default index type for a MEMORY table is HASH. A hashed index is very fast for exact-value lookups, which is the typical way MEMORY tables are used. However, if you plan to use a MEMORY table for comparisons that can match a range of values (for example, id < 100), hashed indexes do not work well. You’ll be better off creating a BTREE index instead, by adding a USING BTREE clause to the index definition:
CREATE TABLE namelist
(
    id INT NOT NULL,
    name CHAR(100),
    INDEX (id) USING BTREE
) ENGINE=MEMORY;

To index a prefix of a string column, the syntax for naming the column in the index definition is `col_name(n)` rather than simply `col_name`. The prefix value, `n`, indicates that the index should include the first `n` bytes of column values for binary string types, or the first `n` characters for nonbinary string types. For example, the following statement creates a table with a `CHAR` column and a `BINARY` column. It indexes the first 10 characters of the `CHAR` column and the first 15 bytes of the `BINARY` column:

```sql
CREATE TABLE addresslist
(
    name CHAR(30) NOT NULL,
    address BINARY(60) NOT NULL,
    INDEX (name(10)),
    INDEX (address(15))
);
```

When you index a prefix of a string column, the prefix length, just like the column length, is specified in the same units as the column data type—that is, bytes for binary strings and characters for nonbinary strings. However, the maximum size of index entries are measured internally in bytes. The two measures are the same for single-byte character sets, but not for multi-byte character sets. For nonbinary strings that have multi-byte character sets, MySQL stores into index values as many complete characters as fit within the maximum permitted byte length.

In some circumstances, you may find it not only desirable but necessary to index a column prefix rather than the entire column:

- A prefix is required to index a `BLOB` or `TEXT` column.
- The length of index rows is equal to the sum of the length of the index parts of the columns that make up the index. If this length exceeds the maximum permitted number of bytes in index rows, you can make the index “narrower” by indexing a column prefix. Suppose that a MyISAM table that uses the latin1 single-byte character set contains four `CHAR(255)` columns named `c1` through `c4`. An index value for each full column value takes 255 bytes, so an index on all four columns would require 1,020 bytes. However, the maximum length of a MyISAM index row is 1,000 bytes, so you cannot create a composite index that includes the entire contents of all four columns. However, you can create the index by indexing a shorter part of some or all of them. For example, you could index the first 250 characters from each column.

Columns in `FULLTEXT` indexes are indexed in full and do not have prefixes. If you specify a prefix length for a column in a `FULLTEXT` index, MySQL ignores it.
2.6.4.3 Dropping Indexes

To drop an index, use either a DROP INDEX or an ALTER TABLE statement. To use DROP INDEX, you must name the index to be dropped:

```
DROP INDEX index_name ON tbl_name;
```

To drop a PRIMARY KEY with DROP INDEX, specify the name PRIMARY as a quoted identifier:

```
DROP INDEX `PRIMARY` ON tbl_name;
```

That statement is unambiguous because a table is permitted only one PRIMARY KEY and its name is always PRIMARY.

Like the CREATE INDEX statement, DROP INDEX is handled internally as an ALTER TABLE statement. The preceding DROP INDEX statements correspond to the following ALTER TABLE statements:

```
ALTER TABLE tbl_name DROP INDEX index_name;
```

```
ALTER TABLE tbl_name DROP PRIMARY KEY;
```

If you don’t know the names of a table’s indexes, use SHOW CREATE TABLE or SHOW INDEX to find out.

When you drop columns from a table, indexes may be affected implicitly. Dropping a column that is a part of an index removes the column from the index as well. If you drop all columns in an index, MySQL drops the entire index.

2.6.5 Altering Table Structure

ALTER TABLE is a versatile statement and has many uses. We’ve already seen a few of its capabilities earlier in this chapter (for changing storage engines and for creating and dropping indexes). ALTER TABLE can also rename tables, add or drop columns, change column data types, and more. This section covers some of its features. For its complete syntax, see Appendix E, “SQL Syntax Reference.”

ALTER TABLE is useful when you find that the structure of a table no longer reflects its intended use. Perhaps you want to record additional information, or the table contains information that has become superfluous. Maybe existing columns are too small, or it turns out that you’ve defined columns larger than you need and you’d like to make them smaller to save space and improve query performance. Here are some situations for which ALTER TABLE is valuable:

- You assign case numbers to records for a research project using an AUTO_INCREMENT column. You didn’t expect your funding to last long enough to generate more than about 50,000 records, so you made the data type SMALLINT UNSIGNED, which holds a maximum of 65,535 unique values. However, the funding for the project was renewed, and it looks like you might generate another 50,000 records. You need a bigger type to accommodate more case numbers.
Size changes can go the other way, too. Maybe you created a \texttt{CHAR(255)} column but now recognize that no value in the table is more than 100 characters long. You can shorten the column or convert it to \texttt{VARCHAR(255)} to save space.

You want to convert a table to use a different storage engine to take advantage of features offered by that engine. For example, MyISAM tables are not transaction-safe, but you have an application that needs transactional capabilities. You can convert the affected tables to use InnoDB, which supports transactions. Or you might be using MyISAM in MySQL 5.5 because it supports \texttt{FULLTEXT} capabilities, but now you have upgraded to MySQL 5.6, which expands \texttt{FULLTEXT} support to InnoDB.

The syntax for \texttt{ALTER TABLE} looks like this:

\begin{verbatim}
ALTER TABLE tbl_name action [, action] ... ;
\end{verbatim}

Each action specifies a modification to make to the table. Some database systems permit only a single action in an \texttt{ALTER TABLE} statement, but MySQL supports multiple actions, separated by commas.

\textbf{Tip}

If you need to remind yourself about a table's current definition before using \texttt{ALTER TABLE}, issue a \texttt{SHOW CREATE TABLE} statement. This statement is also useful after \texttt{ALTER TABLE} to verify that the alteration affected the table definition as you expect.

The following examples discuss some of the capabilities of \texttt{ALTER TABLE}.

**Changing a column's data type.** To change a data type, use either a \texttt{CHANGE} or \texttt{MODIFY} clause. Suppose that the column \texttt{i} in a table \texttt{mytbl} is \texttt{SMALLINT UNSIGNED}. To change it to \texttt{MEDIUMINT UNSIGNED}, use either of the following statements:

\begin{verbatim}
ALTER TABLE mytbl MODIFY i MEDIUMINT UNSIGNED;
ALTER TABLE mytbl CHANGE i i MEDIUMINT UNSIGNED;
\end{verbatim}

Why is the column named twice in the statement that uses \texttt{CHANGE}? Because one thing that \texttt{CHANGE} can do that \texttt{MODIFY} cannot is to rename the column in addition to changing the type. If you had wanted to rename \texttt{i} to \texttt{k} at the same time you changed the type, you'd do so like this:

\begin{verbatim}
ALTER TABLE mytbl CHANGE i k MEDIUMINT UNSIGNED;
\end{verbatim}

Remember that with \texttt{CHANGE}, you name the column you want to change and then specify its new name and definition. To retain the same column name, you must specify the name twice.

To rename a column without changing its data type, use \texttt{CHANGE old_name new_name} followed by the column's current definition.

To change a column's character set, use the \texttt{CHARACTER SET} attribute in the column definition:

\begin{verbatim}
ALTER TABLE t MODIFY c CHAR(20) CHARACTER SET ucs2;
\end{verbatim}
An important reason for changing data types is to improve query efficiency for joins that compare columns from two tables. Indexes often can be used for comparisons in joins between similar column types, but comparisons are quicker when both columns are exactly the same type. Suppose that you’re running a query like this:

```
SELECT ... FROM t1 INNER JOIN t2 WHERE t1.name = t2.name;
```

If `t1.name` is `CHAR(10)` and `t2.name` is `CHAR(15)`, the query won’t run as quickly as if they were both `CHAR(15)`. You can make them the same by changing `t1.name` using either of these statements:

```
ALTER TABLE t1 MODIFY name CHAR(15);
ALTER TABLE t1 CHANGE name name CHAR(15);
```

**Converting a table to a different storage engine.** To convert a table from one storage engine to another, use an `ENGINE` clause that specifies the new engine name:

```
ALTER TABLE tbl_name ENGINE=engine_name;
```

`engine_name` is a name such as `InnoDB`, `MyISAM`, or `MEMORY`. Lettercase does not matter.

One reason to change a storage engine is to make it transaction-safe. Suppose that you have a MyISAM table and discover that an application that uses it needs to perform transactional operations, including rollback in case failures occur. MyISAM tables do not support transactions, but you can make the table transaction-safe by converting it to use InnoDB:

```
ALTER TABLE tbl_name ENGINE=InnoDB;
```

When you convert a table to a different engine, the permitted or sensible conversions may depend on the feature compatibility of the old and new engines. For example, if you have a table that includes a `BLOB` column, you cannot convert the table to use the MEMORY engine because MEMORY tables do not support `BLOB` columns.

There are circumstances under which you should not use `ALTER TABLE` to convert a table to use a different storage engine. For example:

- An InnoDB table can be converted to use another storage engine. However, if the table has foreign key constraints, they will be lost because only InnoDB supports foreign keys.
- MEMORY tables are held in memory and disappear when the server exits. If you require a table’s contents to persist across server restarts, do not convert it to use the MEMORY engine.

**Renaming a table.** Use a `RENAME` clause that specifies the new table name:

```
ALTER TABLE tbl_name RENAME TO new_tbl_name;
```

Another way to rename tables is with `RENAME TABLE`. The syntax looks like this:

```
RENAME TABLE tbl_name TO new_tbl_name;
```
One thing that RENAME TABLE can do that ALTER TABLE cannot is rename multiple tables in the same statement. For example, you can swap the names of two tables like this:

RENAME TABLE t1 TO tmp, t2 TO t1, tmp TO t2;

If you qualify a table name with a database name, you can move a table from one database to another by renaming it. Either of the following statements move the table t from the sampdb database to the test database:

ALTER TABLE sampdb.t RENAME TO test.t;
RENAME TABLE sampdb.t TO test.t;

You cannot rename a table to a name that already exists.

### 2.7 Obtaining Database Metadata

MySQL provides several ways to obtain database metadata—that is, information about databases and the objects in them:

- **SHOW** statements such as SHOW DATABASES or SHOW TABLES
- Tables in the INFORMATION_SCHEMA database
- Command-line programs such as mysqlshow or mysqldump

The following sections describe how to use each of these information sources to access metadata.

#### 2.7.1 Obtaining Metadata with SHOW

MySQL provides a SHOW statement that displays many types of database metadata. SHOW is helpful for keeping track of the contents of your databases and reminding yourself about the structure of your tables. The following examples demonstrate a few uses for SHOW statements.

List the databases you can access:

SHOW DATABASES;

Display the CREATE DATABASE statement for a database:

SHOW CREATE DATABASE db_name;

List the tables in the default database or a given database:

SHOW TABLES;
SHOW TABLES FROM db_name;

SHOW TABLES doesn’t show TEMPORARY tables.

Display the CREATE TABLE statement for a table:

SHOW CREATE TABLE tbl_name;
Display information about columns or indexes in a table:

SHOW COLUMNS FROM `tbl_name`;
SHOW INDEX FROM `tbl_name`;

The `DESCRIBE tbl_name` and `EXPLAIN tbl_name` statements are synonymous with `SHOW COLUMNS FROM tbl_name`.

Display descriptive information about tables in the default database or in a given database:

SHOW TABLE STATUS;
SHOW TABLE STATUS FROM `db_name`;

Several forms of the `SHOW` statement take a `LIKE 'pattern'` clause permitting a pattern to be given that limits the scope of the output. MySQL interprets `'pattern'` as an SQL pattern that may include the ‘%’ and ‘_’ wildcard characters. For example, this statement displays the names of columns in the `student` table that begin with ‘s’:

```sql
mysql> SHOW COLUMNS FROM student LIKE 's%';
```

```
+------------+------------------+------+-----+---------+----------------+
| Field      | Type             | Null | Key | Default | Extra          |
+------------+------------------+------+-----+---------+----------------+
| sex        | enum('F','M')    | NO   |     | NULL    |                |
| student_id | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
+------------+------------------+------+-----+---------+----------------+
```

To match a literal instance of a wildcard character in a `LIKE` pattern, precede it with a backslash. This is commonly done to match a literal ‘_’, which occurs frequently in database, table, and column names.

Any `SHOW` statement that supports a `LIKE` clause can also be written to use a `WHERE` clause. The statement displays the same columns, but `WHERE` provides more flexibility about specifying which rows to return. The `WHERE` clause should refer to the `SHOW` statement column names. If the column name is a reserved word such as `KEY`, specify it as a quoted identifier. This statement determines which column in the `student` table is the primary key:

```sql
mysql> SHOW COLUMNS FROM student WHERE `Key` = `PRI`;
```

```
+------------+------------------+------+-----+---------+----------------+
| Field      | Type             | Null | Key | Default | Extra          |
+------------+------------------+------+-----+---------+----------------+
| student_id | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
+------------+------------------+------+-----+---------+----------------+
```

It’s sometimes useful to be able to tell from within an application whether a given table exists. You can use `SHOW TABLES` to find out (unless the table is a `TEMPORARY` table):

SHOW TABLES LIKE 'tbl_name';
SHOW TABLES FROM `db_name` LIKE 'tbl_name';

If the `SHOW TABLES` statement lists information for the table, it exists. It’s also possible to determine table existence, even for `TEMPORARY` tables, with either of the following statements:
SELECT COUNT(*) FROM tbl_name;
SELECT * FROM tbl_name WHERE FALSE;

Each statement succeeds if the table exists, and fails if it doesn’t. The first statement is most appropriate for MyISAM tables, for which `COUNT(*)` with no `WHERE` clause is highly optimized. It’s not so good for InnoDB tables, which require a full scan to count the rows. The second statement is more general because it runs quickly for any storage engine. These statements are most suitable for use within application programming languages such as Perl or PHP because you can test the success or failure of the query and take action accordingly. They’re not especially useful in a batch script that you run from `mysql` because you can’t do anything if an error occurs except terminate (or ignore the error, but then there’s obviously no point in running the query at all). Another strategy, which works in any context without failure, is to query the `INFORMATION_SCHEMA` database. See Section 2.7.2, “Obtaining Metadata with `INFORMATION_SCHEMA`.”

To determine the storage engine for individual tables, you can use `SHOW TABLE STATUS` or `SHOW CREATE TABLE`. The output from either statement includes a storage engine indicator.

### 2.7.2 Obtaining Metadata with `INFORMATION_SCHEMA`

Another way to obtain information about databases is to access the `INFORMATION_SCHEMA` database. `INFORMATION_SCHEMA` is based on the SQL standard. That is, the access mechanism is standard, even though some of the content is MySQL-specific. This makes `INFORMATION_SCHEMA` more portable than the various `SHOW` statements, which are entirely MySQL-specific.

`INFORMATION_SCHEMA` is accessed through `SELECT` statements and can be used in a flexible manner. `SHOW` statements always display a fixed set of columns and you cannot capture the output in a table. With `INFORMATION_SCHEMA`, the `SELECT` statement can name specific output columns and a `WHERE` clause can specify any expression required to select the information that you want. Also, you can use joins or subqueries, and you can use `CREATE TABLE ... SELECT` or `INSERT INTO ... SELECT` to save the result of the retrieval in another table for further processing.

You can think of `INFORMATION_SCHEMA` as a virtual database in which the tables are views for different kinds of database metadata. To see what tables `INFORMATION_SCHEMA` contains, use `SHOW TABLES`:

```
mysql> SHOW TABLES IN INFORMATION_SCHEMA;
+---------------------------------------+
| Tables_in_information_schema          |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| ENGINES                               |
| EVENTS                                |
```
2.7 Obtaining Database Metadata

The following list briefly describes some of the INFORMATION_SCHEMA tables just shown:

- **SCHEMATA, TABLES, VIEWS, ROUTINES, TRIGGERS, EVENTS, PARAMETERS, PARTITIONS, COLUMNS**
  Information about databases; tables, views, stored routines, triggers, and events within databases; routine parameters; table partitions; and columns within tables

- **FILES**
  Information about the files used to store tablespace data

- **TABLE_CONSTRAINTS, KEY_COLUMN_USAGE**
  Information about tables and columns that have constraints such as unique-valued indexes or foreign keys

- **STATISTICS**
  Information about table index characteristics

- **REFERENTIAL_CONSTRAINTS**
  Information about foreign keys

- **CHARACTER_SETS, COLLATIONS, COLLATION_CHARACTER_SET_APPLICABILITY**
  Information about supported character sets, collations for each character set, and mapping from each collation to its character set
Chapter 2  Using SQL to Manage Data

■ ENGINES, PLUGINS
  Information about storage engines and server plugins

■ USER_PRIVILEGES, SCHEMA_PRIVILEGES, TABLE_PRIVILEGES, COLUMN_PRIVILEGES
  Global, database, table, and column privilege information from the `user`, `db`, `tables_priv`, and `columns_priv` tables in the `mysql` database

■ GLOBAL_VARIABLES, SESSION_VARIABLES, GLOBAL_STATUS, SESSION_STATUS
  Global and session values of system and status variables

■ PROCESSLIST
  Information about the threads executing within the server

Individual storage engines may add their own tables to INFORMATION_SCHEMA. For example, InnoDB does this.

To determine the columns contained in a given INFORMATION_SCHEMA table, use `SHOW COLUMNS` or `DESCRIBE`:

```
  mysql> DESCRIBE INFORMATION_SCHEMA.CHARACTER_SETS;
  +----------------------+-------------+------+-----+---------+-------+
  | Field                | Type        | Null | Key | Default | Extra |
  +----------------------+-------------+------+-----+---------+-------+
  | CHARACTER_SET_NAME   | varchar(32) | NO   |     |         |       |
  | DEFAULT_COLLATE_NAME | varchar(32) | NO   |     |         |       |
  | DESCRIPTION          | varchar(60) | NO   |     |         |       |
  | MAXLEN               | bigint(3)   | NO   |     | 0       |       |
  +----------------------+-------------+------+-----+---------+-------+
```

To display information from a table, use a `SELECT` statement. (Neither INFORMATION_SCHEMA nor any of its table or column names are case sensitive.) The general query to see all the columns in any given INFORMATION_SCHEMA table is as follows:

```
  SELECT * FROM INFORMATION_SCHEMA.tbl_name;
```

Include a `WHERE` clause to be specific about what you want to see.

The preceding section described the use of `SHOW` statements to determine whether a table exists or which storage engine it uses. INFORMATION_SCHEMA tables can provide the same information. This query uses INFORMATION_SCHEMA to test for the existence of a particular table, returning 1 or 0 to indicate that the table does or does not exist, respectively:

```
  mysql> SELECT COUNT(*) FROM INFORMATION_SCHEMA.TABLES
           -> WHERE TABLE_SCHEMA='sampdb' AND TABLE_NAME='member';
    +-------+
    | COUNT(*) |
    +-------+
    | 1 |
    +-------+
```
Use this query to check which storage engine a table uses:

```sql
mysql> SELECT ENGINE FROM INFORMATION_SCHEMA.TABLES
       -> WHERE TABLE_SCHEMA='sampdb' AND TABLE_NAME='student';
+--------+
| ENGINE |
+--------+
| InnoDB |
+--------+
```

### 2.7.3 Obtaining Metadata from the Command Line

The `mysqlshow` command provides some of the same information as certain `SHOW` statements, which enables you to get database and table information at your command prompt.

**List databases managed by the server:**

```
% mysqlshow
```

**List tables in a database:**

```
% mysqlshow db_name
```

**Display information about columns in a table:**

```
% mysqlshow db_name tbl_name
```

**Display information about indexes in a table:**

```
% mysqlshow --keys db_name tbl_name
```

**Display descriptive information about tables in a database:**

```
% mysqlshow --status db_name
```

The `mysqldump` client program enables you to see the structure of your tables in the form of a `CREATE TABLE` statement (much like `SHOW CREATE TABLE`). If you use `mysqldump` to review table structure, invoke it with the `--no-data` option so that you don’t get swamped with your table’s data!

```
% mysqldump --no-data db_name [tbl_name] ...
```

If you specify only the database name with no table names, `mysqldump` displays the structure for all tables in the database. Otherwise, it shows information only for the named tables.

For both `mysqlshow` and `mysqldump`, specify the usual connection parameter options as necessary, such as `--host`, `--user`, or `--password`. 
2.8 Performing Multiple-Table Retrievals with Joins

It does no good to put records in a database unless you retrieve them eventually and do something with them. That’s the purpose of the `SELECT` statement: to help you get at your data. `SELECT` probably is used more often than any other statement in the SQL language, but it can also be the trickiest; the conditions you use for choosing rows can be arbitrarily complex and can involve comparisons between columns in many tables.

The basic syntax of the `SELECT` statement looks like this:

```
SELECT select_list  # the columns to select
FROM table_list    # the tables from which to select rows
WHERE row_constraint # the conditions rows must satisfy
GROUP BY grouping_columns # how to group results
ORDER BY sorting_columns # how to sort results
HAVING group_constraint # the conditions groups must satisfy
LIMIT count;              # row count limit on results
```

Everything in this syntax is optional except the word `SELECT` and the `select_list` part that specifies what you want to produce as output. Some databases require the `FROM` clause as well. MySQL does not, which enables you to evaluate expressions without referring to any tables:

```
SELECT SQRT(POW(3,2)+POW(4,2));
```

In Chapter 1, “Getting Started with MySQL,” we devoted quite a bit of attention to single-table `SELECT` statements, concentrating primarily on the output column list and the `WHERE`, `GROUP BY`, `ORDER BY`, `HAVING`, and `LIMIT` clauses. This section covers an aspect of `SELECT` that is often confusing: writing joins; that is, `SELECT` statements that retrieve rows from multiple tables. We’ll discuss the types of join MySQL supports, what they mean, and how to specify them. This should help you employ MySQL more effectively because, in many cases, the real problem of figuring out how to write a query is determining the proper way to join tables.

One problem with using `SELECT` is that when you first encounter a new type of problem, it’s not always easy to see how to write a `SELECT` query to solve it. However, after you figure it out, you can use that experience when you run across similar problems in the future. `SELECT` is probably the statement for which past experience plays the largest role in being able to use it effectively, simply because of the sheer variety of problems to which it applies. As you gain experience, you’ll be able to adapt joins more easily to new problems, and you’ll find yourself thinking things like, “Oh, yes, that’s one of those `LEFT JOIN` things,” or, “Aha, that’s a three-way join restricted by the common pairs of key columns.” (You may find it encouraging to hear that experience helps you. Or you may find it alarming to consider that you could wind up thinking in terms like that.)

Many of the examples that demonstrate how to use the forms of join operations that MySQL supports use the following two tables, `t1` and `t2`:

```
| i1 | c1 |
|----+----|
```

```
| i2 | c2 |
```

Table t1: Table t2:
The tables are deliberately small so the effect of each type of join can be readily seen.

Other types of multiple-table SELECT statement are subqueries (one SELECT nested within another) and UNION statements. These are covered in Section 2.9, “Performing Multiple-Table Retrievals with Subqueries,” and Section 2.10, “Performing Multiple-Table Retrievals with UNION.”

A related multiple-table feature that MySQL supports is the capability of deleting or updating rows in one table based on the contents of another. For example, you might want to remove rows in one table that aren’t matched by any row in another, or copy values from columns in one table to columns in another. Section 2.11, “Multiple-Table Deletes and Updates,” discusses these types of operations.

### 2.8.1 Inner Joins

If a SELECT statement names multiple tables in the FROM clause with the names separated by INNER JOIN, MySQL performs an inner join, which produces results by matching rows in one table with rows in another table. For example, if you join t1 and t2 as follows, each row in t1 is combined with each row in t2:

```sql
mysql> SELECT * FROM t1 INNER JOIN t2;
```

```
+----+----+----+----+
| i1 | c1 | i2 | c2 |
+----+----+----+----+
|  1 | a  |  2 | c  |
|  2 | b  |  2 | c  |
|  3 | c  |  2 | c  |
|  1 | a  |  3 | b  |
|  2 | b  |  3 | b  |
|  3 | c  |  3 | b  |
|  1 | a  |  4 | a  |
|  2 | b  |  4 | a  |
|  3 | c  |  4 | a  |
+----+----+----+----+
```

In this statement, SELECT * means “select every column from every table named in the FROM clause.” You could also write this as SELECT t1.*, t2.*:

```sql
SELECT t1.*, t2.* FROM t1 INNER JOIN t2;
```

If you don’t want to select all columns or you want to display them in a different left-to-right order, name each desired column, separated by commas.
A join that combines each row of each table with each row in every other table to produce all possible combinations is known as the “cartesian product.” Joining tables this way has the potential to produce a very large number of rows because the possible row count is the product of the number of rows in each table. A join between three tables that contain 100, 200, and 300 rows, respectively, could return $100 \times 200 \times 300 = 6$ million rows. That’s a lot of rows, even though the individual tables are small. In cases like this, normally a WHERE clause is useful for reducing the result set to a more manageable size.

If you add a WHERE clause causing tables to be matched on the values of certain columns, the join selects only rows with equal values in those columns:

```
mysql> SELECT t1.*, t2.* FROM t1 INNER JOIN t2 WHERE t1.i1 = t2.i2;
+----+----+----+----+
| i1 | c1 | i2 | c2 |
+----+----+----+----+
|  2 | b  |  2 | c  |
|  3 | c  |  3 | b  |
+----+----+----+----+
```

The CROSS JOIN and JOIN join types are the same as INNER JOIN, so these statements are equivalent:
```
SELECT t1.*, t2.* FROM t1 INNER JOIN t2 WHERE t1.i1 = t2.i2;
SELECT t1.*, t2.* FROM t1 CROSS JOIN t2 WHERE t1.i1 = t2.i2;
SELECT t1.*, t2.* FROM t1 JOIN t2 WHERE t1.i1 = t2.i2;
```

The ‘,’ (comma) join operator is similar as well:
```
SELECT t1.*, t2.* FROM t1, t2 WHERE t1.i1 = t2.i2;
```

However, the comma operator has a different precedence from the other join types, and it can sometimes produce syntax errors when the other types will not. I recommend that you avoid the comma operator.

INNER JOIN, CROSS JOIN, and JOIN (but not the comma operator) support alternative syntaxes for specifying how to match table columns:

- One syntax uses an ON clause rather than a WHERE clause. The following example shows this using INNER JOIN:
  ```
  SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ON t1.i1 = t2.i2;
  ```
  ON can be used regardless of whether the joined columns have the same name.

- The other syntax involves a USING() clause; this is similar in concept to ON, but the name of the joined column or columns must be the same in each table. For example, the following query joins mytbl1.b to mytbl2.b:
  ```
  SELECT mytbl1.*, mytbl2.* FROM mytbl1 INNER JOIN mytbl2 USING (b);
  ```
2.8.2 Qualifying References to Columns from Joined Tables

References to each table column throughout a SELECT statement must resolve unambiguously to a single table named in the FROM clause. If only one table is named, there is no ambiguity; all columns must be columns of that table. If multiple tables are named, any column name that appears in only one table is similarly unambiguous. However, if a column name appears in multiple tables, references to the column must be qualified with a table identifier using \texttt{tbl\_name.col\_name} syntax to specify which table you mean. Suppose that a table \texttt{mytbl1} contains columns \texttt{a} and \texttt{b}, and a table \texttt{mytbl2} contains columns \texttt{b} and \texttt{c}. References to columns \texttt{a} or \texttt{c} are unambiguous, but references to \texttt{b} must be qualified as either \texttt{mytbl1.b} or \texttt{mytbl2.b}:

\begin{verbatim}
SELECT a, mytbl1.b, mytbl2.b, c FROM mytbl1 INNER JOIN mytbl2 ...
\end{verbatim}

Sometimes a table name qualifier is not sufficient to resolve a column reference. For example, if you’re performing a self-join (that is, joining a table to itself), you’re using the table multiple times within the query and it doesn’t help to qualify a column name with the table name. In this case, table aliases are useful for communicating your intent. You can assign an alias to any instance of the table and refer to columns from that instance as \texttt{alias\_name.col\_name}. The following query joins a table to itself, but assigns an alias to one instance of the table to enable column references to be specified unambiguously:

\begin{verbatim}
SELECT mytbl.col1, m.col2 FROM mytbl INNER JOIN mytbl AS m
WHERE mytbl.col1 > m.col1;
\end{verbatim}

2.8.3 Left and Right (Outer) Joins

An inner join shows only rows where a match can be found in both tables. An outer join shows matches, too, but can also show rows in one table that have no match in the other table. Two kinds of outer joins are left and right joins. Most of the examples in this section use \texttt{LEFT JOIN}, which identifies rows in the left table that are not matched by the right table. \texttt{RIGHT JOIN} is the same except that the roles of the tables are reversed.

A \texttt{LEFT JOIN} works like this: You specify the columns to be used for matching rows in the two tables. When a row from the left table matches a row from the right table, the contents of the rows are selected as an output row. When a row in the left table has no match, it is still selected for output, but joined with a “fake” row from the right table that contains \texttt{NULL} in each column.

In other words, a \texttt{LEFT JOIN} forces the result set to contain a row for every row selected from the left table, whether or not there is a match for it in the right table. The left-table rows with no match can be identified by the fact that all columns from the right table are \texttt{NULL}. These result rows tell you which rows are missing from the right table. That is an interesting and important property, because this kind of problem comes up in many different contexts. Which customers have not been assigned an account representative? For which inventory items have no sales been recorded? Or, closer to home with our \texttt{sampdb} database: Which students have not taken a particular exam? Which students have no rows in the \texttt{absence} table (that is, which students have perfect attendance)?
Consider once again our two tables, \( t_1 \) and \( t_2 \):

|  | +----+----+ | +----+----+ |
|  | | i1 | c1 | | i2 | c2 |
|  | +----+----+ | +----+----+ |
|  | | 1 | a | | 2 | c |
|  | | 2 | b | | 3 | b |
|  | | 3 | c | | 4 | a |

If we use an inner join to match these tables on \( t_1.i1 \) and \( t_2.i2 \), we'll get output only for the values 2 and 3, because those are the values that appear in both tables:

```
mysql> SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ON t1.i1 = t2.i2;
```

```
+----+----+----+----+
<table>
<thead>
<tr>
<th>i1</th>
<th>c1</th>
<th>i2</th>
<th>c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>b</td>
<td>2</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>3</td>
<td>b</td>
</tr>
</tbody>
</table>
+----+----+----+----+
```

A left join produces output for every row in \( t_1 \), whether or not \( t_2 \) matches it. To write a left join, name the tables with \texttt{LEFT JOIN} in between rather than \texttt{INNER JOIN}:

```
mysql> SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2;
```

```
+----+----+------+------+
<table>
<thead>
<tr>
<th>i1</th>
<th>c1</th>
<th>i2</th>
<th>c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>2</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>3</td>
<td>b</td>
</tr>
</tbody>
</table>
+----+----+------+------+
```

Now there is an output row even for the \( t_1.i1 \) value of 1, which has no match in \( t_2 \). All the columns in this row that correspond to \( t_2 \) columns have a value of \texttt{NULL}.

One thing to watch out for with \texttt{LEFT JOIN} is that unless right-table columns are defined as \texttt{NOT NULL}, you may get problematic rows in the result. For example, if the right table contains columns with \texttt{NULL} values, you won't be able to distinguish those \texttt{NULL} values from \texttt{NULL} values that identify unmatched rows.

As mentioned earlier, a \texttt{RIGHT JOIN} is like a \texttt{LEFT JOIN} with the roles of the tables reversed. These two statements are equivalent:

```
SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2;
SELECT t1.*, t2.* FROM t2 RIGHT JOIN t1 ON t1.i1 = t2.i1;
```

The following discussion is phrased in terms of \texttt{LEFT JOIN}. To adjust it for \texttt{RIGHT JOIN}, reverse the table roles.
LEFT JOIN is especially useful when you want to find only those left table rows that are unmatched by the right table. Do this by adding a WHERE clause that selects only the rows that have NULL values in a right table column—in other words, the rows in one table that are missing from the other:

```sql
mysql> SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2
    \-> WHERE t2.i2 IS NULL;
```

```
+----+----+------+------+
| i1 | c1 | i2   | c2   |
+----+----+------+------+
|  1 | a  | NULL | NULL |
+----+----+------+------+
```

Normally, when you write a query like this, your real interest is in the unmatched values in the left table. The NULL columns from the right table are of no interest for display purposes, so you would omit them from the output column list:

```sql
mysql> SELECT t1.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2
    \-> WHERE t2.i2 IS NULL;
```

```
+----+----+
| i1 | c1 |
+----+----+
|  1 | a  |
+----+----+
```

Like INNER JOIN, a LEFT JOIN can be written using an ON clause or a USING() clause to specify the matching conditions. As with INNER JOIN, ON can be used whether or not the joined columns from each table have the same name, but USING() requires that they have the same names.

NATURAL LEFT JOIN is similar to LEFT JOIN; it performs a LEFT JOIN, matching all columns that have the same name in the left and right tables. (Thus, no ON or USING clause is given.)

As already mentioned, LEFT JOIN is useful for answering “Which values are missing?” questions. Let’s apply this principle to the tables in the sampdb database and consider a more complex example than those shown earlier using t1 and t2.

For the grade-keeping project, first mentioned in Chapter 1, “Getting Started with MySQL,” we have a student table listing students, a grade_event table listing the grade events that have occurred, and a score table listing scores for each student for each grade event. However, if a student was ill on the day of some quiz or test, the score table wouldn’t contain any score for the student for that event. A makeup quiz or test should be given in such cases, but how do we find these missing rows?

The problem is to determine which students have no score for a given grade event, and to do this for each grade event. That is, we want to find which combinations of student and grade event are not present in the score table. This “which values are not present” wording is a tip-off that we want a LEFT JOIN. The join isn’t as simple as in the previous examples, though:
We aren’t just looking for values that are not present in a single column, we’re looking for a two-column combination. The combinations we want are all the student/event combinations. These are produced by joining the student table to the grade_event table:

```sql
FROM student INNER JOIN grade_event
```

Then we take the result of that join and perform a LEFT JOIN with the score table to find the matches for student ID/event ID pairs:

```sql
FROM student INNER JOIN grade_event
    LEFT JOIN score ON student.student_id = score.student_id
    AND grade_event.event_id = score.event_id
```

Note that the ON clause causes the rows in the score table to be joined according to matches in different tables named earlier in the join. That’s the key for solving this problem. The LEFT JOIN forces a row to be generated for each row produced by the join of the student and grade_event tables, even when there is no corresponding score table row. The result set rows for these missing score rows can be identified by the fact that the columns from the score table will all be NULL. We can identify these rows by adding a condition in the WHERE clause. Any column from the score table will do, but because we’re looking for missing scores, it’s probably conceptually clearest to test the score column:

```sql
WHERE score.score IS NULL
```

We can also sort the results using an ORDER BY clause. The two most logical orderings are by event per student and by student per event. I’ll choose the first:

```sql
ORDER BY student.student_id, grade_event.event_id
```

Now all we need to do is name the columns we want to see in the output, and we’re done. Here is the final statement:

```sql
SELECT
    student.name, student.student_id,
    grade_event.date, grade_event.event_id, grade_event.category
FROM
    student INNER JOIN grade_event
    LEFT JOIN score ON student.student_id = score.student_id
    AND grade_event.event_id = score.event_id
WHERE
    score.score IS NULL
ORDER BY
    student.student_id, grade_event.event_id;
```

Running the query produces these results:

```
+-----------+------------+------------+----------+----------+
| name      | student_id | date       | event_id | category |
|-----------+------------+------------+----------+----------|
| Megan     | 1          | 2012-09-16 | 4        | Q        |
| Joseph    | 2          | 2012-09-03 | 1        | Q        |
```
Here’s a subtle point. The output displays the student IDs and the event IDs. The student_id column appears in both the student and score tables, so at first you might think that the output column list could name either student.student_id or score.student_id. That’s not the case, because the entire basis for being able to find the rows we’re interested in is that all the score table columns are returned by the LEFT JOIN as NULL. Selecting score.student_id would produce only a column of NULL values in the output. The same principle applies to deciding which event_id column to display. It appears in both the grade_event and score tables, but the query selects grade_event.event_id because the score.event_id values will always be NULL.

### 2.9 Performing Multiple-Table Retrievals with Subqueries

A subquery is a SELECT statement written within parentheses and nested inside another statement. Here’s an example that looks up the IDs for grade event rows that correspond to tests ('T') and uses them to select scores for those tests:

```sql
SELECT * FROM score
WHERE event_id IN (SELECT event_id FROM grade_event WHERE category = 'T');
```

Subqueries can return different types of information:

- A scalar subquery returns a single value.
- A column subquery returns a single column of one or more values.
- A row subquery returns a single row of one or more values.
- A table subquery returns a table of one or more rows of one or more columns.

Subquery results can be tested in different ways:

- Scalar subquery results can be evaluated using relative comparison operators such as = or <.
IN and NOT IN test whether a value is present in a set of values returned by a subquery.

ALL, ANY, and SOME compare a value to the set of values returned by a subquery.

EXISTS and NOT EXISTS test whether a subquery result is empty.

A scalar subquery is the most restrictive because it produces only a single value. But as a consequence, scalar subqueries can be used in the widest variety of contexts. They are applicable essentially anywhere that you can use a scalar operand, such as a term of an expression, as a function argument, or in the output column list. Column, row, and table subqueries that return more information cannot be used in contexts that require a single value.

Subqueries can be correlated or uncorrelated. This is a function of whether a subquery refers to and is dependent on values in the outer query.

You can use subqueries with statements other than SELECT. However, for statements that modify tables (DELETE, INSERT, REPLACE, UPDATE, LOAD DATA), MySQL enforces the restriction that the subquery cannot select from the table being modified.

In some cases, subqueries can be rewritten as joins. You might find subquery rewriting techniques useful to see whether the MySQL optimizer does a better job with a join than the equivalent subquery.

The following sections discuss the kinds of operations you can use to test subquery results, how to write correlated subqueries, and how to rewrite subqueries as joins.

2.9.1 Subqueries with Relative Comparison Operators

The =, <>, >, >=, <, and <= operators perform relative-value comparisons. When used with a scalar subquery, they find all rows in the outer query that stand in particular relationship to the value returned by the subquery. For example, to identify the scores for the quiz that took place on '2012-09-23', use a scalar subquery to determine the quiz event ID and then match score table rows against that ID in the outer SELECT:

```
SELECT * FROM score
WHERE event_id =
  (SELECT event_id FROM grade_event
   WHERE date = '2012-09-23' AND category = 'Q');
```

With this form of statement, where the subquery is preceded by a value and a relative comparison operator, the subquery must produce a only single value. That is, it must be a scalar subquery; if it produces multiple values, the statement will fail. In some cases, it may be appropriate to satisfy the single-value requirement by limiting the subquery result with LIMIT 1.

Use of scalar subqueries with relative comparison operators is handy for solving problems for which you’d be tempted to use an aggregate function in a WHERE clause. For example, to determine which of the presidents in the president table was born first, you might try this statement:

```
SELECT * FROM president WHERE birth = MIN(birth);
```
That doesn’t work because you can’t use aggregates in WHERE clauses. (The WHERE clause determines which rows to select, but the value of \texttt{MIN()} isn’t known until after the rows have already been selected.) However, you can use a subquery to produce the minimum birth date like this:

\begin{verbatim}
SELECT * FROM president
WHERE birth = (SELECT MIN(birth) FROM president);
\end{verbatim}

Other aggregate functions can be used to solve similar problems. The following statement uses a subquery to select the above-average scores from a given grade event:

\begin{verbatim}
SELECT * FROM score WHERE event_id = 5
AND score > (SELECT AVG(score) FROM score WHERE event_id = 5);
\end{verbatim}

If a subquery returns a single row, you can use a row constructor to compare a set of values (that is, a tuple) to the subquery result. This statement returns rows for presidents who were born in the same city and state as John Adams:

\begin{verbatim}
mysql> SELECT last_name, first_name, city, state FROM president
    -> WHERE (city, state) =
    -> (SELECT city, state FROM president
    -> WHERE last_name = 'Adams' AND first_name = 'John');
\end{verbatim}

\begin{verbatim}
+-----------+-------------+-----------+-------+
| last_name | first_name  | city      | state |
| Adams     | John        | Braintree | MA    |
| Adams     | John Quincy | Braintree | MA    |
+-----------+-------------+-----------+-------+
\end{verbatim}

You can also use \texttt{ROW(city, state)} notation, which is equivalent to \texttt{(city, state)}. Both act as row constructors.

### 2.9.2 IN and NOT IN Subqueries

The \texttt{IN} and \texttt{NOT IN} operators can be used when a subquery returns multiple rows to be evaluated in comparison to the outer query. They test whether a comparison value is present in a set of values. \texttt{IN} is true for rows in the outer query that match any row returned by the subquery. \texttt{NOT IN} is true for rows in the outer query that match no rows returned by the subquery. The following statements use \texttt{IN} and \texttt{NOT IN} to find those students who have absences listed in the absence table, and those who have perfect attendance (no absences):

\begin{verbatim}
mysql> SELECT * FROM student
    -> WHERE student_id IN (SELECT student_id FROM absence);
\end{verbatim}

\begin{verbatim}
+-------+-----+------------+
| name  | sex | student_id |
| Kyle  | M   | 3          |
| Abby  | F   | 5          |
\end{verbatim}
| name      | sex | student_id |
|-----------+-----+------------|
| Megan     | F   | 1          |
| Joseph    | M   | 2          |
| Katie     | M   | 4          |
| Nathan    | M   | 6          |
| Liesl     | F   | 7          |

IN and NOT IN also work for subqueries that return multiple columns. In other words, you can use them with table subqueries. In this case, use a row constructor to specify the comparison values to test against each column:

```sql
mysql> SELECT last_name, first_name, city, state FROM president
    -> WHERE (city, state) IN
        -> (SELECT city, state FROM president
            -> WHERE last_name = 'Roosevelt');
```

IN and NOT IN actually are synonyms for = ANY and <> ALL, which are covered in the next section.

### 2.9.3 ALL, ANY, and SOME Subqueries

The ALL and ANY operators are used in conjunction with a relative comparison operator to test the result of a column subquery. They test whether the comparison value stands in particular relationship to all or some of the values returned by the subquery. For example, <= ALL is true if the comparison value is less than or equal to every value that the subquery returns, whereas <= ANY is true if the comparison value is less than or equal to any value that the subquery returns. SOME is a synonym for ANY.

This statement determines which president was born first by selecting the row with a birth date less than or equal to all the birth dates in the president table (only the earliest date satisfies this condition):
Performing Multiple-Table Retrievals with Subqueries

mysql> SELECT last_name, first_name, birth FROM president
   -> WHERE birth <= ALL (SELECT birth FROM president);
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
| Washington | George     | 1732-02-22 |
+------------+------------+------------+

Less usefully, the following statement returns all rows because every date is less than or equal to at least one other date (itself):

mysql> SELECT last_name, first_name, birth FROM president
   -> WHERE birth <= ANY (SELECT birth FROM president);
+------------+---------------+------------+
| last_name  | first_name    | birth      |
+------------+---------------+------------+
| Washington | George        | 1732-02-22 |
| Adams      | John          | 1735-10-30 |
| Jefferson  | Thomas        | 1743-04-13 |
| Madison    | James         | 1751-03-16 |
| Monroe     | James         | 1758-04-28 |

When ALL, ANY, or SOME are used with the = comparison operator, the subquery can be a table subquery. In this case, you test return rows using a row constructor to provide the comparison values.

mysql> SELECT last_name, first_name, city, state FROM president
   -> WHERE (city, state) = ANY
      -> (SELECT city, state FROM president
         -> WHERE last_name = 'Roosevelt');
+-----------+-------------+-----------+-------+
| last_name | first_name  | city      | state |
+-----------+-------------+-----------+-------+
| Roosevelt | Theodore    | New York  | NY    |
| Roosevelt | Franklin D. | Hyde Park | NY    |
+-----------+-------------+-----------+-------+

As mentioned in the previous section, IN and NOT IN are shorthand for = ANY and <> ALL. That is, IN means “equal to any of the rows returned by the subquery” and NOT IN means “unequal to all rows returned by the subquery.”

2.9.4 EXISTS and NOT EXISTS Subqueries

The EXISTS and NOT EXISTS operators merely test whether a subquery returns any rows. If it does, EXISTS is true and NOT EXISTS is false. The following statements show some trivial examples of these subqueries. The first returns 0 if the absence table is empty, the second returns 1:
EXISTS and NOT EXISTS actually are much more commonly used in correlated subqueries. For examples, see Section 2.9.5, “Correlated Subqueries.”

With EXISTS and NOT EXISTS, the subquery uses * as the output column list. There’s no need to name columns explicitly, because the subquery is assessed as true or false based on whether it returns any rows, not based on the particular values that the rows might contain. You can actually write pretty much anything for the subquery column selection list, but if you want to make it explicit that you’re returning a true value when the subquery succeeds, you might write it as SELECT 1 rather than SELECT *

### 2.9.5 Correlated Subqueries

Subqueries can be uncorrelated or correlated:

- An uncorrelated subquery contains no references to values from the outer query, so it could be executed by itself as a separate statement. For example, the subquery in the following statement is uncorrelated because it refers only to the table t1 and not to t2:

  ```sql
  SELECT j FROM t2 WHERE j IN (SELECT i FROM t1);
  ```

- A correlated subquery does contain references to values from the outer query, and thus is dependent on it. Due to this linkage, a correlated subquery cannot be executed by itself as a separate statement. For example, the subquery in the following statement is true for each value of column j in t2 that matches a column i value in t1:

  ```sql
  SELECT j FROM t2 WHERE (SELECT i FROM t1 WHERE i = j);
  ```

Correlated subqueries commonly are used for EXISTS and NOT EXISTS subqueries, which are useful for finding rows in one table that match or don’t match rows in another. Correlated subqueries work by passing values from the outer query to the subquery to see whether they match the conditions specified in the subquery. For this reason, it’s necessary to qualify column names with table names if they are ambiguous (appear in more than one table).

The following EXISTS subquery identifies matches between the tables—that is, values that are present in both. The statement selects students who have at least one absence listed in the absence table:

```sql
SELECT student_id, name FROM student WHERE EXISTS
(SELECT * FROM absence WHERE absence.student_id = student.student_id);
```

NOT EXISTS identifies nonmatches—values in one table that are not present in the other. This statement selects students who have no absences:

```sql
SELECT student_id, name FROM student WHERE NOT EXISTS
(SELECT * FROM absence WHERE absence.student_id = student.student_id);
```
2.9.6 Subqueries in the FROM Clause

Subqueries can be used in the FROM clause to generate values. In this case, the result of the subquery acts like a table. A subquery in the FROM clause can participate in joins, its values can be tested in the WHERE clause, and so forth. With this type of subquery, you must provide a table alias to give the subquery result a name:

```
mysql> SELECT * FROM (SELECT 1, 2) AS t1 INNER JOIN (SELECT 3, 4) AS t2;
+---+---+---+---+
| 1 | 2 | 3 | 4 |
+---+---+---+---+
| 1 | 2 | 3 | 4 |
+---+---+---+---+
```

2.9.7 Rewriting Subqueries as Joins

It’s often possible to rephrase a query that uses a subquery in terms of a join, and it’s not a bad idea to examine queries that you might be inclined to write in terms of subqueries. A join is sometimes more efficient than a subquery, so if a SELECT written as a subquery takes a long time to execute, try writing it as a join to see whether it performs better. The following discussion shows how to do that.

2.9.7.1 Rewriting Subqueries That Select Matching Values

Here’s an example statement containing a subquery; it selects scores from the score table only for tests (that is, it ignores quiz scores):

```
SELECT * FROM score
WHERE event_id IN (SELECT event_id FROM grade_event WHERE category = 'T');
```

The same statement can be written without a subquery by converting it to a simple join:

```
SELECT score.* FROM score INNER JOIN grade_event
ON score.event_id = grade_event.event_id WHERE grade_event.category = 'T';
```

As another example, the following query selects scores for female students:

```
SELECT * FROM score
WHERE student_id IN (SELECT student_id FROM student WHERE sex = 'F');
```

This can be converted to a join as follows:

```
SELECT score.* FROM score INNER JOIN student
ON score.student_id = student.student_id WHERE student.sex = 'F';
```

There is a pattern here. The subquery statements follow this form:

```
SELECT * FROM table1
WHERE column1 IN (SELECT column2a FROM table2 WHERE column2b = value);
```
Such queries can be converted to a join using this form:

```sql
SELECT table1.* FROM table1 INNER JOIN table2 ON table1.column1 = table2.column2a WHERE table2.column2b = value;
```

In some cases, the subquery and the join might return different results. This occurs when `table2` contains multiple instances of `column2a`. The subquery form produces only one instance of each `column2a` value, but the join produces them all and its output includes duplicate rows. To suppress these duplicates, begin the join with `SELECT DISTINCT` rather than `SELECT`.

### 2.9.7.2 Rewriting Subqueries That Select Nonmatching (Missing) Values

Another common type of subquery statement searches for values in one table that are not present in another table. As we've seen before, the “which values are not present” type of problem is a clue that a LEFT JOIN may be helpful. Here's the statement with a subquery seen earlier that tests for students who are not listed in the `absence` table (it finds those students with perfect attendance):

```sql
SELECT * FROM student
WHERE student_id NOT IN (SELECT student_id FROM absence);
```

This query can be rewritten using a LEFT JOIN as follows:

```sql
SELECT student.*
FROM student LEFT JOIN absence ON student.student_id = absence.student_id
WHERE absence.student_id IS NULL;
```

In general terms, the subquery statement form is as follows:

```sql
SELECT * FROM table1
WHERE column1 NOT IN (SELECT column2 FROM table2);
```

A query having that form can be rewritten like this:

```sql
SELECT table1.*
FROM table1 LEFT JOIN table2 ON table1.column1 = table2.column2
WHERE table2.column2 IS NULL;
```

This assumes that `table2.column2` is defined as NOT NULL.

The subquery does have the advantage of being more intuitive than the LEFT JOIN. “Not in” is a concept that most people understand without difficulty, because it occurs outside the context of database programming. The same cannot be said for the concept of “left join,” for which there is no such basis for natural understanding.
2.10 Performing Multiple-Table Retrievals with \texttt{UNION}

To create a result set that combines the results from several queries, use a \texttt{UNION} statement. For the examples in this section, assume that you have three tables, \texttt{t1}, \texttt{t2}, and \texttt{t3}, that look like this:

\begin{verbatim}
mysql> SELECT * FROM t1;
+-----+-----+
| i   | c   |
+-----+-----+
| 1   | red |
| 2   | blue|
| 3   | green|
+-----+-----+

mysql> SELECT * FROM t2;
+-----+-----+
| j   | c   |
+-----+-----+
| -1  | tan |
| 1   | red |
+-----+-----+

mysql> SELECT * FROM t3;
+--------+-----+
| d   | k   |
+--------+-----+
| 1904-01-01 | 100 |
| 2004-01-01 | 200 |
| 2004-01-01 | 200 |
+--------+-----+
\end{verbatim}

Tables \texttt{t1} and \texttt{t2} have integer and character columns, and \texttt{t3} has date and integer columns. To write a \texttt{UNION} statement that combines multiple retrievals, write multiple \texttt{SELECT} statements and put the keyword \texttt{UNION} between them. Each \texttt{SELECT} must retrieve the same number of columns. For example, to select the integer column from each table, do this:

\begin{verbatim}
mysql> SELECT i FROM t1 UNION SELECT j FROM t2 UNION SELECT k FROM t3;
+-----+
| i   |
+-----+
| 1   |
| 2   |
| 3   |
| -1  |
| 100 |
| 200 |
+-----+
\end{verbatim}

\texttt{UNION} has the following properties.
**Column name and data types.** The column names for the `UNION` result come from the names of the columns in the first `SELECT`. The second and subsequent `SELECT` statements in the `UNION` must select the same number of columns, but corresponding columns need not have the same names or data types. (Normally, you write a `UNION` such that corresponding columns do have the same types, but MySQL performs type conversion as necessary if they do not.) Column matching occurs by position rather than by name, which is why the following two statements return different results, even though they select the same values from the two tables:

```sql
mysql> SELECT i, c FROM t1 UNION SELECT k, d FROM t3;
+------+------------+
| i    | c          |
+------+------------+
|    1 | red        |
|    2 | blue       |
|    3 | green      |
|  100 | 1904-01-01 |
|  200 | 2004-01-01 |
+------+------------+
mysql> SELECT i, c FROM t1 UNION SELECT d, k FROM t3;
+------------+-------+
| i          | c     |
+------------+-------+
| 1          | red   |
| 2          | blue  |
| 3          | green |
| 1904-01-01 | 100   |
| 2004-01-01 | 200   |
+------------+-------+
```

In each statement, the data type for each column of the result is determined from the selected values. In the first statement, strings and dates are selected for the second column. The result is a string column. In the second statement, integers and dates are selected for the first column, strings and integers for the second column. In both cases, the result is a string column.

**Duplicate-row handling.** By default, `UNION` eliminates duplicate rows from the result set:

```sql
mysql> SELECT * FROM t1 UNION SELECT * FROM t2 UNION SELECT * FROM t3;
+------------+-------+
| i          | c     |
+------------+-------+
| 1          | red   |
| 2          | blue  |
| 3          | green |
| -1         | tan   |
| 1904-01-01 | 100   |
| 2004-01-01 | 200   |
+------------+-------+
```
t1 and t2 both have a row containing values of 1 and 'red', but only one such row appears in the output. Also, t3 has two rows containing '2004-01-01' and 200, one of which has been eliminated.

**UNION DISTINCT** is synonymous with **UNION**; both retain only distinct rows.

To preserve duplicates, change each **UNION** to **UNION ALL**:

```sql
mysql> SELECT * FROM t1 UNION ALL SELECT * FROM t2 UNION ALL SELECT * FROM t3;
```

<table>
<thead>
<tr>
<th>i</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>red</td>
</tr>
<tr>
<td>2</td>
<td>blue</td>
</tr>
<tr>
<td>3</td>
<td>green</td>
</tr>
<tr>
<td>-1</td>
<td>tan</td>
</tr>
<tr>
<td>1</td>
<td>red</td>
</tr>
<tr>
<td>1904-01-01</td>
<td>100</td>
</tr>
<tr>
<td>2004-01-01</td>
<td>200</td>
</tr>
<tr>
<td>2004-01-01</td>
<td>200</td>
</tr>
</tbody>
</table>

If you mix **UNION** or **UNION DISTINCT** with **UNION ALL**, any distinct union operation takes precedence over any **UNION ALL** operations to its left.

**ORDER BY** and **LIMIT handling**. To sort a **UNION** result as a whole, place each **SELECT** within parentheses and add an **ORDER BY** clause following the last one. Because the **UNION** uses column names from the first **SELECT**, the **ORDER BY** should refer to those names, not the column names from the last **SELECT**:

```sql
mysql> (SELECT i, c FROM t1) UNION (SELECT k, d FROM t3)
    -> ORDER BY c;
```

<table>
<thead>
<tr>
<th>i</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1904-01-01</td>
</tr>
<tr>
<td>200</td>
<td>2004-01-01</td>
</tr>
<tr>
<td>2</td>
<td>blue</td>
</tr>
<tr>
<td>3</td>
<td>green</td>
</tr>
<tr>
<td>1</td>
<td>red</td>
</tr>
</tbody>
</table>

If a sort column is aliased, an **ORDER BY** at the end of the **UNION** must refer to the alias. Also, the **ORDER BY** cannot refer to table names. If you need to sort by a column specified as `tbl_name.col_name` in the first **SELECT**, alias the column and refer to the alias in the **ORDER BY** clause.

Similarly, to limit the number of rows returned by a **UNION**, add **LIMIT** to the end of the statement:
ORDER BY and LIMIT also can be used within a parenthesized individual SELECT to apply only to that SELECT:

```
mysql> (SELECT * FROM t1 ORDER BY i LIMIT 2)
    -> UNION (SELECT * FROM t2 ORDER BY j LIMIT 1)
    -> UNION (SELECT * FROM t3 ORDER BY d LIMIT 2);
```

```
+----------+------+
| i        | c    |
+----------+------+
| 1        | red  |
| 2        | blue |
+----------+------+
```

ORDER BY within an individual SELECT is used only if LIMIT is also present, to determine which rows the LIMIT applies to. It does not affect the order in which rows appear in the final UNION result.

## 2.11 Multiple-Table Deletes and Updates

Sometimes it’s useful to delete rows based on whether they match or don’t match rows in another table. Similarly, it’s often useful to update rows in one table using the contents of rows in another table. This section describes how to perform multiple-table DELETE and UPDATE operations. These types of statements draw heavily on the concepts used for joins, so be sure you’re familiar with the material discussed earlier in Section 2.8, “Performing Multiple-Table Retrievals with Joins.”

To perform a single-table DELETE or UPDATE, you refer only to the columns of one table and thus need not qualify the column names with the table name. For example, this statement deletes all rows in a table `t` that have `id` values greater than 100:

```
DELETE FROM t WHERE id > 100;
```

But what if you want to delete rows based not on properties inherent in the rows themselves, but rather on their relationship to rows in another table? Suppose that you want to delete from `t` those rows with `id` values that are present in or missing from another table `t2`?
To write a multiple-table DELETE, name all the tables in a FROM clause and specify the conditions used to match rows in the tables in the WHERE clause. The following statement deletes rows from table t1 where there is a matching id value in table t2:

```
DELETE t1 FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

Notice that if a column name appears in more than one of the tables, it is ambiguous and must be qualified with a table name.

The syntax also supports deleting rows from multiple tables at once. To delete rows from both tables where there are matching id values, name them both after the DELETE keyword:

```
DELETE t1, t2 FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

What if you want to delete nonmatching rows? A multiple-table DELETE can use any kind of join that you can write in a SELECT, so employ the same strategy that you’d use when writing a SELECT that identifies the nonmatching rows. That is, use a LEFT JOIN or RIGHT JOIN. For example, to identify rows in t1 that have no match in t2, write a SELECT like this:

```
SELECT t1.* FROM t1 LEFT JOIN t2 ON t1.id = t2.id WHERE t2.id IS NULL;
```

The analogous DELETE statement to find and remove those rows from t1 uses a LEFT JOIN as well:

```
DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id = t2.id WHERE t2.id IS NULL;
```

MySQL supports a second multiple-table DELETE syntax. This syntax uses a FROM clause to list the tables from which rows are to be deleted and a USING clause to join the tables that determine which rows to delete. The preceding multiple-table DELETE statements can be rewritten using this syntax as follows:

```
DELETE FROM t1 USING t1 INNER JOIN t2 ON t1.id = t2.id;
DELETE FROM t1, t2 USING t1 INNER JOIN t2 ON t1.id = t2.id;
DELETE FROM t1 USING t1 LEFT JOIN t2 ON t1.id = t2.id WHERE t2.id IS NULL;
```

The principles involved in writing multiple-table UPDATE statements are quite similar to those used for DELETE: Name all the tables that participate in the operation and qualify column references as necessary. Suppose that the quiz you gave on September 23, 2012 contained a question that everyone got wrong, and then you discover that the reason for this is that your answer key was incorrect. As a result, you want to add a point to everyone’s score. With a multiple-table UPDATE, you can do this as follows:

```
UPDATE score, grade_event SET score.score = score.score + 1
WHERE score.event_id = grade_event.event_id
AND grade_event.date = '2012-09-23' AND grade_event.category = 'Q';
```

In this case, you could accomplish the same objective using a single-table update and a subquery:

```
UPDATE score SET score = score + 1
WHERE event_id = (SELECT event_id FROM grade_event
WHERE date = '2012-09-23' AND category = 'Q');
```
But other updates cannot be written using subqueries. For example, you might want to not only identify rows to update based on the contents of another table, but to copy column values from one table to another. The following statement copies \texttt{t1.a} to \texttt{t2.a} for rows that have a matching \texttt{id} column value:

\begin{verbatim}
UPDATE t1, t2 SET t2.a = t1.a WHERE t2.id = t1.id;
\end{verbatim}

To perform multiple-table deletes or updates for InnoDB tables, you need not use the syntax just described. Instead, set up a foreign key relationship between tables that includes an \texttt{ON DELETE CASCADE} or \texttt{ON UPDATE CASCADE} constraint. For details, see Section 2.13, “Foreign Keys and Referential Integrity.”

### 2.12 Performing Transactions

A transaction is a set of SQL statements that execute as a unit and can be canceled if necessary. Either all the statements execute successfully, or none of them have any effect. This is achieved through the use of commit and rollback capabilities. If all of the statements in the transaction succeed, you commit it to record their effects permanently in the database. If an error occurs during the transaction, you roll it back to cancel it. Any statements executed up to that point within the transaction are undone, leaving the database in the state it was in prior to the point at which the transaction began.

Commit and rollback provide the means to ensure that halfway-done operations don’t make their way into your database and leave it in a partially updated (inconsistent) state. The canonical example involves a financial transfer where money from one account is placed into another account. Suppose that Bill writes a check to Bob for $100.00 and Bob cashes the check. Bill’s account should be decremented by $100.00 and Bob’s account incremented by the same amount:

\begin{verbatim}
UPDATE account SET balance = balance - 100 WHERE name = 'Bill';
UPDATE account SET balance = balance + 100 WHERE name = 'Bob';
\end{verbatim}

If a crash occurs between the two statements, the operation is incomplete. Depending on which statement executes first, Bill is $100 short without Bob having been credited, or Bob is given $100 without Bill having been debited. Neither outcome is correct. If transactional capabilities are not available, you must figure out the state of ongoing operations at crash time by examining your logs manually to determine how to undo them or complete them. The rollback capabilities of transaction support enable you to handle this situation properly by undoing the effect of the statements that executed before the error occurred. (You may still have to determine which transactions weren’t entered and re-issue them, but at least you don’t have to worry about half-transactions making your database inconsistent.)

Another use for transactions is to make sure that the rows involved in an operation are not modified by other clients while you’re working with them. MySQL automatically performs locking for single SQL statements to keep clients from interfering with each other, but this is not always sufficient to guarantee that a database operation achieves its intended result, because some operations are performed over the course of several statements. In this case,
Performing Transactions

Different clients might interfere with each other. A transaction groups statements into a single execution unit to prevent concurrency problems that could otherwise occur in a multiple-client environment.

Transactional systems typically are characterized as providing ACID properties. ACID is an acronym for Atomic, Consistent, Isolated, and Durable, referring to four properties transactions should have:

- **Atomicity**: The statements comprising a transaction form a logical unit. You can’t have just some of them execute.
- **Consistency**: The database is consistent before and after the transaction executes. For example, if rows in one table cannot have an ID that is not listed in another table, a transaction that attempts to insert a row with an invalid ID will fail and roll back.
- **Isolation**: One transaction has no effect on another, so that transactions executed concurrently have the same effect as if done one after the other.
- **Durability**: When a transaction executes successfully to completion, its effects are recorded permanently in the database.

Transactional processing provides stronger guarantees about the outcome of database operations, but also requires more overhead in CPU cycles, memory, and disk space. MySQL offers storage engines that are transaction-safe (such as InnoDB), and that are not transaction-safe (such as MyISAM and MEMORY). Transactional properties are essential for some applications and not for others, and you can choose which ones make the most sense for your applications. Financial operations typically need transactions, and the guarantees of data integrity outweigh the cost of additional overhead. On the other hand, for an application that logs web page accesses to a database table, a loss of a few rows if the server host crashes might be tolerable. In this case, using a nontransactional storage engine avoids the overhead required for transactional processing.

### 2.12.1 Using Transactions to Ensure Safe Statement Execution

Use of transactions requires a transactional storage engine such as InnoDB. Engines such as MyISAM and MEMORY will not work. If you’re not sure whether your MySQL server supports transactional storage engines, see Section 2.6.1.1, “Checking Which Storage Engines Are Available.”

By default, MySQL runs in autocommit mode, which means that changes made by individual statements are committed to the database immediately to make them permanent. In effect, each statement is its own transaction implicitly. To perform transactions explicitly, disable autocommit mode and then tell MySQL when to commit or roll back changes.

One way to perform a transaction is to issue a `START TRANSACTION` (or `BEGIN`) statement to suspend autocommit mode, execute the statements that make up the transaction, and end the transaction with a `COMMIT` statement to make the changes permanent. If an error occurs during the transaction, cancel it by issuing a `ROLLBACK` statement instead to undo the changes.
START TRANSACTION suspends the current autocommit mode, so after the transaction has been committed or rolled back, the mode reverts to its state prior to the START TRANSACTION. If autocommit was enabled beforehand, ending the transaction puts you back in autocommit mode. If it was disabled, ending the current transaction causes you to begin the next one.

The following example illustrates this approach. First, create a table to use:

```sql
mysql> CREATE TABLE t (name CHAR(20), UNIQUE (name)) ENGINE=InnoDB;
```

Next, initiate a transaction with START TRANSACTION, add a couple of rows to the table, commit the transaction, and then see what the table looks like:

```sql
mysql> START TRANSACTION;
mysql> INSERT INTO t SET name = 'William';
mysql> INSERT INTO t SET name = 'Wallace';
mysql> COMMIT;
```

```sql
+---------+
| name    |
+---------+
| Wallace |
| William |
+---------+
```

You can see that the rows have been recorded in the table. If you had started up a second instance of mysql and selected the contents of t after the inserts but before the commit, the rows would not show up. They would not become visible to the second mysql process until the COMMIT statement had been issued by the first one.

If an error occurs during a transaction, you can cancel it with ROLLBACK. Using the t table again, you can see this by issuing the following statements:

```sql
mysql> START TRANSACTION;
mysql> INSERT INTO t SET name = 'Gromit';
mysql> INSERT INTO t SET name = 'Wallace';
ERROR 1062 (23000): Duplicate entry 'Wallace' for key 'name'
mysql> ROLLBACK;
```

```sql
+---------+
| name    |
+---------+
| Wallace |
| William |
+---------+
```

The second INSERT attempts to place a row into the table that duplicates an existing name value, but fails because name has a UNIQUE index. After issuing the ROLLBACK, the table has only the two rows that it contained prior to the failed transaction. In particular, the successful INSERT that was performed before the failed one has been undone and its effect is not recorded in the table.
Issuing a `START TRANSACTION` statement while a transaction is in process commits the current transaction implicitly before beginning a new one.

Another way to perform transactions is to manipulate the autocommit mode directly using `SET` statements:

```
SET autocommit = 0;
SET autocommit = 1;
```

Setting the `autocommit` variable to zero disables autocommit. The effects of any statements that follow become part of the current transaction, which you end by issuing a `COMMIT` or `ROLLBACK` statement to commit or cancel it. With this method, autocommit remains off until you turn it back on, so ending one transaction also begins the next one. You can also commit a transaction by re-enabling autocommit.

To see how this approach works, begin with the same table as for the previous examples:

```
mysql> DROP TABLE t;
```

Then disable autocommit mode, insert some rows, and commit the transaction:

```
mysql> SET autocommit = 0;
mysql> INSERT INTO t SET name = 'William';
mysql> INSERT INTO t SET name = 'Wallace';
mysql> COMMIT;
```

At this point, the two rows have been committed to the table, but autocommit mode remains disabled. If you issue further statements, they become part of a new transaction, which may be committed or rolled back independently of the first transaction. To verify that autocommit is still off and that `ROLLBACK` will cancel uncommitted statements, issue the following statements:

```
mysql> INSERT INTO t SET name = 'Gromit';
mysql> INSERT INTO t SET name = 'Wallace';
ERROR 1062 (23000): Duplicate entry 'Wallace' for key 'name'
```

To verify that autocommit is still off and that `ROLLBACK` will cancel uncommitted statements, issue the following statements:

```
mysql> ROLLBACK;
```

```
mysql> SELECT * FROM t;
+---------+
<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wallace</td>
</tr>
<tr>
<td>William</td>
</tr>
</tbody>
</table>
+---------+
```
To re-enable autocommit mode, use this statement:

```sql
mysql> SET autocommit = 1;
```

As just described, a transaction ends when you issue a `COMMIT` or `ROLLBACK` statement, or when you re-enable autocommit while it is disabled. Transactions also end under other circumstances. In addition to the `SET autocommit`, `START TRANSACTION`, `BEGIN`, `COMMIT`, and `ROLLBACK` statements that affect transactions explicitly, certain other statements do so implicitly because they cannot be part of a transaction. In general, these tend to be DDL (data definition language) statements that create, alter, or drop databases or objects in them, or statements that are lock-related. For example, if you issue any of the following statements while a transaction is in progress, the server commits the transaction first before executing the statement:

- `ALTER TABLE`
- `CREATE INDEX`
- `DROP DATABASE`
- `DROP INDEX`
- `DROP TABLE`
- `LOCK TABLES`
- `RENAME TABLE`
- `SET autocommit = 1` (if not already set to 1)
- `TRUNCATE TABLE`
- `UNLOCK TABLES` (if tables currently are locked)

For a complete list of statements that cause implicit commits in your version of MySQL, see the MySQL Reference Manual.

A transaction also ends if a client’s session ends or is broken before a commit occurs. In this case, the server automatically rolls back any transaction the client had in progress.

If a client program automatically reconnects after its session with the server is lost, the connection is reset to its default state of having autocommit enabled.

Transactions are useful in all kinds of situations. Suppose that you’re working with the `score` table that is part of the grade-keeping project and you discover that the grades for two students have gotten mixed up and need to be switched. The incorrectly entered grades are as follows:

```sql
mysql> SELECT * FROM score WHERE event_id = 5 AND student_id IN (8,9);
```

```
+------------+----------+-------+
| student_id | event_id | score |
+------------+----------+-------+
|          8 |        5 |    18 |
|          9 |        5 |    13 |
+------------+----------+-------+
```

To fix this, student 8 should be given a score of 13 and student 9 a score of 18. That can be done easily with two statements:

```sql
UPDATE score SET score = 13 WHERE event_id = 5 AND student_id = 8;
UPDATE score SET score = 18 WHERE event_id = 5 AND student_id = 9;
```
However, it’s necessary to ensure that both statements succeed as a unit. This is a problem to which transactional methods may be applied. To use `START TRANSACTION`, do this:

```sql
mysql> START TRANSACTION;
mysql> UPDATE score SET score = 13 WHERE event_id = 5 AND student_id = 8;
mysql> UPDATE score SET score = 18 WHERE event_id = 5 AND student_id = 9;
mysql> COMMIT;
```

To accomplish the same thing by manipulating the autocommit mode explicitly instead, do this:

```sql
mysql> SET autocommit = 0;
mysql> UPDATE score SET score = 13 WHERE event_id = 5 AND student_id = 8;
mysql> UPDATE score SET score = 18 WHERE event_id = 5 AND student_id = 9;
mysql> COMMIT;
mysql> SET autocommit = 1;
```

Either way, the result is that the scores are swapped properly:

```sql
mysql> SELECT * FROM score WHERE event_id = 5 AND student_id IN (8,9);
+------------+----------+-------+
<table>
<thead>
<tr>
<th>student_id</th>
<th>event_id</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>18</td>
</tr>
</tbody>
</table>
+------------+----------+-------+
```

### 2.12.2 Using Transaction Savepoints

MySQL enables you to perform a partial rollback of a transaction. To do this, issue a `SAVEPOINT` statement within the transaction to set a named marker. To roll back to just that point in the transaction later, use a `ROLLBACK` statement that names the savepoint. The following statements illustrate how this works:

```sql
mysql> CREATE TABLE t (i INT) ENGINE=InnoDB;
mysql> START TRANSACTION;
mysql> INSERT INTO t VALUES(1);
mysql> SAVEPOINT my_savepoint;
mysql> INSERT INTO t VALUES(2);
mysql> ROLLBACK TO SAVEPOINT my_savepoint;
mysql> INSERT INTO t VALUES(3);
mysql> COMMIT;
```

```sql
mysql> SELECT * FROM t;
+----+
| i  |
+----+
| 1  |
| 3  |
+----+
```
After executing these statements, the first and third rows have been inserted, but the second one has been canceled by the partial rollback to the `my_savepoint` savepoint.

### 2.12.3 Transaction Isolation

Because MySQL is a multiple-user database system, different clients can attempt to use any given table at the same time. Storage engines such as MyISAM use table locking to keep clients from modifying a table at the same time, but this does not provide good concurrency performance when there are many updates. The InnoDB storage engine takes a different approach. It uses row-level locking for finer-grained control over table access by clients. One client can modify a row at the same time that another client reads or modifies a different row in the same table. If both clients want to modify a row at the same time, whichever of them acquires a lock on the row gets to modify it first. This provides better concurrency than table locking. However, there is the question of whether one client's transaction should be able to see the changes made by another client's transaction.

InnoDB implements transaction isolation levels to give clients control over what kind of changes made by other transactions they want to see. Different isolation levels permit or prevent problems that can occur when different transactions run simultaneously:

- **Dirty reads.** A dirty read occurs when a change made by one transaction can be seen by other transactions before the transaction has been committed. Another transaction thus might think the row has been changed, even though that will not really be true if the transaction that changed the row later is rolled back.

- **Nonrepeatable reads.** A nonrepeatable read refers to failure by a transaction to get the same result for a given `SELECT` statement each time it executes it. This might happen if one transaction performs a `SELECT` twice but another transaction changes some of the rows in between the two executions.

- **Phantom rows.** A phantom is a row that becomes visible to a transaction when it was not previously. Suppose that a transaction performs a `SELECT` and then another transaction inserts a row. If the first transaction runs the same `SELECT` again and sees the new row, that is a phantom.

To deal with these problems, InnoDB supports four transaction isolation levels. These levels determine which modifications made by one transaction can be seen by other transactions that execute at the same time:

- **READ UNCOMMITTED:** A transaction can see row modifications made by other transactions even before they have been committed.

- **READ COMMITTED:** A transaction can see row modifications made by other transactions only if they have been committed.

- **REPEATABLE READ:** If a transaction performs a given `SELECT` twice, the result is repeatable. That is, it gets the same result each time, even if other transactions have changed or inserted rows in the meantime.
- **SERIALIZABLE**: This isolation level is similar to **REPEATABLE READ** but isolates transactions more completely: Rows examined by one transaction cannot be modified by other transactions until the first transaction completes. This enables one transaction to read rows and at the same time prevent them from being modified by other transactions until it is done with them.

Table 2.4 shows for each isolation level whether it permits dirty reads, nonrepeatable reads, or phantom rows. The table is InnoDB-specific in that **REPEATABLE READ** does not permit phantom rows to occur. Some database systems do permit phantoms at the **REPEATABLE READ** isolation level.

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Dirty Reads</th>
<th>Nonrepeatable Reads</th>
<th>Phantom Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ UNCOMMITTED</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>READ COMMITTED</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>REPEATABLE READ</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SERIALIZABLE</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

The default InnoDB isolation level is **REPEATABLE READ**. This can be changed at server startup with the **--transaction-isolation** option, or at runtime with the **SET TRANSACTION** statement. The statement has three forms:

```sql
SET GLOBAL TRANSACTION ISOLATION LEVEL level;
SET SESSION TRANSACTION ISOLATION LEVEL level;
SET TRANSACTION ISOLATION LEVEL level;
```

A client that has the **SUPER** privilege can use **SET TRANSACTION** to change the global isolation level, which then applies to any clients that connect thereafter. In addition, any client can change its own transaction isolation level, either for all subsequent transactions within its session with the server (if **SESSION** is specified) or for its next transaction only (if **SESSION** is omitted). No special privileges are required for the client-specific levels.

### Can You Mix Transactional and Nontransactional Tables?

It is possible to use both transactional and nontransactional tables during the course of a transaction, but the result might not be what you expect. Statements for nontransactional tables always take effect immediately, even when **autocommit** is disabled. In effect, nontransactional tables are always in autocommit mode and each statement commits immediately. As a result, if you change a nontransactional table within a transaction and then attempt a rollback, the nontransactional table changes cannot be undone.
2.13 Foreign Keys and Referential Integrity

A foreign key relationship enables you to declare that an index in one table is related to an index in another. It also enables you to place constraints on what may be done to the tables in the relationship. The database enforces the rules of this relationship to maintain referential integrity. For example, the `score` table in the `sampdb` sample database contains a `student_id` column, which we use to relate score rows to students in the `student` table. When we created these tables in Chapter 1, “Getting Started with MySQL,” we set up some explicit relationships between them. For example, we declared `score.student_id` to be a foreign key for the `student.student_id` column. That prevents a row from being entered into the `score` table unless its `student_id` value exists in the `student` table. In other words, the foreign key prevents entry of scores for nonexistent students.

Foreign keys are not useful just for row entry, but for deletes and updates as well. For example, we could set up a constraint such that if a student is deleted from the `student` table, all corresponding rows for the student in the `score` table are deleted automatically as well. This is called “cascaded delete” because the effect of the delete cascades from one table to another. Cascaded update is possible as well. For example, with cascaded update, changing a student’s `student_id` value in the `student` table also changes the value in the student’s corresponding `score` table rows.

Foreign keys maintain the consistency of your data, and they provide a certain measure of convenience. Without foreign keys, you are responsible for keeping track of inter-table dependencies and maintaining their consistency from within your applications. In some cases, doing this might not be much more work than issuing a few extra `DELETE` statements to make sure that when you delete a row from one table, you also delete the corresponding rows in any related tables. But it is extra work, and if the database engine will perform consistency checks for you, why not let it? Automatic checking capability is especially useful if your tables have particularly complex relationships. You likely will not want to be responsible for implementing these dependencies in your applications.

In MySQL, the InnoDB storage engine provides foreign key support. This section describes how to set up InnoDB tables to define foreign keys, and how foreign keys affect the way you use tables. First, it’s necessary to define some terms:

- The parent is the table that contains the original key values.
- The child is the related table that refers to key values in the parent.

Parent table key values are used to associate the two tables. Specifically, an index in the child table refers to an index in the parent. The child index values must match those in the parent or else be set to `NULL` to indicate that there is no associated parent table row. The index in the child table is known as the “foreign key”—that is, the key that is foreign (external) to the parent table but contains values that point to the parent. A foreign key relationship can be set up to reject `NULL` values, in which case all foreign key values must match a value in the parent table.

InnoDB enforces these rules to guarantee that the foreign key relationship stays intact with no mismatches. This is called “referential integrity.”
The following syntax shows how to define a foreign key in a child table:

```sql
[CONSTRAINT constraint_name]
FOREIGN KEY [fk_name] (index_columns)
    REFERENCES tbl_name (index_columns)
    [ON DELETE action]
    [ON UPDATE action]
    [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
```

Although all parts of this syntax are parsed, InnoDB does not implement the semantics for all the clauses: The `MATCH` clause is not supported and is ignored if you specify it. Also, some `action` values are recognized but have no effect. (For storage engines other than InnoDB, the entire `FOREIGN KEY` definition is parsed but ignored.)

InnoDB pays attention to the following parts of the definition:

- The `CONSTRAINT` clause, if given, supplies a name for the foreign key constraint. If you omit it, InnoDB creates a name.
- `FOREIGN KEY` indicates the indexed columns in the child table that must match index values in the parent table. `fk_name` is the foreign key ID. If given, it is ignored unless InnoDB automatically creates an index for the foreign key; in that case, `fk_name` becomes the index name.
- `REFERENCES` names the parent table and the index columns in that table to which the foreign key in the child table refers. The `index_columns` part of the `REFERENCES` clause must have the same number of columns as the `index_columns` that follows the `FOREIGN KEY` keywords.
- `ON DELETE` enables you to specify what happens to the child table when parent table rows are deleted. The default if no `ON DELETE` clause is present is to reject any attempt to delete rows in the parent table that have child rows pointing to them. To specify an `action` value explicitly, use one of the following clauses:
  - `ON DELETE NO ACTION` and `ON DELETE RESTRICT` are the same as omitting the `ON DELETE` clause. Some database systems have deferred checks, and `NO ACTION` is a deferred check. For InnoDB, foreign key constraints are checked immediately, so `NO ACTION` and `RESTRICT` are the same.
  - `ON DELETE CASCADE` causes matching child rows to be deleted when the corresponding parent row is deleted. In essence, the effect of the delete is cascaded from the parent to the child. This enables you to perform multiple-table deletes by deleting rows only from the parent table and letting InnoDB delete the corresponding rows from the child table.
  - `ON DELETE SET NULL` causes index columns in matching child rows to be set to `NULL` when the parent row is deleted. If you use this option, all the indexed child table columns named in the foreign key definition must be defined to permit `NULL` values. (One implication of using this action is that you cannot define the foreign key to be a `PRIMARY KEY`; primary keys do not permit `NULL` values.)
  - `ON DELETE SET DEFAULT` is recognized but unimplemented and InnoDB issues an error.
ON UPDATE enables you to specify what happens to the child table when parent table rows are updated. The default if no ON UPDATE clause is present is to reject any inserts or updates in the child table that result in foreign key values that don’t have any match in the parent table index, and to prevent updates to parent table index values to which child rows point. The possible action values are the same as for ON DELETE and have similar effects.

To set up a foreign key relationship, follow these guidelines:

- The child table must have an index where the foreign key columns are listed as its first columns. The parent table must also have an index in which the columns in the REFERENCES clause are listed as its first columns. (In other words, the key columns must be indexed in the tables on both ends of the foreign key relationship.) You must create the parent table index explicitly before defining the foreign key relationship. InnoDB automatically creates an index on foreign key columns (the referencing columns) in the child table if the CREATE TABLE statement does not include such an index. This makes it easier to write the CREATE TABLE statement in some cases. However, an automatically created index will be a nonunique index and will include only the foreign key columns. You should define the index in the child table explicitly if you want it to be a PRIMARY KEY or UNIQUE index, or if it should include other columns in addition to those in the foreign key.

- Corresponding columns in the parent and child indexes must have compatible types. For example, you cannot match an INT column with a CHAR column. Corresponding character columns must be the same length. Corresponding integer columns must have the same size and must both be signed or both UNSIGNED.

- You cannot index prefixes of string columns in foreign key relationships. (That is, for string columns, you must index the entire column, not just a leading prefix of it.)

In Chapter 1, “Getting Started with MySQL,” we created tables for the grade-keeping project that have simple foreign key relationships. Now let’s work through an example that is more complex. Begin by creating tables named parent and child, such that the child table contains a foreign key that references the par_id column in the parent table:

```
CREATE TABLE parent
(
  par_id    INT NOT NULL,
  PRIMARY KEY (par_id)
) ENGINE = INNODB;

CREATE TABLE child
(
  par_id    INT NOT NULL,
  child_id  INT NOT NULL,
  PRIMARY KEY (par_id, child_id),
  FOREIGN KEY (par_id) REFERENCES parent (par_id)
```
ON DELETE CASCADE
ON UPDATE CASCADE
} ENGINE = INNODB;

The foreign key in this case uses ON DELETE CASCADE to specify that when a row is deleted from the parent table, MySQL also should remove child rows with a matching par_id value automatically. ON UPDATE CASCADE indicates that if a parent row par_id value is changed, MySQL also should change any matching par_id values in the child table to the new value.

Now insert a few rows into the parent table, and then add some rows to the child table that have related key values:

```sql
mysql> INSERT INTO parent (par_id) VALUES(1),(2),(3);
mysql> INSERT INTO child (par_id,child_id) VALUES(1,1),(1,2);
mysql> INSERT INTO child (par_id,child_id) VALUES(2,1),(2,2),(2,3);
mysql> INSERT INTO child (par_id,child_id) VALUES(3,1);
```

These statements result in the following table contents, where each par_id value in the child table matches a par_id value in the parent table:

```sql
mysql> SELECT * FROM parent;
+--------+
| par_id |
+--------+
|      1 |
|      2 |
|      3 |
mysql> SELECT * FROM child;
+--------+----------+
| par_id | child_id |
+--------+----------+
|      1 |        1 |
|      1 |        2 |
|      2 |        1 |
|      2 |        2 |
|      2 |        3 |
|      3 |        1 |
```

To verify that InnoDB enforces the key relationship for insertion, try adding a row to the child table that has a par_id value not found in the parent table:

```sql
mysql> INSERT INTO child (par_id,child_id) VALUES(4,1);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`sampdb`.`child`, CONSTRAINT `child_ibfk_1` FOREIGN KEY (`par_id`) REFERENCES `parent` (`par_id`) ON DELETE CASCADE ON UPDATE CASCADE)
To test cascaded delete, see what happens when you delete a parent row:

```
mysql> DELETE FROM parent WHERE par_id = 1;
```

MySQL deletes the row from the `parent` table:

```
mysql> SELECT * FROM parent;
```

```
+--------+
| par_id |
+--------+
| 2      |
| 3      |
```

In addition, it cascades the effect of the `DELETE` statement to the `child` table:

```
mysql> SELECT * FROM child;
```

```
+--------+----------+
| par_id | child_id |
+--------+----------+
| 2      | 1        |
| 2      | 2        |
| 2      | 3        |
| 3      | 1        |
```

To test cascaded update, see what happens when you update a `parent` row:

```
mysql> UPDATE parent SET par_id = 100 WHERE par_id = 2;
```

```
mysql> SELECT * FROM parent;
```

```
+--------+
| par_id |
+--------+
| 3      |
| 100    |
```

```
mysql> SELECT * FROM child;
```

```
+--------+----------+
| par_id | child_id |
+--------+----------+
| 3      | 1        |
| 100    | 1        |
| 100    | 2        |
| 100    | 3        |
```

The preceding example shows how to arrange for deletes or updates of a `parent` row to cause cascaded deletes or updates of any corresponding `child` rows. The `ON DELETE` and `ON UPDATE` clauses permit other actions. For example, one possibility is to let the `child` rows remain in the
Foreign Keys and Referential Integrity

To do this, it's necessary to make several changes to the definition of the child table:

- **Use ON DELETE SET NULL rather than ON DELETE CASCADE.** This tells InnoDB to set the foreign key column (par_id) to NULL instead of deleting the rows.
- **Use ON UPDATE SET NULL rather than ON UPDATE CASCADE.** This tells InnoDB to set the foreign key column (par_id) to NULL when matching parent rows are updated.
- The original definition of child defines par_id as NOT NULL. That won't work with ON DELETE SET NULL or ON UPDATE SET NULL, so the column definition must be changed to permit NULL.
- The original definition of child also defines par_id to be part of a PRIMARY KEY. However, a PRIMARY KEY cannot contain NULL values. Changing par_id to permit NULL therefore also requires that the PRIMARY KEY be changed to a UNIQUE index. UNIQUE indexes enforce uniqueness except for NULL values, which can occur multiple times in the index.

To see the effect of these changes, re-create the parent table using the original definition and load the same initial rows into it. Then create the child table using the new definition shown here:

```sql
CREATE TABLE child
(
  par_id    INT NULL,
  child_id  INT NOT NULL,
  UNIQUE (par_id, child_id),
  FOREIGN KEY (par_id) REFERENCES parent (par_id)
    ON DELETE SET NULL
    ON UPDATE SET NULL
) ENGINE = INNODB;
```

With respect to inserting new rows, the child table behaves similarly to the original definition. That is, it permits insertion of rows with par_id values found in the parent table, but prohibits entry of values that aren’t listed there:

```
mysql> INSERT INTO child (par_id,child_id) VALUES(1,1),(1,2);
mysql> INSERT INTO child (par_id,child_id) VALUES(2,1),(2,2),(2,3);
mysql> INSERT INTO child (par_id,child_id) VALUES(3,1);
mysql> INSERT INTO child (par_id,child_id) VALUES(4,1);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`sampdb`.`child`, CONSTRAINT `child_ibfk_1` FOREIGN KEY (`par_id`) REFERENCES `parent` (`par_id`) ON DELETE SET NULL ON UPDATE SET NULL)
```

There is one difference with respect to inserting rows. Because the par_id column now is defined as NULL, you can explicitly insert rows into the child table that contain NULL and no error occurs. A difference in behavior also occurs when you delete a parent row. Try removing a parent row and then check the contents of the child table to see what happens:
In this case, the child rows that had 1 in the par_id column are not deleted. Instead, the par_id column is set to NULL, as specified by the ON DELETE SET NULL constraint.

Updating a parent row has a similar effect:

mysql> UPDATE parent SET par_id = 100 WHERE par_id = 2;
mysql> SELECT * FROM child;

To see what foreign key relationships an InnoDB table has, use the SHOW CREATE TABLE statement.

If an error occurs when you attempt to create a table that has a foreign key, use the SHOW ENGINE INNODB STATUS statement to get the full error message.

### 2.14 Using FULLTEXT Searches

MySQL is capable of performing full-text searches, which enables you to look for words or phrases without using pattern-matching operations. There are three kinds of full-text search:

- Natural language searching (the default). MySQL parses the search string into words and searches for rows containing these words.
- Boolean mode searching. Words in the search string can include modifier characters that indicate specific requirements, such as that a given word should be present or absent in matching rows, or that rows must contain an exact phrase.
Query expansion searching. This kind of search occurs in two phases. The first phase is a natural language search. Then a second search is done using the original search string concatenated with the most highly relevant matching rows from the first search. This expands the search on the basis of the assumption that words related to the original search string will match relevant rows that the original string did not.

Full-text search capability is enabled for a given table by creating a special kind of index and has the following characteristics:

- Full-text searches are based on `FULLTEXT` indexes. In MySQL 5.5, these can be created only for MyISAM tables. MySQL 5.6 introduces full-text support for InnoDB, but we'll stick with MyISAM here because you might not have 5.6. Only `CHAR`, `VARCHAR`, and `TEXT` columns can be included in a `FULLTEXT` index.

- Common words are ignored for `FULLTEXT` searches, where “common” means “present in at least half the rows.” It’s especially important to remember this when you’re setting up a test table to experiment with the `FULLTEXT` capability. Be sure to insert at least three rows into your test table. If the table has just one or two rows, every word in it will occur at least 50% of the time and you’ll never get any results!

- There is a built-in list of common words such as “the,” “after,” and “other” that are called “stopwords” and that are always ignored.

- Words that are too short are ignored. By default, “too short” is defined as fewer than four characters, but you can reconfigure the server to set the minimum length to a different value. (See Section 2.14.4, “Configuring the `FULLTEXT` Search Engine”.)

- Words are defined as sequences of characters that include letters, digits, apostrophes, and underscores. This means that a string like “full-blooded” is considered to contain two words, “full” and “blooded.” Normally, a full-text search matches whole words, not partial words, and the `FULLTEXT` engine considers a row to match a search string if it includes any of the words in the search string. If you use a boolean full-text search, you can impose the additional constraint that all the words must be present (either in any order, or, to perform a phrase search, in exactly the order listed in the search string). With a boolean search, it’s also possible to match rows that do not include certain words, or to add a wildcard modifier to match all words that begin with a given prefix.

- A `FULLTEXT` index can be created for a single column or multiple columns. If it spans multiple columns, searches based on the index look through all the columns simultaneously. The flip side of this is that when you perform a search, you must specify a column list that corresponds exactly to the set of columns that matches some `FULLTEXT` index. For example, if you want to search `col1` sometimes, `col2` sometimes, and both `col1` and `col2` sometimes, you must create three indexes: one for each of the columns separately, and one that includes both columns.

The following examples show how to use full-text searching by creating `FULLTEXT` indexes and then performing queries on them using the `MATCH` operator. A script to create the table and some sample data to load into it are available in the `full-text` directory of the `sampdb` distribution.
Create a `FULLTEXT` index much the same way as other indexes: Define it with `CREATE TABLE` when creating the table initially, or add it afterward with `ALTER TABLE` or `CREATE INDEX`. Because `FULLTEXT` indexes require you to use MyISAM tables, you can take advantage of one of the properties of the MyISAM storage engine if you’re creating a new table to use for `FULLTEXT` searches: Table loading proceeds more quickly if you populate the table first and then add the indexes afterward, rather than loading data into an already indexed table. Suppose that you have a data file named `apothegm.txt` containing famous sayings and the people to whom they’re attributed:

<table>
<thead>
<tr>
<th>Attribution</th>
<th>Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aeschylus</td>
<td>Time as he grows old teaches many lessons</td>
</tr>
<tr>
<td>Alexander Graham Bell</td>
<td>Mr. Watson, come here. I want you!</td>
</tr>
<tr>
<td>Benjamin Franklin</td>
<td>It is hard for an empty bag to stand upright</td>
</tr>
<tr>
<td>Benjamin Franklin</td>
<td>Little strokes fell great oaks</td>
</tr>
<tr>
<td>Benjamin Franklin</td>
<td>Remember that time is money</td>
</tr>
<tr>
<td>Miguel de Cervantes</td>
<td>Bell, book, and candle</td>
</tr>
<tr>
<td>Proverbs 15:1</td>
<td>A soft answer turneth away wrath</td>
</tr>
<tr>
<td>Theodore Roosevelt</td>
<td>Speak softly and carry a big stick</td>
</tr>
<tr>
<td>William Shakespeare</td>
<td>But, soft! what light through yonder window breaks?</td>
</tr>
<tr>
<td>Robert Burton</td>
<td>I light my candle from their torches.</td>
</tr>
</tbody>
</table>

If you want to search by phrase and attribution separately or together, you need to index each column separately, and also create an index that includes both columns. You can create, populate, and index a table named `apothegm` as follows:

```
CREATE TABLE apothegm (attribution VARCHAR(40), phrase TEXT) ENGINE=MyISAM;
LOAD DATA LOCAL INFILE 'apothegm.txt' INTO TABLE apothegm;
ALTER TABLE apothegm
    ADD FULLTEXT (phrase),
    ADD FULLTEXT (attribution),
    ADD FULLTEXT (phrase, attribution);
```

### 2.14.1 Natural Language `FULLTEXT` Searches

After setting up the table, perform natural language full-text searches on it using `MATCH` to name the column or columns to search and `AGAINST()` to specify the search string. For example:

```
mysql> SELECT * FROM apothegm WHERE MATCH(attribution) AGAINST('roosevelt');
+-------------------+-------------------------------------------+
| attribution        | phrase                                    |
+-------------------+-------------------------------------------+
| Theodore Roosevelt | Speak softly and carry a big stick        |
```

```
mysql> SELECT * FROM apothegm WHERE MATCH(phrase) AGAINST('time');
+-------------------------------------------------------------+
| attribution        | phrase                                           |
+-------------------------------------------------------------+
In the last example, note how the query finds rows that contain the search word in different columns, which demonstrates the `FULLTEXT` capability of searching multiple columns at once. Also note that the order of the columns as named in the query is `attribute`, `phrase`. That differs from the order in which they were named when the index was created (``attribute`, `phrase``), which illustrates that order does not matter. What matters is that there must be some `FULLTEXT` index that consists of exactly the columns named.

To see only how many rows a search matches, use `COUNT(*)`:

```sql
SELECT * FROM apothegm WHERE MATCH (attribute, phrase) -> AGAINST('bell');
```

<table>
<thead>
<tr>
<th>attribution</th>
<th>phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexander Graham Bell</td>
<td>Mr. Watson, come here. I want you!</td>
</tr>
<tr>
<td>Miguel de Cervantes</td>
<td>Bell, book, and candle</td>
</tr>
</tbody>
</table>

Output rows for natural language `FULLTEXT` searches are ordered by decreasing relevance when you use a `MATCH` expression in the `WHERE` clause. Relevance values are nonnegative floating point values, with zero indicating “no relevance.” To see these values, use a `MATCH` expression in the output column list:

```sql
SELECT phrase, MATCH (phrase) AGAINST('time') AS relevance
-> FROM apothegm;
```

<table>
<thead>
<tr>
<th>phrase</th>
<th>relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time as he grows old teaches many lessons</td>
<td>1.3253291845321655</td>
</tr>
<tr>
<td>Mr. Watson, come here. I want you!</td>
<td>0</td>
</tr>
<tr>
<td>It is hard for an empty bag to stand upright</td>
<td>0</td>
</tr>
<tr>
<td>Little strokes fell great oaks</td>
<td>0</td>
</tr>
<tr>
<td>Remember that time is money</td>
<td>1.340062141418457</td>
</tr>
<tr>
<td>Bell, book, and candle</td>
<td>0</td>
</tr>
<tr>
<td>A soft answer turneth away wrath</td>
<td>0</td>
</tr>
<tr>
<td>Speak softly and carry a big stick</td>
<td>0</td>
</tr>
<tr>
<td>But, soft! what light through yonder window breaks?</td>
<td>0</td>
</tr>
<tr>
<td>I light my candle from their torches.</td>
<td>0</td>
</tr>
</tbody>
</table>
A natural language search finds rows that contain any of the search words, so a query such as the following returns rows that contain either “hard” or “soft”:

```sql
mysql> SELECT * FROM apothegm WHERE MATCH(phrase) -> AGAINST('hard soft');
```

| attribution         | phrase                                              |
|---------------------+-----------------------------------------------------|
| Benjamin Franklin   | It is hard for an empty bag to stand upright        |
| Proverbs 15:1       | A soft answer turneth away wrath                    |
| William Shakespeare | But, soft! what light through yonder window breaks? |

Natural language mode is the default full-text search mode. To specify this mode explicitly, add `IN NATURAL LANGUAGE MODE` after the search string. The following statement performs the same search as the preceding example:

```sql
SELECT * FROM apothegm WHERE MATCH(phrase) AGAINST('hard soft' IN NATURAL LANGUAGE MODE);
```

### 2.14.2 Boolean Mode **FULLTEXT** Searches

Greater control over multiple-word matching can be obtained by using boolean mode **FULLTEXT** searches. This type of search is performed by adding `IN BOOLEAN MODE` after the search string in the `AGAINST()` function. Boolean searches have the following characteristics:

- The 50% rule is ignored. Searches find words even if they occur in more than half of the rows.
- No sorting by relevance occurs.
- A search can require all words in a phrase to be present in a particular order. To match a phrase, specify it within double quotes. Matches occur for rows that contain the same words together in the same order as listed in the phrase:

```sql
mysql> SELECT * FROM apothegm
    -> WHERE MATCH(attribution, phrase)
    -> AGAINST('"bell book and candle"' IN BOOLEAN MODE);
```

| attribution         | phrase                        |
|---------------------+-------------------------------|
| Miguel de Cervantes | Bell, book, and candle        |

- It’s possible to perform a boolean mode full-text search on columns that are not part of a **FULLTEXT** index, although this is much slower than using indexed columns.

For boolean searches, modifiers may be applied to words in the search string. A leading plus or minus sign requires a word to be present or not present in matching rows. For example, a search string of `"bell"` matches rows that contain “bell,” but a search string of `"+bell`
-candle' in boolean mode matches only rows that contain “bell” and do not contain “candle.”

```sql
mysql> SELECT * FROM apothegm
    -> WHERE MATCH(attribution, phrase)
    -> AGAINST('bell');
```

| attribution           | phrase                             |
|-----------------------+------------------------------------|
| Alexander Graham Bell | Mr. Watson, come here. I want you! |
| Miguel de Cervantes   | Bell, book, and candle             |

```sql
mysql> SELECT * FROM apothegm
    -> WHERE MATCH(attribution, phrase)
    -> AGAINST('+bell -candle' IN BOOLEAN MODE);
```

| attribution           | phrase                             |
|-----------------------+------------------------------------|
| Alexander Graham Bell | Mr. Watson, come here. I want you! |

A trailing asterisk acts as a wildcard so that any row containing words beginning with the search word match. For example ‘soft*’ matches “soft,” “softly,” “softness,” and so forth:

```sql
mysql> SELECT * FROM apothegm WHERE MATCH(phrase)
    -> AGAINST('soft*' IN BOOLEAN MODE);
```

| attribution         | phrase                                             |
|---------------------+----------------------------------------------------|
| Proverbs 15:1       | A soft answer turneth away wrath                   |
| William Shakespeare | But, soft! what light through yonder window breaks?|
| Theodore Roosevelt  | Speak softly and carry a big stick                 |

However, the wildcard feature cannot be used to match words shorter than the minimum index word length.


Stopwords are ignored just as for natural language searches, even if marked as required. A search for '+Alexander +the +great' finds rows containing “Alexander” and “great,” but ignores “the” as a stopword.

### 2.14.3 Query Expansion FULLTEXT Searches

A full-text search with query expansion performs a two-phase search. The initial search is like a regular natural language search. Then the most highly relevant rows from this search are used for the second phase. The words in these rows are used along with the original search terms to
perform a second search. Because the set of search terms is larger, the result generally includes rows that are not found in the first phase but are related to them.

To perform this kind of search, add `WITH QUERY EXPANSION` following the search terms. The following example provides an illustration. The first query shows a natural language search. The second query shows a query expansion search. Its result includes an extra row that contains none of the original search terms. This row is found because it contains the word “candle” that is present in one of the rows found by the natural language search.

```
mysql> SELECT * FROM apothegm
    ...> WHERE MATCH(attribution, phrase)
    ...> AGAINST('bell book');
+-----------------------+------------------------------------+
| attribution           | phrase                             |
|-----------------------+------------------------------------+
| Miguel de Cervantes   | Bell, book, and candle             |
| Alexander Graham Bell | Mr. Watson, come here. I want you! |
+-----------------------+------------------------------------+
mysql> SELECT * FROM apothegm
    ...> WHERE MATCH(attribution, phrase)
    ...> AGAINST('bell book' WITH QUERY EXPANSION);
+-----------------------+---------------------------------------+
| attribution           | phrase                                |
|-----------------------+---------------------------------------+
| Miguel de Cervantes   | Bell, book, and candle                |
| Alexander Graham Bell | Mr. Watson, come here. I want you!    |
| Robert Burton         | I light my candle from their torches. |
+-----------------------+---------------------------------------+
```

### 2.14.4 Configuring the **FULLTEXT** Search Engine

Several full-text parameters are configurable and can be modified by setting system variables. The `ft_min_word_len` and `ft_max_word_len` variables determine the shortest and longest words to index in **FULLTEXT** indexes. Words with lengths outside the range defined by these two variables are ignored when **FULLTEXT** indexes are built. The default minimum and maximum values are 4 and 84.

Suppose that you want to change the minimum word length from 4 to 3. Do so like this:

1. Start the server with the `ft_min_word_len` variable set to 3. To ensure that this happens whenever the server starts, it’s best to place the setting in an option file such as `/etc/my.cnf`:

   ```
   [mysqld]
   ft_min_word_len=3
   ```
2. For any existing tables that already have FULLTEXT indexes, you must rebuild those indexes. You can drop and add the indexes, but it's easier and sufficient to perform a quick repair operation:

   REPAIR TABLE tbl_name QUICK;

3. Any new FULLTEXT indexes that you create after changing the parameter will use the new value automatically.

   For more information on setting system variables, see Appendix D, “System, Status, and User Variable Reference.” For details on using option files, see Appendix F, “MySQL Program Reference.”

   **Note**

   If you use myisamchk to rebuild indexes for a table that contains any FULLTEXT indexes, see the FULLTEXT-related notes in the myisamchk description in Appendix F, “MySQL Program Reference.”
Index

Symbols

+ (addition operator), 58, 241, 766
& (AND) operator, 242, 773
&& (AND) operator, 241, 773
\ (backslashes), strings, 184
, (comma) join operator, 138
/ (division operator), 57, 241, 767
$ (dollar signs), PHP, 492
... (ellipsis), operators/functions, 764
= (equal to operator), 57, 243, 769
<= (less than or equal to operator), 57, 243, 770
>= (greater than or equal to operator), 57, 243, 770
< (less than operator), 57, 243, 770
% (modulo operator), 57, 241, 767
* (multiplication operator), 57, 241, 767
~ (negation operator), 774
! (NOT) operator, 241, 772
!= (not equal operator), 57, 243, 770
< > (not equal to operator), 57, 243
<=> (null-safe equality operator), 770
| (OR operator), 242, 773
|| (OR operator), 241-242, 773
( ) (parentheses), 401
; (semicolons), 27, 266
#! (shebang), 396
<< (shift left operator), 242, 773
>> (shift right operator), 242, 773
[ ] (square brackets), operators/functions, 764
- (subtraction operator), 57, 241, 767
!= (unequal) operator, 770
|| (vertical bars), operators/functions, 764
% (wildcard character), 244
_ (wildcard character), 244
A

aborted_clients status variable, 881
aborted_connects status variable, 881
ABS() function, 784
absence table, 45, 49
access control, 540
  administrative-only, setting, 649-651
directory exception, 651
directories outside base directory, 651
Innodb directory, 651
servers, running, 652
symlinks, 651
authentication plugins, 676-679
  proxy users, creating, 677-679
server connections, 677
server side/client side, 677
specifying, 676
base directory insecurities, checking, 648
clients, 686-687
column information structures, 364
CREATE USER statements
  account operations, 655
  selecting, 656
data directory, 546-547, 648
DROP USER statement, 656
external risks, 646
grant tables
  administrative privilege columns, 680
  authentication columns, 680, 684
  listing of, 680
  object privileges, 681-682
  privilege columns, 683-684
  privilege tables, 683
  resource management columns, 680, 685-686
  scope-of-access columns, 680-681, 683
  SSL-related columns, 680, 685
  user table authentication, 680
internal risks, 645
metadata, 130
  command line, 135
  INFORMATION_SCHEMA database, 132-135
multiple-user benefit, 13
option files, 653-654
overview, 646-647

privileges
  account administering, enabling, 669-670
  administrative, 661-663, 666
  ALL specifier, 661, 666
  combining, 665
database-level, 666
displaying, 671
global, 666
granting, 660-661
level-specifiers, 665
no privileges, 668
object, 663-665
ON specifier, 665
PROXY, 667
quoting, 667
revoking, 671-672
secure connections, requiring, 668-669
stored routines, 667
table/column level, 667
USAGE specifier, 661
remote, 13
RENAMe USER statement, 656
resource consumption limits, 670-671
risks, 673-676
  ALTER privilege, 676
  anonymous-user accounts, 673
  FILE privilege, 674-676
  GRANT OPTION privileges, 674
  insecure accounts, 673-674
  mysql database privileges, 674
  passwords in old hash format, 673
  PROCESS privileges, 676
  RELOAD privileges, 676
  SUPER privileges, 674, 676
scope columns
  case sensitivity, 689
  column_name, 688
  Db, 688
  host, 687-689
  listing of, 687-689
  matching order, 690-691
  routine_name, 688
  table_name, 688
  user, 688
server table, preventing, 701
   internal locking, 702-703
   locking all tables at once, 705
   read-only locking, 703-704
   read/write locking, 704-705
   shutting down servers, 702
statement access verification, 689-690
stealing data example, 647
Unix socket file, 652-653
user accounts
   account-management statements, 654-655
   authentication, 659-660
   grant tables, upgrading, 655
   matching host values to DNS, 658-659
   names, 656-658
   passwords, changing/resetting, 672
user table row matching example, 691-694
accessor macros, Web: 1087-1088
account clause
   account-management statements, 656
   GRANT statement, 660
accounts
   administrator, creating, 24
   anonymous-user
   deleting, 568-569
   passwords, assigning, 567-568
   security risk, 673
initial user, 564-569
   available on all platforms, 566
   client program connections, 566
   displaying, 565
   passwords, assigning, 567-569
   platform specific, 567
login, creating, 738-739
   mysqld login, 571-572
   root passwords, 567-569, 582-583
user. See user accounts
ACID (Atomic, Consistent, Isolated, and Durable) properties, 157
ACOS() function, 784
action parameter, 513
activation state (plugins), 592
add_new_event() function, 517-518
ADDDATE() function, 803
addition (+) operator, 57, 241, 766
ADDTIME() function, 803

administration
   access control, 540
databases
   backups, 540
   migration, 541
   preventive maintenance, 540
   recovery, 541
   replication, 541
default character set/collation, 603-604
derent message language, setting, 604
initial user accounts, 564-569
   available on all platforms, 566
   client program connections, 566
   displaying, 565
   passwords, assigning, 567-569
   platform specific, 567
locale, 604-605
logs
   age-based expiration, 625
   binary, 618, 622-623
   enabling, 619
   error, 618, 620-621
   expiring, 629-631
   fixed-name, rotating, 626-629
   flushing, 626
   general query, 618, 621
   listing of, 617-618
   maintenance, 539
   output destination, selecting, 624-625
   relay, 618, 624
   replication-related expiration, 625
   rotating, 625
   slow query, 618,
   table truncation/rotation, 625
   tables, 625, 631
multiple servers, 539, 632
   client programs, running, 641
   configuring, 635
   directory options, 633
   error log file names, creating, 634
   InnoDB log location, 634
   issues, 632-635
   login account options, 635
   network interface options, 633
   replication slave options, 634
   startup options strategies, 636-637
status/log file names options, 634
Unix, 637-639
Windows, 639-641

mysqld
configuration and tuning, 539
connections, listening, 579-580
restarting manually, 581-582
root password, resetting, 582-583
startup/shutdown, 539, 577-579
stopping, 580-581

mysqld on Unix
running, 570
starting, 572-574
unprivileged login account, configuring, 571-572

mysqld on Windows, 575
running as Windows service, 576-577
running manually, 575

new server passwords, setting, 569

plugins
activation state, 592
case sensitivity, 591
displaying, 592
interface components, 590
library suffix, 590
loading at runtime, 591
loading at startup, 591
operations, 590
uninstalling, 592
software updates, 539

status variables
displaying, 584
overview, 584
values, checking, 588-589

storage engines
available, displaying, 593
default, selecting, 594
status/startup options, 593-594

system variables, 584-585
displaying, 583
overview, 583
setting at runtime, 587-588
setting at server startup, 586-587
values, checking, 585-586

time zones, 602-603
updates, 641-643

user account maintenance, 539

administrative functions
C, Web: 1117-1119
Perl DBI, Web: 1147-1148
administrative-only access, configuring, 649-651
data directory exception, 651
directories outside base directory, 651
innodb directory, 651
servers, running, 652
symlinks, 651
administrative privileges, 661-663, 666, 680
administrator accounts, creating, 24

advisory locking functions, 824-826
AES_DECRYPT() function, 821
AES_ENCRYPT() function, 821

aliases
case sensitivity, 100
quoting with identifiers, 98

ALL specifier, 146-147, 661, 666
ALTER DATABASE statements, 107, 898
ALTER EVENT statements, 898-899
ALTER FUNCTION statements, 899
ALTER privilege, 663, 676
ALTER PROCEDURE statements, 899
ALTER ROUTINE privilege, 663
ALTER TABLE statements, 899-904
action values, 899-903
benefits, 127-128
clauses
CHANGE, 128
CHARACTER SET, 128-129
ENGINE, 129
MODIFY, 128
RENAME, 129-130
indexes, adding, 124
partitioning options, 904
resequencing existing columns, 237
sequence columns, adding, 236
syntax, 128
table files, 549

ALTER VIEW statements, 905
ANALYZE TABLE statements, 905

AND (&) operator, 57, 773
AND (&&) operator, 241, 774

anonymous-user accounts
deleting, 568-569
passwords, assigning, 567-568
security risk, 673
ANSI_QUOTES mode, 96, 182, 863
ANSI SQL mode, 96
ANY subqueries, 146-147
Apache
configuring, 460-461
APIs
C. See C client programs
Perl DBI. See Perl DBI API
PHP. See PHP API
selecting, 314
development time, 316-317
execution environment, 315
performance, 315-316
portability, 317
SSL capabilities, 698
app_type member (my_option structures), 341
approximate-value numbers, 181-182
architecture
data directory, 545
Perl DBI API, 311-312
pluggable, 589-590
storage engines, 108
terminology, 21-22
ARCHIVE storage engine, 108, 112
arg_type member (my_option structures), 341
arguments
connect() function, 399-400
expression functions, 240
fetch() function, 502-503
undef, 422
vectors, processing, 342
arithmetic operators, 57, 766-767
addition, 766
DIV, 767
division, 767
listing of, 241
modulo, 767
multiplication, 767
NULL values, 246
rules, 766
subtraction, 767
ASCII conversions, 254
ASCII() function, 254, 790
ASIN() function, 784
ATAN() function, 785
ATAN2() function, 785
Atomic, Consistent, Isolated, and Durable (ACID) properties, 157
attributes. See also clauses
account, 660
collation, 102-103
columns, 35, 660
global, 747
Perl DBI, Web: 1149
database-handles, Web: 1149
dynamic, Web: 1155
general handle, Web: 1149-1150
MySQL-specific database handle,
Web: 1150-1152
MySQL-specific statement handle,
Web: 1154-1155
statement-handles, Web: 1152-1153
PDO database-handles, Web: 1173-1174
PrintError, 403
privileges, 660
RaiseError, 403
temporal data types, 223
TraceLevel, 428
what, 660
auth_info clause
CREATE USER statement, 659-660
GRANT statement, 661
authentication
columns (grant tables), 684
plugins, 676-679
proxy users, creating, 677-679
server connections, 677
server side/client side, 677
specifying, 676
user accounts, 659-660
AUTO_INCREMENT clause, 202
AUTO_INCREMENT columns, 230
adding to tables, 235-236
creating, 47
member table column example, 36
nonpositive numbers, 235
properties
general, 230-232
InnoDB, 234
MEMORY, 234-235
MyISAM, 232-234
ranges, 235
resequencing existing columns, 236-237
resets, 235
unsigned, 235
auto_increment_increment system variable, 836
auto_increment_offset system variable, 836
autocommit system variable, 836
automatic_sp_privileges system variable, 270, 837
automating
    initialization, 224-226
    log expiration, 630-631
    update properties, 224-226
auto-recovery, 706
    failure, 725-726
    performing, 700
available_drivers() function, Web: 1132
availability
    character sets, 103-104, 185-186
collations, 103-104, 185-186
result set metadata, 359
SSL support, 370
storage engines, 108-109
average values summaries, 76
AVG() functions, 76, 818

back_log system variable, 837
backslashes (\), strings, 184
backups, 707-709
    best practices, 709
    binary, 714-715
        complete, 714-715
    logs, 623
    partial, 715
databases, 540
InnoDB tables, 715-716
selecting, 708
slave, creating, 732-733
storage engine portability, 709-710
text, 711-714
    all tables from all databases, 711
    compressing, 712
database transfers, 716-717
individual files, 711
mysqldump options, 712-714
mysqldump output, 711-712
table subsets into separate files, creating, 712
types, 708

bail_out() function, 405
banner advertisement tables example, 19
basedir system variable, 837
Basic Multilingual Plane (BMP), 104
BEGIN...END statements, 988
BEGIN statement, 905
begin_work() function, Web: 1137
beginTransaction() function, Web: 1162
BENCHMARK() function, 829
BETWEEN operator, 243
big_tables system variable, 837
BIGINT data type, 193, 750-751
    ranges, 197
    storage requirements, 197
BIN() function, 200, 790
binary backups, 714-715
    best practices, 709
    complete, 714-715
    defined, 708
    partial, 715
    text-format backups, compared, 708
binary character sets, 216
binary data
    printing, 353
    statements, 367-368
BINARY data types 204, 207
binary logs, 556, 618
    administration, 622-623
    expiring, 629-630
    formats, 731
    index file, 623
    post-backup statements, re-executing, 723-725
    system backups, 623
binary protocol
    disadvantages, 378
    prepared statements, 377
        executing, 378-379
        inserting rows and retrieving them
        program, writing, 379-388
        parameterizing, 377-378
binary strings, 194, 755-756
    BINARY, 755
    BLOB, 755-756
    conversions, 255
    defined, 185
    LONGBLOB, 756
MEDIUMBLOB, 756
nonbinary strings, compared, 188-189
sorting properties, 186
TINYBLOB, 755
VARBINARY, 755

**BINARY str operator, 773**
bind_address system variable, 765
bind_col() function, 421, Web: 1142
bind_columns() function, 421, Web: 1142
bind_param() function, Web: 1142
bind_param_array() function, Web: 1143
bindColumn() function, 503, Web: 1166
bindParam() function, Web: 1166-1167
bindValue() function, Web: 1167
BINLOG statement, 906
binlog_cache_disk_use status variable, 881
binlog_cache_size system variable, 837
binlog_cache_use status variable, 881
binlog_checksum system variable, 837
binlog_direct_non_transactional_updates system variable, 838
binlog_format system variable, 838
binlog_row_image system variable, 838
binlog_rows_query_log_events system variable, 838
binlog_stmt_cache_disk_use status variable, 881
binlog_stmt_cache_size system variable, 838
binlog_stmt_cache_use status variable, 881
biographical information table, creating, 32, 34-35
BIT_AND() function, 818
BIT_COUNT() function, 829
BIT data types, 193, 197, 200-201, 752
ranges, 197
storage requirements, 197

**bit-field numbers, 182**

**BIT_LENGTH() function, 829**

**bit operators**
AND, 773
exclusive-OR, 773
listing of, 242
negation, 774
NULL values, 246
OR, 773
shift left, 773
shift right, 773, 818

**BIT_OR() function, 818**

**BIT_XOR() function, 809**
BLACKHOLE storage engine, 108, 112
BLOB data types
indexes, 207
overview, 207-208
query optimization, 298
size, 204, 207
special care, 208
storage requirements, 204
BLOB strings, 194, 755-756
block_size member (my_option structures), 341
BMP (Basic Multilingual Plane), 104
boolean mode searches, 170, 174-175
boolean values, 192
bulk_insert_buffer_size system variable, 838
bytes_received status variable, 881
bytes_sent status variable, 882

**C**

**C client programs, 311**
accessor macros, Web: 1087-1088
client library, 320
compiling/linking, 321, Web: 1074-1075
connect1, 323-326
connect2, compared, 347
establishing connections, 324-325
header files, 324
initialization macro, 326
initializing client library, 326
running, 326
shortcomings, 326
source file, 323-324
terminating client library, 326
terminating connections, 325
variables, declaring, 324
connect1/show_opt programs, compared, 347
connection parameters, specifying, 348
error-checking, 330
running, 347
source file, 344-347
connection parameters at runtime, specifying, 331
command-line option-handling, 335-343
option files, reading, 332-335
parameter formats, 331
data structures, Web: 1075
non scalar. See non scalar data structures
scalar data types, Web: 1075-1076
error-checking, 327-330
example resources, 320
functions
administrative, Web: 1125-1126
client library initialization/ termination, Web: 1088-1089
connection management, listing of, Web: 1089-1100
debugging, Web: 1127
error-reporting, Web: 1101
information, Web: 1113-1116
multiple result sets, Web: 1113
parameter names, Web: 1087-1088
prepared statement construction/ execution, Web: 1118-1120
prepared statement error-reporting, Web: 1117-1118
prepared statement result set processing, Web: 1120-1125
prepared statements, Web: 1116-1117
result sets processing, Web: 1104-1113
statement construction/execution, Web: 1102-1104
threaded clients, Web: 1126-1127
transaction control, Web: 1116
header files, 320
interactive statement-execution, 368-369
Makefiles, 322-323
multiple servers, running, 641
new client, 348
prepared call, 390-393
output, 393
parameter setup, 390-391
prepared statement handler, initializing, 390
results, processing, 391
retrieval loop, 392
server versions, verifying, 390
prepared statements, 377
executing, 378-379
inserting rows and retrieving them program, writing, 379-388
parameterizing, 377-378
result sets
metadata, 359-364
returning, 351-353
row-modifying, 350
sending to server functions, 349
special characters, 365-366
SSL support, 370-374
availability, 370
enabling, 696
holding option values variables, 372-373
options, adding, 370-372
passing SSL option information to client library, 374
statements, handling, 348-350
alternative approaches, 356-357
binary data, 367-368
causes of failures, 349
character-escaping operations, 349
general-purpose statement handler, 354-355
multiple-statement execution, 375-377
mysql_store_result() versus mysql_use_result() functions, 357-359
result sets, returning, 351-353
row-modifying, 350
sending to server functions, 349
special characters, 365-366
CACHE INDEX statement, 906
CALL prepared statements, 389-393
output, 393
parameter setup, 390-391
prepared statement handlers, initializing, 390
results, processing, 391
retrieval loop, 392
server versions, verifying, 390
CALL statement, 906
CAs (Certificate Authorities), 695
cascaded deletes, 164, 167-168
cascaded updates, 164, 168
CASE [expr] WHEN expr1 THEN result1 ... [ELSE default] END operator, 771
case sensitivity, 881
aliases, 100
columns, 100
database names, 100
filenames, 100
forcing lowercase, 100
functions, 99
index names, 100
keywords, 99
LIKE operator, 244
MySQL utilities, 1001
Perl DBI scripts, 400
plugins, 591
scope columns, 689
SQL statements, 29, 99-101
stored program names, 100
strings, 100, 183
system variables, 836
table names, 100
trigger names, 100
view names, 100
CASE statements, 988
CAST() function, 253, 783-784
cast operators, 775-776
category columns, creating, 47
CEIL() function, 781
CEILING() function, 785
certificate files (SSL), 695
CGI scripts, 459-460
functions
  HTML structure, 461
  importing, 461
  object-oriented interface, 461-462
HTML
  text, escaping, 464-465
  versus XHTML, 464
input parameters, 462
multiple-purpose pages, writing, 465-468
output, generating, 462-464
portability, 463
URL text, escaping, 464-465
CHANGE clause, 128
CHANGE MASTER statement, 907-908
CHAR data types
  size/storage requirements, 204
  VARCHAR data types, compared, 206
CHAR() function, 254, 790
CHAR strings, 194, 756-757
CHAR_LENGTH() function, 790
character data, retrieving, 56
CHARACTER_LENGTH() function, 786

CHARACTER SET clause
  ALTER TABLE statement, 128-129
  CREATE DATABASE statement, 106
  rules, 102-103
character_set_client system variable, 838
character_set_connection system variable, 838
character_set_database system variable, 839
character_set_filesystem system variable, 839
character_set_results system variable, 839
character_set_server system variable, 102, 603, 839
character_set_system system variable, 839
character sets
  availability, 103-104, 185-186
  columns, editing, 128-129
  conversions, 254
  current, displaying, 104
  default, setting, 603-604
  features, 101-102
  mixing, 102
  setting, 102-103
  strings, 214-216, 753-754
    binary, 188-189, 216
    CONVERT() function, 187-188
    displaying, 215
    example, 214
    introducers, 187-188
    nonbinary, 188-189
    rules, 214
    selecting, 215
    variables, 189-191
  Unicode support, 104-105
  variables, 189-191
character_sets_dir system variable, 839
CHARSET clause
  CHARACTER SET statement, 102-103
  string data types, 214-216
    character sets, displaying, 215
    example, 214
    rules, 214
CHARSET() function, 790
charset notation, 187
_charset str operator, 773
check_pass() function, 533
check_response() function, 527
CHECK TABLE statement, 719-720, 909-910
checking tables
  CHECK TABLE statement, 719-720
  InnoDB, 718
  MyISAM, 719
  mysqlcheck utility, 720-721
CHECKSUM TABLE statement, 910
chk_mysql_opt_files.pl script, 653-654
clauses. See also attributes
  auth_info, 659-661
  AUTO_INCREMENT, 36, 202
  CHANGE, 128
  CHARACTER SET, 106, 128-129
  CHARSET, 102-103, 214-216
  COLLATE, 102-103, 106, 214-216
  CREATE DATABASE statement, 106
  data types, 201-203, 214-216, 747
  DEFAULT, 203
  DEFINER, 276
  ENGINE, 46-47, 129
  FROM, 54-56, 149
  GRANT statement, 660-661
  GROUP BY, 74-76
  IDENTIFIED WITH, 676
  IF NOT EXISTS, 106
  LIKE, 131, 585, 589
  LIMIT, 63-64, 475
  MODIFY, 128
  NOT NULL, 216, 223
  NULL, 216, 223
  numeric data types, 749
  ON DELETE CASCADE, 166-167
  ON DELETE SET NULL, 169
  ON UPDATE CASCADE, 166-167
  ON UPDATE SET NULL, 169
  PARTITION BY, 120-121
  RENAME, 129-130
  REPLACE, 232
  REQUIRE
    GRANT statement, 661
    GRANT USAGE statement, 697
    secure connections, 668-669
  RETURNS, 268
  ROLLUP, 77
  SELECT statements, 956-957
  SIGNED, 201
  TEMPORARY, 115-116
  UNSIGNED
  AUTO_INCREMENT, 235
  numeric data types, 201
  UPDATE, 232
WHERE
  COUNT() function, 72
  DELETE statement, 85
  query optimizer, 288-289
  SELECT statement, 56
  SHOW statement, 131
  SHOW STATUS statement, 589
  SHOW VARIABLES statement, 585
  UPDATE statement, 86
  WITH
    GRANT statement, 661
    resource consumption limits, 670
  WITH GRANT OPTION, 669-670
  WITH ROLLUP, 77-78, 263
  ZEROFILL, 201-202
client access, 686-687
  scope columns
    case sensitivity, 689
    column name, 688
    Db, 688
    host, 687-689
    listing of, 687-689
    matching order, 690-691
    proxied_host, 689
    proxied_user, 689
    routine_name, 688
    routine_type, 688
    table_name, 688
    user, 688
    statement access verification, 689-690
    user table row matching example, 691-694
client programs. See C client programs
clone() function, Web: 1137
CLOSE statements, 992
closeCursor() function, Web: 1168
COALESCE() function, 790
COERCIBILITY() function, 791
col_prompt() function, 451
COLLATE clause, 102-103
  CREATE DATABASE statement, 106
  string data types, 214-216
collations, displaying, 215
dexample, 214
  rules, 214
columns 1083
collation attribute, 102-103
collation_connection system variable, 839
collation_database system variable, 839
COLLATION() function, 255, 787
collations
  availability, 103-104, 185-186
  current, displaying, 104
  default, setting, 603-604
  names, 186
  setting, 102-103
  strings, 753-754
    binary versus nonbinary, 188-189
    displaying, 215
    example, 214
    rules, 214
  suffixes, 186
  type conversions, 255
collation_server system variable, 102, 603, 839
column_name columns, 688
columnCount() function, Web: 1168
columns
  aliases, quoting with identifiers, 98
  attributes, 35
  AUTO_INCREMENT, 36
    adding to tables, 235-236
    creating, 47
    member table column example, 36
  nonpositive numbers, 235
  properties, 230-235
  ranges, 235
  resequencing existing columns, 236-237
  resets, 235
  unsigned, 235
category, creating, 47
character sets, editing, 128-129
contents, retrieving, 54-56
currency information, 258
data types
  editing, 128
  specifying, 193-195
date
  creating, 47
  information, 258-259
  values, 35
deleting, 85-86
displaying, 131
enumeration, creating, 46
expiration, creating, 36
grant tables
  administrative privilege, 680
  authentication, 684
  object privilege, 682-681
  privilege columns, 683-684
  resource management, 685-686
  scope. See scope columns
  scope-of-access, 680-681, 683
  user table authentication, SSL, resource management, 680
height information, 257-258
identical data types, comparing, 288
identifiers, 99
indexing, selecting, 281-285
  badly performing queries, identifying, 285
  cardinality, 282
  comparisons, matching to index types, 284-285
  overindexing, 284
  prefixes, 283-284
  short values, 283
individual values, retrieving, 503
information, displaying, 135
INFORMATION_SCHEMA database, displaying, 134
information structures, accessing, 364
integer, creating, 46, 48
joined table references, qualifying, 138-139
member_id, creating, 36
names
  case sensitivity, 100
  views, 263-264
output
  restrictions, 37
  values, naming, 64-66
PRIMARY KEY clauses, 36
privileges, 667
references, 240
scope. See scope columns
sequence
  adding to tables, 235-236
  creating, 47, 237-239
  general properties, 230-232
InnoDB characteristics, 234
MEMORY characteristics, 234-235
MyISAM characteristics, 232-234
nonpositive numbers, 235
ranges, 235
resequencing existing, 236-237
resets, 235
unsigned, 235
unsetting, 87
updating, 86
values, specifying, 196
variable-length, 35, 46
columns attribute, 660
columns_priv table, 680
com_xxx status variable, 882
comma (,) join operator, 138
command line
metadata access, 135
mysqld startup options, 578
option-handling, 335-343
argument vector, processing, 342
option information, defining, 339-341
show_opt, invoking, 342-343
show_opt program, 336-338
SSL options, 697
system variables, setting, 586
commands
input editing, 90-91
mysql utility, 1026-1028
mysqladmin client, 1035-1037
mysqlshow, 135
perldoc, 743
comments
my_option structures, 340
Perl DBI scripts, adding, 398
syntax, 996-997
commit() function, Web: 1137, Web: 1162
COMMIT statement, 910-911
comparison functions, 781-783
comparison operators, 57, 768-772
CASE [expr WHEN expr1 THEN result1 ...
... [ELSE default] END, 771
equal, 769
expr BETWEEN min AND max, 770-771
expr IN (value1,value2,...), 772
expr IS, 772
expr IS NULL/expr IS NOT NULL, 772
expr NOT BETWEEN min AND max, 770-771
expr NOT IN (value1,value2,...), 772
greater than, 770
greater than or equal to, 770
less than, 770
less than or equal to, 770
listing of, 243
NULL values, 247
null-safe equality, 769
rules, 768-769
unequal, 770
comparisons
data types, 748
index type matching, 284-285
complete binary backups, 714-715
completion_type system variable, 839
composite indexes, 233
compound statements, 266-267, 987-996
condition-handling, 992-996
control structure, 987-989
cursor, 991-992
declaration, 989-991
COMPRESS() function, 821
compressing dump files, 712
compression functions, 821-824
compression status variable, 882
CONCAT() function, 248, 253, 791
CONCAT_WS(), 792
concurrency
problems, preventing, 156
storage engine locking levels, 303-305
concurrent_insert system variable, 840
condition-handling statements, 992-996
configuring
administrative-only access, 649-651
data directory exception, 651
directories outside base directory, 651
innodb directory, 651
servers, running, 652
symlinks, 651
character sets, 102-103
collations, 102-103
full-text searches, 176-177
InnoDB tablespace, 595-598
auto-extend increments, 596
file pathnames, 596
file specification syntax, 596
per-table, 599-600
raw partitions, 597-598
regular files, 597
system variables, 595
Windows, 598
master-slave replication, 728-731
master server settings, 728-729
master.info file, 730
separate slave accounts, 730
server ID values, assigning, 728
slave settings, 729-730
statements, 730-731
threads, starting/stopping, 731
multiple servers, 635
MYSQL_BIND arrays
  insert_rows() function, 384
  select_rows() function, 385-388
mysqld, 539
SQL mode, 96-97
SSL, 695-698
  accounts requiring SSL, creating, 697-698
  certificate/key files, 697
  client programs SSL support, enabling, 696
  command-line options, 697
  language APIs, 698
  option files, 697
  server SSL support, enabling, 695-696
  SSL-related server status variables values, displaying, 697
system variables
  runtime, 587-588
  server startup, 586-587
tables, 111
time zones, 602-603
unprivileged mysqld login accounts, 571-572
utility variables, 1006-1007
Web servers, 460-461
connect() function, Web: 1132-1136
  connection parameters, 432, Web: 1135-1136
  driver options, Web: 1133-1135
  Perl DBI scripts, 399-400
connect_cached() function, Web: 1136
connect_timeout system variable, 840
connect1 client program, 323-326
  client library
    initializing, 326
    terminating, 326
connect2, compared, 347
connections
  establishing, 324-325
  terminating, 325
  header files, 324
  initialization macro, 326
  running, 326
  shortcomings, 326
  source file, 323-324
  variables, declaring, 324
connect2 client program, 327, 344-348
  connect1 program, compared, 347
  connection parameters, specifying, 348
  error-checking, 330
  new client programs based on, writing, 348
  running, 347
  show_opt programs, compared, 347
  source file, 344-347
connection_errors_xxx status variable, 882
CONNECTION_ID() function, 829
CONNECTION_USER() function, 830
connections
  databases (Perl DBI scripts), 312, 400
  handlers, 325
  management functions, Web: 1088-1099
  mysql utility, 87
    option files, 87-88
    shell aliases/scripts, 89
    shell command history, 88
  mysqld
    restarting manually, 581-582
    root password, resetting, 582-583
  parameters, specifying
    C client programs, 331, 336-341
    command-line option-handling, 335-343
    connect2 program, 348
    option files, reading, 332-335
    parameter formats, 331
    Perl DBI, 423-426
  secure, requiring. See also SSL, 668-669
servers
  authentication plugins, 677
  establishing, 25-26
  PHP scripts, 490-491
  programs. See connect1 client program; connect2 client program
  terminating, 26-27
  Web scripts, 468-469
TCP/IP
  listening (mysqld), 579
connections status variable, 882
constants (PDO), Web: 1173-1174
  general database-handle attributes, Web: 1173-1174
  fetch-mode values, Web: 1174
  parameter-type values, Web: 1174
constructor (PDO), Web: 1159-1161
Content-Type: header, 463
control structure statements, 987-989
CONV() function, 792
CONVERT() function, 187-188, 254-255, 784
CONVERT_TZ() function, 803
copying
  databases to other servers, 716
  text backup files, 716-717
  writing directly to other server, 717-718
  tables, 117-120
core_file system variable, 840
correlated subqueries, 148
COS() function, 785
costs (indexing), 281
COT() function, 785
COUNT() function, 819
  GROUP BY clause, 74-76
  ROLLUP clause, 77
  summaries, 72-76
  WHERE clause, 72
  WITH ROLLUP clause, 77-78
counters, incrementing, 238-239
counting summaries, 72-76
  distinct non-NULL values, 73
  groups, 74-76
  minimum/maximum/total/average values, 76
  non-NULL values, 73
  number of rows clause matches, 72
  number of rows selected, 72
  overall count of values, 73
  summary, 77-78
CRC32() function, 785
CREATE DATABASE statement, 30, 106-107, 130, 547, 911
CREATE EVENT statement, 274, 912-913
CREATE FUNCTION statement, 268, 913-915
CREATE INDEX statement, 915-916
CREATE privilege, 664
CREATE PROCEDURE statement, 268, 913-915
CREATE ROUTINE privilege, 664
CREATE TABLE statement, 113-114, 916-926
  AVG_ROW_LENGTH option, 115
  column definitions, 926
  data type keywords, 918-919
  ENGINE clause, 46-47, 114
  foreign key support, 922-923
  IF NOT EXISTS modifier, 115
  index clauses, 919
  MAX_ROWS option, 115
  options, 919-922
  PARTITION BY clause, 120-121
  partitioning, 923-925
  student table, 45-46
  table files, creating, 549
  TEMPORARY keyword, 115-116
CREATE TABLE...LIKE statement, 117-118
CREATE TABLE...SELECT statement, 117-119
create_tmp_disk_tables status variable, 882
core_file system variable, 840
currency information, storing, 258
cur_case() function, 68, 803
csv storage engine, 108, 112, 710
currval() function, 68, 803
current_user() function, 803
CURRENT_TIME() function, 803
CURRENT_TIMESTAMP() function, 804
CURRENT_USER() function, 276, 816
cursor statements, 991-992
CURTIME() function, 804

D

damages (tables)
  checking
    CHECK TABLE statement, 719-720
  InnoDB tables, 718
  MyISAM, 719
  mysqlcheck utility, 720-721
  overview, 718
  repairing
    InnoDB, 718
    MyISAM, 719
    mysqlcheck utility, 720-721
    REPAIR TABLE statement, 720

data
  adding to tables
    data files, 52-53
    INSERT statement, 50-52
  binary
    printing, 353
    statements, 367-368
  C API structures, Web: 1075
    nonscalar. See nonscalar data structures
    scalar data types, Web: 1075-1076
  format options, 1067-1068
  loading efficiency, 300-303
  dropping/deactivating indexes, 302-303
  index flushing, reducing, 301-302
  INSERT statement, 301
  LOAD DATA statement, 300-301
  mixed query environments, 303
  shorter statements, 302
  recovering. See recovery
  retrieving
    column values, naming, 64-66
    criteria, specifying, 56-59
    dates, 57, 66-69
    multiple tables, 78-85. See also joins; subqueries
    NULL values, 60-61
  numeric ranges, 56
  pattern matching, 69-70
  Perl DBI script. See dump_members.pl script
  PHP script, 497-499
  SELECT statements, 54-56
  several individual values, 59
  string values containing character data, 56
  summaries, 72-78
  table contents, displaying, 54
  user-defined variables, 71

data directory
  access
    control, 546-547
    exception, 651
  architecture, 545
  defined, 539
  file representations
    databases, 547
    tables, 548
    triggers, 549
    views, 549
  files, 545
  grant tables. See grant tables
  identifier constraints, 550-551
  initializing, 740-741
  insecurities, checking, 648
  location, 544-545
  log files, 554-556
  maximum table size, 551-553
  performance, 553-554
  permissions, displaying, 650
  PID files, 555
  relocating, 556-557
  assessing, 558-559
  entire directory, 559
  function, selecting, 557
  individual databases, 559-560
  individual tables, 560
  InnoDB tablespace, 561
  precautions, 558
  startup option, 557
  status/log files, 561-562
  symlink, 557
  status files, 554
  table operations statements, 549-550
  Unix, 543
data_sources() function, Web: 1136

data types
- attributes, 747
- character sets
  - features, 101-102
  - mixing, 102
  - setting, 102-103
- characteristics, 192
- collations, 102-103
- columns
  - editing, 128
  - specifying, 193-195
- comparisons, 748
- conversion, 247-251
  - binary/nonbinary strings, 255
  - character sets, 254
  - collations, 255
  - comparisons, 251
  - CONCAT() function, 248
- dates, 254
  - explicit, 247
- floating-point and integer values, 248
  - forcing, 253-255
- hexadecimal, 248-249, 253
- illegal values, 248
- implicit, 247
- operands to operator expected types, 249
  - string-to-number, 249-250
  - temporal values, 251
  - testing, 252-253
- time parts, 254
  - values into strings, 253

date. See temporal data types

default values, 748
- ENUM, 297
- explicit, 179
- global attributes, 747
- implicit, 95
- length, 748
- MYSQL_ROW, 352
- names, 747
- numeric, 193, 748-749
  - attributes, 201-203, 749
- BIT, 197, 200-201, 752
- exact-value, 197-199
- fixed-point, 751
- floating-point, 197, 200, 751-752
- improper values, 228
- integer, 749-751
- listing of, 193
- NULL/NOT NULL values, 203
  - ranges, 197
- selecting, 203, 257-258
  - storage requirements, 197-198
- Perl DBI. See handles
  - query performance, selecting, 296-298
  - BLOB/TEXT, 298
  - ENUM, 297
  - NOT NULL, 297
  - numbers, 296
  - PROCEDURE ANALYSE() function, 297
  - smallest types, 296-297
  - strings, 296
  - tables, defragmenting, 297
- ranges, 748
- scalar, Web: 1075-1076
  - selecting, 255-256
  - currency, 258
  - dates, 258-259
  - height information, 257-258
  - performance/efficiency, 256
  - ranges, 256, 259-260
  - storage size, 256
  - value types in column, 256-259
- storage, 748
- string, 193, 204, 753-754
  - attributes, 214-216
  - binary, 204-205, 207, 755-756
  - BLOB, 207-208
  - CHAR/VARCHAR, 206
  - character sets/collations, 753-754
  - ENUM, 208-213, 758
  - improper values, 228
  - lengths, 205, 753
  - listing of, 194
  - nonbinary, 204-205, 756-758
  - selecting, 217-218
  - SET, 208-213, 759
  - size, 204
  - storage requirements, 204
  - TEXT, 207-208
  - trailing pad values, 218, 754
- VARBINARY, 207
temporal, 193, 759
  attributes, 223
automatic initialization/update properties, 224-226
DATE, 220-221, 760
DATETIME, 221, 760
fractional seconds, 223-224
improper values, 228
input dates, 220
listing of, 193
MySQL 5.6 improvements, 218
ranges, 218-219
storage requirements, 219, 759
temporal values, 226-227
TIME, 221, 760-761
TIMESTAMP, 221-222, 761-762
two-digit years, 227-228
YEAR, 222-223, 762
zero values, 220
type conversions
  ASCII, 254
binary/nonbinary strings, 255
character sets, 254
collations, 255
comparisons, 251
CONCAT() function, 248
dates, 254
explicit, 247
floating-point and integer values, 248
forcing, 253-255
hexadecimal, 248-249, 253
illegal values, 248
implicit, 247
operands to operator expected types, 249
string-to-number, 249-250
temporal values, 251
testing, 252-253
time parts, 254
values into strings, 253
variable-length characters, creating, 35
zero values, 748
data values
  boolean, 192
columns, specifying, 196
improper handling, 228-230
NULL, 192
numeric, 181-182
permitted lists, defining, 209
spatial, 191-192
strings. See strings, values
temporal, 191
DATABASE() function, 830
databases
  access interfaces (PHP), 485-486
  backups, 540, 707-709
    best practices, 709
    binary, 714-715
    selecting types, 708
    storage engine portability, 709-710
text, 711-714
  browser script, 471-475
data limits, 475
  empty values into nonbreaking spaces, converting, 475
  HTML table, creating, 475
  initial page, generating, 472-473
  main body, 471-472
  security warning, 471
table contents, displaying, 473
tbl_name parameter, 472
connections, 400
copying to other servers, 716
text backup files, 716-717
  writing directly to other server, 717-718
crash recovery. See recovery
creating, 30-31, 106-107
data, loading, 300-303
dropping/deactivating indexes, 302-303
  index flushing, reducing, 301-302
  INSERT statement, 301
LOAD DATA statement, 300-301
mixed-query environments, 303
shorter statements, 302
data directory, relocating, 559-560
default, setting, 30-31
definition, displaying, 106-107
deleting, 107
documentation, 107
table representations, 547
handles
  attributes, Web: 1149
  functions, Web: 1137-1142
MySQL-specific attributes, Web: 1150-1152
PDO attributes, Web: 1173-1174
Identifiers, 98
INFORMATION_SCHEMA
  columns, displaying, 134
displaying, 132
metadata access, 132-135
tables, 133-134
Integrity, maintaining
  auto-recovery, 706
  preventive maintenance, scheduling, 707
Listing, 38, 130, 135
Metadata, accessing, 130
  command line, 135
INFORMATION_SCHEMA database, 132-135
SHOW statement, 130-132
Migration, 541
MySQL privileges, 673-674
Names, case sensitivity, 100
Preventive maintenance, 540, 699-700
Privileges, 666
Recovering, 541, 722
Replication, 541
  compatibility guidelines, 727-728
  master-slave, 728-731
  overview, 727
Resetting to known state, 53-54
Selecting, 105-106
Server connectivity, 312, 400
tables, listing, 37
types, 708
datadir system variable, 840
DATE() function, 804
DATE_ADD() function, 68, 254, 804-805
Date and time
  columns, creating, 47
data types. See temporal data types
differences between, 68
expiration columns, creating, 36
formats, 226
functions, 802-821
  ADDDATE(), 803
  ADDTIME(), 803
  CONVERT_TZ(), 803
CURDATE(), 803
CURRENT_DATE(), 803
CURRENT_TIME(), 803
CURRENT_TIMESTAMP(), 804
CURTIME(), 804
DATE(), 804
DATE_ADD(), 804-805
DATE_FORMAT(), 806
DATE_SUB(), 807
DATEDIFF(), 807
DAY(), 808
DAYNAME(), 808
DAYOFMONTH(), 808
DAYOFWEEK(), 808
DAYOFYEAR(), 808
EXTRACT(), 808-809
FROM_DAYS(), 809
FROM_UNIXTIME(), 809
GET_FORMAT(), 809-810
HOUR(), 810
LAST_DAY(), 810
listing of, 802-821
LOCALTIME(), 810
LOCALTIMESTAMP(), 810
MAKEDATE(), 810
MAKETIME(), 811
MICROSECOND(), 811
MINUTE(), 811
MONTH(), 811
MONTHNAME(), 811
NOW(), 811
PERIOD_ADD(), 812
PERIOD_DIFF(), 812
QUARTER(), 812
SEC_TO_TIME(), 812
SECOND(), 812
STR_TO_DATE(), 813
SUBDATE(), 813
SUBTIME(), 813
SYSDATE(), 813
TIME(), 813
TIME_FORMAT(), 813
TIME_TO_SEC(), 814
TIMEDIFF(), 814
TIMESTAMP(), 814
TIMESTAMPADD(), 814
TIMESTAMPDIFF(), 814
非断言空格，475
安全警告，471
tbl_name参数，472
Db列，688
db表，680
DBI_DRIVER环境变量，Web: 1156
DBI_DSN环境变量，Web: 1156
DBI_PASS环境变量，Web: 1156
DBI_TRACE环境变量，429，Web: 1156
DBI_USER环境变量，Web: 1156
DEALLOCATE PREPARE语句，929
debug系统变量，840
debugging
  函数，Web: 1119-1120
  Perl DBI脚本，426
  打印语句，428
  跟踪，428-429
DECIMAL数据类型，193，751
  范围，197
  存储要求，197
DECLARE语句，989-991
DECODER函数，822
递减数字序列，创建，238
def_value成员（my_option结构），341
DEFAULT属性，203
DEFAULT()函数，830
default数据库设置，30-31
default_storage_engine系统变量，840
default_tmp_storage_engine系统变量，840
default_week_format系统变量，840
definer权限，276
definer特权，276
defragmenting tables，297
date时间格式系统变量，840
day()函数，808
dayname()函数，808
dayofmonth()函数，67，808
dayofweek()函数，808
dayofyear()函数，808
db_browse.pl脚本，471-475
display_table_contents()函数，473-475
display_table_names()函数，472-473
HTML表，创建，475
LIMIT子句，475
main body，471-472
非断言空格，475
安全警告，471
tbl_name参数，472
Db列，688
db表，680
DBI_DRIVER环境变量，Web: 1156
DBI_DSN环境变量，Web: 1156
DBI_PASS环境变量，Web: 1156
DBI_TRACE环境变量，429，Web: 1156
DBI_USER环境变量，Web: 1156
DEALLOCATE PREPARE语句，929
debug系统变量，840
debugging
  函数，Web: 1119-1120
  Perl DBI脚本，426
  打印语句，428
  跟踪，428-429
DECIMAL数据类型，193，751
  范围，197
  存储要求，197
DECLARE语句，989-991
DECODER函数，822
递减数字序列，创建，238
def_value成员（my_option结构），341
DEFAULT属性，203
DEFAULT()函数，830
default数据库设置，30-31
default_storage_engine系统变量，840
default_tmp_storage_engine系统变量，840
default_week_format系统变量，840
definer权限，664
DELETE特权，664
DELETE语句，929-930
  多个表，154-155
  行，85-86
deleting

anonymous-user accounts, 568-569
cascaded deletes, 164, 167-168
columns, 85-86
databases, 107
rows, 85-86
events, 275
  multiple tables, 154-155
  preserving sequencing, 235
tables, 121-122
delimiters (compound statements), 266-267
DES_DECRYPT() function, 822
DES_ENCRYPT() function, 822-823
DESCRIBE statement, 36-37, 930-931
development releases, 643
directories
  creating (Perl DBI), 436-442
  plain text, 439-440
  RTF version, 440-442
  online, creating, 455-458
  sampdb distribution, 735-736
    Perl DBI scripts, 476-477
    PHP, 514-515
dirty reads, 162
disaster planning. See recovery
disconnect() function, Web: 1137
display_cell() function, 516
display_column() function, 535
display_entry() function, 531-533
display_events() function
display_form() function, 525-526
display_login_form() function, 530
display_login_page() function, 530
display_scores() function, 477-479, 518-519
display_table_contents() function, 473-475
display_table_names() function, 472-473
displaying
  character sets available, 185-186
  collations available, 185-186
  columns, 131, 134
  CREATE DATABASE statement, 130
  current character sets/collations, 104
  database definitions, 106-107
databases, 130, 135
  errors, 170
  foreign keys, 170
help messages (utilities), 1000-1001
indexes, 131
INFORMATION_SCHEMA database, 132
initial user accounts, 565
plugins, 592
privileges, 671
result set metadata, 360-364
column display width, 361-362
final code, 362-364
printing
  boxed column labels, 362
  values, 362
row storage formats, 300
SSL-related server status variable values, 697
statement results, 28
status variables, 584
storage engines available, 593
system variables, 583, 836
tables, 130
  contents, 50, 54
  structure, 36-37
distinct non-NULL values, counting, 73
DIV (integer division) operator, 57, 241, 767
div_precision_increment system variable, 841
division by zero errors, 229
division (/) operator, 57, 241, 767
DNS, account name host values, matching, 658-659
do() function, 406-407, Web: 1137-1138
do statement, 931
dollar signs ($), PHP, 492
DOUBLE data type, 193
  ranges, 197
  storage requirements, 197
double-quoting strings (qq), 417-418
DROP DATABASE statement, 107, 547, 931
DROP EVENT statement, 932
DROP FUNCTION statement, 932
DROP INDEX statement, 127, 302, 932
DROP privilege, 664
DROP PROCEDURE statement, 932
DROP TABLE statement, 121-122, 549, 932
DROP TRIGGER statement, 932-933
DROP USER statement, 656, 933
DROP VIEW statement, 933
dropping. See deleting
dump_members.php script, 497-499
- display values, encoding, 498
- error handling, 498
- home page link, creating, 498-499
- installing/accessing, 498
- result set, returning, 498

dump_members.pl script, 397-398
- case sensitivity, 400
- comments, adding, 398
- connect() function arguments, 399-400
- connections, 400
- disconnecting, 402
- finish() function, 402
- result sets, retrieving, 400-401
- row-fetching loop, 401-402
- statement terminators, 401
- use DBI statement, 399
- use strict statement, 399
- use warnings statement, 399
- warnings, 401

dump_members2.php script, 499-500

dump_members2.pl script, 404-405
dump_results() function, Web: 1143

dynamic attributes (Perl DBI), Web: 1145-1155

E

edit_member() function, 452

edit_member.php script
- editing form, 533-534
- framework, 529-530
- member login page, 530-531
- null values, 535-536
- password verification, 531-533
- updating entries, 534-535

edit_member.pl script, 448-454

editing
- columns
  - character sets, 128-129
  - data types, 128
- databases, 107
- rows
  - storage formats, 300
  - with statements, 350
- tables
  - storage characteristics, 114-115
  - structure, 127-130

user account passwords, 672
U.S. Historical League member entries
command-line script, 448-454
online, 527-536
ellipsis (...), operators/functions, 764
ELT() function, 779
empty values, 475
ENCODE() function, 823
ENCRYPT() function, 823
ending
- server connections, 26-27
- statements, 27-28
- transactions, 160

ENGINE clause
- ALTER TABLE statement, 129
- CREATE TABLE statement, 46-47

enter_scores() function, 520-521

entering statements, 27
- case-sensitivity, 29
- function syntax, 29
- multiple-lines, 28
- multiple statements on single line, 28-29

ENUM data type, 208-213
- creating, 46, 208
- improper values, 228
- numeric form, 210-211
- permitted value lists, defining, 209
- query optimization, 297
- SET data type, compared, 208
- size/storage requirements, 204
- sorting/indexing, 212-213

ENUM strings, 194, 758

environment variables
- DBI_TRACE, 429
- PATH, configuring, 739-740
- Perl DBI, Web: 1156
- utility options, checking, 1011-1012

eq_range_index_divide_limit system variable, 841

equal to (=) operator, 57, 243, 769

equal to (<=) operator, 57, 769

err() function, Web: 1146

error_count system variable, 842

error handling
- foreign keys, displaying, 170
- improper values. See improper values
- message language, setting, 604
- PDO exceptions, 491
Perl DBI, 402-405
  automatic, 403-404
  checking, 400
default error messages, replacing, 404
default settings, 403
dump_members2.pl script example, 404-405
  manually checking/printing, 403
PrintError attribute, 403
RaiseError attribute, 403
PHP, 507-509
  prepared statement functions,
    Web: 1112-1113
  reporting functions, Web: 1099
error logs, 556, 620-621
  defined, 618
event scheduler, 274
  levels, selecting, 620
multiple servers, 634
Unix, 620
Windows, 621
errorCode() function, 508, Web: 1162, Web: 1168
ERROR_FOR_DIVISION_BY_ZERO, 863
errorInfo() function, 508, Web: 1163, Web: 1168
errstr() function, Web: 1146
escape demo.pl script, 464-465
escape sequences
  strings, 183
  utility option files, 1010
escapeHTML() function, 464-465
EVENT privilege, 664
event_scheduler system variable, 842
events, 274-275
  creating, 274
defined, 274
  deleting old rows from table example, 275
enabling/disabling, 275
IDs, 41-42
one time only, 275
privileges, 274
scheduler
  enabling, 274
  logging, 274
starting/Stopping at runtime, 274
status, verifying, 274
security, 276

exact-value data types, 197-199
exact-value numbers, 181-182
exception functions, Web: 1172-1173
exclusive-OR (XOR) operator, 773
exec() function, Web: 1163
  prepared statements, 505
  row-modifying statements, 501
exec_stmt program, 368-369
exec_stmt_ssl.c, creating, 370-374
  availability, 370
  holding option values variables, 372-373
  options, adding, 370-372
  running, 374
execute() function, Web: 1143, Web: 1168
execute_array() function, Web: 1143
EXECUTE privilege, 664
EXECUTE statement, 933
EXISTS subqueries, 147-148
EXP() function, 786
expiration column, 36
  automating, 630-631
  binary, 629-630
  relay, 630
expiration column, 36
expiring logs, 625, 629-631
  automating, 630-631
  binary, 629-630
  running, 630
EXPLAIN statement, 290-296, 933-936
explicit data types, 179
EXPORT_SET() function, 792
expr BETWEEN min AND max operator, 770-771
expr IN (value1,value2,...), 772
expr IS operator, 772
expr NOT BETWEEN min AND max operator, 770-771
expr NOT IN (value1,value2,...) operator, 772
expressions, 239-240
  NULL values, 246-247
  operators, 241-243
    arithmetic, 241
    bit, 242
    comparison, 243
    logical, 241-242
    precedence, 246
  pattern matching, 243-245
    LIKE operator, 243-244
    REGEXP operator, 244
type conversions, 247-251
  ASCII, 254
F

FEDERATED storage engine, 108, 113
fetch() function
  arguments, 502-503
  example, 501
  Perl DBI, Web: 1143
  PDO, Web: 1168-1169
  PHP data-retrieval script, 498
FETCH statements, 992
fetchAll() function, 504, Web: 1169
fetchAll_arrayref() function, 415, Web: 1144
fetchAll_hashref() function, Web: 1144
fetchColumn() function, 491, Web: 1169
fetchObject() function, Web: 1169
fetchrow_array() function, 401, 408-409, Web: 1144
fetchrow_arrayref() function, 409-410, Web: 1145
fetchrow_hashref() function, 410-411
FIELD() function, 779
FIELDS clause, 943-944
FILE privilege, 662, 674-675
files
  data, loading, 52-53
  data directory, 545
  .frm defined, 548
  MEMORY tables, 548
  MyISAM tables, 548
  views, 549
  include, 491-497
InnoDB tablespace, 595-598
  adding, 599
  auto-extend increments, 596
  file specification syntax, 596
  pathnames, 596
  raw partitions, 597-598
  regular files, 597
  startup failure, troubleshooting, 598
  system variables, 595
  Windows, 598
log. See logs
Makefiles, 322-323
master.info, 730
MYISAM table, 548
names
  case sensitivity, 100
  identifier constraints, 550
  option, 1008
  connection parameters, reading, 424
  logging, enabling, 619
  mysql utility connection parameters, 87-88
  mysqld startup, 578
  plugins, loading, 591
  reading, 332-335
  securing, 653-654
  SSL, 697
  system variables, setting, 586
  Unix, 1007
  utility, 1007-1011
  Web script security, 470-471
  Windows, 424-425, 1008
PID, 555
retrieving images and storing in tables, 367-368
sampdb distribution, 735-736
source
   connect1.c, 323-324
   connect2, 344-347
   show_opt, 336-338
SSL status, 695-696
   listing of, 554
   multiple servers, 634
   relocating, 561-562
statements, storing, 29
table-specific, 109-110
TRG, 549
TRN, 549
Unix socket, securing, 652-653
filesystem security, 540
FIND_IN_SET() function, 793
finish() function, 402, Web: 1145
fixed-length string types, 205
fixed-name logs, rotating, 626-629
fixed-point types, 751
flip_flop.pl script, 467-468
FLOAT data type, 193
   ranges, 197
   storage requirements, 197
FLOAT[(M,D)] type, 752
FLOAT(p) type, 751
floating-point data types, 197, 200, 751-752
   FLOAT[(M,D)], 752
   FLOAT(p), 751
FLOOR() function, 253, 786
flush_commands status variable, 882
FLUSH PRIVILEGES statement, 583
FLUSH statement, 936-937
flush system variable, 842
FLUSH TABLES statement, 302, 703
flush_time system variable, 842
flushing logs, 626
footers, 495-497
forcing type conversions, 253-255
foreign_key_checks system variable, 842
foreign keys
   absence table example, 49
   benefits, 164
   cascaded deletes
creating, 166-168
testing, 167-168
cascaded updates
   creating, 166-168
testing, 168
defining in child table, 164-165
deletes/updates, 164
displaying, 170
ersors, displaying, 170
guidelines, 166
insertion, verifying, 167
null values, 168-170
parent/child values, 164
referential integrity, 164
row entries, 164
score table example, 48
unique indexes, creating, 169
FORMAT() function, 793
format_entry() function, 455
formats
   binary logs, 731
   row storage
      displaying/editing, 300
      InnoDB, 299-300
      MEMORY, 299
      MyISAM, 299
forms
   hidden fields, creating, 525-526
text input fields, 530
FOUND_ROWS() function, 830
.frm files
   defined, 548
   MEMORY tables, 548
   MyISAM tables, 548
   views, 549
FROM clause
   SELECT statements, 54-56
   subqueries, 149
FROM_BASE64() function, 793
FROM_DAYS() function, 809
FROM_UNIXTIME() function, 809
ft_boolean_syntax system variable, 842
ft_max_word_len system variable, 842
ft_min_word_len system variable, 843
ft_queryExpansion_limit system variable, 843
ft_stophword_file system variable, 843
full-text searches
    boolean mode, 174-175
    characteristics, 171
    configuring, 176-177
    natural language, 172-174
    query expansion, 175-176
    types, 170
FULLTEXT indexes, 124, 126
    configuring, 176-177
    creating, 171-172
    Web table searches, 482-483
func() function, Web: 1147-1148
functions
    add_new_event(), 517-518
    advisory locking, 824-826
    ASCII(), 254, 790
    AVG(), 76, 818
    bail_out(), 405
    BENCHMARK(), 829
    BIN(), 200, 790
    bind_col(), 421
    bindColumn(), 503
    bind_columns(), 421
    BIT_COUNT(), 829
    BIT_LENGTH(), 829
C API
    administrative, Web: 1125-1126
    client library initialization/termination, Web: 1088-1089
    connection management, listing of, Web: 1089-1100
    debugging, Web: 1127
    error-reporting, Web: 1101
    information, Web: 1113-1116
    multiple result sets, Web: 1113
    parameter names, Web: 1087-1088
    prepared statement construction/execution, Web: 1118-1120
    prepared statement error-reporting, Web: 1117-1118
    prepared statement result set processing, Web: 1120-1125
    prepared statements, Web: 1116-1117
    result sets processing, listing of, Web: 1104-1113
    statement construction/execution, Web: 1102-1104
    threaded clients, Web: 1126-1127
    transaction control, Web: 1116
    cast, 783-784
    CAST(), 253, 783-784
    CGI.module, 462
    CGI.pm
        HTML structures, 461
        HTML/URL text, escaping, 464-465
        importing, 461
        object-oriented interface, 461-462
        output, 462-464
    CHAR(), 254, 790
    check_pass(), 533
    check_response(), 527
    col_prompt(), 451
    COLLATION(), 255, 787
    comparison, 781-783
    compression, 821-824
    CONCAT(), 248, 253, 791
    connect()  connection parameters, 423
        Perl DBI scripts, 399-400
    CONNECTION_ID(), 829
    CONNECTION_USER(), 830
    CONVERT(), 187-188, 254-255, 784
    COUNT(), 819
        GROUP BY clause, 74-76
        ROLLUP clause, 77
        WITH ROLLUP clause, 77-78
        summaries, 72-76
        WHERE clause, 72
    CURDATE(), 68, 803
    CURRENT_USER(), 276, 816
    DATABASE(), 830
    date and time, listing of, 802-821
    DATE_ADD(), 68, 254, 804-805
    DATE_SUB(), 69, 807
    DAYOFMONTH(), 67, 808
    DEFAULT(), 830
    display_cell(), 516
    display_column(), 535
    display_entry(), 531-533
    display_events()  Perl DBI, 476-477
        PHP, 514-515
    display_form(), 525-526
    display_login_form(), 530
    display_login_page(), 530
    display_scores(), 477-479, 518-519
display_table_contents(), 473-475
display_table_names(), 472-473
do(), 406-407
edit_member(), 452
enter_scores(), 520-521
errorCode(), 508
errorInfo(), 508
escapeHTML(), 464-465
exec()
    prepared statements, 505
    row-modifying statements, 501
expressions, 240
fetch()
    arguments, 502-503
    example, 501
    PHP data-retrieval script, 498
fetchAll(), 504
fetchall_arrayref(), 415
fetchColumn(), 491
fetchrow_array(), 401
finish(), 402
FLOOR(), 253, 786
format, 763
format_entry(), 455
FOUND_ROWS(), 830
getCode(), 508
getMessage(), 508
handle_options(), 342
header(), 463
HEX(), 201, 253
hidden_field(), 526
html_begin(), 495-497
html_end(), 495-497
html_format_entry(), 456, 481
htmlspecialchars(), 498
insert_rows(), 381-385
interpret_argument(), 445
IP address, 826-828
is_null(), 504
LAST_INSERT_ID(), 237-239, 831
li(), 473
load_defaults()
    defined, 332
    security, 335
    show_argv program example, 332-333
LOAD_FILE(), 831
load_image(), 367
MASTER_POS_WAIT(), 831
MAX(), 76, 820
MIN(), 76, 820
MONTH(), 67, 811
MONTHNAME(), 67, 811
my_init(), 326
mysql_affected_rows(), 350
mysql_close(), 325
mysql_errno(), 328
mysql_error(), 328
mysql_fetch_row(), 351-352
mysql_free_result(), 351
mysql_init(), 325
mysql_library_end(), 326
mysql_library_init(), 326
mysql_more_results(), 375
mysql_next_result(), 375
mysql_query(), 349
mysql_real_connect(), 325, 375
mysql_real_escape_string(), 365
mysql_real_query(), 349
mysql_real_connect(), 325, 375
mysql_real_escape_string(), 365
mysql_real_query(), 349
mysql_set_server_option(), 375
mysql_sqlstate(), 328
mysql_stmt_close(), 388
mysql_stmt_fetch(), 388
mysql_stmt_free_result(), 388
mysql_stmt_init(), 380
mysql_store_result(), 351, 357-359
mysql_use_result(), 351, 357-359
NAME_CONST(), 831
names
    case sensitivity, 99
    identifiers, 98
new PDO(), 490
notify_member(), 446
numeric, 784-789
OCT(), 201, 797
ORDER BY RAND(), 523
param(), 462
parentheses, 401
password_field(), 531
PDO, Web: 1159
    constants, Web: 1173-1174
    security, Web: 1172-1173
    show_argv program example, 332-333
PDO class, Web: 1159-1166
    statement handles, Web: 1166-1172
show_argv program example, 332-333
Perl DBI
administrative, Web: 1147-1148
%attr hash argument, Web: 1130
calling sequence, Web: 1130
database-handle, Web: 1137-1142
DBI class, Web: 1132-1136
general handle, Web: 1146
statement-handle, Web: 1142-1145
utility, Web: 1148-1149
prepare(), 505
present_question(), 525
print_dashes(), 362
print_error(), 329-330
PROCEDURE ANALYSE(), 297
process_call_result(), 392
process_multi_statement(), 376
process_real_statement(), 356-357
process_result_set() function, 352-353
process_statement(), 355
prompt(), 451
query(), 491
quote(), 418-419, 505-506
radio_button(), 526
read_file(), 445
remove_backslashes(), 512
ROUND(), 253, 788
ROW_COUNT(), 831-832
rowCount(), 505
row-fetching
fetchrow_array(), 408-409
fetchrow_arrayref(), 409-410
fetchrow_hashref(), 410-411
listing of, 407
SCHEMA(), 832
script_name(), 516
script_param(), 512
search_members()
ushl_browse.pl script, 480
ushl_ft_browse.pl, 482-483
security, 821-824
select_rows(), 385-388
selectrow_array(), 413
SESSION_USER(), 832
SLEEP(), 832
solicit_event_info(), 516-517
spatial, 828

start_html(), 463
stored, 268-271
creating, 268
defined, 268
integer-valued parameter representing
a year example, 268
multiple values, 269
names, 269
privileges, 270-271
security, 276
tables, updating, 270
STR_TO_DATE(), 66, 813
string, listing of, 789-802
submit_button(), 526
SUM(), 76, 820
summary, listing of, 817-821
text, 29, 764, 780
SYSTEM_USER(), 832
table(), 475
td(), 475
text_field(), 481
th(), 475
TO_DAYS(), 68, 815
TRACE(), 428
undef argument, 422
USER(), 832
UUID(), 832
UUID_SHORT(), 833
VALUES(), 833
VERSION(), 833
XML, 828

G

gen_dir.pl script
entry-fetching loop, 439
format selection code, 438-439
HTML format, 456-458
switchbox, 437-438
general_log system variable, 621, 843
general_log_file system variable, 621, 772
general-purpose statement handlers, 354-355
general query logs, 556, 618, 621
GET_BOOL var_type, 340
GET DIAGNOSTICS statements, 992-994
GET.Disabled var_type, 340
GET_Double var_type, 340
GET.Enum var_type, 340
GET_FORMAT() function, 809-810
get_info() function, Web: 1138
GET_INT var_type, 340
GET_LL var_type, 340
GET_LOCK() function, 825
GET_LONG var_type, 340
GET_NO_ARG var_type, 340
GET_SET var_type, 340
GET_STR_ALLOC var_type, 340
GET_STR var_type, 340
GET_UINT var_type, 340
GET_ULL var_type, 340
GET_ULONG var_type, 340
getAttribute() function, Web: 1163, Web: 1170
getAvailableDrivers function, Web: 1163
getCode() function, 508
getColumnMeta() function, Web: 1170
getCodeMessage() function, 508
global attributes, 747
global privileges, 666
GLOBAL qualifier
SHOW STATUS statement, 589
SHOW VARIABLES statement, 586
global variables, 97
globalization
default character set/collation, 603-604
error message language, 604
internationalization, 601
locale, 604-605
localization, 601
time zones, configuring, 602-603
grade_event table
creating, 40, 47
linking with score table
dates, 41
event IDs, 41-42
grade-keeping project, 17
above-average scores for a grade event,
finding, 145
absences
finding, 145
summarizing, 82-83
table, creating, 45, 49
grade_event table, 40, 47
gradebook example, 39
incorrectly entered grades, swapping,
160-161
linking tables, 41
missing tests/quizzes for students,
finding, 141-143
online score-entry application, 510-511
action input parameter, 513
editing scores, 520-522
event table cells, generating, 516
events, displaying, 514-515
framework, 513-514
hyperlink URLs, 516
new event entry form, 516-517
scores, entering, 517-518
scores for selected events, displaying,
518-519
security, 522
transactional data-entry operations,
520
perfect attendance, 84, 145
quiz/test scores for given date,
retrieving, 78-81
rows, adding
from files, 52-53
INSERT statement, 50-52
scores
browser, creating, 475-479
retrieving, 43-44
table, creating, 39-40, 48-49
total score per student at end of
semester, 82
student table, creating, 44-47
tables, linking, 41-42
test/quiz statistics view, 264-265
GRANT OPTION privilege, 662, 669-670, 674
GRANT statements, 938-943
clauses, 660-661
ON, 939-940
REQUIRE, 668-669, 941
WITH, 941-942
examples, 942-943
privileges, revoking, 672
privileges to be granted, 938-939
selecting, 661
grant tables
account-management statements
affected, 654-655
accounts, 564-569
available on all platforms, 566
client program connections, 566
displaying, 565
passwords, assigning, 567-569
platform specific, 567
administrative privilege columns, 680
columns
  authentication, 684
  privilege, 683-684
  resource management, 685-686
  scope. See scope columns
SSL-related, 685
initializing, 740-741
listing of, 680
object privileges, 681-682
privilege tables, 683
source, 566
statement access verification, 689-690
upgrading, 655
user tables
  authentication, 680
  row matching example, 691-694
GRANT USAGE statement, 697
greater than (>) operator, 57, 243, 770
greater than or equal to (>=) operator, 57, 243, 770
GREATEST() function, 779
GROUP BY clause, 74-76
GROUP_CONCAT() function, 819-820
group_concat_max_len system variable, 843
  groups
    operators, 765-766
    option, 578
    values, counting, 74-76
H
handle_options() function, 342
HANDLER statement, 943
handler_commit status variable, 882
handler_delete status variable, 883
handler_external_lock status variable, 883
handler_mrr_init status variable, 883
handler_prepare status variable, 883
handler_read_first status variable, 883
handler_read_key status variable, 883
handler_read_last status variable, 883
handler_read_next status variable, 883
handler_read_prev status variable, 883
handler_read_rnd status variable, 883
handler_read_rnd_next status variable, 883
handler_rollback status variable, 883
handler_savepoint status variable, 883
handler_savepoint_rollback status variable, 884
handler_update status variable, 884
handler_write status variable, 884
handles, 397
database
  attributes, Web: 1149
  functions, Web: 1137-1142
MySQL-specific attributes,
  Web: 1150-1152
PDO attributes, Web: 1173-1174
general
  attributes, Web: 1149-1150
  functions, Web: 1146
names, 397
PDOStatement, 501
statements
  attributes, Web: 1152-1153
  functions, Web: 1142-1145
MySQL-specific attributes,
  Web: 1154-1155
PDO functions, Web: 1166-1172
HASH indexes, 124-125
have_compress system variable, 843
have_crypt system variable, 843
have_dynamic_loading system variable, 843
have_geometry system variable, 843
have_openssl system variable, 844
have_query_cache system variable, 844
have_rtree_keys system variable, 844
have_ssl system variable, 844
have_symlink system variable, 844
header() function, 463
headers
  connect1 client program, 324
    Content-Type:, 463
    html_begin() function, 495-497
height information, storing, 257-258
hello world script examples, 487-488
help messages, displaying, 1000-1001
HEX() function, 201, 253, 793
hexadecimal notation
  conversions, 248-249
  strings, 184
hidden_field() function, 526
hidden fields (forms)
  creating, 525-526
  security, 528
HIGH_NOT_PRECEDENCE, 863
host access
  limited, 657
  matching host values to DNS, 658-659
  single, 657
  unlimited, 657
host_cache_size system variable, 844
host columns, 687-688
hostname system variable, 844
HOUR() function, 810
HTML
  escaping, 464-465
  structure, 455-456
  tables, creating, 475
  XHTML, compared, 464
html_begin() function, 495-497
html_end() function, 495-497
html_format_entry() function, 456, 481
htmlspecialchars() function, 498
hyperlinks, creating, 499-500
id (my_option structures), 339
IDENTIFIED WITH clause, 676
identifiers, 97
  aliases, 98
  columns, 99
  constraints
    MySQL, 550
    operating systems, 550-551
database, 98
function names, 98
length, 98
qualified names, 99
qualifiers, 98
quoting, 97-98
tables, 98
unquoted, 97
views, 98
identity system variable, 844
IF() function, 779
IF statements, 988
IF NOT EXISTS clause, 106
IFNULL() function, 779
ignore_builtin_innodb system variable, 870
ignore_db_dirs system variable, 844
IGNORE_SPACE, 863
images, retrieving from files and storing in tables, 367-368
implicit data types, 179
importing CGI.pm functions, 461
improper values, handling, 228-230
division by zero errors, 229
strict mode
  turning on, 230
  weakening, 230
transactional/nontransactional tables, 229
warnings, 229
zero date errors, 229
IN operator, 59, 243
IN subqueries, 145-146
include files (PHP)
  benefits, 491-493
  Historical League example, 495
  locations, establishing, 493-504
  referencing, 494
increasing number sequences, creating, 237-238
INDEX privilege, 664
indexes, 278
  benefits, 278-281
    multiple tables, 280
    single-table queries, 279
  binary logs, 623
  BLOB/TEXT data types, 207
  case sensitivity, 100
columns, selecting, 281-285
  badly performing queries, identifying, 283
cardinality, 282
  comparisons, matching to index types, 284-285
  overindexing, 284
  prefixes, 283-284
  short values, 283
  composite, 233
innodb_buffer_pool_dump_now system variable, 871
innodb_buffer_pool_dump_status status variable, 888
innodb_buffer_pool_filename system variable, 871
innodb_buffer_pool_instances system variable, 871
innodb_buffer_pool_load_abort system variable, 871
innodb_buffer_pool_load_at_startup system variable, 871
innodb_buffer_pool_load_now system variable, 888
innodb_buffer_pool_load_status status variable, 888
innodb_buffer_pool_pages_data status variable, 888
innodb_buffer_pool_pages_dirty status variable, 888
innodb_buffer_pool_pages_flushed status variable, 888
innodb_buffer_pool_pages_free status variable, 888
innodb_buffer_pool_pages_latched status variable, 888
innodb_buffer_pool_pages_misc status variable, 888
innodb_buffer_pool_pages_total status variable, 889
innodb_buffer_pool_read_ahead status variable, 889
innodb_buffer_pool_read_ahead_evicted status variable, 889
innodb_buffer_pool_read_requests status variable, 889
innodb_buffer_pool_reads status variable, 889
innodb_buffer_pool_size system variable, 600, 872
innodb_buffer_pool_wait_free status variable, 889
innodb_buffer_pool_write_free status variable, 889
innodb_change_buffer_max_requests status variable, 889
innodb_change_buffer_max_size system variable, 872
innodb_change_buffering system variable, 872
innodb_checksum_algorithm system variable, 872
innodb_checksums system variable, 873
innodb_commit_concurrency system variable, 873
innodb_concurrency_tickets system variable, 873
innodb_data_file_path system variable, 595, 873
innodb_data_fsyncs status variable, 889
innodb_data_home_dir system variable, 595, 873
innodb_data_pending_fsyncs status variable, 889
innodb_data_pending_reads status variable, 889
innodb_data_pending_writes status variable, 889
innodb_data_read status variable, 889
innodb_data_reads status variable, 889
innodb_data_writes status variable, 889
innodb_data_written status variable, 890
innodb_dblwr_pages_written status variable, 890
innodb_dblwr_writes status variable, 890
innodb_doublewrite system variable, 873
innodb_fast_shutdown system variable, 873
innodb_file_format_check system variable, 873
innodb_file_format_max system variable, 874
innodb_file_format system variable, 873
innodb_file_io_threads system variable, 874
innodb_file_per_table system variable, 111, 599, 874
innodb_flush_log_at_trx_commit system variable, 874
innodb_flush_method system variable, 874
innodb_flush_neighbors system variable, 874
innodb_force_load_corrupted system variable, 875
innodb_force_recovery system variable, 875
innodb_ft_xxx system variable, 875
innodb_have_atomic_builtins status variable, 890
innodb_iocapacity system variable, 875
innodb_iocapacity_max system variable, 875
innodb_large_prefix system variable, 875
innodb_lock_wait-timeout system variable, 875
innodb_locks_unsafe_for_binlog system variable, 875
innodb_log_buffer_size system variable, 600, 876
innodb_log_file_size system variable, 601, 876
innodb_log_files_in_group system variable, 601, 876
innodb_log_group_home_dir system variable, 601, 876
innodb_log_waits status variable, 890
innodb_log_write_requests status variable, 890
innodb_log_writes status variable, 890
innodb_lru_scan_depth system variable, 876
innodb_max_dirty_ages_pct_lwm system variable, 876
innodb_max_dirty_pages_pct system variable, 876
innodb_max_purge_lag system variable, 876
innodb_max_purge_lag_delay system variable, 876
innodb_mirrored_log_groups system variable, 877
innodb_monitor_disable system variable, 877
innodb_monitor_enable system variable, 877
innodb_monitor_reset system variable, 877
innodb_monitor_reset_all system variable, 877
innodb_num_open_files status variable, 890
innodb_old_blocks_pct system variable, 877
innodb_old_blocks_time system variable, 877
innodb_open_files system variable, 877
innodb_os_log_fsyncs status variable, 890
innodb_os_log_pending_fsyncs status variable, 890
innodb_os_log_pending_writes status variable, 890
innodb_os_log_written status variable, 890
innodb_page_size system variable, 877
innodb_pages_created status variable, 890
innodb_pages_read status variable, 890
innodb_pages_written status variable, 891
innodb_print_all_deadlocks system variable, 877
innodb_purge_batch_size system variable, 877
innodb_purge_threads system variable, 877
innodb_random_read_ahead system variable, 878
innodb_read_ahead_threshold system variable, 878
innodb_read_io_threads system variable, 878
innodb_replication_delay system variable, 878
innodb_rollback_on_timeout system variable, 891
innodb_rollback_segments system variable, 878
innodb_row_lock_current_waits status variable, 891
innodb_row_lock_current_max status variable, 891
innodb_row_lock_waits status variable, 891
innodb_rows_deleted status variable, 891
innodb_rows_inserted status variable, 891
innodb_rows_read status variable, 891
innodb_rows_updated status variable, 891
innodb_sort_buffer_size system variable, 878
innodb_spin_wait_delay system variable, 878
innodb_stats_method system variable, 878
innodb_stats_on_metadata system variable, 879
innodb_stats_persistent_sample_pages system variable, 879
innodb_stats_sample_pages system variable, 879
innodb_stats_transient_sample_pages system variable, 879
innodb_strict_mode system variable, 879
innodb_support_xa system variable, 879
innodb_sync_spin_loops system variable, 879
innodb_table_locks system variable, 879
innodb_thread_concurrency system variable, 880
innodb_thread_sleep_delay system variable, 880
innodb_truncated_status_writes status variable, 891
innodb_undo_directory system variable, 880
innodb_undo_logs system variable, 880
innodb_undo_tablespaces system variable, 880
innodb_version system variable, 880
innodb_write_io_threads system variable, 880
innodb_xxx status variable, 884
input editing commands, 90-91
input line editing, 90-91
input parameters
  CGI.pm function, 462
  PHP, 511-512
INSERT() function, 794
INSERT privilege, 664
INSERT statement, 943-946
data loading, 301
double-quoting strings in Perl DBI, 417-418
  rows, adding, 50-52
insert_id system variable, 845
insert_rows() function, 381-385
INSTALL PLUGIN statement, 591, 946
install_driver() function, Web: 1136
installed_drivers() function, Web: 1136
installing
MySQL, 737-739
   data directory, initializing, 740-741
   grant tables, initializing, 740-741
   login accounts, creating, 738-739
PATH environment variable,
   configuring, 739-740
   system tables, initializing, 742-743
   Unix, 739
   Windows, 739
PDO, 743
Perl DBI software, 743
PHP, 743-745
INSTR() function, 794
INT data type, 193, 750
   ranges, 197
   storage requirements, 197
integer columns, creating, 46, 48
integer data types, 749-751
   BIGINT, 750-751
   INT, 750
   MEDIUMINT, 750
   SMALLINT, 750
   TINYINT, 749
integer division (DIV) operator, 57
interactive online quizzes, creating, 522-527
   checking user responses, 527
   creating questions, 523-525
   form hidden fields, creating, 525-526
   presenting questions, 525
   user response submissions, 526
interactive statement-execution program, 368-369
interactive_timeout system variable, 845
interfaces
database-access (PHP), 485-486
   plugin
      activation state, 592
      case sensitivity, 591
      components, 590
      displaying plugins, 592
      library suffix, 590
      loading plugins at runtime, 591
loading plugins at startup, 591
   operations, 590
   uninstalling plugins, 592
internal locking, 702-703
   all tables at once, 705
   read-only access, 703-704
   read/write, 704-705
   single sessions, 703
   statements, 703
internal security risks, 645
internationalization
default character set/collation, 603-604
   defined, 601
   error message language, 604
   locale, 604-605
   time zones, configuring, 602-603
interpret_argument() function, 445
INTERVAL() function, 779
inTransaction() function, Web: 1164
introducers, 187
invoker privileges, 276
IP address functions, 826-828
IPv4/IPv6 addresses, 657
IS_FREE_LOCK() function, 826
IS_IPV4() function, 827
IS_IPV4_COMPAT() function, 827
IS_IPV4_MAPPED() function, 827
IS_IPV6() function, 828
IS NOT NULL operator, 243
IS NULL operator, 243
is_null() function, 504
IS_USED_LOCK() function, 826
ISNULL() function, 779
ITERATE() statements, 988
J
join_buffer_size system variable, 845
joins
column references, qualifying, 138-139
   inner, 137-138
   LEFT, 82
   multiple tables example, 78-83
   outer, 139-143
   query optimizer support, 289
   SELECT statements, 955-956
LENGTH() function, 753, 794
less than (<) operator, 57, 243, 770
less than or equal to (<=) operator, 57, 243, 770
like() function, 473
license system variable, 847
LIKE clause
   SHOW statements, 131
   SHOW STATUS statement, 589
   SHOW VARIABLES statement, 585
LIKE/NOT LIKE operators, 243-244, 776-777
LIMIT clause
   db_browse.pl script, 475
   query results, limiting, 63-64
   limiting query results, 63-64
LINES clause, 944
Linux, log rotating, 628
live hyperlinks, creating, 499-500
LN() function, 786
LOAD DATA statement, 300-301, 946-951
   data files, loading, 52-53
   data formats, 948
   FIELDS clause options, 948-949
   LINES clause, 949-950
   LOCAL keyword, 947
   special characters, 948
LOAD INDEX INTO CACHE statement, 951
LOAD XML statement, 951-952
load_defaults() function
   defined, 332
   security, 335
   show_argv program example, 332-333
LOAD_FILE() function, Web: 1164
load_image() function, 367
loading
   data, 300-303
      dropping/deactivating indexes, 302-303
      index flushing, reducing, 301-302
      INSERT statement, 301
      LOAD DATA statement, 300-301
      mixed-query environments, 303
      shorter statements, 302
plugins
  runtime, 591
  startup, 591
LOCAL keyword, 941
local_infile system variable, 847
locale, selecting, 604-605
localization
  default character set/collation, 603-604
  defined, 601
  error message language, 604
  locale, 604-605
  time zones, configuring, 602-603
LOCALTIME() function, 810
LOCALTIMESTAMP() function, 810
LOCATE() function, 794
LOCK TABLES privilege, 664
LOCK TABLES statement, 304, 702, 952-953
lock_wait_timeout system variable, 786, 847
locked_in_memory system variable, 847
locking
  advisory functions, 813-814
  all tables at once, 705
  levels, 303-305
  overview, 702-703
  read-only access, 703-704
  read/write, 704-705
  single sessions, 703
  statements, 703
  tables, 303-305
LOG() function, 783
log system variable, 847
LOG2() function, 787
LOG10() function, 787
log_bin system variable, 847
log_bin_basename system variable, 847
log_bin_index system variable, 847
log_bin_trust_function_creators system variable, 847
log_error system variable, 847
log_output system variable, 848
log_queries_not_using_indexes system variable, 848
log_slave_updates system variable, 848
log_slow_queries system variable, 848
log_throttle_queries_not_using_indexes system variable, 848
log_warnings system variable, 620, 848
logical operators, 772
  AND (&&), 773
  listing of, 57, 241-242
  natural language distinctions, 59
  NOT (!), 772
  NULL values, 247
  OR (||), 773
  XOR, 773
login accounts, creating, 738-739
logrotate utility, 628
logs
  age-based expiration, 625
  binary, 556, 618
    administration, 622-623
    expiring, 629-630
    formats, 731
    index files, 623
    post-backup statements, re-executing, 723-725
    system backups, 623
  enabling, 619
  error, 556, 620-621
    defined, 618
    event scheduler, 274
    levels, selecting, 620
    multiple servers, 634
    Unix, 620
    Windows, 621
  expiring, 629-631
    automating, 630-631
    binary, 629-630
    relay, 630
  fixed-name, rotating, 626-629
  flushing, 626
  general query, 556, 618, 621
    listing of, 554, 617-618
    maintenance, 539
    multiple servers, 634
    output destination, selecting, 624-625
    relay, 618, 624, 630
    relocating, 561-562
    replication-related expiration, 625
    rotating, 625-629
  security, 556
  slow query, 618
tables
  rotating, 625, 631
  truncating, 625, 631
  writing to, 625
long_query_time system variable, 848
LONGBLOB data type, 204, 207-208
LONGBLOB strings, 194, 756
LONGTEXT data type, 204, 207-208
LONGTEXT strings, 194, 758
looks_like_number() function, Web: 1148
LOOP statements, 989
lower_case_file_system system variable, 849
LOWER() function, 795
lower_case_table_names system variable, 100, 849
LPAD() function, 795
LTRIM() function, 795
mailing lists, 80
maintenance
  backups, 707-709
    best practices, 709
    binary, 714-715
    InnoDB, 715-716
    selecting, 708
    slave, creating, 732-733
    storage engine portability, 709-710
    text, 711-714
    types, 708
checking tables
  CHECK TABLE statement, 719-720
  InnoDB tables, 718
  MyISAM tables, 719
  mysqlcheck utility, 720-721
databases
  backing up, 540
  crash recovery, 541
  preventive, 540
logs, 539
preventive
  auto-recovery, 706
databases, 699-700
  scheduling, 707
server cooperation, 700-701
tables, 700
tools, 700
Unix login, 701
repairing tables
  InnoDB tables, 718
  MyISAM tables, 719
mysqlcheck utility, 720-721
REPAIR TABLE statement, 720
replication
  binary logging formats, 731
  compatibility guidelines, 727-728
  master-slave, 728-731
  overview, 727
  slave backups, creating, 732-733
server interference, preventing, 701
  internal locking, 702-703
  locking all tables at once, 705
  read-only locking, 703-704
  read/write locking, 704-705
  shutting down servers, 702
user accounts, 539
MAKE_SET() function, 795
MAKEDATE() function, 810
Makefiles, 322-323
MAKETIME() function, 811
master_info_repository system variable, 849
MASTER_POS_WAIT() function, 831
master-slave replication, 728-731
  master server settings, 728-729
  master.info file, 730
  relay logs, 731
  separate slave accounts, 730
  server ID values, assigning, 728
  slave settings, 729-730
  statements, 730-731
  threads, starting/stopping, 731
master_verify_checksum system variable, 850
master.info file, 730
MATCH() function, 796-797
MATCH operator, full-text searches
  boolean mode, 174-175
  natural language, 172-174
  query expansion, 175-176
MAX() function, 76, 820
max_allowed_packet system variable, 850
max_binlog_cache_size system variable, 850
max_binlog_size system variable, 850
max_binlog_stmt_cache_size, 850
max_connect_errors system variable, 850
max_connections system variable, 850
max_delayed_threads system variable, 850
max_error_count system variable, 851
max_heap_table_size system variable, 851
max_insert_delayed_threads system variable, 851
max_join_size system variable, 851
max_length_for_sort_data system variable, 851
max_prepared_stmt_count system variable, 851
max_relay_log_size system variable, 624, 630, 851
max_seeks_for_key system variable, 852
max_sort_length system variable, 852
max_sp_recursion_depth system variable, 852
max_tmp_tables system variable, 852
max_used_connections status variable, 884
max_user_connections system variable, 852
max_value member (my_option structures), 341
max_write_lock_count system variable, 852
maximum value summaries, 76
MD5() function, 823
MEDIUMBLOB data type, 204, 207-208
MEDIUMBLOB strings, 194, 756
MEDIUMINT data type, 193, 750
ranges, 197
storage requirements, 197
MEDIUMTEXT data type, 204, 207-208
MEDIUMTEXT strings, 194
member_id columns, creating, 36
member table, 33
creating, 35-38
expiration column, 36
member_id column, 36
membership
list tables, creating, 33
renewal notifications, sending, 443-448
tables, creating, 35-38
MEMORY storage engine, 108
data, representing, 548
locking levels, 304
overview, 111-112
portability, 710
row storage formats, 299
sequence characteristics, 234-235
MERGE storage engine, 108, 113, 304
metadata
accessing, 130
command line, 135
INFORMATION_SCHEMA database, 132-135
SHOW statement, 130-132
result sets
C client programs, 359-364
availability, 359
column information structures, accessing, 364
defined, 359
displaying, 360-364
result set data processing decisions, 359
metadata, displaying, 364
Perl DBI scripts, 430-434
metadata_locks_cache_size system variable, 852
methods. See functions
MICROSECOND() function, 811
MID() function, 797
migrating databases, 541
MIN() function, 76, 820
min_examined_row_limit system variable, 852
min_value member (my_option structures), 341
minimum value summaries, 76
MINUTE() function, 811
mixed format logging, 731
MOD() function, 787
MODIFY clause, 128
modules (CGI.pm), 459-460
HTML
structures, 461
text, escaping, 464-465
XHTML, compared, 464
importing functions, 461
input parameters, 462
multiple-purpose pages, writing, 465-468
object-oriented, 461-462
output, generating, 462-464
portability, 463
URL text, escaping, 464-465
modulo (%) operator, 57, 241, 767
MONTH() function, 67, 811
MONTHNAME() function, 67, 811
multiple-client environments, 156
multiple-data retrieval
  joins, 138-139
  subqueries, 144
multiple-line SQL statements, 28
multiple-purpose pages, writing, 465-468
multiple servers, 632
  administration, 539
  client programs, running, 641
  configuring, 635
  error log file names, 634
  InnoDB log location, 634
  issues, 632-635
  new servers, passwords, 569
  options
    directory, 633
    login accounts, 635
    network interface, 633
    replication slaves, 634
    startup, 636-637
    status/log file names, 634
  Unix, 637-639
  Windows, 639-641
multiple-statement execution, 375-377
  enabling, 375
  process_multi_statement() function, 376-377
  result retrieval functions, 375
multiple-tables
  deletes, 154-155
  queries, 78-84
  retrievals
    joins. See joins
    subqueries. See subqueries
    UNION statements, 151-154
  updates, 155-156
multiple-user access benefit, 13
multiplication (*) operator, 57, 241, 767
my_init() function, 326
my_option structures, 339
  app_type, 341
  arg_type, 341
  block_size, 341
  comment, 340
  def_value, 341
  id, 339
  max_value, 341
  min_value, 341
  name, 339
  sub_size, 341
  typelib, 340
  u_max_value, 340
  value, 340
  var_type, 340-341
my_print_defaults program, 1011
MyISAM storage engine, 108
  auto-recovery, 706
  checking/repairing tables, 719
  data, representing, 548
  features, 111
  locking levels, 304
  portability, 710
  row storage formats, 299
  sequence characteristics, 232-234
  table maximum size, 552
myisam_data_pointer_size system variable, 852
myisam_max_sort_file_size system variable, 853
myisam_mmap_size system variable, 853
myisam_recover_options system variable, 853
myisam_repair_threads system variable, 853
myisam_sort_buffer_size system variable, 853
myisam_stats_method system variable, 853
myisam_use_mmap system variable, 853
myisamchk utility
  defined, 538
  maintenance advantages, 719
  options
    specific to myisamchk, 1015-1018
    standard, 1014
  overview, 1013-1014
  table maintenance, 700
  variables, 1018-1019
MySQL
  benefits, 11-12
    availability, 14
    capabilities, 14
    client/server architecture, 22
    connectivity, 22
    cost, 17
    easy, 16
    flexible output format, 13
    flexible retrieval order, 13
multiple-user access, 13
open distribution/source code, 21
portability, 16
query language support, 20
record filing time reduction, 13
record retrieval time reduction, 13
remote access, 13
security, 22
speed, 13
Web-based inventory searches, 14
installing, 737-738, 739
data directory, initializing, 740-741
grant tables, initializing, 740-741
login account, creating, 738-739
PATH environment variable, configuring, 739-740
system tables, initializing, 742-743
Unix, 739
Windows, 739
mailing lists, 80, 642
needs scenarios, 12
pronunciation, 22
reference manual website, 7
server. See mysqld
software, updating, 539
Workbench website, 21
mysql database privileges, 673-674
MYSQL structure, Web: 1076
mysql utility, 21
commands, 1026-1028
connections, 87
option files, 87-88
shell aliases/scripts, 89
shell command history, 88
databases, resetting, 53
defined, 538
invoking, 25-26
options
specific to mysql, listing of,
1021-1025
standard, 1021
overview, 1019-1021
prompt definition sequences, 1028-1029

function syntax, 29
multiple-lines, 28
multiple statements on single line, 28-29
reading from files, 29
results, displaying, 28
table structure, displaying, 36-37
typing less, 90-93
typing tips
copy/paste, 92
script files, 92-93
variables, 1025-1026

MySQL Workbench program, 21
mysql_affected_rows() function, Web: 1104
mysql_autocommit() function, Web: 1116
MYSQL_BIND arrays
configuring, 384
select_rows() function, 385-388

MYSQL_BIND data structure, Web: 1082-1086
input values, Web: 1085
member purpose, Web: 1083-1084
output values, Web: 1083-1085
public members, Web: 1082-1083
mysql_change_user() function, Web: 1090
mysql_character_set_name() function, Web: 1113

mysql_close() function, 325, Web: 1090
mysql_commit() function, Web: 1116
mysql_config utility
defined, 1030
options, 1030-1031
mysql_data_seek() function, Web: 1106
mysql_debug() function, Web: 1127
mysql_dump_debug_info() function, Web: 1127
mysql_errno() function, 328, Web: 1101
mysql_error() function, 328, Web: 1101
mysql_fetch_field() function, Web: 1106
mysql_fetch_field_direct() function, Web: 1107
mysql_fetch_fields() function, Web: 1106-1107
mysql_fetch_lengths() function, Web: 1107-1108
mysql_fetch_row() function, 351-352, Web: 1108

MYSQL_FIELD data structure, Web: 1076-1080
mysql_field_count() function, Web: 1110
mysql_field_seek() function, Web: 1109
mysql_free_result() function, 351, Web: 1110
**mysql_warning_count() function, Web: 1116**

**mysqldadmin utility, 1034**
- commands, 1035-1037
- defined, 538
- options
  - specific to mysqldadmin, 1034-1035
  - standard, 1034
- variables, 1013-1035

**mysqlbinlog utility, 622**
- options
  - specific to mysqlbinlog, 1038-1041
  - standard, 1038
- overview, 1038
- variables, 1041

**mysqldcheck utility**
- checking/repairing tables, 720-721
- defined, 538
- maintenance, scheduling, 707
- options
  - specific to mysqlcheck, 1042-1044
  - standard, 1041-1042
  - table analysis, 1044
  - table checking, 1044
  - table optimization, 1044-1045
  - table repair, 1044
- overview, 1041
- table maintenance, 700

**mysqld, 21**
- administration
  - configuration and tuning, 539
  - log maintenance, 539
  - multiple servers, 539
  - MySQL software updates, 539
  - startup/shutdown, 539
  - user account maintenance, 539
- client access control, 686-687
- connections, listening, 579-580
- data directory access, 546-547
- defined, 538
- login accounts, 571-572
- maintenance interference, preventing, 701
  - internal locking, 702-703
  - locking all tables at once, 705
  - read-only locking, 703-704
  - read/write locking, 704-705
  - shutting down mysqld, 702
- options
  - replication, 1053-1056
  - specific to mysqld, listing of, 1046-1053
    - standard, 1045-1046
    - Windows, 1053
  - overview, 1045
  - restarting manually, 581-582
  - root password, resetting, 582-583
  - security, 540
  - starting, 741
    - Unix, 741
    - Windows, 742
  - startup options, 577-579
  - stopping, 580-581
  - Unix
    - connections, listening, 579
    - running, 570
    - starting, 572-574
    - unprivileged login account, configuring, 571-572
  - variables, 1056
  - Windows, 575
    - connections, listening, 580
    - running as Windows service, 576-577
    - running manually, 575

**mysqld_multi script, 637-639, 1056**
- specific to mysqld_multi option, 1057
- standard options, 1056-1057

**mysqld_safe, 1058**
- specific to mysqld_safe option, 1058-1059
- standard options, 1058

**mysqldump utility, 21, 135**
- data format options, 1067-1068
- database maintenance, 700
- defined, 538
- options, 712-714
  - specific to mysqldump, 1061-1067
    - standard, 1060
- overview, 1060
- text dump files
  - all tables from all databases, 711
  - compressing, 712
  - creating, 711-714
  - individual files, 711
  - output, 711-712
table subsets into separate files, creating, 712
variables, 1068

mysqldumpslow utility, 621
mysql utility
  data files, loading, 53
  options
     data format, 1070
     specific to mysqlimport, 1069-1070
     standard, 1068
  overview, 1068
mysql.server utility, 1029-1030
mysqlshow utility, 38, 135
  options
     specific, 1071
     standard, 1071
  overview, 1070-1071

mytbl.frm file, 548
mytbl.MYD file, 548
mytbl.MYI file, 548

NAME_CONST() function, 831
named_pipe system variable, 853

names
  aliases
     case sensitivity, 100
     quoting with identifiers, 98
  case sensitivity
     aliases, 100
     columns, 100
     databases, 100
     files, 100
     functions, 99
     indexes, 100
     stored programs, 100
     tables, 100
     triggers, 100
     views, 100
  collations, 186
  columns, 64-66, 263-264
  data types, 747
  files, 550-551
  functions, 98
  my_option structures, 339
Perl DBI handles, 397
Perl DBI non handle variables, 397
PHP scripts, 486
qualified, 99
stored functions, 269
system variables, 836
  tables, 32
     files, 109
     renaming, 129-130
     temporary, 116
  triggers, 272
user accounts, 656-658
  account value, 656
  hostnames, 656-657
  IPv4/IPv6 addresses, 657
  localhost, 658
  matching host values to DNS, 658-659
  quoting, 658
  usernames, 657
  wildcards, 657
variables, Web: 1131
  Windows file paths, 424-425
natural language searches, 170, 172-174
NDB storage engine, 108, 112
neat() function, Web: 1148
neat_list() function, Web: 1148-1149
need_renewal.pl script, 443-444
negation operator (~), 774
net_buffer_length system variable, 854
net_read_timeout system variable, 854
net_retry_count system variable, 854
net_write_timeout system variable, 854
network interface options (multiple servers), 633
new PDO() function, 490
new system variable, 854
nextRowset() function, Web: 1170-1171
NO_ARG arg_type, 341
NO_AUTO_CREATE_USER, 863
NO_AUTO_VALUE_ON_ZERO, 863
NO_BACKSLASH_ESCAPES, 863
NO_DIR_IN_CREATE, 863
NO_ENGINE_SUBSTITUTION, 864
NO_FIELD_OPTIONS, 864
NO_KEY_OPTIONS, 864
NO_TABLE_OPTIONS, 864
NO_UNSIGNED_SUBTRACTION, 864
null-safe equality operator (\(\iff\)), 769

**NULL values, 60-61, 192**
- AUTO_INCREMENT columns, 231
- column sort, 62
- directory membership updates, 535-536
- expressions, 246-247
- foreign key relationships, 168-170
- numeric data types, 203
- result sets, checking, 416, 504
- sequence columns, 231
- string data types, 216
- temporal data types, 223

**NULLIF() function, 780**

**numbers**
- hexadecimal, 253
- sequences. See sequences
- string conversions, 249-250

**numeric data types, 193, 748-749**
- attributes, 201-203, 749
- BIT, 197, 200-201, 752
- exact-value, 197-199
- fixed-point, 751
- floating-point, 197, 200, 751-752
- DOUBLE, 752
- FLOAT\([(M,D)]\), 752
- FLOAT(p), 751
- improper values, 228
- integer, 749-751
- BIGINT, 750-751
- INT, 750
- MEDIUMINT, 750
- SMALLINT, 750
- TINYINT, 749
- listing of, 193
- NULL/NOT NULL values, 203
- query optimization, 296
- ranges, 197
- selecting, 203, 257-258
- storage requirements, 197-198

**numeric functions, 784-789**

**numeric values, 181**
- approximate, 181-182
- bit-field, 182
- exact, 181-182
- retrieving, 56
object privileges, 663-665, 681-682
OCT() function, 201, 797
OCTET_LENGTH() function, 797
old system variable, 854
old_alter_table system variable, 854
OLD_PASSWORD() function, 823
old_passwords system variable, 854
ON clause, 937-938
ON DELETE CASCADE clause, 166-167
ON DELETE SET NULL clause, 169
ON specifier, 665
ON UPDATE CASCADE clause, 166-167
ON UPDATE SET NULL clause, 169
online score-entry script. See score_entry.php script
ONLY_FULL_GROUP_BY, 864
open_files status variable, 885
open_files_limit system variable, 854
Open Geospatial Consortium Web site, 191
OPEN statements, 992
open_streams status variable, 885
open_table_definitions status variable, 885
open_tables status variable, 885
opened_files status variable, 885
opened_table_definitions status variable, 885
opened_tables status variable, 885
operand conversions, 187
operating systems, identifier constraints, 550-551
operators
IN, 145-146
ALL, 146-147
ANY, 146-147
arithmetic, 57, 766-767
addition, 766
DIV, 767
division, 767
listing of, 241
modulo, 767
multiplication, 767
NULL values, 246
rules, 766
subtraction, 767
bit
AND, 773
exclusive-OR, 773
listing of, 242
negation, 774
NULL values, 246
OR, 773
shift left, 773
shift right, 773
cast, 775-776
comparison, 57, 768-772
CASE [expr] WHEN expr1 THEN result1 ... [ELSE default] END, 771
equal, 769
expr BETWEEN min AND max, 770-771
expr IN (value1,value2,...), 772
expr IS, 772
expr IS NULL/expr IS NOT NULL, 772
expr NOT BETWEEN min AND max, 770-771
expr NOT IN (value1,value2,...), 772
greater than, 770
greater than or equal to, 770
less than, 770
less than or equal to, 770
listing of, 243
NULL values, 247
null-safe equality, 769
rules, 768-769
unequal, 770
EXISTS, 147-148
format, 763
grouping, 765-766
IN(), 59
logical, 772
AND (&&), 774
listing of, 57, 241-242
natural language distinctions, 59
NOT (!), 772
NULL values, 247
OR (||), 773
XOR, 773
MATCH
boolean mode, 174-175
natural language, 172-174
query expansion, 175-176
NOT EXISTS, 147-148
NOT IN, 145-146
row storage formats, 299
displaying/editing, 300
InnoDB, 299-300
MEMORY, 299
MyISAM, 299
scheduling policies, 303
storage engine locking levels, 303-305
OPTIMIZE TABLE statement, 953-954
optimizer_prune_level system variable, 855
optimizer_search_depth system variable, 855
optimizer_switch system variable, 855
optimizer_trace_xxx system variable, 855
option files
connection parameters, reading, 424
logging, enabling, 619
mysql program connection parameters, 87-88
mysqld startup, 578
plugins, loading, 591
reading, 332-335, 424
securing, 653-654
SSL, 697
system variables, setting, 586
Unix, 1007
utility, 1007-1011
escape sequences, 1010
leading spaces, 1010
read directives, 1010-1011
user-specific option privacy, 1011
Web scripts security, 470-471
Windows, 424-425, 1008
options
CHANGE MASTER statement, 907
CHECK TABLE statement, 909
command-line, 335-343
argument vector, processing, 342
option information, defining, 339-341
show_opt, invoking, 342-343
show_opt program source file, 336-338
connect2 program, 344-348
connect1/show_opt programs, compared, 347
connection parameters, specifying, 348
running, 347
source file, 344-347
CREATE TABLE statement, 916-918
FLUSH statement, 933-934
groups, 578
myisamchk utility
   specific to myisamchk, listing of, 1015-1018
   standard, 1014
mysql utility
   specific to mysql, listing of, 1021-1025
   standard, 1021
mysql_config utility, 1030-1031
mysql_install_db script
   specific, 1032
   standard, 1032
mysqladmin client
   specific to mysqladmin, 1034-1035
   standard, 1034
mysqlbinlog
   specific to mysqlbinlog, 1038-1041
   standard, 1038
mysqlcheck
   specific to mysqlcheck, 1042-1044
   standard, 1041-1042
   table analysis, 1044
   table checking, 1044
   table optimization, 1044-1045
   table repair, 1044
mysqld
   replication, 1053-1056
   specific to mysqld, listing of, 1046-1053
   standard, 1045-1046
   Windows, 1053
mysqld_multi
   specific to mysqld_multi, 1057
   standard, 1056-1057
mysqld_safe
   specific to mysqld_safe, 1058-1059
   standard, 1058
mysql_upgrade
   specific options, 1033-1034
   standard, 1033
mysqldump utility, 712-714
   data format, 1067-1068
   specific to mysqldump, 1061-1067
   standard, 1060
mysqlimport
   data format, 1070
   specific, 1069-1070
   standard, 1068
mysql.server utility, 1030
mysqlshow
   specific to mysqlshow, 1071
   standard, 1071
option files, 332-335
perror, 1072
SELECT statements, 954-955
SSL, adding to clients, 370-372
utilities
   case sensitivity, 1001
   checking, 1011
   escape sequences, 1010
   group names, 1009
   leading spaces, 1010
   long-form/short-form, 1001
   option-file processing, 1008-1009
   option files, 1007-1011
   processing features, 1002
   quoting, 1010
   read directives, 1010-1011
   SSL, 1006
   standard, 1003-1005
   user-specific option privacy, 1011
   variables, 1006-1007
values, holding, 372-373
OR operator (||), 57, 241-242, 773
ORD() function, 797
ORDER BY RAND() function, 64, 523
outer joins, 139-143
output
   CGI.pm generating, 462-464
   column values, naming, 64-66
   format flexibility, 13
   query optimizer EXPLAIN statements, 290-296
      efficiency with indexes, 292-296
      expression writing style, selecting, 290-292
overall count of values, counting, 73
overindexing, 284
override parameter, 481
ownership
   administrative-only, setting, 649-651
   base directory, displaying, 650
PDO (PHP Data Objects) extension, 314
classes, Web: 1158
errors
   exceptions, 491
   handling, 507-509
functions, Web: 1159
   constants, Web: 1173-1174
   exceptions, Web: 1172-1173
PDO class, Web: 1159-1166
statement handles, Web: 1166-1172
installing, 743
placeholders, 506-507
statements, handling, 500-501
result sets, 501-504
row-modifying, 501
transaction processing, 520
Web site, 486
PDOStatement statement handle, 501
performance
   APIs, selecting, 315-316
   data directory, 553-554
   optimizing. See optimization queries, identifying, 285
performance_schema_xxx status variable, 885
performance_schema_xxx system variable, 855
PERIOD_ADD() function, 812
PERIOD_DIFF() function, 812
Perl DBI API, 311-312
   architecture, 311-312
   attributes, Web: 1149
      database-handles, Web: 1149
dynamic, Web: 1155
general handle, Web: 1149-1150
mysql-specific database-handle,
   Web: 1150-1152
mysql-specific statement-handle,
   Web: 1154-1155
statement-handles, Web: 1152-1153
database server connections, 312
defined, 310
evironment variables, Web: 1156
functions
   administrative, Web: 1147-1148
   %attr hash argument, Web: 1131
calling sequence, Web: 1131
database-handle, Web: 1137-1142
DBI class, Web: 1132-1136

P

PAD_CHAR_TO_FULL_LENGTH, 865
param() function, 462
parameters
   action, 513
   binding (Perl DBI), 421-423
   connection, specifying
      C client programs, 331
      command-line option-handling, 335-343
      connect2 program, 348
      option files, reading, 332-335
Perl DBI, 423-426
input
   CGI.pm, 462
   PHP, 511-512
mysql_real_connect() function, 325
override, 481
prepared statements, 377-378
tbl_name
   checking, 473
   db_browse.pl script, 472
types (stored procedures), 271-272
parentheses ( ), 401
partial binary backups, 715
PARTITION BY clause, 120-121
partitions, creating, 120-121
PASSWORD() function, 823-824
password_field() function, 531
passwords
   Historical League member entries online editing script, 527-529
   initial user accounts, assigning, 567-569
   new servers, setting, 569
   old hash format security risk, 673
   root accounts, resetting, 582-583
   user accounts, 672
PATH environment variable, 739-740
pattern matching, 69-70, 243-245
   NULL values, 247
   operators, 776-780
      LIKE/NOT LIKE, 243-244, 776-777
      REGEXP/NOT REGEXP pattern, 777-780
      RLIKE/NOT LIKE pattern, 780
Web table searches, 479-482
general handle, Web: 1146
statement-handle, Web: 1142-1145
utility, Web: 1148-1149
portability, 312
scripts, writing. See Perl DBI scripts
variable names, Web: 1131
Web site, 395,
Perl DBI scripts, Web: 1130
case sensitivity, 400
characteristics, 396
comments, adding, 398
connect() function arguments, 399-400
connections, 400, 425-426
parameters, specifying, 423-426
data-retrieval script, 397-398
debugging, 426
print statements, 426-428
tracing, 428-429
disconnecting, 402
error handling, 402-405
automatic, 403-404
default error messages, replacing, 404
default settings, 403
dump_members2.pl script example, 404-405
manually checking/printing, 403
PrintError attribute, 403
RaiseError attribute, 403
finish() function, 402
function parentheses, 401
handles, 397
names, 397
nonhandle variables, 397
invoking, 396
option files, securing, 653-654
parameter binding, 421-423
placeholders, 419-421
prepared statements, 421
quoting special characters, 416-419
requirements, 395
result sets
metadata, 430-434
retrieving. See result sets, Perl DBI scripts
row-fetching loop, 401-402
row-modifying statements, 406-407
software, installing, 743
statement terminators, 401
transactions, 434-436
undef argument, 422
U.S. Historical League
directory, generating, 436-442
member entries, editing, 448-454
membership renewal notices, sending, 443-448
members with common interests, finding, 454-455
online directory, creating, 455-458
use DBI statement, 397
use strict statements, 399
use warnings statement, 399
warnings mode, 401
Web-based, 459
CGI.pm module. See CGI scripts
database browser, 471-475
grade-keeping project score browser, 475-479
security, 470-471
server connections, 468-469
table searches, 479-483
Web server, configuring, 460-461
where-to-find-Perl indicator, 398
Perl modules Web site, 743
perldoc command, 743
permissions
administrative-only, setting, 649-651
base directory, 650
data directory, 650
permission utility
options, 1072
overview, 1072
phantom rows, 162
PHP API
client host that requested the page IP
address, displaying, 313
database interfaces, 314, 485-486
defined, 310
functions, Web: 1159
constants, Web: 1173-1174
exceptions, Web: 1172-1173
PDO class, Web: 1159-1166
statement handles, Web: 1166-1172
installing, 743-745
PDO. See PDO
scripts, writing. See PHP scripts
tag styles, Web: 1157-1158
up-to-the-minute information to visitors script, 313
Web site, 486, 743

**PHP scripts**
data-retrieval, 497-499
display values, encoding, 498
error handling, 498
home page link, creating, 498-499
installing/accessing, 498
result set, returning, 498
directory member entries, editing online, 527-536
editing form, 533-534
framework, 529-530
member login page, 530-531
null values, 535-536
passwords, 527-529, 531-533
updating entries, 534-535
error handling, 507-509
headers/footers functions, 495-497
hello world examples, 487-488
home page, 488-491
include files
benefits, 491-493
Historical League example, 495
locations, establishing, 493-494
referencing, 494
input parameters, 511-512
interactive online quiz, 522-527
checking user responses, 527
creating questions, 523-525
form hidden fields, creating, 525-526
presenting questions, 525
user response submissions, 526
live hyperlinks, creating, 499-500
names, 486
online score-entry, 510-511
action input parameter, 513
editing scores, 520-521
event table cells, generating, 516
events, displaying, 514-515
framework, 513-514
hyperlink URLs, 516
new event entry form, 516-517
scores, entering, 517-518
scores for selected events, displaying, 518-519
transactional data-entry operations, 520
online score-entry application, 522
overview, 487
PDO error exceptions, 491
placeholders, 506-507
prepared statements, 505
quoting special characters, 505-507
row-modifying statements, 501
rows, retrieving, 501-504
all at once, 504
arrays, 503
calculated columns, 503
default fetch mode, setting, 502
individual column values, 503
NULL values, checking, 504
row-fetching loop, 501-503
statement handle, 501
samples, installing, 486-487
security, 491
server connection, 490-491
standalone, 489
statements, handling, 500-501
tag styles supported, Web: 1157-1158
variables, 492
Web site
welcome message with membership count home page, 491

**PI() function**, 787

**PID (Process ID) files**
overview, 555
relocating, 561-562

**pid_file system variable**, 855

**ping() function**, Web: 1138

**PIPS_AS_CONCAT mode**, enabling, 242
**PIPS_AS_CONCAT SQL mode**, 96, 865

**placeholders**
Perl DBI scripts, 419-421
PHP, 506-507
plain text directories, creating, 439-440
pluggable architecture, 589-590
**plugin_dir system variable**, 855

**plugins**
activation state, 592
authentication, 676-679
proxy users, creating, 677-679
server connections, 677
server side/client side, 677
specifying, 676
case sensitivity, 591
displaying, 592
interface
components, 590
library suffix, 590
operations, 590
loading
runtime, 591
startup, 591
uninstalling, 592
port system variable, 855
portability
APIs, selecting, 317
CGI.pm module, 463
Perl DBI API, 312
storage engines, 709-710
POSITION() function, 798
POSIX character class constructions, 778
POW() function, 787
POWER() function, 787
precedence (operators), 246, 764-765
prefixes (indexing), 283-284
preload_buffer_size system variable, 855
prepare() function, 505, Web: 1138, Web: 1164
PREPARE statement, 954
prepare_cached() function, Web: 1138
prepared statements, 377
C client programs
call. See CALL prepared statements
executing, 378-379
inserting rows and retrieving them program, writing, 379-388
parameterizing, 377-378
functions, Web: 1112
construction/execution,
Web: 1113-1114
error-reporting, Web: 1112-1113
result set processing, Web: 1114-1117
Perl DBI, 421
PHP, 505
prepared_stmt_count status variable, 885
pres_quiz.php script, 522-527
checking user responses, 527
creating questions, 523-525
form hidden fields, creating, 525-526
presenting questions, 525
user response submissions, 526
present_question() function, 525
president table, creating, 32, 34-35
preventive maintenance
auto-recovery, 706
backups, 707-709
best practices, 709
binary, 714-715
InnoDB, 715-716
selecting, 708
storage engine portability, 709-710
text, 711-714
types, 708
databases, 699-700
scheduling, 707
server cooperation, 700-701
server interference, preventing, 701
internal locking, 702-703
locking all tables at once, 705
read-only locking, 703-704
read/write locking, 704-705
shutting down servers, 702
tables, 700
tools, 700
Unix login, 701
primary keys
absence table example, 49
converting to unique indexes, 169
defining, 166
score table example, 48
print statements, 427-428
print_dashes() function, 362
print_error() function, 329-330
PrintError attribute, 403
printing binary data, 353
privileges
account administering, enabling, 669-670
administrative, 666
combining, 665
database-level, 666
definer, 276
displaying, 671
events, 274
global, 666
grant tables
  administrative, 682
  object, 682-681
invoker, 276
no privileges, 668
object, 663-665
PROXY, 667
quoting, 667
resource consumption limits, 670-671
revoke, 671-672
secure connections, requiring, 668-669
specifiers
  ALL, 666
  ALL/USAGE, 661
  levels, 665
stored functions/procedures, 270-271
stored routines, 667
super, 276, 673-674
table/column level, 667
triggers, 273
user accounts
  administrative, 661-663
  granting, 660-661
views, 263
privileges clause, 660
PROCEDURE ANALYSE() function, 297
procedures (stored)
  creating, 268
  defined, 268
  example, 269
  invoking, 269
  parameter types, 271-272
  privileges, 270-271
  security, 275-276
  tables, updating, 270
Process ID files. See PID files
PROCESS privilege, 662, 674-675
process_call_result() function, 392
process_multi_statement() function, 376-377
process_real_statement() function, 356-357
process_result_set() function, 352-353
process_statement() function, 355
processing statements, 348-350
  alternative approaches, 356-357
  binary data, 367-368
  causes of failures, 349
  character-escaping operations, 349
general-purpose statement handler, 354-355
multiple-statement execution, 375-377
mysql_store_result() function, 357-359
mysql_use_result() function, 357-359
prepared statements, 377
  executing, 378-379
  inserting rows and retrieving them
  program, writing, 379-388
  parameterizing, 377-378
quoting special characters, 365-366
result set metadata, 359-364
  availability, 359
  column information structures, accessing, 364
  defined, 359
  displaying, 360-364
  result set data processing decisions, 359
  result sets, returning, 351-353
  row-modifying, 350, 406-407
  sending to server functions, 349
procs_priv table, 680
prompt definition sequences, 1028-1029
prompt() function, 451
protocol_version system variable, 856
proxied_host columns, 689
proxied_user columns, 689
proxies_priv table, 680
PROXY privilege, 662, 667
proxy_user system variable, 856
proxying authentication plugins, 677-679
pseudo_thread_id system variable, 856
PURGE BINARY LOGS statement, 730, 954-955

Q

Qcache_free_blocks status variable, 891
Qcache_free_memory status variable, 891
Qcache_hits status variable, 891
Qcache_inserts status variable, 892
Qcache_lowmem_prunes status variable, 892
Qcache_not_cached status variable, 892
Qcache_queries_in_cache status variable, 892
Qcache_total_blocks status variable, 892
Qcache_xxx status variable, 885
qq (double-quoting strings), 417-418
read_buffer_size system variable  1125

qualifiers
  identifiers, 98
  joined table column references, 138-139
QUARTER() function, 812
queries
  alternative forms, testing, 289
  badly performing, identifying, 285
  criteria, specifying, 56-59
  dates, 57
    differences between, 68
    operations supported, 66
  parts, retrieving, 67-68
  specific, searching, 66-67
  syntax, 66
  expansion searches, 170, 175-176
  multiple tables. See joins; subqueries
NULL value, 60-61
numeric ranges, 56
optimizer, 286-290
  alternative forms of queries, testing, 289
  EXPLAIN output, 290-296
  hints/overrides, 287
  identical data type columns, comparing, 288
  joins versus subquery support, 289
  operation, verifying, 287
  restrictive tests, 286
  stand alone indexed columns in comparison expressions, 288-289
  table order, forcing, 287
  tables, analyzing, 287
  type conversions, 289-290
pattern matching, 69-70
results
  binding to variables, 421-423
  limiting, 63-64
  sorting, 61-63
  several individual values, 59
  string values containing character data, 56
summaries
  counting, 72-76
  unique values present in a set of values, 72
table contents, displaying, 54
terminology, 20-21
user-defined variables, creating, 71
query() function, 491, Web: 1164-1165
query_alloc_block_size system variable, 856
query_cache status variables, listing of, 891-892
query_cache_limit system variable, 856
query_cache_min_res_unit system variable, 856
query_cache_size system variable, 856
query_cache_type system variable, 856
query_cache_wlock_invalidate system variable, 856
query_prealloc_size system variable, 857
questions status variable, 885
quote() function, 418-419, 505-506, 798, Web: 1139, Web: 1165
quote_identifier() function, Web: 1139
quoting
  C client programs, 365-366
  identifiers, 97-98
  options, 1008-1009
  Perl DBI, 416-419
  PHP, 505-507
  privileges, 667
  user account names, 658

R

RADIANS() function, 787
dep_button() function, 526
RAND() function, 787-788
rand_seed1 system variable, 857
rand_seed2 system variable, 857
range_alloc_block_size system variable, 857
ranges
  data types, 748
  numeric data types, 197
  sequence columns, 235
  temporal data types, 218-219
RasieError attribute, 403
raw data values, loading, 52-53
raw partitions, 597-598
RDBMS (relational database management system), 18
  banner advertisement table example, 19
defined, 18-19
READ COMMITTED isolation level, 162
READ UNCOMMITTED isolation level, 162
read_buffer_size system variable, 857
read_file() function, 445
read_only system variable, 857
read-only table locking
  all tables at once, 705
  individual tables, 703-704
read_rnd_buffer_size system variable, 857
reading option files, 332-335
  connection parameters, 424
  Windows, 424-425
read/write table locking, 704-705
REAL_AS_FLOAT, 865
records
  filing time benefit, 13
  multiple-user access, 13
  remote access, 13
  retrieval benefits, 13
recovery, 722
  auto-recovery, 700, 706
  backups, 707-709
    best practices, 709
    binary, 714-715
    InnoDB, 715-716
    selecting, 708
    storage engine portability, 709-710
    text, 711-714
    types, 708
  binary log file statements, re-executing, 723-725
  databases, 541, 722
  InnoDB auto-recovery failure, 725-726
  tables, 723
REFERENCES privilege, 664
referential integrity, 164. See also foreign keys
REGEXP/NOT REGEXP operators, 243-244, 777-780
regular expressions
  pattern matching, 775-778
  POSIX character class constructions, 778
regular indexes, 123
relational database management system. See RDBMS
relative comparison operators, 144-145
relay_log system variable, 857
relay_log_basename system variable, 857
relay_log_index system variable, 857
relay_log_info_file system variable, 857
relay_log_info_repository system variable, 858
relay_log_purge system variable, 858
relay_log_recovery system variable, 858
relay_log_space_limit system variable, 858
relay logs, 618, 624
  expiring, 630
  master-slave replication, 731
RELEASE SAVEPOINT statement, 955
RELEASE_LOCK() function, 825
RELOAD privilege, 662, 676
relocating data directory contents, 556-557
  assessing, 558-559
  entire directory, 559
  function, selecting, 557
  individual databases, 559-560
  individual tables, 560
  InnoDB tablespace, 560
  precautions, 558
  startup option, 557
  status/log files, 561-562
  symlink, 557
remote access, 13
remove_backslashes() function, 512
RENAME clause, 129-130
RENAME TABLE statement, 955
RENAME USER statement, 656, 955
renaming tables, 129-130
renewal notices, sending, 443-448
renewal_notify.pl script, 444
REPAIR TABLE statement, 720, 956
repairing tables
  InnoDB, 718
  MyISAM, 719
  mysqlcheck utility, 720-721
  REPAIR TABLE statement, 720
REPEAT() function, 798
REPEAT statements, 989
REPEATABLE READ isolation level, 162
REPLACE attribute, 232
REPLACE() function, 798
REPLACE statement, 956-957
replication
  binary logging formats, 731
  compatibility guidelines, 727-728
  databases, 541
  master-slave, 728-731
    master.info file, 730
    master server settings, 728-729
Perl DBI scripts, 400-401
entire sets, returning at once, 413-415
null values, checking, 416
number of rows returned, counting, 411
row-fetching loops, 407-411
PHP, 501-504
all rows at once, 504
arrays, 503
calculated columns, 503
default fetch mode, setting, 502
individual column values, 503
NULL values, 504
row-fetching loop, 501-503
statement handle, 501
processing functions, Web: 1103-1108,
Web: 1114-1117
returning (statements)
C client programs, 351-353
Perl DBI. See result sets, Perl DBI scripts
single-row, 411-413
results
binding to variables, 421-423
limiting, 63-64
retrieving, 54-56
column values, naming, 64-66
criteria, 56-59
dates, 66-69
multiple tables, 78-85
NULL values, 60-61
pattern matching, 69-70
summaries, 72-78
table contents, 54
user-defined variables, 71
sorting, 61-63
subqueries, testing, 143-144
retrieving
data
column values, naming, 64-66
criteria, specifying, 56-59
dates, 57, 66-69
multiple tables. See joins; subqueries
NULL values, 60-61
numeric ranges, 56
pattern matching, 69-70
SELECT statements, 54-56
several individual values, 59
string values containing character
data, 56
summaries, 72-78
table contents, displaying, 54
user-defined variables, 71
rows
  all at once, 413-415, 504
  arrays, 503
  calculated columns, 503
default fetch mode, 502
  individual column values, 503
  null values, checking, 416, 504
  number returned, counting, 411
Perl DBI, 401-402
PHP, 501-504
row-fetching loops, 407-411, 501-503
single-row results, 411-413
RETURN statement, 268, 989
RETURNS clause, 268
REVERSE() function, 798
REVOKE statement, 671-672, 957-958
revoking privileges, 671-672
RIGHT() function, 798
right joins, 139-143
RLIKE/NOT RLIKE operators, 780
rollback() function, Web: 1139, Web: 1165
ROLLBACK statement, 958-959
ROLLUP clause, 77
root accounts passwords
  assigning, 567-569
  resetting, 582-583
rotate_fixed_logs.sh script, 626
rotating logs, 625
  fixed-name, 626-629
tables, 631
ROUND() function, 253, 788
routine_name columns, 688
routine_type columns, 688
row-based logging, 731
ROW_COUNT() function, 831-832
row-fetching loops
  Perl DBI, 407-411
    fetchrow_array() function, 408-409
    fetchrow_arrayref(), 409-410
    fetchrow_hashref(), 410-411
    functions, listing of, 407
  PHP, 501-503
rowCount() function, 505, Web: 1171
rows
  adding
data files, 52-53
    INSERT statement, 50-52
  deleting, 85-86
    events, 275
    preserving sequencing, 235
  modifying statements, 350
    Perl DBI, 406-407
    PHP, 501
  multiple-table
    deleting, 154-155
    updates, 155-156
  phantom, 162
  randomly selecting, 64
retrieving
  all at once, 413-415, 504
  arrays, 503
  calculated columns, 503
default fetch mode, 502
  individual column values, 503
  null values, checking, 416, 504
  number returned, counting, 411
Perl DBI, 401-402
PHP, 501-504
row-fetching loops, 407-411, 501-503
single-row results, 411-413
storage formats, 299
  displaying/editing, 300
  InnoDB, 299-300
  MEMORY, 299
  MyISAM, 299
updating, 86
rows() function, Web: 1145
RPAD() function, 799
RTF directories, creating, 440-442
RTRIM() function, 799
S
sampdb distribution
  files/directories, 735-736
  unpacking, 735
  Web site, 735
sampdb_pdo.php script, 495
search_members() function

proxied_host/proxied_user, 689
routine_name, 688
routine_type, 688
table_name, 688
user, 688

score table
creating, 39-40, 48-49
linking with grade_event table
dates, 41
event IDs, 41-42

score_browse.pl script, 475-479
display_events() function, 476-477
display_scores() function, 477-479

score_entry.php script, 510-511
action input parameter, 513
add_new_event() function, 517-518
display_cell() function, 516
display_events() function, 514-515
display_scores() function, 518-519
enter_scores() function, 520-522
framework, 513-514
PDO transaction processing, 520
script_name() function, 516
security, 522
solicit_event_info() function, 516-517

script_name() function, 516
script_param() function, 512

scripts
C client. See C client programs
Perl DBI. See Perl DBI scripts
PHP. See PHP scripts
stored
benefits, 261-262
compound statements, 266-267
defined, 261
events, 274-275
security, 275-276
single statement example, 266
stored functions, 268-271
stored procedures, 268-272
triggers, 272-273
Web-based. See Web-based scripts

search_members() function
ushl_browse.pl script, 480
ushl_ft_browse.pl script, 482-483
searches
  full-text
    boolean mode, 174-175
    characteristics, 171
    configuring, 176-177
    natural language, 172-174
    query expansion, 175-176
    types, 170
  tables, 479-483
SEC_TO_TIME() function, 812
SECOND() function, 812
secure_auth system variable, 858
secure_file_priv system variable, 858
Secure Sockets Layer. See SSL
security
  access control risks, 673-676
    ALTER privilege, 676
    anonymous-user accounts, 673
    FILE privilege, 674-676
    GRANT OPTION privilege, 674
    insecure accounts, 673-674
    mysql database privileges, 673-674
    passwords in old hash format, 673
    PROCESS/SUPER privileges, 674-675
    RELOAD privilege, 676
    superuser privileges, 673-674
  db_browse.pl script, 471
  external risks, 646
  filesystem access, 540
    administrative-only, setting, 649-651
    base directory insecurities, checking, 648
    data directory insecurities, checking, 648
    overview, 646-647
    stealing data example, 647
  functions, 821-824
  hidden fields, 528
  Historical League member entries online editing script, 527-529
  initial user accounts, 564-569
    available on all platforms, 566
    client program connections, 566
    displaying, 565
    passwords, assigning, 567-569
    platform specific, 567
  internal risks, 645
  load_defaults() function, 335
  log files, 556
  mysqld, 540
  new server passwords, 569
  online score-entry script, 520-521
  option files, 653-654
  PHP, 491
  SSL
    benefits, 694
    configuring, 695-698
  storage engine locking levels, 303-305
  stored programs, 275-276
  Unix socket file, 652-653
  user-specific options (programs), 1011
  views, 275-276
  Web-based scripts, 470-471
SELECT privilege, 665
SELECT statements, 959-965
  clauses
    FROM, 54-56
    FOR UPDATE, 964
    GROUP BY, 963
    HAVING, 963
    INTO, 964
    LIMIT, 963
    LOCK IN SHARE MODE, 964
    ORDER BY, 963
    PROCEDURE, 963
    WHERE, 56
  data, retrieval, 54-56
  examples, 964-965
  indexes, 962
  joins, 961-963
    column references, qualifying, 138-139
    inner, 137-138
    outer, 139-143
  NULL values, checking, 504
  number of rows returned, 411
  options, 960
  overview, 954-958
  results, writing to files, 964
  single-row results, retrieving, 412-413
  subqueries, 143-144
    ALL/ANY/SOME, 146-147
    correlated, 148
    FROM clause, 149
EXIST/NOT EXISTS, 147-148
IN/NOT IN, 145-146
relative comparison operators, 144-145
rewriting as joins, 149
uncorrelated, 148
syntax, 136
select_full_join status variable, 886
select_full_range_join status variable, 886
select_range status variable, 886
select_range_check status variable, 886
select_rows() function, 385-388
select_scan status variable, 886
selectall_arrayref() function, Web: 1140
selectall_hashref() function, Web: 1140
selectcol_arrayref() function, Web: 1140
selecting
APIs, 314-314
development time, 316-317
execution environment, 315
performance, 315-316
portability, 317
columns for indexing, 281-285
badly performing queries, identifying, 285
cardinality, 282
comparisons, matching to index types, 284-285
overindexing, 284
prefixes, 283-284
short values, 283
data types, 255-256
currency, 258
dates, 258-259
height information, 257-258
performance/efficiency, 256
ranges of values, 256, 259-260
storage size, 256
string, 217-218
value types in column, 256-259
data types for query optimization, 296-298
BLOB/TEXT, 298
ENUM, 297
NOT NULL, 297
numbers, 296
PROCEDURE ANALYSE() function, 297
smallest types, 296-297
strings, 296
tables, defragmenting, 297
databases, 105-106
error message language, 604
expression writing style, 290-292
GRANT statements, 661
locale, 604-605
numeric data types, 203, 257-258
rows, 64
storage engines, 594
selectrow_array() function, 413, Web: 1141
selectrow_arrayref() function, Web: 1141
selectrow_hashref() function, Web: 1141
semicolons (;), statements, 27, 266
sequences, 230
adding to tables, 235-236
arbitrary values, creating, 238
AUTO_INCREMENT properties
general, 230-232
InnoDB, 234
MEMORY, 234-235
MyISAM, 232-234
creating without AUTO_INCREMENT, 237-239
decreasing numbers, creating, 238
increasing numbers, creating, 237-238
incrementing counters, 238-239
multiple independent, creating, 233
mysql prompt definition, 1028-1029
nonpositive numbers, 235
ranges, 235
resequencing existing columns, 236-237
resets, 235
unsigned, 235
SERIALIZABLE isolation level, 162
server_id system variable, 858
server_uuid system variable, 858
servers
connections
authentication plugins, 677
establishing, 25-26
Perl DBI API, 312
PHP scripts, 490-491
programs. See connect1 client program; connect2 client program
terminating, 26-27
Web scripts, 468-469
database transfers, 716
text backup files, 716-717
writing directly to other server, 717-718
maintenance interference, preventing, 701
internal locking, 702-703
locking all tables at once, 705
read-only locking, 703-704
read/write locking, 704-705
shutting down servers, 702
multiple, 632
administration, 539
client programs, running, 641
configuring, 635
directory options, 633
error log file names, 634
InnoDB log location, 634
issues, 632-635
login account option, 635
network interface options, 633
replication slave options, 634
startup option strategies, 636-637
status/log file names, 634
Unix, 637-639
Windows, 639-641
MySQL. See mysqld
new passwords, setting, 569
option groups, 578
replication
compatibility guidelines, 727-728
master-slave, 728-731
overview, 727
running as administrator, 652
shutting down, 702
SQL mode, 96-97
setting, 96-97
values, 96
Web, configuring, 460-461
SESSION qualifier
SHOW STATUS statement, 589
SHOW VARIABLES statement, 586
SESSION_USER() function, 832
SET data type, 208-213
creating, 209
ENUM data type, compared, 208
improper values, 228
numeric form, 211-212
permitted value lists, defining, 209
size/storage requirements, 204
sorting/indexing, 212-213
SET PASSWORD statement, 567-568, 966-967
SET statement, 159-160, 965-966
SET strings, 194, 759
SET TRANSACTION statement, 967-968
setAttribute() function, Web: 1165, Web: 1171
setFetchMode() function, Web: 1171-1172
SHA() function, 824
SHA1() function, 824
SHA2() function, 824
shared_memory system variable, 859
shared_memory_base_name system variable, 859
shebang (#!), 396
shells
aliases, 89
command history, 88
scripts, 89
shift left (<<) operator, 773
shift right (>>) operator, 773
SHOW BINARY LOGS statement, 969
SHOW BINLOG EVENTS statement, 969-970
SHOW CHARACTER SET statement, 970
SHOW COLLATION statement, 970-971
SHOW COLUMNS statement, 37, 131, 971
SHOW CREATE DATABASE statement, 106-107, 130
SHOW CREATE statement, 972
SHOW DATABASES privilege, 663
SHOW DATABASES statement, 38, 130, 547, 972
SHOW ENGINE statement, 972
SHOW ENGINE INNODB STATUS statement, 170
SHOW ENGINES statement, 108-109, 593, 973
SHOW ERRORS statement, 973
SHOW EVENTS statement, 973
SHOW FULL COLUMNS statement, 37
SHOW FUNCTION STATUS statement, 973
SHOW GRANTS statement, 671, 974
SHOW INDEX statement, 131, 974-975
SHOW MASTER STATUS statement, 975
SHOW OPEN TABLES statement, 975
SHOW PLUGINS statement, 976
SHOW PRIVILEGES statement, 976
SHOW PROCEDURE STATUS statement, 973
SHOW PROCESSLIST statement, 976
SHOW RELAYLOG EVENTS statement, 976
SHOW SLAVE HOSTS statement, 977
SHOW SLAVE STATUS statement, 730, 977-979
SHOW statement, 969
LIKE pattern clause, 131
metadata, accessing, 130-132
WHERE clause, 131
SHOW STATUS statement, 979-980
GLOBAL/SESSION qualifiers, 589
LIKE/WHERE clauses, 589
status variables, displaying, 584
SHOW TABLE STATUS statement, 131, 980-981
SHOW TABLES statement, 37, 130, 981
SHOW TRIGGERS statement, 981
SHOW VARIABLES statement, 982
GLOBAL/SESSION qualifiers, 586
LIKE clause, 585
system variables, displaying, 583
WHERE clause, 585
SHOW VIEW privilege, 665
SHOW WARNINGS statement, 982
show_argv program, 332-335
show_opt program
connect2 program, compared, 347
invoking, 342-343
overview, 336
source file, 336-338
SHUTDOWN privilege, 663
shutting down
mysql, 539
servers, 702
SIGN() function, 789
SIGNAL statements, 995-996
SIGNED attribute, 201
SIN() function, 789
single-row result sets, returning, 411-413
size
BLOB/TEXT data types, 207
string data types, 204
tables, 551-553
skip_external_locking system variable, 859
skip_name_resolve system variable, 859
skip_networking system variable, 859
skip_show_database system variable, 859
slave_allow_batching system variable, 859
slave backups, creating, 732-733
slave_checkpoint_group system variable, 859
slave_checkpoint_period system variable, 859
slave_compressed_protocol system variable, 860
slave_exec_mode system variable, 860
slave_heartbeat_period status variable, 886
slave_last_heartbeat status variable, 886
slave_load_tmpdir system variable, 860
slave_max_allowed_packet system variable, 860
slave_net_timeout system variable, 860
slave_open_temp_files status variable, 886
slave_parallel_workers system variable, 860
slave_pending_jobs_size_max system variable, 860
slave_received_heartbeats status variable, 886
slave_retried_transactions status variable, 886
slave_running status variable, 886
slave_skip_errors system variable, 860
slave_sql_verify_checksum system variable, 860
slave_transaction_retries system variable, 861
slave_type_conversions system variable, 861
sleep() function, 832
slow query logs, 618
slow_launch_threads status variable, 886
slow_launch_time system variable, 861
slow_queries status variable, 886
slow_query_log system variable, 193, 621,
slow_query_log_file system variable, 861 750, 861
SMALLINT data types
ranges, 197
storage requirements, 197
socket system variable, 861
software required, 736-737
solicit_event_info() function, 516-517
SOME subqueries, 146-147
sort_buffer_size system variable, 861
sort_merge_passes status variable, 886
sort_range status variable, 887
sort_rows status variable, 887
sort_scan status variable, 887
**SQL mode, 96-97**
- `sql_auto_is_null` system variable, 861
- `sql_big_selects` system variable, 861
- `sql_buffer_result` system variable, 862
- `sql_log_bin` system variable, 862
- `sql_log_off` system variable, 862
- Composite modes, 862-866
  - Setting, 96-97
  - Values, 96, 862-865
- `sql_mode` system variable, 96, 862-866
  - Composite modes, 865
  - Values, 862-865
- `sql_notes` system variable, 866
- `sql_quote_show_create` system variable, 866
- `sql_safe_updates` system variable, 866
- `sql_select_limit` system variable, 866
- `sql_slave_skip_counter` system variable, 866
- `sql_warnings` system variable, 866

**SQL (Structured Query Language), 20**
- Case sensitivity, 99-101
- Aliases, 100
- Column names, 100
- Database names, 100
- Filenames, 100
- Forcing lowercase, 100
- Function names, 99
- Index names, 100
- Keywords, 99
- Stored program names, 100
- String values, 100
- Table names, 100
- Trigger names, 100
- View names, 100

**SSL (Secure Sockets Layer)**
- Benefits, 694
- Client support, 370-374
  - Availability, 370
  - Holding option values variables, 372-373
  - Options, adding, 370-372
  - Passing SSL option information to client library, 374
- Configuring, 695-698
  - Accounts requiring SSL, creating, 697-698
  - Certificate/key files, 697
  - Client programs SSL support, enabling, 696
  - Command-line options, 697
  - Language APIs, 698
  - Option files, 697
  - Server SSL support, enabling, 695-696
  - SSL-related server status variable values, displaying, 697

**SQRT() function, 789**

**SQL mode, 96-97**
- `sql_auto_is_null` system variable, 861
- `sql_big_selects` system variable, 861
- `sql_buffer_result` system variable, 862
- `sql_log_bin` system variable, 862
- `sql_log_off` system variable, 862
- Composite modes, 862-866
  - Setting, 96-97
  - Values, 96, 862-865
- `sql_mode` system variable, 96, 862-866
  - Composite modes, 865
  - Values, 862-865
- `sql_notes` system variable, 866
- `sql_quote_show_create` system variable, 866
- `sql_safe_updates` system variable, 866
- `sql_select_limit` system variable, 866
- `sql_slave_skip_counter` system variable, 866
- `sql_warnings` system variable, 866

**SQRT() function, 789**

**SQL (Structured Query Language), 20**
- Case sensitivity, 99-101
- Aliases, 100
- Column names, 100
- Database names, 100
- Filenames, 100
- Forcing lowercase, 100
- Function names, 99
- Index names, 100
- Keywords, 99
- Stored program names, 100
- String values, 100
- Table names, 100
- Trigger names, 100
- View names, 100

**SSL (Secure Sockets Layer)**
- Benefits, 694
- Client support, 370-374
  - Availability, 370
  - Holding option values variables, 372-373
  - Options, adding, 370-372
  - Passing SSL option information to client library, 374
- Configuring, 695-698
  - Accounts requiring SSL, creating, 697-698
  - Certificate/key files, 697
  - Client programs SSL support, enabling, 696
  - Command-line options, 697
  - Language APIs, 698
  - Option files, 697
  - Server SSL support, enabling, 695-696
  - SSL-related server status variable values, displaying, 697

**SQRT() function, 789**

**SQL (Structured Query Language), 20**
- Case sensitivity, 99-101
- Aliases, 100
- Column names, 100
- Database names, 100
- Filenames, 100
- Forcing lowercase, 100
- Function names, 99
- Index names, 100
- Keywords, 99
- Stored program names, 100
- String values, 100
- Table names, 100
- Trigger names, 100
- View names, 100

**SSL (Secure Sockets Layer)**
- Benefits, 694
  - Client support, 370-374
    - Availability, 370
    - Holding option values variables, 372-373
    - Options, adding, 370-372
    - Passing SSL option information to client library, 374
  - Configuring, 695-698
    - Accounts requiring SSL, creating, 697-698
    - Certificate/key files, 697
    - Client programs SSL support, enabling, 696
    - Command-line options, 697
    - Language APIs, 698
    - Option files, 697
    - Server SSL support, enabling, 695-696
    - SSL-related server status variable values, displaying, 697

**SQRT() function, 789**

**SQL (Structured Query Language), 20**
- Case sensitivity, 99-101
- Aliases, 100
- Column names, 100
- Database names, 100
- Filenames, 100
- Forcing lowercase, 100
- Function names, 99
- Index names, 100
- Keywords, 99
- Stored program names, 100
- String values, 100
- Table names, 100
- Trigger names, 100
- View names, 100

**SSL (Secure Sockets Layer)**
- Benefits, 694
  - Client support, 370-374
    - Availability, 370
    - Holding option values variables, 372-373
    - Options, adding, 370-372
    - Passing SSL option information to client library, 374
  - Configuring, 695-698
    - Accounts requiring SSL, creating, 697-698
    - Certificate/key files, 697
    - Client programs SSL support, enabling, 696
    - Command-line options, 697
    - Language APIs, 698
    - Option files, 697
    - Server SSL support, enabling, 695-696
    - SSL-related server status variable values, displaying, 697
plugins, loading, 591
storage engines status change options, 593-594

state
databases, resetting, 53-54
plugin activation, 592

statements
; (semicolons), 27
access verification, 689-690
account-management, 654-655
ALTER DATABASE, 107, 898
ALTER EVENT, 898-899
ALTER FUNCTION, 899
ALTER PROCEDURE, 899
ALTER TABLE, 899-904
action values, 899-903
benefits, 127-128
CHANGE clause, 128
CHARACTER SET clause, 128-129
ENGINE clause, 129
indexes, adding, 124
MODIFY clause, 128
partitioning options, 904
RENAME clause, 129-130
resequencing existing columns, 237
sequence columns, adding, 236
syntax, 128
table files, 549
ALTER VIEW, 905
ANALYZE TABLE, 905
BEGIN, 905
binary logging, 731
BINLOG, 906
CACHE INDEX, 906
CALL, 906
case-sensitivity, 29, 99-101
aliases, 100
column names, 100
database names, 100
filenames, 100
forcing lowercase, 100
function names, 99
index names, 100
keywords, 99
stored program names, 100
string values, 100
table names, 100

program options, 1006
requiring, 668
status variables, 892-894
ssl_accept_renegotiates status variable, 892
ssl_accepts status variable, 892
ssl_callback_cache_hits status variable, 892
ssl_cipher status variable, 892
ssl_cipher_list status variable, 892
ssl_client_connects status variable, 892
ssl_connect_renegotiates status variable, 892
ssl_ctx_verify_depth status variable, 893
ssl_ctx_verify_mode status variable, 893
ssl_default_timeout status variable, 893
ssl_finished_accepts status variable, 893
ssl_finished_connects status variable, 893
ssl_server_not_after status variable, 893
ssl_server_not_before status variable, 893
ssl_session_cache_hits status variable, 893
ssl_session_cache_misses status variable, 893
ssl_session_cache_mode status variable, 893
ssl_session_cache_overflows status variable, 893
ssl_session_cache_size status variable, 893
ssl_session_cache_timeouts status variable, 893
ssl_sessions_reused status variable, 893
ssl_used_session_cache_entries status variable, 893
ssl_verify_depth status variable, 894
ssl_verify_mode status variable, 894
ssl_version status variable, 894
ssl_xxx status variable, 887
ssl_xxx system variable, 867
standalone PHP scripts, 489
START SLAVE statement, 731, 983
START TRANSACTION statement, 157-158, 983-984
start_html() function, 463
starting mysqld, 539, 741
options, 577-579
Unix, 572-574, 741
Windows, 742
startup
character set/collation, setting, 603
InnoDB tablespace failure, 598
logging options, 619
multiple server options, 636-637
mysqld options, 577-579
triggers, 100
view names, 100
CHANGE MASTER, 907-908
CHARACTER SET, 102-103
CHECK TABLE, 719-720, 909-910
CHECKSUM TABLE, 910
client programs, 348-350
  alternative approaches, 356-357
  binary data, 367-368
  causes of failures, 349
  character-escaping operations, 349
  general-purpose statement handler, 354-355
  mysql_store_result() functions, 357-359
  mysql_use_result() functions, 357-359
quoting special characters, 365-366
result set metadata, 359-364
result sets, returning, 351-353
row-modifying statements, 350
  sending to server functions, 349
COLLATE, 102-103
comments, adding, 996-997
COMMIT, 910-911
compound, 266-267, 987-996
  condition-handling, 992-996
  control structure, 987-989
cursor, 991-992
declaration, 989-991
construction/execution functions, Web: 1099-1102
CREATE DATABASE, 30, 106-107, 130, 547, 911
CREATE EVENT, 274, 912-913
CREATE FUNCTION, 268, 913-915
CREATE INDEX, 915-916
CREATE PROCEDURE, 268, 913-915
CREATE TABLE, 113-114, 916-926
  AVG_ROW_LENGTH option, 115
column definitions, 926
data type keywords, 918-919
ENGINE clause, 46-47, 114
foreign key support, 922-923
IF NOT EXISTS modifier, 115
index clauses, 919
MAX_ROWS, 115
options, 919-922
PARTITION BY clause, 120-121
partitioning, 923-925
student table, 45-46
table files, creating, 549
  TEMPORARY keyword, 115-116
CREATE TABLE ...LIKE, 117-118
CREATE TABLE ...SELECT, 117-119
CREATE TRIGGER, 272, 926-927
CREATE USER, 927-928
  account operations, 655
  account value, 656
  auth_info clause, 659-660
  IDENTIFIED WITH clause, 676
selecting, 656
CREATE VIEW, 928-929
date/time retrieval, 27
DEALLOCATE PREPARE, 929
DELETE, 85-86, 154-155, 929-930
DESCRIBE, 36-37, 930-931
DO, 931
DROP DATABASE, 107, 547, 931
DROP EVENT, 932
DROP FUNCTION, 932
DROP INDEX, 127, 302, 932
DROP PROCEDURE, 932
DROP TABLE, 121-122, 549, 932
DROP TRIGGER, 932-933
DROP USER, 656, 933
DROP VIEW, 933
  ending, 27-28
  entering, 27
  multiple-lines, 28
  multiple statements on single line, 28-29
  typing less, 90-93
EXECUTE, 933
EXPLAIN, 290-296, 933-936
FLUSH, 936-937
FLUSH PRIVILEGES, 583
FLUSH TABLES, 302, 703
function syntax, 29
GRANT, 938-943
  clauses, 660-661
  examples, 942-943
  ON clause, 939-940
  privileges, revoking, 672
  privileges to be granted, 938-939
  REQUIRE clause, 668-669, 941
number of rows returned, 411
placeholders, 419-421
prepared, 421
quoting special characters, 416-419
result sets, returning. See result sets,
Perl DBI scripts
row-fetching loops, 407-411
row-modifying, 406-407
single-row results, returning, 411-413

PHP, 500-501
NULL values, checking, 504
prepared, 505
quoting special characters, 505-507
row-modifying, 501
rows, retrieving, 501-504

PREPARE, 954
prepared. See prepared statements
print, 426-428
PURGE BINARY LOGS, 954-955
reading from files, 29
RELEASE SAVEPOINT, 955
RENAME TABLE, 955
RENAME USER, 656, 955
REPAIR TABLE, 720, 956
REPLACE, 956-957
RESET, 957
result sets, returning, 351-353
results, displaying, 28
RETURN, 268
REVOKE, 671-672, 957-958
ROLLBACK, 958-959
row-modifying, handling
C client, 350
Perl DBI, 406-407
SAVEPOINT, 161, 959
SELECT, 959-965
data retrieval, 54-56
examples, 964-965
FOR UPDATE clause, 964
FROM clause, 54-56
GROUP BY clause, 963
HAVING clause, 963
indexes, 962
INTO clauses, 964
joins syntax. See SELECT statements,
joins
LIMIT clause, 963
LOCK IN SHARE MODE clause, 964
nesting with another SELECT statement. See subqueries
NULL values, checking, 504
number of rows returned, 411
options, 960
ORDER BY clause, 963
PROCEDURE clause, 963
results, writing to files, 964
single-row results, retrieving, 412-413
subqueries. See SELECT statements, subqueries
syntax, 136
WHERE clause, 56, 963
SET, 159-160, 965-966
SET PASSWORD, 567-568, 966-967
SET TRANSACTION, 967-968
SHOW, 969
LIKE pattern clause, 131
metadata, accessing, 130-132
WHERE clause, 131
SHOW BINARY LOGS, 969
SHOW BINLOG EVENTS, 969-970
SHOW CHARACTER SET, 970
SHOW COLLATION, 970-971
SHOW COLUMNS, 37, 131, 971
SHOW CREATE, 972
SHOW CREATE DATABASE, 106-107, 130
SHOW DATABASES, 38, 130, 547, 972
SHOW ENGINE, 972
SHOW ENGINE INNODB STATUS, 170
SHOW ENGINES, 108-109, 593, 973
SHOW ERRORS, 973
SHOW EVENTS, 973
SHOW FULL COLUMNS, 37
SHOW FUNCTION STATUS, 973
SHOW GRANTS, 671, 974
SHOW INDEX, 131, 974-975
SHOW MASTER STATUS, 975
SHOW OPEN TABLES, 975
SHOW PLUGINS, 976
SHOW PRIVILEGES, 976
SHOW PROCEDURE STATUS, 973
SHOW PROCESSLIST, 976
SHOW RELAYLOG EVENTS, 976
SHOW SLAVE HOSTS, 977
SHOW SLAVE STATUS, 977-979
SHOW STATUS, 979-980
GLOBAL/SESSION qualifiers, 589
LIKE/WHERE clauses, 589
status variables, displaying, 584
SHOW TABLE STATUS, 131, 980-981
SHOW TABLES, 37, 130, 981
SHOW TRIGGERS, 981
SHOW VARIABLES, 982
GLOBAL/SESSION qualifiers, 586
LIKE clause, 585
system variables, displaying, 583
WHERE clause, 585
SHOW WARNINGS, 982
START SLAVE, 983
START TRANSACTION, 157-158, 983-984
STOP SLAVE, 984
synonyms, 898
syntax, 897
table file operations, 549-550
table locking, 703
TRUNCATE TABLE, 984
UNINSTALL PLUGIN, 592, 984
UNION, 151-154, 985
UNLOCK TABLE, 304, 703, 985
UPDATE, 986-987
multiple-table, 155-156
root/anonymous-user passwords, 568
rows, 86
USE, 987
use DBI, 399
use strict, 399
use warnings, 399
status files
listing of, 554
multiple servers, creating, 634
relocating, 561-562
status variables, 881
case sensitivity, 881
displaying, 584
general, listing of, 881-888
InnoDB, listing of, 888-891
overview, 584
query cache, listing of, 891-892
SSL, 892-894
values, checking, 588-589
STD() function, 820
STDDEV() function, 820
STDDEV_POP() function, 820
STDDEV_SAMP() function, 820
stealing data example, 647
STOP SLAVE statement, 731, 984
stopping mysqld, 580-581
storage
  data type requirements
    numeric, 197-198
    temporal, 219
data types, 204, 748
images from files, 367-368
row formats, 299
displaying/editing, 300
InnoDB, 299-300
MEMORY, 299
MyISAM, 299
SQL statements in files, 29
temporal data types, 759
storage_engine system variable, 867
storage engines
ARCHIVE, 112
auto-recovery, performing, 700
availability, 108-109
BLACKHOLE, 112
converting tables to different, 129
CSV, 112, 710
default, selecting, 594
default status/startup option, 593-594
defined, 46
displaying available, 593
FEDERATED, 113
file representations, 548
grade-keeping student table example, 46-47
index characteristics, 123
InnoDB
  auto-recovery, 706, 725-726
  backing up, 715-716
  checking/repairing tables, 718
data, representing, 548
features, 110
innodb directory access mode, setting, 651
locking levels, 303
portability, 710
row storage formats, 299-300
sequence characteristics, 234
status variables, listing of, 888-891
system variables, listing of, 870-880
tablespace. See tablespaces (InnoDB)
transaction isolation levels, 162-163
variables, 600-601
listing of, 108
locking levels, 303-305
MEMORY
  data, representing, 548
  locking levels, 304
  overview, 111-112
  portability, 710
  row storage formats, 299
  sequence characteristics, 234-235
MERGE, 113, 304
MyISAM
  auto-recovery, 706
  checking/repairing tables, 719
data, representing, 548
features, 111
portability, 710
row storage formats, 299
sequence characteristics, 232-234
table maximum size, 552
NDB, 112
pluggable architecture, 108
portability, 709-710
specifying for tables, 114
table-specific files, 109-110
stored functions, 268-271
  creating, 268
defined, 268
  integer-valued parameter representing a
    year example, 268
  multiple values, 269
  names, 269
  privileges, 270-271
  security, 276
tables, updating, 270
stored procedures
  creating, 268
defined, 268
  example, 269
  invoking, 269
  parameter types, 271-272
stored procedures

privileges, 270-271
security, 275-276
tables, updating, 270
triggers, 273

stored routines

defined, 272
names, 272
privileges, 273
security, 276

stored programs

benefits, 261-262
case sensitivity, 100
compound statements, 266-267
defined, 261
events, 274-275
creating, 274
defined, 274
deleting old rows from table example, 275
enabling/disabling, 275
enabling scheduler, 274
logging, 274
one time only, 275
privileges, 274
scheduler status, verifying, 274
security, 276
starting/stopping scheduler, 274
security, 275-276

single statement example, 266
stored functions, 268-271
creating, 268
defined, 268
integer-valued parameter representing a year example, 268
multiple values, 269
names, 269
privileges, 270-271
tables, updating, 270

stored procedures

creating, 268
defined, 268
element, 269
invoking, 269
parameter types, 271-272
privileges, 270-271
security, 275-276
tables, updating, 270

triggers

actions, 273
benefits, 272
creating, 272

defined, 272
names, 272
privileges, 273
security, 276

str COLLATE collation operator, 773
str NOT REGEXP pattern, 775-778
STR_TO_DATE() function, 66, 813
STRAIGHT_JOIN, 287
STRICT_TRANS_TABLES SQL mode, 96, 865
strings, 182-184, 193

binary/nonbinary, 255
case sensitivity, 100
character data, retrieving, 56
converting to numbers, 249-250
data types, 204, 753-754
attributes, 214-216
binary, 755-756
binary/nonbinary corresponding types, 204-205
BINARY/VARBINARY, 207
BLOB, 207-208
CHAR/VARCHAR, 206
character sets/collations, 753-754
ENUM, 208-213, 758
improper values, 228
length, 205
lengths, 753
nonbinary, 756-758
query performance, improving, 296
selecting, 217-218
SET, 208-213, 759
size, 204
storage requirements, 204
TEXT, 207-208
trailing pad values, 218, 754
types, listing of, 194
escape sequences, 183
functions, listing of, 789-802
quoting characters
  C client programs, 365-366
  Perl DBI, 416-419
  PHP, 505-507
surrounding quotes, 182-183
values
  backslashes, turning off, 184
  binary, 185
  binary versus nonbinary, 188-189
  character set variables, 189-191
  CONVERT() function, 187-188
  escape sequences, 183
  hexadecimal notation, 184
  introducers, 187-188
  length, 188
  nonbinary, 185
  quote characters, including, 183-184
  sorting properties, 186
  surrounding quotes, 182-183
structural terminology, 18-19
Structured Query Language. See SQL
student table
  columns, creating, 46
  creating, 44-47
sub_size member (my_option structures), 341
SUBDATE() function, 813
submit_button() function, 526
subqueries, 84-85, 143-144
  ALL/ANY/SOME, 146-147
  correlated, 148
  correlated/uncorrelated, 144
  defined, 78
  EXISTS/NOT EXISTS, 147-148
  FROM clause, 149
  IN/NOT IN, 145-146
  query optimizer support, 289
  relative comparison operators, 144-145
  results, testing, 143-144
  rewriting as joins, 149
    matching values, selecting, 149-150
    nonmatching values, 150
scalar, 144, 241
types, 143
  uncorrelated, 148
SUBSTR() function, 799
SUBSTRING() function, 799-800
SUBSTRING_INDEX() function, 800
SUBTIME() function, 813
subtraction operator (-), 57, 241, 766
SUM() function, 76, 820
summaries, 72-78
  counting, 72-76
    distinct non-NULL values, 73
    groups, 74-76
    minimum/maximum/total/average values, 76
    non-NULL values, 73
    number of rows, 72
    overall count of values, 73
    summary, 77-78
  functions, listing of, 817-821
  unique values present in a set of values, 72
SUPER privilege, 663, 673-674
  definer privileges, setting, 276
  security risks, 674-675
switchboxes, 437-438
symlinks, 557, 651
sync_binlog system variable, 867
sync_frm system variable, 867
sync_master_info system variable, 867
sync_relay_log system variable, 867
sync_relay_log_info system variable, 867
synthetic indexes, 298
SYSDATE() function, 813
system tables, initializing, 742-743
system_time_zone system variable, 602, 867
SYSTEM_USER() function, 832
system variables, 584-585, 835-836
  character_set_server, 603
  collation_server, 603
  displaying, 583, 836
  error message language selection, 604
  expire_logs_days, 629
  general, listing of, 836-870
  general_log, 621
  general_log_file, 621
  InnoDB, listing of, 870-880
innodb_autoextend_increment, 596
innodb_data_file_path, 595
innodb_data_home_dir, 595
innodb_file_per_table, 599
lc_time_names, 604
log_warnings, 620
max_relay_log_size, 624, 630
names, 836
overview, 583
setting, 764
  runtime, 587-588
  server startup, 586-587
slow_query_log, 621
time zones, 602-603
values, checking, 585-586

T

table() function, 475
table handlers. See storage engines
table_definition_cache system variable, 868
table_locks_immediate status variable, 887
table_locks_waited status variable, 887
table_name columns, 688
table_open_cache, 868
table_open_cache_hits status variable, 887
table_open_cache_instances system variable, 868
table_open_cache_misses status variable, 887
table_open_cache_overflows status variable, 887
tables
  absence, 45, 49
  aliases, 98
  backups
    best practices, 709
    binary, 714-715
    InnoDB, 715-716
    selecting, 708
    storage engine portability, 709-710
text, creating, 711-714
  types, 708
  banner advertisement example, 19
  columns. See columns
  contents, displaying, 50, 54
copying from other tables, 117-120
creating, 113-114
  CREATE TABLE statement, 45-46
  if it doesn't already exist, 115
  from other tables/query results, 117-120
  partitions, 120-121
  storage characteristics, 114-115
temporary, 115-116
damages
  checking with CHECK TABLE statement, 719-720
  checking with mysqlcheck utility, 720-721
  InnoDB, checking and repairing, 718
  MyISAM, checking and repairing, 719
  overview, 718
  repairing with mysqlcheck utility, 720-721
  repairing with REPAIR TABLE statement, 720
data directory, relocating, 560
default database, listing, 37
defragmenting, 297
deleting, 121-122
descriptive information, displaying, 131, 135
file representations, 548
files created by storage engines, 109-110
grade_event
  creating, 40, 47
  linking with score table on dates, 41
  linking with score table on event IDs, 41-42
  grants. See grant tables
HTML, creating, 475
identifiers, 98
indexes, 122-123
column prefixes, 126
deleting, 127
existing tables, 124
flexibility, 122
FULLTEXT, 126
HASH, 125
ID numbers, generating, 36
new tables, 125
storage engine characteristics, 123
types, 123-124
unique, 124-125
td() function  1143

INFORMATION_SCHEMA database, 133-134
linking
dates, 41
event IDs, 41-42
listing, 130, 135
locking, 303-305
all at once, 705
overview, 702-703
read-only access, 703-704
read/write, 704-705
single session, 703
statements, 702
logs
rotating, 631
truncating, 625, 631
rotation, 625
writing to, 625
multiple
deleting rows, 154-155
queries, 78-84
retrievals. See joins; subqueries
updating rows, 155-156
names, 32, 100
operations statements, 549-550
partitions, creating, 120-121
preventive maintenance, 700
privileges, 667
recovering, 723
renaming, 129-130
rows. See rows
score
creating, 39-40, 48-49
linking with grade_event table on dates, 41
linking with grade_event table on event IDs, 41-42
sequence columns, adding, 235-236
server access, preventing, 701
internal locking, 702-703
locking all tables at once, 705
read-only locking, 703-704
read/write locking, 704-705
shutting down servers, 702
size, 551-553
storage characteristics, editing, 114-115
storage engines, converting, 129
structure
displaying, 36-37
editing, 127-130
student
columns, creating, 46
creating, 44-47
system, initializing, 742-743
temporary
creating, 115-116
names, 116
transactional/nontransactional mixing, 163
strict mode, 229
updating, 270
U.S. Historical League, creating, 31
member, creating, 35-38
member table, 33
president, 32, 34-35
Web searches, 479-483
FULLTEXT indexes, 482-483
pattern matching, 479-482
tables_priv table, 680
tablespace (InnoDB), 548
components, adding, 599
configuring, 111, 595-598
auto-extend increments, 596
file pathnames, 596
file specification syntax, 596
raw partitions, 597-598
regular files, 597
system variables, 595
Windows, 598
contents, 595
defined, 111
individual (per-table), 599-600
relocating, 560
startup failure, troubleshooting, 598
TAN() function, 789
tbl_name parameter
checking, 473
db_browse.pl script, 472
tc_log_max_pages_used, 887
tc_log_page_size status variable, 887
tc_log_page_waits status variable, 887
TCP/IP connections, listening, 579
td() function, 475
temporal data types, 193, 759
attributes, 223
automatic initialization/update properties, 224-226
DATE, 220-221, 760
DATETIME, 221, 760
formats, 226
fractional seconds, 223-224
improper values, 228
input dates, 220
listing of, 193
MySQL 5.6 improvements, 218 ranges, 218-219
storage requirements, 219, 759
TIME, 221, 760-761
TIMESTAMP, 221-222, 761-762
two-digit years, 227-228
values, 191, 226-227
temporal data types, 226-227
type conversions, 251
YEAR, 222-223, 762
zero values, 220

**TEMPORARY clause, 115-116**

temporary tables
creating, 115-116
names, 116

terminology
architectural, 21-22
query language, 20-21
structural, 18-19

testing
alternative forms of queries, 289
cascaded deletes, 167-168
cascaded updates, 168
subquery results, 143-144
type conversions, 252-253

text backups
all tables from all databases, 711
binary backups, compared, 708
compressing, 712
creating, 711-714
database transfers, 716-717
individual files, 711
mysqldump options, 712-714
output, 711-712
table subsets into separate files, creating, 712

**TEXT data type**
indexes, 207
overview, 207-208
query optimization, 298
size, 204, 207
special care, 208
storage requirements, 204

text-format backups
best practices, 709
defined, 708
text input fields, 530
TEXT strings, 194, 757-758
text_field() function, 530
textfield() function, 481
th() function, 475

thread_cache_size system variable, 868
thread_concurrency system variable, 868
thread_handling system variable, 868
thread_stack system variable, 868
threaded client functions, Web: 1119
threads_cached status variable, 887
threads_connected status variable, 887
threads_created status variable, 887
threads_running status variable, 888
TIME data type, 193, 221, 228, 760-761
time formats, 226
TIME() function, 813
TIME_FORMAT() function, 813
time_format system variable, 868
TIME_TO_SEC() function, 814
time_zone system variable, 602, 868
time zones, configuring, 602-603
timed_mutexes system variable, 880
TIMEDIFF() function, 814
TIMESTAMP() function, 814
TIMESTAMP() function, 814
TIMESTAMP data type, 193, 221-222, 761-762
automatic initialization/update properties, 224-226
current timestamp, 222
formats, 226-227
ranges, 222
time zones, 222
timestamp system variable, 868
TIMESTAMPADD() function, 814
TIMESTAMPDIFF() function, 68, 814
TINYBLOB data type, 204, 207-208
TINYBLOB strings, 194, 755
UNION statements

UNION statements, 151-154, 985
file representations, 549
names, 100, 272
privileges, 273
security, 276

TRIM() function, 800
TRN files, 549
troubleshooting
InnoDB tablespace startup failure, 598
multiple server issues, 632-635
mysqld connectivity
restarting manually, 581-582
root password, resetting, 582-583
table damages, 718
checking with CHECK TABLE statement, 719-720
InnoDB, checking and repairing, 718
MyISAM tables, 719
mysqlcheck utility, 720-721
repairing with REPAIR TABLE statement, 720

TRUNCATE() function, 789
TRUNCATE TABLE statement, 984
truncating tables, 631
tuning mysqld, 539
tx_isolation system variable, 869
tx_read_only system variable, 869
typelib member (my_option structures), 340
typing tips
copy/paste, 92
input line editing, 90-91
script files, 92-93

U

u_max_value (my_option structures), 340
UCASE() function, 801
ucs2 character set, 104
unary minus (-) operator, 241
UNCOMPRESS() function, 824
UNCOMPRESSED_LENGTH() function, 824
uncorrelated subqueries, 148
undefined argument, 422
unequal operators, 770
UNHEX() function, 801
Unicode character sets, 104-105
UNINSTALL PLUGIN statement, 592, 984
uninstalling plugins, 592
UNION statements, 151-154, 985
unique indexes, 123
   creating, 124-125
   primary key conversions, 169
unique values present in a set of values summary, 72
unique_checks system variable, 869
Universal Coordinated Time (UTC), 222
Unix
   compressing dump files, 712
   database relocation, 559
   error logs, 620
   initial user accounts, 567
   input editing commands, 90-91
   logs, rotating, 626-628
   multiple servers, 637-639
   MySQL, installing, 739
   mysqld
      connections, listening, 579
      running, 570
      starting, 572-574, 741
      stopping, 580-581
      unprivileged login account, configuring, 571-572
   PATH environment variable, configuring, 739
   preventive maintenance login, 701
   program option files, 1007
   socket file, securing, 652-653
   symlinks, 557
UNIX_TIMESTAMP() function, 815
UNLOCK TABLE statement, 304, 703, 985
unpacking sampdb distribution, 735
unquoted identifiers, 97
unsetting columns, 87
UNSIGNED attribute
   AUTO_INCREMENT, 235
   numeric data types, 201, 749
updatable_views_with_limit system variable, 869
UPDATE privilege, 665
UPDATE statement, 986-987
   AUTO_INCREMENT columns, 232
   multiple-table, 155-156
   root/anonymous-user account passwords, 568
   rows, 86
update_expiration() routine, 270
updates
   automatic, 224-226
   cascaded, 164, 168
   columns, 86, 155-156
   deciding to upgrade or not, 641-643
   grant tables, 655
   MySQL software, 539
   rows, 86, 155-156
   tables, 270
   views, 265
UPDATEXML() function, 828
UPPER() function, 801
uptime status variable, 888
uptime_since_flush_status status variable, 888
URL text, escaping, 464-465
USAGE privilege, 668
USAGE specifier, 661
use DBI statement, 399
USE statement, 105-106, 987
use strict statements, 399
use warnings statement, 399
user accounts
   access control risks, 673-676
   ALTER privilege, 676
   anonymous-user accounts, 673
   FILE privilege, 674-676
   GRANT OPTION privilege, 674
   insecure accounts, 673-674
   mysql database privileges, 674
   passwords in old hash format, 673
   PROCESS/SUPER privilege, 676
   RELOAD privilege, 676
   superuser privileges, 673-674
account-management statements, 654-655
anonymous
   deleting, 568-569
   passwords, assigning, 567-568
   security risk, 673
authentication, 659-660
authentication plugins, 676-679
proxy users, creating, 677-679
server connections, 677
server side/client side, 677
specifying, 676
CREATE USER statement
   account operations, 655
   selecting, 656
resetting passwords, 582-583
SSL required, creating, 697-698

USER() function, 832

user table, 680
authentication columns, 680, 684
privilege columns, 683
resource management columns, 680, 685-686
SSL-related columns, 680, 685

users
accounts. See user accounts
column values, 688
defined variables, 71, 894-895
proxy, creating, 677-679

U.S. Historical League project, 15-17
common-interest Web searches
FULLTEXT indexes, 482-483
pattern matching, 479-482
creative ideas, 15-16
directory, generating, 436-442
HTML format, 455-458
plain text version, 439-440
RTF version, 440-442
directory member entries, editing online, 527-536
editing form, 533-534
framework, 529-530
member login page, 530-531
null values, 535-536
password verification, 531-533
passwords, 527-529
updating entries, 534-535
displaying current membership count to visitors script, 313
home page, 488-491
interactive online quiz, creating, 522-527
member entries, editing, 448-454
members with common interests, finding, 454-455
membership renewal notices, sending, 443-448
membership updates, 270
objectives, 15
practical questions, 16-17
president born first, finding, 144-146
presidents born before Andrew Jackson
subquery, 84
tables, creating, 31
member, 33, 35-38
president, 32, 34-35
ushl_browse.pl script, 479-482
ushl_ft_browse.pl script, 482-483
UTC (Universal Coordinated Time), 222
UTC_DATE() function, 815
UTC_TIME() function, 815
UTC_TIMESTAMP() function, 816
utf8 character set, 104
utf8mb4 character set, 105
utf16 character set, 105
utf16le character set, 105
utf32 character set, 105
utilities
environment variables, checking, 1011-1012
functions, Web: 1148-1149
help messages, displaying, 1000-1001
my_print_defaults, 1011
myisamchk
defined, 538
maintenance advantages, 719
options specific to myisamchk, listing of, 1015-1018
overview, 1013-1014
standard options, 1014
table maintenance, 700
variables, 1018-1019
mysql. See mysql utility
mysql_config
defined, 1030
options, 1030-1031
mysql_install_db
specific options, 1032
standard options, 1032
mysql.server, 1029-1030
mysql_upgrade, 1033
specific options, 1033-1034
standard options, 1033
mysqld
commands, 1035-1037
defined, 538
options, 1034-1035
variables, 1013-1035
mysqldbinlog, 622
overview, 1038
specific options to mysqldbinlog,
1038-1041
standard options, 1038
variables, 1041
mysqldcheck
checking/repairing tables, 720-721
defined, 538
maintenance, scheduling, 707
overview, 1041
specific options to mysqldcheck,
1042-1044
standard options, 1041-1042
table analysis options, 1044
table checking options, 1044
table maintenance, 700
table optimization, 1044-1045
table repair options, 1044
mysqld_multi
specific options to mysqld_multi,
1057
standard options, 1056-1057
mysqld_safe, 1058
specific options to mysqld_safe,
1058-1059
standard options, 1058
mysqldump, 135
data format options, 1067-1068
database maintenance, 700
defined, 538
options, 712-714
overview, 1060
specific options to mysqldump,
1061-1067
standard options, 1060
text dump files, creating, 711-714
variables, 1068
mysqldumpslow, 621
mysqldump
specific options, 1033-1034
standard options, 1033
mysqldump, 621
mysqldumpslow, 621
mysqldump
specific options, 1033-1034
standard options, 1033
mysqldump, 621
mysqldumpslow, 621
specific options, 1070
standard options, 1071
options
  case sensitivity, 1001
  checking, 1011
  escape sequences, 1010
  group names, 1009
  leading spaces, 1010
  long-form/short-form, 1001
  option files, 1007-1011
  processing features, 1002
  quoting, 1010
  read directives, 1010-1011
  SSL, 1006
  standard, 1003-1005
  user-specific privacy, 1011
  variables, 1006-1007
pererror
  options, 1072
  overview, 1072
rotate, 628
UUID() function, 832
UUID_SHORT() function, 833

V
value member (my_option structures), 340
values
  account, 656
  boolean, 192
  columns, specifying, 196
  data types
    default, 748
    zero, 748
  empty, 475
  improper, handling, 228-230
    division by zero errors, 229
    transactional/nontransactional tables, 229
    turning on strict mode, 230
    warnings, 229
    weakening strict mode, 230
    zero date errors, 229
MYSQL_BIND array parameters, assigning, 385
NOT NULL
  data types for query optimization, 297
  numeric data types, 203
  string data types, 216
  temporal data types, 223
NULL, 192
  AUTO_INCREMENT columns, 231
  column sort, 62
  directory membership updates, 535-536
  expressions, 246-247
  foreign key relationships, 168-170
  numeric data types, 203
  result sets, checking, 416, 504
  sequence columns, 231
  string data types, 216
  temporal data types, 223
numeric, 181
  approximate, 181-182
  bit-field, 182
  exact, 181-182
options, holding, 372-373
  permitted lists, defining, 209
scope columns, 687-689
  column_name, 688
  Db, 688
  host, 687-689
  proxied_host, 689
  proxied_user, 689
  routine_name, 688
  routine_type, 688
  table_name, 688
  user, 688
spatial, 191-192
sql_mode system variable, 862-865
SSL-related server status variables, displaying, 697
status variables, checking, 588-589
strings, 182-184
  backslashes, turning off, 184
  binary, 185, 188-189
  case sensitivity, 100
  character set variables, 189-191
  CONVERT() function, 187-188
  escape sequences, 183
  hexadecimal notation, 184
  introducers, 187-188
  length, 188
  nonbinary, 185, 188-189
  quote characters, including, 183-184
  sorting properties, 186
  surrounding quotes, 182-183
system variables, checking, 585-586
temporal, 191
temporal data types, 226-227
type conversions, 251
trailing pad, 754
type conversions, 247-251
ASCII, 254
binary/nonbinary strings, 255
collation, 255
comparisons, 251
CONCAT() function, 248
dates, 254
explicit, 247
floating-point and integer values, 248
forcing, 253-255
hexadecimal, 248-249, 253
illegal values, 248
implicit, 247
operands to operator expected types, 249
string-to-number, 249-250
temporal values, 251
testing, 252-253
time parts, 254
values into strings, 253
VALUES() function, 833
VAR_POP() function, 821
VAR_SAMP() function, 821
var_type member (my_option structures), 340-341
VARBINARY data type
overview, 207
size/storage requirements, 204
VARBINARY strings, 194, 755
VARCHAR data type
CHAR data types, compared, 206
columns, creating, 35
size/storage requirements, 204
VARCHAR strings, 194, 757
variable-length character data types, 35
variable-length string types, 45-46, 205
variables
automatic_sp_privileges, 270
character set, 189-191
character_set_server, 102
collation_server, 102
connect1 client program, declaring, 324
connection handlers, 325
DBI_TRACE, 429
environment
PATH, 739-740
Perl DBI, Web: 1156
program options, checking, 1011-1012
global, 97
InnoDB storage engine, 600-601
innodb_file_per_table, 111
lower_case_table_names, 100
myisamchk utility, 1018-1019
MySQL programs, setting, 1006-1007
mysql utility, 1025-1026
MYSQL_BIND array parameters, assigning, 385
mysqladmin client, 1013-1035
mysqlbinlog, 1041
mysqld, 1056
mysqldump, 1068
option values, holding, 372-373
Perl DBI, 397, Web: 1131
PHP, 492
query results, binding, 421-423
sql_mode, 96
SSL-related server status values, displaying, 697
status, 881
general, listing of, 881-888
InnoDB, listing of, 888-891
overview, 584
query cache, listing of, 891-892
SSL, 892-894
values, checking, 588-589
system, 584-585, 835-836
character_set_server, 603
collation_server, 603
displaying, 583, 836
error message language selection, 604
expire_logs_days, 629
general, listing of, 836-870
general_log, 621
general_log_file, 621
InnoDB, listing of, 870-880
innodb_autoextend_increment, 596
innodb_data_file_path, 595
innodb_file_per_table, 599
cftime_names, 604
log_warnings, 620
max_relay_log_size, 624, 630
names, 836
overview, 583
setting, 587-588, 836
slow_query_log, 621
time zones, 602-603
values, checking, 585-586
user-defined, 71, 894-895
VARIANCE() function, 821
verifying
  event scheduler status, 274
  query optimizer operation, 287
VERSION() function, 833
version system variable, 869
version_comment system variable, 870
version_compile_machine system variable, 870
version_compile_os system variable, 870
vertical bars (||), operators/functions, 764
viewing. See displaying
views
  column names, 263-264
  defined, 261-263
  file representations, 549
  grade-keeping project test/quiz statistics
    example, 264-265
  identifiers, 98
  names, 100
  privileges, 263
  referring to columns, 263
  security, 275-276
  updating, 265
  WHERE clauses, 263

Web

Web-based scripts, 459
CGI
  HTML structure functions, 461
  HTML/URL text, escaping, 464-465
  HTML versus XHTML, 464
  importing functions, 461
  input parameters, 462
  multiple-purpose pages, writing, 465-468
  object-oriented interface, 461-462
  output, generating, 462-464
  portability, 463
database browser, 471-475
  data limits, 475
  empty values into nonbreaking spaces, converting, 475
  HTML table, creating, 475
  initial page, generating, 472-473
  main body of script, 471-472
  security warning, 471
  table contents, displaying, 473
tbl_name parameter, 472
grade-keeping project score browser, 475-479
  displaying events as a table, 476-477
  scores for specified event, listing, 477-479
Perl DBI scripts, 459-460
PHP
  data-retrieval, 497-499
  error handling, 507-509
  headers/footers functions, 495-497
  home page, 488-491
  include files, 491-495
  input parameters, 511-512
  interactive online quiz, 522-527
  live hyperlinks, creating, 499-500
  member entries online editing, 527-536
  NULL values, checking, 504
  online score-entry. See PHP scripts, online score-entry
  placeholders, 506-507
  prepared statements, 505
  quoting special characters, 505-507
row-modifying statements, 501
rows, retrieving, 501-504
security, 470-471, 491
statements, handling, 500-501
servers, connecting, 468-469
table searches, 479-483
FULLTEXT indexes, 482-483
pattern matching, 479-482
Web server, configuring, 460-461

Web sites
MySQL
mailing list, 80, 642
reference manuals, 7
Workbench, 21
Open Geospatial Consortium, 191
PDO, 486
Perl DBI, 395
Perl modules, 743
PHP, 486, 743
sampdb distribution, 735
WWW security FAQ, 471
XPath, 816
WEEK() function, 816-817
WEEKDAY() function, 817
WEEKOFYEAR() function, 817
WEIGHT_STRING() function, 801-802
WHAT clause, 660
WHERE clause
COUNT() function, 72
DELETE statement, 85
query optimizer, 288-289
SELECT statement, 56
SHOW statement, 131
SHOW STATUS statement, 589
SHOW VARIABLES statement, 585
UPDATE statement, 86
views, 263
where-to-find-Perl indicators, 398
WHILE statements, 983-984, 989
wildcards
LIKE operator, 244
REGEXP operator, 244
user account names, 657
Windows
compressing dump files, 712
database relocation, 560
initial user accounts, 567
InnoDB tablespace, configuring, 598
input editing commands, 90-91
logs, 621, 628
multiple servers, 639-641
MySQL, installing, 739
mysqld, 575
connections, listening, 580
options, 1053
running as Windows service, 576-577
running manually, 575
starting, 742
stopping, 581
PATH environment variable, configuring, 740
program option files, 1007-1008
WITH clause
GRANT statement, 661, 938
resource consumption limits, 670
WITH GRANT OPTION clause, 669-670
WITH ROLLUP clause, 77-78
writing expressions, 240-241
column references, 240
functions/arguments, 240
NULL values, 246-247
operators, 241-243
arithmetic, 241
bit, 242
comparison, 243
logical, 241-242
precedence, 246
pattern matching, 243-245
LIKE operator, 243-244
REGEXP operator, 244
scalar subqueries, 241
writing scripts
APIs. See APIs
benefits, 307-308
C client. See C client programs
goals, 308
Perl DBI, Web: 1130
case sensitivity, 400
characteristics, 396
comments, adding, 398
connect() function arguments, 399-400
connection parameters, specifying, 423-426
connections, 400, 425-426
debugging. See debugging, Perl DBI scripts
disconnecting, 402
dump_members.pl, 397-398
etire result sets, returning at once, 413-415
error handling, 402-405
finish() function, 402
function parentheses, 401
handles, 397
invoking scripts, 396
nonhandle variables, 397
null values, checking, 416
number of rows returned, 411
parameter binding, 421-423
placeholders, 419-421
prepared statements, 421
quoting special characters, 416-419
requirements, 395
result set metadata, 430-434
result sets, displaying, 400
row-fetching loops, 401-402, 407-411
row-modifying statements, 406-407
single-row results, returning, 411-413
statement terminators, 401
transactions, 434-436
undef argument, 422
use DBI statement, 399
use strict statements, 399
use warnings statement, 399
U.S. Historical League examples.
See Perl DBI scripts, U.S. Historical League
warnings mode, 401
Web-based. See Perl DBI scripts, Web-based
where-to-find-Perl indicator, 398
PHP, Web: 1157-1158
data-retrieval, 497-499
database-access interfaces, 485-486
error handling, 507-509
headers/footers functions, 495-497
hello world examples, 487-488
home page, 488-491
include files, 491-495
input parameters, 511-512
interactive online quiz, 522-527
live hyperlinks, creating, 499-500
member entries online editing,
names, 486
NULL values, checking, 504
online score-entry. See PHP scripts,
online score-entry overview, 487
PDO classes, Web: 1158
PDO error exceptions, 491
placeholders, 506-507
prepared statements, 505
quoting special characters, 505-507
row-modifying statements, 501
rows, retrieving, 501-504
samples, installing, 486-487
security, 491
server connections, 490-491
standalone, 489
statements, handling, 500-501
variables, 492
Web site, 486
Web site, 486
Web integration, 309
WWW security FAQ Web site, 471

X

X509, 668
XHTML, 464
XML functions, 828
XOR operator, 57, 241, 773
XPath, 816

Y

YEAR data type, 193, 222-223, 762
YEAR() function, 817
YEARWEEK() function, 817

Z

zero date errors, 229
ZEROFILL attribute, 201-202, 749
zero values, 748