API Reference

This chapter contains a reference to the APIs (application programming interfaces) that were introduced in this book for the programming languages PHP, Perl, Java, and C.

PHP API (mysql Interface)

Since PHP 5 two interfaces have been available for interacting with MySQL: the *mysql* functions and the *mysqli* classes and methods. This section gives a compact overview of the *mysql* functions and their parameters for access to MySQL databases. In the next section follows a reference to the *mysqli* interface.

First a few formal remarks to the following explanation of syntax:

- Square brackets in the left column indicate optional parameters.
- For all functions for which there is an enumeration over an index *n*, this index is in the range 0 to *nmax -1*.
- Examples of the application of these functions can be found particularly in Chapters 3 and 15. Further references can be found in the Index.

Establishing a Connection	
$id = mysql_connect($host, $user, $pw);	Establishes a connection.
$id = mysql_connect($host, $user, $pw, $new_link, $client_flags);	As above, except that *new_link* specifies whether a new connection should be made if a similar connection already exists (default *false*); *client_flags* specifies whether particular connection properties should be used (e.g., *MYSQL_CLIENT_COMPRESS*); these two optional parameters have been available only since PHP 4.3.
$id = mysql_pconnect($host, $user, $pw [, $new_link [, $client_flags]]);	Establishes a persistent connection or attempts to reuse a still open connection of another PHP page.
mysql_change_user($newuser, $passw);	Changes the user name for the connection.
mysql_select_db($dbname);	Determines the default database.
mysql_close([$id]);	Closes the connection.

Generally, the specification *id* can be omitted as long as there is only one connection to MySQL and thus no possibility of confusion.

Administration

$result = mysql_list_dbs([$id]);	Determines a list of all known databases; the evaluation is like that of *SELECT* queries.
$result = mysql_list_tables($dbname [,$id]);	Determines a list of all tables of the database; the evaluation is like that of *SELECT* queries.
$result = mysql_list_fields($dbn, $tbln [,$id]);	Determines a list of all fields of the table; the evaluation is like that of *SELECT* queries.
mysql_create_db($dbname [,$id]);	Creates a new database.
mysql_drop_db($dbname [,$id]);	Deletes a database.

Error Evaluation

$n = mysql_errno([$id]);	Determines the number of the most recent error.
$txt = mysql_error([$id]);	Determines the error message.

Information Functions

$txt = mysql_get_client_info([$id]);	Returns a character string with the version number of the client library.
$txt = mysql_get_host_info([$id]);	Returns a character string that describes the connection with the server (including host name, e.g., *"localhost via TCP/IP"*).
$n = mysql_get_proto_info([$id]);	Returns an integer with the number of the communication protocol in use (e.g., 10).
$txt = mysql_get_server_info([$id]);	Returns a character string with the version number of the server (e.g., *"5.0.2-alpha-standard"*).
$n = mysql_thread_id([$id]);	Returns an integer with the thread number of the given connection.
$txt = mysql_stat([$id]);	Returns a character string with a brief status report on the server (e.g., *"Uptime: 24763 Threads: 1 Questions: 65 …"*).

Executing SQL Commands

[$result =] mysql_query($sql [, $id])	Executes an SQL command for the default database; if it is a *SELECT* command, the found records can be evaluated with *$result*.
[$result =] mysql_db_query($db, $sql [, $id]);	Executes a command for the database *db* (which becomes the default database for all further queries).
$result = mysql_unbuffered_query($sql [, $id]);	Functions in principle like *mysql_query*, but is designed only for *SELECT* queries; the difference between this and *mysql_query* is that found records remain at first on the server and are transferred only as needed; the number of found records can be determined only by running through all of them; *mysql_num_rows* cannot be used.
$sql = addslashes($s);	Replaces 0-bytes and the characters *', "*, and \ in *$s* with the strings \0, \', \", and \\.
$sql = mysql_escape_string($s);	Functions like *addslashes*, but also replaces carriage return, line feed, and Ctrl+Z with the strings \n, \r, and \z.
$sql = mysql_real_escape_string($s [,$id]);	Functions like *mysql_escape_string*, but also considers the character set of the MySQL connection.

Output of *SELECT* Query Results	
mysql_data_seek($result, $rownr);	Determines the active data record within the result.
$row = mysql_fetch_array($result);	Returns the next record of the result (or *false*); access to individual fields takes place with *row[n]* or *row['fieldname']*, where case sensitivity is enforced.
$row = mysql_fetch_assoc($result);	Functions like *mysql_fetch_array*, except that field access must be by column name; *row[n]* is not permitted.
$row = mysql_fetch_row($result);	Returns the next record of the result (or *false*); access to individual fields is via *row[n]*.
$row = mysql_fetch_object($result);	Returns the next record of the result (or *false*); access to individual fields is via *row->fieldname*.
$data = mysql_result($result, $rownr, $colnr);	Returns the contents of the field in row *rownr* and column *colnr*; this function is slower than the other functions in this list and should therefore be used only in particular cases (such as to read a single value, e.g., *SELECT COUNT(*)*).
mysql_free_result($result);	Frees the query result immediately (otherwise, not until the end of the script).

All of the above functions except for *mysql_treat_result()* treat *result* as a value that enables access to the list of data records from a *SELECT* query. Usually, all records are output one after the other with *mysql_fetch_array*, *mysql_fetch_row*, or *mysql_fetch_object*, with each subsequent execution of the function setting the next record as the active one. The active record can also be set with *mysql_data_seek*.

The three functions *mysql_fetch_array*, *mysql_fetch_row*, and *mysql_fetch_object* differ only in the way in which individual fields of a record are accessed: *row['fieldname']*, *row[n]*, or *row->fieldname*. Of the three functions, *mysql_fetch_row* is the most efficient, but the difference in speed is negligible.

Metainformation on Query Results	
$n = mysql_num_rows($result);	Determines the number of result records (*SELECT*).
$n = mysql_num_fields($result);	Determines the number of result columns (*SELECT*).
$n = mysql_affected_rows([$id]);	Determines the number of records that were changed by the last SQL command (*INSERT*, *UPDATE*, *DELETE*, *CREATE*, …, *SELECT*).
$autoid = mysql_insert_id([$id]);	Determines the *AUTO_INCREMENT* value generated by the last *INSERT* command.
$txt = mysql_info([$id]);	Returns status information on the last command, e.g., *"Rows matched: 65 Changed: 65 Warnings: 0"*; `mysql_info` is designed only for commands that usually affect large numbers of records (*INSERT INTO*, *UPDATE*, *ALTER TABLE*, etc.).

Metainformation on the Fields (Columns) of Query Results

$fname = mysql_field_name($result, $n);	Returns the field name of column *n*.
$tblname = mysql_field_table($result, $n);	Returns the table name for column *n*.
$typename = mysql_field_type($result, $n);	Returns the data type of column *n* (e.g., *"TINYINT"*).
$length = mysql_field_len($result, $n);	Returns the maximum length of the column.
$lengths = mysql_fetch_length($result);	Returns a field with length information for all fields of the last-read data record (access with *lengths[n]*).
$flags = mysql_field_flags($result, $n);	Returns the attribute properties of a column as a character string (e.g., *"not_null primary_key"*); the properties are separated by spaces; evaluation is done most easily with *explode*.
$info = mysql_fetch_field($result, $n);	Returns information on column *n* as an object; evaluation proceeds with *info->name* (see the list below); note that *info* may contain, in part, properties other than *flags*.

Attributes of mysql_field_flags

auto_increment	Attribute *AUTO_INCREMENT*.
binary	Attribute *BINARY*.
blob	Data type *BLOB*, *TINYBLOB*, etc.
enum	Data type *ENUM*.
multiple_key	The field is part of a nonunique index.
not_null	Attribute *NOT NULL*.
primary_key	Attribute *PRIMARY KEY*.
timestamp	Attribute *TIMESTAMP*.
unique_key	Attribute *UNIQUE*.
unsigned	Attribute *UNSIGNED*.
zerofill	Attribute *ZEROFILL*.

Field Information for mysql_fetch_field

info->name	column name (field name).
info->table	name of the table from which the field comes.
info->max_length	maximum length of the field.
info->type	name of the data type of the field (e.g., *"TINYINT"*).
info->numeric	1 or 0, depending on whether the field contains numeric data.
info->blob, not_null, multiple_key, primary_key, unique_key, unsigned, zerofill	1 or 0; see list above for interpretation.

PHP-API (mysqli Interface)

In addition to the *mysql* interface described in the previous section, since PHP 5 the new *mysqli* interface has been available. Among its advantages are object-oriented programming, greater functionality, and support for new MySQL features (e.g., prepared statements).

The *mysqli* interface offers three classes:

mysqli: Objects of this class manage the connection to the MySQL server.

mysqli_result: Objects of this class contain the results of *SELECT* queries.

mysqli_stmt: Objects of this class enable the definition and execution of prepared statements.

The following syntax tables describe the most important properties and methods of the three classes. Here *$mysqli* is an object of the *mysqli* class, *$result* an object of the *mysqli_result* class, and *$stmt* an object of the *mysqli_stmt* class.

The mysqli Class

Establishing a Connection	
$mysqli = new mysqli("servername", "user", "pw", "dbname");	Connection variant 1: The *mysqli* constructor creates the connection.
$mysqli = mysqli_init();* *$mysqli->options(...); *$mysqli->ssl_set("key", "cert", "ca", "capath", "cipher");* *$mysqli->real_connect("servername", "user", "pw", "dbname", portno, "socketfile", flags);*	Connection variant 2: The *mysqli* object is created with *mysqli_init*. Then you can set options, for example, in the form *$mysqli->options(MYSQLI_OPT_CONNECT_TIMEOUT, 10)*. The actual creation of the connection finally takes place via *real_connect*, where the last three parameters are optional. Allowable flags include *MYSQLI_CLIENT_COMPRESS* and *MYSQL_CLIENT_SSL*.
$err = mysqli_connect_errno();	Tests whether an error has occurred during creation of the connection. The return value 0 means that the connection has succeeded.

mysqli — Executing SQL Commands	
[$result =] $mysqli->query($sqlstring);	Executes an SQL command. With queries (*SELECT*), moreover, the entire result is transferred to the client and returned as a *mysqli_result* object.
$mysqli->real_query($sqlstring);	Executes an SQL command without transferring the result.
$sql = "sqlcmd1;cmd2;cmd3";* *$ok = $mysqli->multi_query($sql); *if($ok)* *do {* *$result = $mysqli->store_result();* *... process result* *while($mysqli->next_result());*	Executes several SQL commands. The first *SELECT* result can be determined at once with *store_result*. If there is more than one result, these must be activated with *next_result*. This method returns 0 if there are no more results or if an error occurred.
$stmt = $mysqli->prepare($sqlstring);	Prepares a parameterized SQL command and returns a *mysqli_stmt* object; the execution of the command and the evaluation of the result take place via the *mysqli_stmt* methods (see below).

mysqli — Important Methods	
$mysqli->autocommit(0 / 1);	Sets the autocommit mode.
$mysqli->close();	Closes the connection.
$mysqli->commit();	Ends a transaction.
$str = $mysqli->escape_string($str);	Places a backslash in front of special characters in strings or replaces them with SQL-conforming character combinations.
$mysqli->rollback();	Aborts a transaction.

mysqli — Important Properties	
$n = $mysqli->affected_rows;	Returns the number of records that were altered by the most recent SQL command (*INSERT*, *UPDATE*, *DELETE*, etc.).
$n = $mysqli->errno;	Returns the number of the last-occurring error.
$str = $mysqli->error;	Returns the most recent error message.
$str = $mysqli->info;	Returns a string with information about the last-executed SQL command (e.g., after an *UPDATE* command: *Rows matched: nnn Changed: nnn Warnings: nnn*).
$n = $mysqli->insert_id;	Determines the *AUTO_INCREMENT* value created by the last *INSERT* command.
$n = $mysqli->warning_count;	Returns the number of warnings that were triggered by the last SQL command.

mysqli_result Classes

mysqli_result — Important Methods	
$result->close();	Releases memory occupied by an object.
$result->data_seek(n);	Makes record *n* the active record (*n=0* for the first record). This does not work if *query* was executed with the optional parameter *MYSQLI_USE_RESULT*.
$row = $result->fetch_array();	Returns the next record of the result (or *FALSE*). Access to individual fields takes place via *$row[n] or $row['fieldname']*, where case sensitivity is in force.
$row = $result->fetch_assoc();	Like *fetch_array*, except that access to fields can take place only in the form *$row['fieldname']*.
$meta = $result->fetch_fields();	Returns an array of objects that contain metadata on the columns. For example, *$meta[$n]->name* gives the name of a column.
$row = $result->fetch_row();	Like *fetch_array*, except that field access can take place only in the form *$row[n]*.
$row = $result->fetch_object();	Returns the next record of the result (or *FALSE*). Access to individual fields is via *$row->fieldname*, where case sensitivity is in force.

mysqli_result — Important Properties	
$n = $result->affected_rows;	Returns the number of records that were most recently changed with *INSERT*, *DELETE*, or *UPDATE*.
$n = $result->field_count;	Returns the number of columns of the *SELECT* result.
$lenarray = $result->lengths;	Returns an array whose elements contain the number of characters of all columns of the record most recently read with *fetch_xxx*.
$n = $result->num_rows;	Returns the number of records in a *SELECT* result.

mysqli_stmt Class

mysqli_stmt — Important Methods	
$stmt->bind_param('idsb...', $var1, $var2 ...);	Binds the parameters of an SQL command with the associated variables. For each variable the data type must be specified with a character: *i* = integer, *d* = double, *s* = string, *b* = binary (BLOB).
$stmt->bind_result($var1, $var2 ...);	Binds the columns of a *SELECT* result and the associated PHP variables. The method must be executed after *execute*.
$stmt->close();	Releases an object's memory.
$stmt->execute();	Executes an SQL command, where the parameters are passed via *$var1*, *$var2*, etc.
$stmt->fetch();	Transfers the next record of a *SELECT* result into the variables previously specified with *bind_result*. *fetch* returns *FALSE* if all records have been processed. *fetch* normally transfers each record individually from the server to the client, unless *store_result* was executed first.
$stmt->store_result();	Transfers all *SELECT* results to the client.

mysqli_stmt — Important Properties	
$n = $stmt->affected_rows;	Returns the number of records that were changed by *INSERT*, *DELETE*, or *UPDATE*.
$n = $stmt->num_rows;	Returns the number of records in a *SELECT* result; this property can be used only if *store_result* was previously executed.

Perl DBI

This reference does not contain an exhaustive list of all *DBI* methods, functions, and attributes. We have included only those key words that are most relevant for everyday MySQL programming with Perl. A complete reference can be found in the *perldoc* documentation.

In this book we generally place Perl methods in parentheses to improve readability. However, Perl syntax allows the execution of methods without parentheses, as in *$dbh->disconnect*.

The following lines show the principles for building a Perl script file:

```
#!/usr/bin/perl -w
use DBI;                 # database access
use CGI qw(:standard);  # required only with CGI scripts
use CGI::Carp qw(fatalsToBrowser);  # only with CGI scripts
...                    # here follows the actual code
```

Common Variable Names

The Perl *DBI* module is object-oriented. Thus the key words introduced in this section relate in part to methods that can be applied to specific objects (which in Perl are generally called *handles*). In this reference the following variable names will be used for such objects:

Common Variable Names for *DBI* Handles

$dbh	(database handle)	Represents the connection to the database.
$sth	(statement handle)	Enables evaluation of query results (with *SELECT* queries).
$h	(handle)	General handles, used in this section with methods that are available to *$dbh*, *$sth*, and *DBI*.
$drh	(driver handle)	Enables access to many administrative functions.

Establishing the Connection

The Connection

use DBI();	Activates the DBI module.
$datasource = "DBI:mysql:dbname;" . "host=hostname";	Specifies database names and computer names; the database name may be omitted, but then at least the colon must be given.
$dbh = DBI->connect($datasource, $username, $password [, %attributes]);	Creates the connection to the database.

Within the *datasource* character string, further parameters—separated by semicolons—may be given. Details on these parameters can be found in Chapter 22.

Optional Parameters in the *datasource* Character String

host=hostname	Specifies the name of the computer with the MySQL server (default *localhost*).
port=n	Specifies the IP port (default 3306).
mysql_compression=0/1	Compresses communication (default 0).
mysql_read_default_file=filename	Specifies the file name of the MySQL configuration file.
mysql_read_default_group=mygroup	Reads the group *[mygroup]* within the configuration (default group *[client]*).

A list with attributes can be passed as an optional fourth parameter of *connect*. You can supply these attributes either directly or in the form of an array variable:

```
$dbh = DBI->connect($source, $user, $pw, {Attr1=>val1, Attr2=>val2});
%attr = (Attr1=>val1, Attr2=>val2);
$dbh = DBI->connect($source, $user, $pw, \%attr);
```

To a great extent, these attributes can be read and changed with *$dbh* after the connection has been established:

```
$dbh->{'LongReadLen'} = 1000000;
```

The following table describes the most important *connect* attributes.

Optional *connect* Attributes ($*dbh* Attributes)	
RaiseError=>0/1	Displays an error message and ends the program if the connection is not properly established (default 0)
PrintError=>0/1	Displays an error message but continues execution if the connection is not properly established (default 1)
LongReadLen=>n	Determines the maximum size of an individual data field in bytes (0: do not even read long fields)
LongTruncOK=>0/1	Specifies whether data fields that are too long should be truncated (1) or whether an error should be triggered (0)

Terminate the Connection	
$dbh->disconnect();	Terminates connection to the database.

Executing SQL Commands, Evaluating SELECT Queries

Execute Queries Without Return of Data Records	
$n = $dbh->do("INSERT ...");	Executes an SQL query without returning records; *$n* contains the number of records that were changed, *0E0* if no records were changed, *-1* if the number is unknown, or *undef* if an error has occurred.
$n = $dbh->do($sql, \%attr, @values);	Executes a parameterized query; *@values* contains values for the wild card expressed in the SQL command by *?*; these values are handled automatically with *quote()*; *%attr* can contain optional attributes (otherwise, specify *undef*).
$id = $dbh->{'mysql_insertid'};	Returns the *AUTO_INCREMENT* value of the last record to be inserted (caution: the attribute *mysql_insertid* is MySQL-specific).

Execute Queries with Return of Data Records	
$sth = $dbh->prepare("SELECT ...");	Prepares an SQL query (generally *SELECT* queries); all further operations proceed with the help of the *statement handle*.
$sth->execute();	Executes the query.
$sth->execute(@values);	Executes a parameterized query; *@values* contains values for the wild card expressed in the SQL command by *?*.
$sth->fetchxxx();	Evaluates the results (see below).
$sth->finish();	Releases the resources of the statement handle.

If a query was executed with *prepare* and *execute* and a list of records was returned as result, then this list can be evaluated with a number of *fetch* methods.

Evaluating Lists of Data Records

@row = $sth->fetchrow_array();	Reads the next record into the array *@row*; if the end of the list is reached or if an error occurs, then *@row* contains an empty array; access to individual elements proceeds with *$row[n]* (where for the first column, *n=0*).
@row = $sth->fetch();	Equivalent to *fetchrow_array()*.
$rowptr = $sth->fetchrow_arrayref();	Equivalent to *fetchrow_array()*, but returns pointers to arrays (or *undef* if the end of the list of records is reached or an error occurs).
$row = $sth->fetchrow_hashref();	Reads the next record into the associated array *$row*; if the end of the list of records is reached or if an error occurs, then *$row* contains the value *undef*; access to individual elements proceeds with *$row->{'columnname'}*, where case sensitivity is enforced.
$result = $sth->fetchall_arrayref();	Reads all records and returns a pointer to an array of pointers to the individual records; access to individual elements proceeds with *$result->[$row][$col]*.
$result = $sth->fetchall_arrayref({});	As above, but the records are now associative arrays; access is via *$result->[$row]->{'columnname'}*.

Bind Variables to Columns (for *fetchrow_array*)

$sth->bind_col($n, \$var);	Binds the column *n* to the variable *$var* (where for the first column we have, exceptionally, *n=1*); the variable is automatically updated when the next record is read; *bind_col* must be executed after *execute*; the function returns *false* if an error occurs.
$sth->bind_columns(\$var1, \$var2, ...);	Equivalent to *bind_col*, except that variables are assigned to all columns of the query; make sure you have the correct number of variables.

Metainformation on SQL Commands

$n = $sth->{'NUM_OF_FIELDS'};	Returns the number of result columns (after *SELECT*).
$n = $sth->{'NUM_OF_PARAMS'};	Returns the number of parameters in queries with wild cards.
$sql = $sth->{'Statement'};	Returns the underlying SQL command.

Determine Column Names, Data Types, etc., of *SELECT* Results

$array_ref = $sth->{'NAME'};	Returns a pointer to an array with the names of all columns evaluation takes place with *@{$array_ref}[$n]*, where *n* ranges from 0 to *$sth->{'NUM_OF_FIELDS'}-1*.
$array_ref = $sth->{'NAME_lc'};	As above, but names in lowercase.
$array_ref = $sth->{'NAME_uc'};	As above, but names in uppercase.
$array_ref = $sth->{'NULLABLE'};	Specifies for each column whether *NULL* may be stored there (1) or not (0); if this information cannot be determined, then the array contains the value 2 for this column.
$array_ref = $sth->{'PRECISION'};	Specifies the precision in the sense of ODBC (the maximum column width).

Determine Column Names, Data Types, etc., of *SELECT* Results *(Continued)*

$array_ref = $sth->{'SCALE'};	Specifies the number of decimal places for floating-point numbers.
$array_ref = $sth->{'TYPE'};	Specifies the data type of all columns in the form of numerical values; the values relate to the ODBC standard; tests determined the following values: *CHAR*: *12*, *INT*: *4*, *TEXT/BLOB*: *-1*, *DATE*: *9*, *TIME*: *10*, *TIMESTAMP*: *11*, *FLOAT*: *7*, *DECIMAL*: *3*, *ENUM/SET*: *1*.

Shorthand Notation

@row = $dbh->selectrow_array($sql);	Corresponds to a combination of *prepare*, *execute*, and *fetchrow_array*; the result is an array of the first result data record; access to further records is not possible.
$result = $dbh->selectrow_array($sql);	As above, but *$result* contains the value of the first column of the first result record.
$result = $dbh->selectall_arrayref($sql);	Corresponds to *prepare*, *execute*, and *fetchall_arrayref*; for evaluation of *$result*, see *fetchall_arrayref*.

Marking Special Characters in Character Strings and BLOBs with the Backslash

$dbh->quote($data);	Prefixes the contents of *$data* between single quotes, prefixes \ and ‘ with \, and replaces 0-bytes with \0; if *$data* is empty (*undef*), then *quote()* returns the character string *NULL*.

Transactions

$dbh->{'AutoCommit'} = 0;	Deactivates autocommit mode. From now on, all SQL commands form transactions.
$dbh->commit();	Confirms a transaction.
$dbh->rollback();	Aborts a transaction.

Error Handling

Methods for Error Handling

$h->err();	Returns the error number of the last error (0: no error).
$h->errstr();	Describes the last error (empty string: no error).
DBI->trace($n [, $filename]);	Logs all internal data accesses and redirects output to *STDERR* or the given file; *n* specifies the degree of detail to be logged (0 deactivates logging, 1 gives a good idea, 15 logs everything).

Auxiliary Functions

***DBI* Functions**	
@bool = DBI::looks_like_a_number(@data);	Tests for each element in the array *@data* whether it is a number and returns *true* or *undef* in the result array.
$result = DBI::neat($data [, $maxlen]);	Formats the character string contained in *$data* in a form suitable for output; character strings are placed in single quotes; non-ASCII characters are replaced by a period; if the character string is longer than *$maxlen* characters (default 400), then it is truncated and terminated with … .
$result = DBI::neat_list(\@listref, $maxlen, $sep);	As above, but for an entire array of data; the individual elements are separated by *$sep* (default ",").

***$dbh* Methods**	
$ok = $dbh->ping();	Tests whether the connection to MySQL still exists and returns *true* or *false* accordingly.

MySQL-Specific Extension of the DBD::mysql Driver

If you use the *DBI* module for access to MySQL databases, then there are some supplementary functions available via *DBI* methods and attributes, of which we shall now describe some of the most important. The use of these functions can simplify programming and can make Perl programs more efficient. However, the code will no longer be portable; that is, a later change to another database system will require additional work.

Administrative Functions Based on a Separated Connection	
$drh = DBI->install_driver('mysql');	Returns a driver handle.
$drh->func('createdb', $database, $host, $user, $password, 'admin');	Creates a new database; a new connection is used for this.
$drh->func('dropdb', $database, $host, $user, $password, 'admin');	Deletes a database.
$drh->func('shutdown', $host, $user, $password, 'admin');	Shuts down the MySQL server.
$drh->func('reload', $host, $user, $password, 'admin');	Reinputs all MySQL tables (including the *mysql* tables with privilege management).

Administrative Functions Within the Current Connection	
$dbh->func('createdb', $database, 'admin');	Creates a new database.
$dbh->func('dropdb', $database, 'admin');	Deletes a database.
$dbh->func('shutdown', 'admin');	Shuts down the MySQL server.
$dbh->func('reload', 'admin');	Reinputs all MySQL tables.

***$dbh* Attributes**	
$info = $dbh->{'mysql_hostinfo'};	Returns a string with the connection data for the MySQL server (e.g., *192.168.80.128 via TCP/IP*).
$info = $dbh->{'mysql_info'};	After certain special SQL commands returns a character string with information about the command (e.g., after an *UPDATE* command: *Rows matched: 13 Changed: 13 Warnings: 0*).
$id = $dbh->{'mysql_insertid'};	Returns the *AUTO_INCREMENT* value of the last inserted record.
$n = $dbh->{'mysql_protoinfo'};	Returns the version number of the MySQL connection protocol (e.g., 10).
$info = $dbh->{'mysql_serverinfo'};	Returns a string with the version number of the MySQL server (e.g., *5.0.2-alpha-standard*).
$info = $dbh->{'mysql_stat'};	Returns a string with the server status (number of threads, open tables, etc.).
$threadid = $dhb->{'mysql_thread_id'};	Returns the thread ID number of the current connection to MySQL.

***$sth* Methods and Attributes**	
$sth->rows();	Returns after *SELECT* queries the number of data records found by *SELECT*; caution: this does not work if *$sth->{'mysql_use_result'}=1* holds.
$sth->{'mysql_store_result'}=1;	Activates *mysql_store_result*, so that with *SELECT* queries all results are stored temporarily on the client computer (default setting).
$sth->{'mysql_use_result'}=1;	Activates *mysql_use_result*, so that with *SELECT* queries only a single record is stored temporarily on the client.

***$sth* Attributes for Determining Metadata on *SELECT* Results**	
$ar_ref = $sth->{'mysql_is_auto_increment'};	Tells whether for the columns the *AUTO_INCREMENT* attribute holds. This and all additional attributes return a pointer to an array whose values give the status. Evaluation is with *@{$ar_ref}[$n]*, where *n* ranges from 0 to *$sth->{'NUM_OF_FIELDS'}-1*.
$ar_ref = $sth->{'mysql_is_blob'};	Tells whether the columns contain BLOBs.
$ar_ref = $sth->{'mysql_is_key'};	Specifies whether the columns are indexed.
$ar_ref = $sth->{'mysql_is_not_null'};	Specifies whether the attribute *NOT NULL* holds for the columns.
$ar_ref = $sth->{'mysql_is_num'};	Specifies whether numerical data are stored in the columns.
$ar_ref = $sth->{'mysql_is_pri_key'};	Specifies which columns are part of the primary index.
$ar_ref = $sth->{'mysql_max_length'};	Specifies the maximum column width of the query results.
$ar_ref = $sth->{'mysql_table'};	Specifies the underlying table names for all columns.
$ar_ref = $sth->{'mysql_type_name'};	Specifies the names of the data types for all columns.

JDBC (Connector/J)

In order to be able to access MySQL under Java, a JDBC driver for MySQL must be installed. This book assumes that you are using Connector/J version 3.*n*. If this assumption is satisfied, you can use numerous classes and methods of JDBC (Java Database Connectivity) with the names *java.sql.** and *javax.sql.**. The following tables assemble only the most important classes and methods of JDBC. There is simply no room for a complete reference to this complex database programming library.

Establishing a Connection

Connection with *DriverManager*

import java.sql.;*	Enables direct access to the JDBC base classes.
Class.forName("com.mysql.jdbc.Driver").newInstance();	Loads and registers Connector/J, the MySQL driver for JDBC.
Connection conn = DriverManager.getConnection ("jdbc:mysql://hostname/dbname", "username", "password");	Creates a connection to the database *dbname* on the computer *hostname*; in the connection string, a large number of additional optional parameters may be passed, the most important of which appear in two tables in Chapter 17.

Connection with DataSource (Since Java 2, Version 1.4)

import java.sql.;* *import javax.sql.*;*	Enables direct access to the JDBC base and extension classes.
com.mysql.jdbc.jdbc2.optional. *MysqlDataSource ds =* *new com.mysql.jdbc.jdbc2.optional.* *MysqlDataSource();* *ds.setServerName("hostname");* *ds.setDatabaseName("dbname");* *Connection conn =* *ds.getConnection(* *"username", "password");*	Creates an object of the class *com.mysql.jdbc.jdbc2.optional.MysqlDataSource*, sets the host and database names, and finally establishes the connection.
ds.setUrl("...");	To be used instead of *setServerName* and *setDatabaseName* for setting various connection parameters; the syntax of the connection string (URL) is the same as for *DriverManager.getConnection*.

Executing SQL Commands

SQL Commands (*Statement*)

Statement stmt = *conn.createStatement();*	Creates a *Statement* object, necessary to execute an SQL command.
Statement stmt = *conn.createStatement(* *java.sql.ResultSet.* *TYPE_FORWARD_ONLY,* *java.sql.ResultSet.* *CONCUR_READ_ONLY);* *stmt.setFetchSize(* *Integer.MIN_VALUE);*	Defines a *Statement* object for forward only *ResultSets*.
Statement stmt = *conn.createStatement(* *java.sql.ResultSet.* *TYPE_SCROLL_SENSITIVE,* *java.sql.ResultSet.* *CONCUR_UPDATABLE);*	Defines a *Statement* object for a variable *ResultSet*.
int n = stmt.executeUpdate(*"INSERT ...");* *stmt.getWarnings();*	Executes *INSERT*, *UPDATE*, and *DELETE* commands. The return value gives the number of changed records. With *getWarnings* you can determine the warnings issued when the command was executed (equivalent to *SHOW WARNINGS*). This method returns an *SQLWarning* object until all warnings have been processed.
ResultSet res = stmt.executeQuery(*"SELECT ...");*	Executes a *SELECT* query and returns as result a *ResultSet* object.
stmt.addBatch("INSERT ..."); *stmt.addBatch("INSERT ...");* *int[] n = stm.executeBatch();*	Collects a number of SQL commands and executes them as a group; *executeBatch* returns an *int* field that specifies the number of changed records.

Determining *AUTO_INCREMENT* IDs After *INSERT* Commands

stmt.executeUpdate("INSERT ...");	The starting point for the following three variants.
ResultSet newid = *stmt.getGeneratedKeys();* *if(newid.next()) {* *int id = newid.getInt(1); }*	*getGeneratedKeys* returns a *ResultSet* object with the most recently generated ID number(s); normally, that is, after a usual *INSERT* command, *res* contains exactly one ID number that is read with *next()* and *getInt(1)*; *getGeneratedKeys* is available since Java 2, version 1.4.
long id = ((com.mysql.jdbc. *Statement)stmt).getLastInsertID();*	*getLastInsertID* also returns the ID number; however, the method is Connector/J-specific and not portable.
ResultSet newid = *stmt.executeQuery(* *"SELECT LAST_INSERT_ID()");* *if(newid.next()) {* *id = newid.getInt(1); }*	Here the ID number is returned with a separate SQL command; note that the command is executed in the same transaction.

Executing *PreparedStatments*	
PreparedStatement pstmt = conn.prepareStatement("INSERT ... (?, ?)");	Declares an SQL command with two parameters indicated by question marks.
pstmt.setString(1, "O'Reilly"); pstmt.setInt(2, 7878);	Passes the parameter; there are numerous methods in addition to *setString* and *setInt*, e.g., *setNull*, *setDate*, *setTime*, *setFloat*, *setBinaryStream*; to these methods are passed the parameter number (beginning with 1) and the actual data.
int n = pstmt.executeUpdate(); ResultSet res = pstmt.executeQuery(); pstmt.addBatch(); int n pstmt.executeBatch();	Executes the command(s); the methods have the same meaning as for the *Statement* class.

Processing SELECT Results (ResultSet Class)

Changing *ResultSets*	
res.deleteRow();	Deletes the active record.
res.updateXxx(n, data); res.updateRow();	First changes the specified columns of the active record and then stores the changes.
res.moveToInsertRow(); res.updateXxx(n, data); res.insertRow();	Inserts a new record, changes its columns, and then stores the changes.

Evaluating *ResultSet*	
res.getInt(n); res.getString(n); res.getBytes(n); ...	Returns the data field of column *n* of the currently active record; instead of the column number (1 for the first column), the name of the column may be specified, e.g., *getDate("birthdate")*.
res.wasNull();	Tests whether the most recently read data field was *NULL*; this test is necessary with elementary Java data types that cannot store the value *NULL* and instead contain 0; *wasNull* offers the only way of distinguishing between 0 and *NULL*.
res.getBinaryStream(n);	Returns an *InputStream* object for bytewise reading of binary data.
res.getCharacterStream(n);	Returns a *Reader* object for characterwise reading of binary data.
res.getBlob(n);	Returns a *Blob* object for reading binary data.
res.getClob(n);	Returns a *Clob* object for reading binary data.

***ResultSet* Navigation**	
res.next();	Makes the next record in *ResultSet* the active record; the method returns *false* if there are no more records.
res.first();	Activates the first record; the method returns *false* if the *ResultSet* contains no records.
res.previous();	Activates the previous record.
res.last();	Activates the last record.
res.beforeFirst();	Places the record cursor before the first record; the *ResultSet* object is thereby in the same condition as immediately after *executeQuery*; now the first record can be activated with *next*.
res.afterLast();	Places the record cursor after the last record; *previous* activates the last record.
res.isFirst(); *res.isLast();*	Tests whether the current record is the first/last.
int n = res.getRow();	Returns the number of the active record (1 for the first record).
res.absolute(n);	Activates record *n*.

Metadata on *ResultSet*	
ResultSetMetaData meta = res.getMetaData();	Returns a *ResultSetMetaData* object that gives information about the *SELECT* result.
meta.getColumnCount();	Returns the number of columns.
meta.getColumnName(i);	Returns the name (*String*) of column *i*.
meta.getColumnType(i);	Returns the data type of column *i*; the result is an *int* with one of the constants from *java.sql.Types*.
meta.getColumnTypeName(i);	Returns the name (*String*) of the data type of column *i*.
meta.IsNullable(i);	Tells whether the column may contain *NULL*.
meta.IsAutoIncrement(i);	Tells whether it is an *AUTO_INCREMENT* column.

Transactions

Transactions	
conn.setAutoCommit(false);	Enables transactions (of course only if the MySQL tables are transaction-capable).
conn.commit();	Confirms all SQL commands executed in the current transaction and begins the next transaction.
conn.rollback();	Aborts the commands of the current transaction and begins a new one.

ADO.NET (Connector/Net)

The following tables summarize the most important classes and methods provided by Connector/J (that is, the library `MySql.Data.dll`). Visual Basic syntax is used.

Establishing a Connection, Connection Properties

Creating the Connection

Imports MySql.Data.MySqlClient	Enables convenient access to the Connector/Net classes.
Dim myconn As MySqlConnection *myconn = New MySqlConnection(_* *"Data Source=localhost;* *Initial Catalog=mylibrary;* *User ID=root;PWD=xxxxxx")* *myconn.Open()*	Creates a connection to the MySQL server.

Methods and Properties of the *MySqlConnection* Class

BeginTransaction	Begins a transaction and returns a *MySqlTransaction* object (see below).
Close	Ends the connection.
ConnectionString	Contains the connection properties as a character string.
CreateCommand	Creates a *MySqlCommand* object.
Dispose	Releases memory used by the connection.
Ping	Tests whether the connection is active.
ServerThread	Contains the MySQL server thread number for the connection.
ServerVersion	Contains the MySQL version number as a string (e.g., *5.0.2-alpha-standard-log*).
State	Provides information about the state of the connection (data type *System.Data.ConnectionState*).

Executing and Evaluating SQL Commands

Executing SQL Commands (*MySqlCommand* Class)

Dim com As MySqlCommand *com = myconn.CreateCommand("sql")* *com = New MySqlCommand("sql", _* *myconn)*	Creates a *MySqlCommand* object in two different ways.
com.ExecuteNonQuery()	Executes an SQL command that returns no result (e.g., *UPDATE* or *INSERT*).
obj = com.ExecuteScalar()	Returns a single result (a scalar). *ExecuteScalar* is suitable only for *SELECT* queries that return exactly one row and one column. The return value of *ExecuteScalar* has the data type *Object* and must be transformed with a conversion function (e.g., *CInt*) or a cast operator (e.g., *(int)*) into the desired data format.
dr = com.ExecuteReader()	Returns a *MySqlDataReader* object (see below).
Dim n As Long *com.CommandText =* *"SELECT LAST_INSERT_ID()"* *n = CLng(com.ExecuteScalar())*	Returns the ID number (*AUTO_INCREMENT* column) of the last record inserted with *INSERT*.

Executing SQL Commands with Parameters (*MySqlParameter* Class)

Dim com As MySqlCommand *Dim p1, p2, p3 As MySqlParameter* *com = myconn.CreateCommand()* *com.CommandText = _* *"INSERT ... VALUES(?a, ?b, ?c)"* *p1 = com.Parameters.Add("?a", _* *MySqlDbType.VarChar)* *p2 = com.Parameters.Add("?b", _* *MySqlDbType.Int32)* ... *com.Prepare()*	Prepares an SQL command with parameters. Each parameter is specified in the form *?name*. Then an associated *MySqlParameter* object must be created in which the desired data type of the parameter is specified. *Prepare* prepares the command for later execution.
p1.Value = ... *p2.Value = ...* *com.ExecuteXxx()*	Assigns values to the *Value* properties of the *MySqlParameter* object and then executes the command with the method *Execute* described above.

Evaluating SELECT Results (*MySqlDataReader* Class)

Dim dr As MySqlDataReader *dr = com.ExecuteReader()*	Executes a *SELECT* command and returns the result as a *MySqlDataReader* object. Access to the *SELECT* results takes place under *forward-only* and *read-only*.
dr.HasRows	Tests whether the *DataReader* contains any data.
dr.FieldCount	Returns the number of columns of the *DataReader*.
While dr.Read() *n = CInt(dr!publID)* *s = CStr(dr!publName)* *End While*	Outputs the *DataReader* row by row. *Read* returns *False* if there are no more records. Access to the columns of the current record takes place in VB.NET in the form *dr!columnname*, and in C# in the form *dr["columnname"]*. The return values have the data type *Object* and must be transformed into the actual data format using *Cdatatype* functions (VB.NET) or *(datatype)* cast operators.
dr.GetName(n)	Returns the name of column *n* (0 for the first column).
dr.GetDataTypeName(n)	Returns the name of the column's data type.
dr.IsDBNull(n)	Tests whether column *n* of the current record contains *NULL*.
dr.GetByte(n) *dr.GetBytes(n, ...)* *dr.GetChar(n)* *dr.GetDateTime(n)* ...	Reads the data of column *n* directly. These methods are suitable in particular for processing binary data.
bool = dr.NextResult()	Activates the next *SELECT* result. The method returns *False* if there are no further results.
dr.Close()	Closes the *DataReader* and releases the data.

Altering Data in DataSet/DataTable

Application of *MySqlDataAdapter* and *MySqlCommandBuilder*

Dim da As New _ MySqlDataAdapter(com) Dim ds As New DataSet() da.Fill(ds, "dtname") Dim dt As DataTable = _ ds.Tables("dtname")	Creates a *DataTable* on the basis of the SQL command *com* (*MySqlCommand*) and makes it available under the name *dtname* in a *DataSet*.
n = dt.Count	Determines the number of records in the *DataTable*.
Dim row As DataRow For Each row In dt.Rows var = row!columnname Next	Creates a loop over all records of a *DataTable*. In C# column access is in the form *dt["columnname"]*.
row.Delete()	Deletes the current record in *DataSet*.
row!columnname = ... row.Update()	Changes the current record in *DataSet*.
Dim newrow As DataRow newrow = dt.NewRow() newrow!columnname = ... dt.Rows.Add(newrow)	Creates a new record and stores it in *DataSet*.
Dim cb As New _ MySqlCommandBuilder(da) da.Update(ds, "dtname")	Stores the changes made locally in the *DataSet* permanently on the MySQL server. The *MySqlCommandBuilder* object provides the necessary SQL change commands.
cb.GetDeleteCommand() cb.GetInsertCommand() cb.GetUpdateCommand()	Returns the *DELETE*, *INSERT*, or *UPDATE* commands created by *MySqlCommandBuilder* as *MySqlCommand* objects. (The actual SQL code can be read from the property *CommandText*.)

Transactions

Transactions

Dim tr As MySqlTransaction tr = myconn.BeginTransaction()	Creates a *MySqlTransaction* object.
Dim com As New MySqlCommand(_ "UPDATE ...", myconn, tr) com.ExecuteNonQuery() ... further commands of the transaction	Creates a *MySqlCommand* object and executes it in the framework of this transaction.
tr.Commit()	Confirms the transaction.
tr.Rollback()	Aborts the transaction.

C API

The following tables assemble the most important functions and structures of the C API.

Data Structures

Data Structures	
*MYSQL *conn;*	Structure with connection data.
*MYSQL_RES *result;*	Structure with the results of a *SELECT* query.
MYSQL_ROW row;	Pointer to the results of a row (i.e., of a data record).
MYSQL_ROW_OFFSET roffset;	Pointer to a record within the result list.
*MYSQL_FIELD *field;*	Structure for describing a column (column name, data type, number of digits, etc.); details in the next table.
MYSQL_FIELD_OFFSET foffset;	Offset within a record (0 for the first column, 1 for the second, etc.).
*MYSQL_STMT *stmt;*	Structure for processing prepared statements.
MYSQL_BIND bind[n];	Structure for describing the parameters of prepared statements.
MYSQL_TIME mytime;	Structure for passing date and time values in prepared statements.
my_ulonglong n;	64-bit integer; some of the MySQL functions described below return results of this data type.

In the further syntax tables, the variables *conn*, *result*, *row*, *field*, *roffset*, *foffset*, etc., will be used as if they had been declared in the above table. Note that *MYSQL_ROW* is already a pointer and is therefore declared without *.

Elements of the *MYSQL_FIELD* Structure	
*char *name;*	Name of the column.
*char *table;*	Name of the table from which the column comes; if the column was computed or an *ALIAS* was used, then *table* points to a character string with the formula or the *ALIAS* name.
*char *def;*	Default value of the column or *NULL*.
enum enum_field_types type;	Data type of the column; these are the choices: *FIELD_TYPE_BLOB*, *FIELD_TYPE_DATE*, *FIELD_TYPE_DATETIME*, *FIELD_TYPE_DECIMAL*, *FIELD_TYPE_DOUBLE*, *FIELD_TYPE_ENUM*, *FIELD_TYPE_FLOAT*, *FIELD_TYPE_INT24*, *FIELD_TYPE_LONG*, *FIELD_TYPE_LONGLONG*, *FIELD_TYPE_NULL*, *FIELD_TYPE_SET*, *FIELD_TYPE_SHORT*, *FIELD_TYPE_STRING*, *FIELD_TYPE_TIME*, *FIELD_TYPE_TIMESTAMP*, *FIELD_TYPE_TINY*, *FIELD_TYPE_VAR_STRING*, *FIELD_TYPE_YEAR*.

Continued

Elements of the *MYSQL_FIELD* Structure *(Continued)*	
unsigned int length;	Length of the column according to the column definition.
unsigned int max_length;	Maximal length of a column within the query result; the value is always 0 if you use *mysql_use_result()*.
unsigned int flags;	Additional information for describing the column: *AUTO_INCREMENT_FLAG*, *BINARY_FLAG*, *MULTIPLE_KEY_FLAG*, *NOT_NULL_FLAG*, *PRI_KEY_FLAG*, *UNIQUE_KEY_FLAG*, *UNSIGNED_FLAG*, *ZEROFILL_FLAG*.
unsigned int decimals;	Number of places after the decimal point in *DECIMAL* columns (e.g., 5 for *DECIMAL(10,5)*).

Data Structures for Prepared Statements

Elements of the *MYSQL_BIND* Structure	
enum enum_field_types buffer_type;	Data type of the parameter; possible values are collected in the following table.
*void *buffer;*	Pointer to the buffer variable in which the data are passed.
unsigned long buffer_length;	Maximal buffer size (for strings/BLOBs).
*unsigned long *length;*	Actual length of the data passed (for strings/BLOBs).
*my_bool *is_null;*	Specifies whether *NULL* should be passed; in this case the contents of the buffer variable are not evaluated. Caution: *is_null* cannot be directly read; it is a pointer to the variable that contains the relevant information.
my_bool is_unsigned;	For integer data types specifies whether the value is to be interpreted as unsigned.
my_bool error;	Specifies whether an error has occurred in data transport (e.g., exceeding the maximum buffer size).

The following table summarizes the allowable settings for *buffer_type* in a *MYSQL_BIND* structure. The second column gives the matching MySQL data type and the best-fitting C data type.

enum_field_types (Settings)	
MYSQL_TYPE_TINY	For MySQL *TINYINT* values *(C-Typ char)*.
MYSQL_TYPE_SHORT	For *SMALLINT* values *(short int)*.
MYSQL_TYPE_LONG	For *INT* values *(int)*.
MYSQL_TYPE_LONGLONG	For *BIGINT* values *(long long int)*.
MYSQL_TYPE_FLOAT	For *FLOAT* values *(float)*.
MYSQL_TYPE_DOUBLE	For *DOUBLE* values *(double)*.
MYSQL_TYPE_TIME	For *TIME* values *(MYSQL_TIME)*.
MYSQL_TYPE_DATE	For *DATE* values *(MYSQL_TIME)*.
MYSQL_TYPE_DATETIME	For *DATETIME* values *(MYSQL_TIME)*.

enum_field_types* (Settings) *(Continued)	
MYSQL_TYPE_TIMESTAMP	For *TIMESTAMP* values *(MYSQL_TIME)*.
MYSQL_TYPE_STRING	For *CHAR* strings *(char *)*.
MYSQL_TYPE_VAR_STRING	For *VARCHAR* strings *(char *)*.
MYSQL_TYPE_TINY_BLOB	For *TINY_BLOB*s *(char *)*.
MYSQL_TYPE_BLOB	For *BLOB*s *(char *)*.
MYSQL_TYPE_MEDIUM_BLOB	For *MEDIUM_BLOB*s *(char *)*.
MYSQL_TYPE_LONG_BLOB	For *LONG_BLOB*s *(char *)*.

Elements of the *MYSQL_TIME* Structure	
un*signed int year;*	Year.
unsigned int month;	Months.
unsigned int day;	Days.
unsigned int hour;	Hours.
unsigned int minute;	Minutes.
unsigned int second;	Seconds.
my_bool neg;	Boolean value that tells whether a negative value is involved (such as with a result of the function *TIME_DIFF*).
unsigned long second_part;	Microseconds; not used in MySQL 5.0.

Connection and Administration

Establishing a Connection	
*MYSQL *conn;*	
conn = mysql_init(NULL);	Initializes the *MYSQL* data structure.
mysql_options(conn, option, "value");	Sets additional options for the connection; an *option* with one of the following values is passed: *MYSQL_OPT_CONNECT_TIMEOUT*, *MYSQL_OPT_LOCAL_INFILE*, *MYSQL_OPT_NAMED_PIPE*, *MYSQL_INIT_COMMAND*, *MYSQL_READ_DEFAULT_FILE*, *MYSQL_READ_DEFAULT_GROUP*. Many additional options are documented at `http://dev.mysql.com/doc/mysql/en/mysql_options.html`. With some options, *"value"* can be used to specify the desired value; for setting several options, the function must be called repeatedly; *mysql_options* must be exectued before *mysql_real_connect*.

Continued

Establishing a Connection *(Continued)*	
mysql_real_connect(conn, "hostname", "username", "password", "dbname", portnum, "socketname", flags);	Makes a connection to the database and returns *NULL* in case of error; *flags* can contain a combination of the following values: *CLIENT_COMPRESS*, *CLIENT_FOUND_ROWS*, *CLIENT_IGNORE_SPACE*, *CLIENT_INTERACTIVE*, *CLIENT_LOCAL_FILES*, *CLIENT_MULTI_STATEMENTS*, *CLIENT_MULTI_RESULTS*, *CLIENT_NO_SCHEMA*, *CLIENT_ODBC*, *CLIENT_SSL*.
mysql_set_server_option(conn, option);	Changes server options for a long-term connection. Currently, only two values are accepted for server options: *MYSQL_OPTION_MULTI_STATEMENTS_ON*, *MYSQL_OPTION_MULTI_STATEMENTS_OFF*.
mysql_change_user(conn, "username", "password", "dbname");	Changes the user and the default database for an existing connection.
mysql_change_db(conn, "dbname");	Changes the default database; the function assumes that the user has access to the database.
mysql_ping(conn);	Tests whether the connection still exists; if not, the connection is recreated; returns 0 as result if an active connection exists.
mysql_close(conn);	Closes the connection.

Acquiring Information on the Current Connection	
mysql_characterset_name(conn);	Returns a character string with the default character set of the connection.
mysql_get_client_info();	Returns a character string with information on the version of the client library in use (e.g., *"5.0.2"*).
mysql_get_server_info(conn);	Returns a string with the version of the server (e.g., *"5.0.2-alpha-standard"*).
mysql_get_host_info(conn);	Returns a string with information on the connection to the server (e.g., *"localhost via UNIX socket"*).
mysql_get_proto_info(conn);	Returns the version number (*unsigned int*) of the connection protocol, e.g., 10.
mysql_info(conn);	Returns a string with information on the execution of the last *INSERT*, *UPDATE*, *LOAD DATA*, or *ALTER TABLE* command (e.g., *"Rows matched: 3 Changed: 3 Warnings: 0"*).
mysql_stat(conn);	Returns a string with the server status (number of threads, number of open tables, etc.).
mysql_thread_id(conn);	Returns the number (*unsigned long*) of the thread that the current connection is processing on the server.

Adminstrative Functions	
mysql_kill(conn, n);	Ends the thread specified by *n* (requires the *Process* privilege).
mysql_shutdown(conn);	Shuts down the server (execution requires the *Shutdown* privilege).

Error-Handling	
mysql_errno(conn);	Returns the error number (*unsigned int*) for the most recently executed command (or 0 if there was no error).
mysql_error(conn);	Returns a string with the error message (or an empty string *“”* if there was no error).
mysql_warning_count(conn);	Returns the number of warnings that the last command returned. To read the actual warnings, you must execute the command *SHOW WARNINGS*.

Executing and Evaluating SQL Commands

Execution of SQL Commands	
mysql_query(conn, “SELECT …”);	Executes the specified command and returns 0 if the server accepts the command without triggering an error.
mysql_real_query(conn, “SELECT … “, len);	Like *mysql_query*, except that the SQL command may now contain the 0-byte (e.g., to store BLOBs); for this, the length of the string must be specified explicitly.
mysql_affected_rows(conn);	Returns the number (data type *my_ulonglong*) of changed records after a *DELETE*, *INSERT*, or *UPDATE* command; the function does not return the number of results of a *SELECT* command.
mysql_insert_id(conn);	Returns the *AUTO_INCREMENT* value (data type *my_ulonglong*) of the last record created via *INSERT*.

Processing *SELECT* Results	
result = mysql_store_result(conn);	Transfers all results from the server to the client and stores them in the *MYSQL_RES* structure.
result = mysql_use_result(conn);	Represents an alternative to *mysql_store_result*; the transfer of individual records is only prepared; no data are actually transferred.
mysql_num_fields(result);	Returns the number of columns of the result.
row = mysql_fetch_row(result);	Transfers the next record to a *MYSQL_ROW* structure; the function returns *NULL* if there are no further records available, i.e., if all records have already been processed.
row[n];	Returns a 0-terminated string with the content of column *n* of the current record; note that *row[n]* can contain *NULL*; if you are processing binary data with zero bytes, you must make the call with *mysql_fetch_lengths* to determine the size of the data field.
mysql_free_result(result);	Releases the result structure; if you are working with *mysql_use_result*, the result is only now released by the server.

Processing *SELECT* Results: Metainformation on Columns and Fields	
mysql_fetch_lengths(result);	Returns for the current record an *unsigned long* field with the length of the result string in *row[n]*.
field = mysql_fetch_field(result);	Returns a description (*MYSQL_FIELD* structure) of the data type of a column; the first call returns the data for the first column, the next for the second column, etc.; the function returns *NULL* after all columns have been run through; the elements of the *MYSQL_FIELD* structure were described previously.
field = mysql_fetch_field_direct(result, n);	Returns a description of column *n* (0 for the first column, etc.).
mysql_fetch_fields(result);	Returns a description of all columns as a *MYSQL_FIELD* field.

The following functions can be used only if you are using *mysql_store_result* (and are not working with *mysql_use_result*).

Processing *SELECT* Results: Additional Functions for *mysql_store_result*	
mysql_num_rows(result);	Returns the number of found records.
mysql_data_seek(result, n);	Moves the row cursor to record *n* (0 for the first record); then *mysql_fetch_row* must be executed to reinput data.
roffset = mysql_row_tell(result);	Returns a pointer (a sort of bookmark) to the current record.
mysql_row_seek(result, roffset);	Moves the row cursor to a particular location within the result list; then *mysql_fetch_row* must be executed to reinput data; *roffset* must be determined earlier with *mysql_row_tell*.

***MULTI_STATEMENTS* and *MULTI_RESULTS* Mode**	
mysql_real_connect(…, CLIENT_MULTI_STATEMENTS);	Activates the two *MULTI* modes at connection time.
mysql_set_server_option(conn, MYSQL_OPTION_ MULTI_-STATEMENTS_ON);	Activates the two *MULTI* modes at a later time.
mysql_query("command1; command2; command3");	Executes several commands separated by semicolons. The result of the first command can be processed as usual (*mysql_affected_rows*, *mysql_store_result*, etc.).
n = mysql_more_results(conn);	Tests whether there are more results (*n=1*).
n = mysql_next_result(conn);	Activates the next result. This can then be processed with the usual functions. The possible return values are these: *n=0* OK, there are more results, *n=-1* OK, but there are no more results, *n>0* error number.

Auxiliary Functions	
n = mysql_real_escape_string (conn, dest, src, srclen);	Copies the string *src* into the string *dest* while replacing the special characters with \ character combinations (*\0*, *\b*, *\t*, *\"*, *\'*, etc.); *dest* is terminated with a 0-byte; *srclen* specifies the number of characters in *src*; *dest* must be previously initialized with a string of the appropriate length; to be able to copy a string full of special characters without error, *dest* must offer place for *srclen*2+1* characters; the function returns the number of characters in *dest* (without the 0-byte). If *"O'Reilly"* is copied, then afterward, *src* contains the string *O\'Reilly*, and *n* the value 9.
n = mysql_hex_string(to, from, len);	Changes every character of the string *from* into hexadecimal code and writes it to the buffer *to*. The resulting string in *to* ends with a 0-byte. It does not contain the characters `0x`, which in MySQL indicate hexadecimal strings. You must insert `0x` yourself when preparing the SQL command. *len* gives the length of the string in *from*. The buffer *to* must have length *len*2+1* characters. The function returns the length of the resulting string without the closing 0-byte.
destpt = strmov(dest, src);	Like *strcpy*, copies the string *src* to the string *dest*; the difference is in the return value, which points to the end of the string in *dest*; this makes it easy to construct a string from several pieces; to be able to use *strmov*, the files `my_global.h` and `m_string.h` must be included before `mysql.h`.

Prepared Statements

Preparing and Executing Prepared Statements	
stmt = mysql_stmt_init(conn);	Returns a *MYSQL_STMT* structure.
mysql_stmt_prepare(stmt, sqlcmd, strlen(sqlcmd));	Initializes the *MYSQL_STMT* structure; *sqlcmd* is the string (*char[]*) with the SQL command to be executed.
mysql_stmt_bind_param(stmt, bind);	Declares the parameters for the SQL command; *bind* is a *MYSQL_BIND* field, where each field element describes a parameter.
mysql_stmt_execute(stmt);	Executes the SQL command. The parameters are read from the buffer variables given in *bind* and written there as well (*SELECT* results).
mysql_stmt_close(stmt);	Releases the memory for the *MYSQL_STMT* structure.

Evaluating *SELECT* Results of Prepared Statements	
stmt = mysql_stmt_init(conn);	Returns a *MYSQL_STMT* structure.
mysql_stmt_prepare(stmt, sqlcmd, strlen(sqlcmd));	Initializes the *MYSQL_STMT* structure.
mysql_stmt_bind_result(stmt, bind);	Declares the result columns of the *SELECT* command; *bind* is a *MYSQL_BIND* field, where each field element describes a column.
mysql_stmt_execute(stmt);	Executes the SQL command.
mysql_stmt_store_result(stmt);	Transfers all results to the client and stores them in a buffer. Calling this function is optional. If you do not do so, the records remain on the server until the last record is read with *mysql_stmt_fetch*. Then *mysql_stmt_data_seek* is not available, and *mysql_stmt_num_rows* is available only after all records have been processed.
n = mysql_stmt_num_rows(stmt);	Returns the number of result records; *n* has the data type *my_ulonglong*.
mysql_stmt_fetch(stmt);	Reads the next record into the variables given by the *MYSQL_BIND* field. The function returns 0 if there are no further results.
mysql_stmt_data_seek(stmt, n);	Activates record *n*. The next call to *mysql_stmt_fetch* reads this record into the variables.
result = mysql_stmt_result_metadata(stmt);	Returns metainformation on the *SELECT* result. Details can then be determined with *mysql_num_fields*, *mysql_fetch_field*, etc.