

Using MySQL Client Programs

This chapter covers the following exam topics:

- Invoking command-line client programs
- Specifying command-line options
- The `mysql` client
 - Using `mysql` interactively
 - Using script files with `mysql`
 - `mysql` client commands and SQL statements
 - Using the `---safe-updates` option
- Using `mysqlimport`
- Using `mysqldump` and reloading the dump
- Checking tables with `mysqlcheck` and `myisamchk`
- Using MySQLCC
- Using MySQL Connector/ODBC and MySQL Connector/J

Questions on the material in this chapter make up approximately 10% of the exam.

This chapter discusses general principles that are common to most MySQL client programs. It also describes how to use several specific types of clients:

- The interactive graphical client MySQLCC (MySQL Control Center). This general-purpose client provides a graphical interface to the MySQL server. It can be thought of as an extended graphical version of the character-based `mysql` client.

MySQLCC is still being actively developed at the time of publication of the first Core exam. For that reason, you are expected to be familiar with the general properties and capabilities of MySQLCC, but you are *not* expected to know all the ins and outs of the program.

- The character-based client programs. These run either interactively or perform their job based on command-line arguments with no further input from the user. Character-based clients discussed in this chapter include:
 - `mysql`, a general-purpose client for issuing queries and retrieving their results. It can be used interactively or in batch mode to read queries from a file.
 - `mysqladmin`, an administrative client that helps you manage the server.
 - `mysqlimport`, a program for importing text files into databases. It provides a command-line interface to the `LOAD DATA INFILE SQL` statement.
 - `mysqldump`, a program for dumping the contents of databases. It can be used to make database backups or to copy databases to other machines.
 - `mysqlcheck` and `mysamchk`, programs for checking and repairing tables. They're useful for checking and maintaining the integrity of certain types of tables.
- The MySQL Connector drivers. These drivers provide connectivity to the MySQL server for client programs:
 - MySQL Connector/ODBC, the MySQL driver for programs that use the ODBC (Open Database Connectivity) interface.
 - MySQL Connector/J, the MySQL driver for JDBC connectivity to Java programs.

3.1 Invoking Command-Line Client Programs

MySQL client programs can be invoked from the command line; for example, from a Windows console prompt or from a Unix shell prompt. When you invoke a client program, you can specify options to control its behavior. Some options tell the client how to connect to the MySQL server. Other options tell the program what actions to perform.

This section discusses the following option-related topics:

- The general syntax for specifying options
- How to use connection parameter options
- How to specify options in an option file

Most examples in this section use the `mysql` program, but the general principles apply to other MySQL client programs as well.

To determine the options supported by any MySQL program, invoke the program with the `--help` option. For example, to find out how to use `mysql`, use this command:

```
shell> mysql --help
```

To determine the version of a program, use the `--version` option. For example:

```
shell> mysql --version
mysql Ver 12.22 Distrib 4.0.18, for apple-darwin7.2.0 (powerpc)
```

This output indicates that the `mysql` client is from MySQL version 4.0.18.

3.1.1 Specifying Command-Line Options

Typically, you invoke MySQL client programs with options that indicate to the program what you want it to do. This section describes the general syntax for specifying options, as well as some of the options that are common to most MySQL clients.

3.1.1.1 Command Option Syntax

Options to MySQL programs have two general forms:

- Long options consist of a word preceded by double dashes.
- Short options consist of a single letter preceded by a single dash.

In many cases, a given option has both a long and a short form. For example, to display a program's version number, you can use the long `--version` option or the short `-V` option. These two commands are equivalent:

```
shell> mysql --version
shell> mysql -V
```

Options are case sensitive. `--version` is recognized by MySQL programs, but lettercase variations such as `--Version` or `--VERSION` are not. This applies to short options as well. `-V` and `-v` are both legal options, but mean different things.

Some options are followed by values. For example, when you specify the `--host` or `-h` option to indicate the host machine where the MySQL server is running, you must follow the option with the machine's hostname. For a long option, separate the option and the value by an equal sign (`=`). For short options, the option and the value can be, but need not be, separated by a space. The following three option formats are equivalent; each one specifies `myhost.example.com` as the host machine where the MySQL server is running:

```
--host=myhost.example.com
-h myhost.example.com
-hmyhost.example.com
```

In most cases, if you don't specify an option explicitly, a program will use a default value. Default values make it easier to invoke MySQL client programs because you need specify only those options for which the defaults are unsuitable. For example, the default server hostname is `localhost`, so if the MySQL server to which you want to connect is running on the local host, you need not specify `--host` or `-h` at all.

Exceptions to these option syntax rules are noted in the following discussion wherever relevant. The most important exception is that password options have a slightly different behavior than other options.

3.1.1.2 Connection Parameter Options

To connect to a server using a client program, the client must know upon which host the server is running. A connection may be established locally to a server running on the same host as the client program, or remotely to a server running on a different host. To connect, you must also identify yourself to the server with a username and password.

Each MySQL client has its own program-specific options, but all clients support a common set of options for making a connection to the MySQL server. This section describes the options that specify connection parameters, and how to use them if the default values aren't appropriate.

For command-line clients, all connection parameters are specified after the command name. The following discussion lists each option's long form and short form, as well as its default value. (You'll need to specify connection parameters for other types of client programs as well, such as the graphical `MySQLCC` client. Such a client might allow you to specify connection parameters on the command line, but might also provide an additional method of allowing you to indicate them, such as a dialog box.)

There are three options that indicate to the client where the server is running, as well as the type of connection to establish:

- `--host=host_name` or `-h host_name`

This option specifies the machine where the MySQL server is running. The value can be a hostname or an IP number. The hostname `localhost` means the local host (that is, the computer on which you're running the client program). On Unix, `localhost` is treated in a special manner. On Windows, the value `.` also means the local host and is treated in a special manner as well. For a description of this special treatment, refer to the discussion of the `--socket` option.

The default host value is `localhost`.

- `--port=port_number` or `-P port_number`

This option indicates the port number to which to connect on the server host; it applies only to TCP/IP connections. TCP/IP is used unless you connect using a hostname value of `.` on Windows or `localhost` on Unix.

The default MySQL port number is 3306.

- `--socket=socket_name` or `-S socket_name`

This option's name comes from its original use for specifying a Unix domain socket file. On Unix, for a connection to the host `localhost`, a client connects to the server using a Unix socket file. This option specifies the pathname of that file.

On Windows, the `--socket` option is used for specifying a named pipe. For Windows NT-based systems that support named pipes, a client can connect using a pipe by specifying `.` as the hostname. In this case, `--socket` specifies the name of the pipe. Pipe names aren't case sensitive. (Note that NT-specific MySQL servers don't enable named pipe connections by default; the server must be started with the `--enable-named-pipe` option.)

If this option is omitted, the default Unix socket file pathname is `/tmp/mysql.sock` and the default Windows pipe name is `MySQL`.

Two options provide identification information. These are the username and password of the account that you want to use for accessing the server. The server will reject a connection attempt unless you provide values for these parameters that correspond to an account that the server recognizes:

- `--user=user_name` or `-u user_name`

This option specifies the username for your MySQL account. To determine which account applies, the server uses the username value in conjunction with the name of the host from which you connect. This means that there can be different accounts with the same username, which can be used for connections from different hosts.

On Unix, client programs use your system login name as your default MySQL account username. On Windows, the default MySQL account name is `ODBC`.

- `--password=pass_value` or `-ppass_value`

This option specifies the password for your MySQL account. There is no default password. If you omit this option, your MySQL account must be set up to allow you to connect without a password.

MySQL accounts are set up using the `GRANT` statement, which is discussed in the “Professional Study Guide.”

Password options are special in two ways, compared to the other connection parameter options:

- You can omit the password value after the option name. This differs from the other connection parameter options, each of which requires a value after the option name. If you omit the password value, the client program will prompt you interactively for a password, as shown here:

```
shell> mysql -p
Enter password:
```

When you see the `Enter password:` prompt, type in your password and press Enter. The password isn't echoed as you type, to prevent other people from seeing it.

- If you use the short form of the password option (`-p`) and give the password value on the command line, there must be no space between the `-p` and the value. That is, `-ppass_val` is correct, but `-p pass_val` is not. This differs from the short form for other connection parameter options, where a space is allowed between the option and its value. (For example, `-h host_name` and `-h host_name` are both valid.) This exceptional requirement that there be no space between `-p` and the password value is a logical necessity of allowing the option parameter to be omitted.

Another option that affects how the connection setup occurs is `--compress` (or `-C`). This option causes data sent between the client and the server to be compressed before transmission and uncompressed upon receipt. The result is a reduction in the number of bytes sent over the connection, which can be helpful on slow networks. The cost is additional computational overhead for both the client and server to perform compression and uncompression. `--compress` and `-C` take no value after the option name.

Here are some examples that show how to specify connection parameters:

- Connect to the server using the default hostname and username values with no password:

```
shell> mysql
```
- Connect to the server on the local host with a username of `myname`, asking `mysql` to prompt you for a password:

```
shell> mysql --host=localhost --password --user=myname
```
- Connect with the same options as the previous example, but using the corresponding short option forms:

```
shell> mysql -h localhost -p -u myname
```
- Connect to the server at a specific IP address, with a username of `myname` and password of `mypass`:

```
shell> mysql --host=192.168.1.33 --user=myname --password=mypass
```
- Connect to the server on the local host, using the default username and password and compressing client/server traffic:

```
shell> mysql --host=localhost --compress
```

3.1.1.3 Using Option Files

As an alternative to specifying options on the command line, you can place them in an option file. The standard MySQL client programs look for option files at startup time and use any appropriate options they find there. Putting an option in an option file saves you time: You need not specify the option on the command line each time you invoke a program.

Options in option files are organized into groups, with each group preceded by a *[group-name]* line that names the group. Typically, the group name is the name of the program to which the group of options applies. For example, the `[mysql]` and `[mysqldump]` groups are for options to be used by `mysql` and `mysqldump`, respectively. The special `[client]` group can be used to specify options that you want all client programs to use. A common use for the `[client]` group is to specify connection parameters.

To write an option in an option file, use the long option format that you would use on the command line, but omit the leading dashes. Here's a sample option file:

```
[client]
host = myhost.example.com
compress

[mysql]
safe-updates
```

In this example, the `[client]` group specifies the server hostname and indicates that the client/server protocol should use compression for traffic sent over the network. Options in this group apply to all standard clients. The `[mysql]` group applies only to the `mysql` program. The group shown indicates that `mysql` should use the `--safe-updates` option.

Note that if an option takes a value, spaces are allowed around the `=` sign, something that isn't true for options specified on the command line.

Where an option file should be located depends on your operating system. The standard option files are as follows:

- On Windows, programs use the `my.ini` file in the Windows directory and the `C:\my.cnf` file.
- On Unix, the file `/etc/my.cnf` serves as a global option file used by all users. You can set up your own option file by creating a file named `.my.cnf` in your home directory.

To use an option file, create it as a plain text file using an editor. Client programs can access options from multiple option files, if they exist. It isn't an error for an option file to be missing.

To create or modify an option file, you must have write permission for it. Client programs need only read access.

To tell a program to read a specific option file instead of the standard option files, use the `--defaults-file` option. For example, to use the file `C:\my-opts` for `mysql` on Windows, invoke the program like this:

```
shell> mysql --defaults-file=C:\my-opts
```

If you use `--defaults-file`, it must be the first option after the command name.

If a program finds that an option is specified multiple times, either in the same option file or in multiple option files, the option value that occurs last takes precedence. Options specified on the command line take precedence over options found in option files.

3.1.2 Selecting a Default Database

For most client programs, you must specify a database so that the program knows where to find the tables that you want to use. The conventional way to do this is to name the database on the command line following any options. For example, to dump the contents of the `world` database to an output file named `world.sql`, you might run `mysqldump` like this:

```
shell> mysqldump --password --user=user_name world > world.sql
```

For the `mysql` client, a database name on the command line is optional. This is because you can explicitly indicate the database name for any table when you issue queries. For example, the following statement selects rows from the table `Country` in the `world` database:

```
mysql> SELECT * FROM world.Country;
```

To select or change the default database while running `mysql`, issue a `USE db_name` statement, where `db_name` is the name of the database you'd like to use. For example, the following statement makes `world` the default database:

```
mysql> USE world;
```

The advantage of selecting a default database with `USE` is that in subsequent queries you can refer to tables in that database without having to specify the database name. For example, with `world` selected as the default database, the following `SELECT` statements are equivalent, but the second is easier to write because the table name doesn't need to be qualified with the database name:

```
mysql> SELECT * FROM world.Country;
mysql> SELECT * FROM Country;
```

The default database is sometimes called the current database.

3.2 The `mysql` Client Program

The `mysql` client program enables you to send queries to the MySQL server and receive their results. It can be used interactively or it can read query input from a file in batch mode:

- Interactive mode is useful for day-to-day usage, for quick one-time queries, and for testing how queries work.
- Batch mode is useful for running queries that have been prewritten. It's especially valuable for issuing a complex series of queries that's difficult to enter manually, or queries that need to be run automatically by a job scheduler without user intervention.

3.2.1 Using `mysql` Interactively

To invoke `mysql` interactively from the command line, specify any necessary connection parameters after the command name:

```
shell> mysql -u user_name -p -h host_name
```

You can also provide a database name to select that database as the default database:

```
shell> mysql -u user_name -p -h host_name database_name
```

After `mysql` has connected to the MySQL server, it prints a `mysql>` prompt to indicate that it's ready to accept queries. To issue a query, enter it at the prompt. Complete the query with a statement terminator (typically a semicolon). The terminator tells `mysql` that the statement has been entered completely and should be executed. When `mysql` sees the terminator, it sends the query to the server and then retrieves and displays the result. For example:

```
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| world      |
+-----+
```

A terminator is necessary after each statement because `mysql` allows several queries to be entered on a single input line. `mysql` uses the terminators to distinguish where each query ends, and then sends each one to the server in turn and displays its results:

```
mysql> SELECT DATABASE(); SELECT VERSION();
+-----+
| DATABASE() |
+-----+
| world      |
+-----+
+-----+
| VERSION()  |
+-----+
| 4.0.18-log |
+-----+
```

Statement terminators are necessary for another reason as well: `mysql` allows a single query to be entered using multiple input lines. This makes it easier to issue a long query because you can enter it over the course of several lines. `mysql` will wait until it sees the statement terminator before sending the query to the server to be executed. For example:

```
mysql> SELECT Name, Population FROM City
      -> WHERE Country = 'IND'
      -> AND Population > 3000000;
```

```
+-----+-----+
| Name          | Population |
+-----+-----+
| Mumbai (Bombay) | 10500000 |
| Delhi         | 7206704  |
| Calcutta [Kolkata] | 4399819 |
| Chennai (Madras) | 3841396  |
+-----+-----+
```

More information about statement terminators can be found in section 3.2.3, “Statement Terminators.”

In the preceding example, notice what happens when you don’t complete the query on a single input line: `mysql` changes the prompt from `mysql>` to `->` to indicate that it’s still waiting to see the end of the statement. The full set of `mysql` prompts is discussed in section 3.2.4, “The `mysql` Prompts.”

If a statement results in an error, `mysql` displays the error message returned by the server:

```
mysql> This is an invalid statement;
ERROR 1064: You have an error in your SQL syntax.
```

If you change your mind about a statement that you’re composing, enter `\c` and `mysql` will return you to a new prompt:

```
mysql> SELECT Name, Population FROM City
-> WHERE \c
mysql>
```

To quit `mysql`, use `\q`, `QUIT`, or `EXIT`:

```
mysql> \q
```

3.2.2 Using Editing Keys in `mysql`

`mysql` supports input-line editing, which enables you to recall and edit input lines. For example, you can use the up-arrow and down-arrow keys to move up and down through previous input lines, and the left-arrow and right-arrow keys to move back and forth within a line. Other keys, such as Backspace and Delete, erase characters from the line, and you can type in new characters at the cursor position. To submit an edited line, press Enter.

On Windows, you might find that input-line editing does not work for Windows 95, 98, or Me.

`mysql` also supports tab-completion to make it easier to enter queries. With tab-completion, you can enter part of a keyword or identifier and complete it using the Tab key. This feature is supported on Unix only.

3.2.3 Statement Terminators

You may use any of several statement terminators to end a query. Two statement terminators are the semicolon character (;) and the \g sequence. They're equivalent and may be used interchangeably:

```
mysql> SELECT VERSION(), DATABASE();
+-----+-----+
| VERSION() | DATABASE() |
+-----+-----+
| 4.0.18-log | world      |
+-----+-----+
mysql> SELECT VERSION(), DATABASE()\g
+-----+-----+
| VERSION() | DATABASE() |
+-----+-----+
| 4.0.18-log | world      |
+-----+-----+
```

The \G sequence also terminates queries, but causes mysql to display query results in a different style. The new style shows each output row with each column value on a separate line:

```
mysql> SELECT VERSION(), DATABASE()\G
***** 1. row *****
VERSION(): 4.0.18-log
DATABASE(): world
```

The \G terminator is especially useful if a query produces very wide output lines. It can make the result much easier to read.

3.2.4 The mysql Prompts

The mysql> prompt displayed by mysql is just one of several different prompts that you might see when entering queries. Each type of prompt has a functional significance because mysql varies the prompt to provide information about the status of the statement you're entering. The following table shows each of these prompts:

Prompt	Meaning of Prompt
mysql>	Ready for new statement
->	Waiting for next line of statement
'>	Waiting for end of single-quoted string
">	Waiting for end of double-quoted string or identifier
`>	Waiting for end of backtick-quoted identifier

The `mysql>` prompt is the main (or primary) prompt. It signifies that `mysql` is ready for you to begin entering a new statement.

The other prompts are continuation (or secondary) prompts. `mysql` displays them to indicate that it's waiting for you to finish entering the current statement. The `->` prompt is the most generic continuation prompt. It indicates that you have not yet completed the current statement, for example, by entering `;` or `\G`. The `'>`, `">`, and ``>` prompts are more specific. They indicate not only that you're in the middle of entering a statement, but that you're in the middle of entering a single-quoted string, a double-quoted string, or a backtick-quoted identifier, respectively. When you see one of these prompts, you'll often find that you have entered an opening quote on the previous line without also entering the proper closing quote.

If in fact you did mistype the current query by forgetting to close a quote, you can cancel the query by entering the closing quote followed by the `\c` clear-query command.

3.2.5 Using Script Files with `mysql`

When used interactively, `mysql` reads queries entered at the keyboard. `mysql` can also accept input from a file. An input file containing SQL statements to be executed is known as a "script file" or a "batch file." A script file should be a plain text file containing statements in the same format that you would use to enter the statements interactively. In particular, each statement must end with a terminator.

One way to process a script file is to execute it with a `SOURCE` command from within `mysql`:

```
mysql> SOURCE input_file;
```

Notice that there are no quotes around the name of the file.

`mysql` executes the queries in the file and displays any output produced.

The file must be located on the client host where you're running `mysql`. The filename must either be an absolute pathname listing the full name of the file, or a pathname that's specified relative to the directory in which you invoked `mysql`. For example, if you started `mysql` on a Windows machine in the `C:\mysql\` directory and your script file is `my_commands` in the `C:\scripts` directory, either of the following `SOURCE` commands would tell `mysql` to run the file:

```
mysql> SOURCE C:\scripts\my_commands;  
mysql> SOURCE ..\scripts\my_commands;
```

The other way to execute a script file is by naming it on the `mysql` command line. Invoke `mysql` and use the `<` input redirection operator to specify the file from which to read query input:

```
shell> mysql db_name < input_file
```

If a statement in a script file fails with an error, `mysql` ignores the rest of the file. To execute the entire file regardless of whether errors occur, invoke `mysql` with the `--force` or `-f` option.

A script file can itself contain `SOURCE` commands to execute other files. But be careful not to create a `SOURCE` loop. For example, if `file1` contains a `SOURCE file2` command, `file2` should not contain a `SOURCE file1` command.

3.2.6 `mysql` Output Formats

By default, `mysql` produces output in one of two formats, depending on whether you use it in interactive or batch mode:

- When invoked interactively, `mysql` displays query output in a tabular format that uses bars and dashes to display values lined up in boxed columns.
- When you invoke `mysql` with a file as its input source on the command line, `mysql` runs in batch mode with query output displayed using tab characters between data values.

To override the default output format, use these options:

- `--batch` or `-B`
Produce batch mode (tab-delimited) output, even when running interactively.
- `--table` or `-t`
Produce tabular output format, even when running in batch mode.

In batch mode, you can use the `--raw` or `-r` option to suppress conversion of characters such as newline and carriage return to escape-sequences like `\n` or `\r`. In raw mode, the characters are printed literally.

To select a different output format than either of the default formats, use these options:

- `--html` or `-H`
Produce output in HTML format.
- `--xml` or `-X`
Produce output in XML format.

3.2.7 `mysql` Client Commands and SQL Statements

When you issue an SQL statement while running `mysql`, the program sends the statement to the MySQL server to be executed. `SELECT`, `INSERT`, `UPDATE`, and `DELETE` are examples of this type of input. `mysql` also understands a number of its own commands that aren't SQL statements. The `QUIT` and `SOURCE` commands that have already been discussed are examples of `mysql` commands. Another example is `STATUS`, which displays information about the current connection to the server, as well as status information about the server itself. Here is what a status display might look like:

```
mysql> STATUS;
-----
mysql Ver 12.22 Distrib 4.0.18, for apple-darwin7.2.0 (powerpc)

Connection id:          14498
Current database:      world
Current user:          myname@localhost
SSL:                   Not in use
Current pager:         stdout
Using outfile:         ''
Server version:        4.0.18-log
Protocol version:      10
Connection:            Localhost via UNIX socket
Client characterset:   latin1
Server characterset:   latin1
UNIX socket:           /tmp/mysql.sock
Uptime:                15 days 1 hour 9 min 27 sec

Threads: 4  Questions: 78712  Slow queries: 0  Opens: 786  Flush tables: 1
Open tables: 64  Queries per second avg: 0.061
-----
```

A full list of `mysql` commands can be obtained using the `HELP` command.

`mysql` commands have both a long form and a short form. The long form is a full word (such as `SOURCE`, `STATUS`, or `HELP`). The short form consists of a backslash followed by a single character (such as `\.`, `\s`, or `\h`). The long forms may be given in any lettercase. The short forms are case sensitive.

Unlike SQL statements, `mysql` commands cannot be entered over multiple lines. For example, if you issue a `SOURCE file_name` command to execute statements stored in a file, `file_name` must be given on the same line as `SOURCE`. It may not be entered on the next line.

By default, the short command forms are recognized on any input line, except within quoted strings. The long command forms aren't recognized except at the `mysql>` primary prompt. For example, `CLEAR` and `\c` both clear (cancel) the current command, which is useful if you change your mind about issuing the statement that you're currently entering. But `CLEAR` isn't recognized after the first line of a multiple-line statement, so you should use `\c` instead. To have `mysql` recognize the long command names on any input line, invoke it with the `--named-commands` option.

3.2.8 Using the `--safe-updates` Option

It's possible to inadvertently issue statements that modify many rows in a table or that return extremely large result sets. The `--safe-updates` option helps prevent these problems. The option is particularly useful for people who are just learning to use MySQL. `--safe-updates` has the following effects:

- `UPDATE` and `DELETE` statements are allowed only if they include a `WHERE` clause that specifically identifies which records to update or delete by means of a key value, or if they include a `LIMIT` clause.
- Output from single-table `SELECT` statements is restricted to no more than 1,000 rows unless the statement includes a `LIMIT` clause.
- Multiple-table `SELECT` statements are allowed only if MySQL will examine no more than 1,000,000 rows to process the query.

The `--i-am-a-dummy` option is a synonym for `--safe-updates`.

3.3 Using `mysqlimport`

The `mysqlimport` client program loads datafiles into tables. It provides a command-line interface to the `LOAD DATA INFILE` statement. That is, `mysqlimport` examines the options given on the command line and builds a `LOAD DATA INFILE` statement that corresponds to the actions specified by those options. It then connects to the server and, for each input file, issues the `LOAD DATA INFILE` statement that correctly loads the file into the appropriate table.

Because `mysqlimport` works this way, to use it most effectively, you should be familiar with the `LOAD DATA INFILE` statement. This section describes `mysqlimport` invocation syntax and how its options correspond to various clauses of the `LOAD DATA INFILE` statement. For a discussion specifically about `LOAD DATA INFILE`, you should also read Chapter 9, “Importing and Exporting Data.” The details in that chapter aren't repeated here.

Invoke `mysqlimport` from the command line as follows:

```
shell> mysqlimport options db_name input_file
```

`db_name` names the database containing the table to be loaded and `input_file` names the file that contains the data to be loaded.

`mysqlimport` uses the filename to determine the name of the table into which the data should be loaded. The program does this by stripping off any filename extension (the last period and anything following it); the result is then used as the table name. For example, `mysqlimport` treats a file named `City.txt` or `City.dat` as input to be loaded into a table named `City`.

If you name multiple files on the command line after the database name, `mysqlimport` issues a `LOAD DATA INFILE` statement for each file.

Each table to be loaded by `mysqlimport` must already exist, and each input file should contain only data values. `mysqlimport` isn't intended for processing files that consist of SQL statements (such files can be created with the `mysqldump` program). For instructions on processing an SQL-format dump file produced by `mysqldump` that contains SQL statements, see section 3.4, "Using `mysqldump`."

The *options* part of the `mysqlimport` command may include any of the standard connection parameter options, such as `--host` or `--user`. You'll need to supply these options if the default connection parameters aren't appropriate. `mysqlimport` also understands options specific to its own operation. Invoke `mysqlimport` with the `--help` option to see a complete list of the options that can be used to tell `mysqlimport` the actions you want it to perform.

By default, input files for `mysqlimport` are assumed to contain lines terminated by newlines, with each line containing tab-separated data values. For an input file that's in a different format, use the following options to tell `mysqlimport` how to interpret the file:

- `--lines-terminated-by=string`
string specifies the character sequence that each input line ends with. The default is `\n` (linefeed, also known as newline); other common line terminators are `\r` (carriage return) and `\r\n` (carriage return/linefeed pairs).
- `--fields-terminated-by=string`
string specifies the delimiter between data values within input lines. The default delimiter is `\t` (tab).
- `--fields-enclosed-by=char` or `--fields-optionally-enclosed-by=char`
char indicates a quote character that surrounds data values in the file. By default, values are assumed not to be quoted. Use this option if values are quoted. A common value for *char* is the double quote character (`"`). If quote characters enclose a data value, they're removed before the value is loaded into the table.
- `--fields-escaped-by=char`
By default, `\` within the input is taken as an escape character that signifies a special sequence. For example, if the `\N` sequence occurs alone in a field, it's interpreted as a NULL value. Use this option to specify a different escape character. To turn escaping off (no escape character), specify an empty value for *char*.

The preceding options give you the flexibility to load input files containing data in a variety of formats. Some examples follow; each one loads an input file named `City.txt` into the `City` table in the `world` database. Commands that are shown on multiple lines should be entered on a single line.

The following command loads a file that has lines ending in carriage return/linefeed pairs:

```
shell> mysqlimport --lines-terminated-by="\r\n" world City.txt
```

Note that the `--lines-terminated-by` value is quoted with double quotes. Format option values often contain special characters, such as backslash, that might have special meaning to your command interpreter. It might be necessary to quote such characters to tell your command interpreter to pass them, unchanged, to `mysqlimport`.

The syntax for specifying a double quote is trickier and depends on which command interpreter you use. The following command loads a datafile containing values quoted by double quote characters:

```
shell> mysqlimport --fields-terminated-by='"' world City.txt
```

This command should work on most Unix shells, which allow the double quote character to be quoted within single quotes. This doesn't work on Windows, where you must specify a double quote within a double-quoted string by escaping it:

```
shell> mysqlimport --fields-terminated-by="\\"" world City.txt
```

The following command loads a file that has data values separated by commas, and lines ending with carriage returns:

```
shell> mysqlimport --fields-terminated-by=,
                --lines-terminated-by="\r" world City.txt
```

Other `mysqlimport` options provide additional control over datafile loading. The following list discusses some of those you're likely to find useful:

- `--ignore` or `--replace`

These options tell `mysqlimport` how to handle input records that contain unique key values that are already present in the table. Such records result in duplicate-key errors and cannot be loaded by default. `--ignore` causes duplicates in the input file to be ignored. `--replace` causes existing records in the table to be replaced by duplicates in the input file.

- `--local`

By default, a datafile to be loaded into a table is assumed to reside on the server host, allowing the server to read the file directly. This is very efficient, but requires the user running `mysqlimport` to have the `FILE` privilege (a powerful privilege normally reserved for administrators). The `--local` option allows use of a datafile that's located locally on the client host where `mysqlimport` is invoked. With `--local`, `mysqlimport` reads the datafile and sends it over the network to the server. This allows `mysqlimport` to read any file on the client host to which the invoker has access, without requiring the invoker to have the `FILE` privilege. For the `--local` option to work, the server must be configured to allow local files to be transferred to it.

3.4 Using `mysqldump`

The `mysqldump` client program dumps table contents to files. The program is useful for making database backups or for transferring database contents to another server. `mysqldump` can produce SQL-format dump files that contain `CREATE TABLE` and `INSERT` statements for re-creating the dumped files or it can produce tab-delimited datafiles. This section describes how to produce SQL-format dump files. Knowledge of tab-delimited dump files is not needed for the Core exam; that topic is discussed in the “Professional Study Guide.”

3.4.1 General `mysqldump` Operation

`mysqldump` has three general modes of operation, depending on how it's invoked:

- By default, `mysqldump` interprets its first nonoption argument as a database name and dumps all the tables in that database. If any other arguments follow the database name, `mysqldump` interprets them as table names and dumps just those tables. The following command dumps the contents of all the tables in the `world` database into a file named `world.sql`:

```
shell> mysqldump world > world.sql
```

The contents of the `world.sql` file will begin something like this (statements to create and load the other tables in the database would follow the partial display shown here):

```
-- MySQL dump 9.10
--
-- Host: localhost    Database: world
-----
-- Server version    4.0.18-log
--
-- Table structure for table 'City'
--
CREATE TABLE City (
  ID int(11) NOT NULL auto_increment,
  Name char(35) NOT NULL default '',
  CountryCode char(3) NOT NULL default '',
  District char(20) NOT NULL default '',
  Population int(11) NOT NULL default '0',
  PRIMARY KEY (ID)
) TYPE=MyISAM;
--
-- Dumping data for table 'City'
--
```

```

INSERT INTO City VALUES (1, 'Kabul', 'AFG', 'Kabul', 1780000);
INSERT INTO City VALUES (2, 'Qandahar', 'AFG', 'Qandahar', 237500);
INSERT INTO City VALUES (3, 'Herat', 'AFG', 'Herat', 186800);
INSERT INTO City VALUES (4, 'Mazar-e-Sharif', 'AFG', 'Balkh', 127800);
INSERT INTO City VALUES (5, 'Amsterdam', 'NLD', 'Noord-Holland', 731200);
INSERT INTO City VALUES (6, 'Rotterdam', 'NLD', 'Zuid-Holland', 593321);
...

```

The following command names just the `City` and `Country` tables after the database name, so `mysqldump` dumps just those tables to a file called `city_country.sql`:

```
shell> mysqldump world City Country > city_country.sql
```

- With the `--databases` (or `-B`) option, `mysqldump` interprets any nonoption argument as a database name and dumps all the tables in each of the named databases. For example, the following command dumps both the `world` and `test` databases into a single file:

```
shell> mysqldump --databases world test > world_and_test.sql
```

- With the `--all-databases` (or `-A`) option, `mysqldump` dumps all tables in all databases. For example, this command writes a backup for all databases to the file `alldb.sql`:

```
shell> mysqldump --all-databases > alldb.sql
```

If you manage a lot of data, `alldb.sql` will be very large. Be sure that you have sufficient free disk space before issuing such a command.

`mysqldump` understands the standard connection parameter options, such as `--host` and `--user`. You'll need to supply these options if the default connection parameters aren't appropriate. `mysqldump` also understands options that provide more specific control over the dump operation. Invoke `mysqldump` with the `--help` option to see a list of available options. Those options described here are ones you're likely to find most useful:

- `--add-drop-table`

Instructs `mysqldump` to precede the dump output for each table with a `DROP TABLE` statement that drops the table. This option ensures that when you reload the dump output, the reload operation removes any existing copy of the table before re-creating it.

- `--all` or `-a`

Instructs `mysqldump` to produce `CREATE TABLE` statements that include all the MySQL-specific options (such as the table type and table comment) with which each table was created. By default, `mysqldump` does not include all these options, resulting in dump files that might be more portable for loading with a DBMS other than MySQL. With the `--all` option, tables created during reloading into MySQL will have the same options as the original tables.

- `--extended-insert` or `-e`

By default, `mysqldump` writes each row as a separate `INSERT` statement. This option produces multiple-row `INSERT` statements that add several rows to the table at a time. Multiple-row statements can be reloaded more efficiently, although they're less readable than single-row statements if you examine the dump output. They're also less portable and might not be understood by other database systems.

- `--no-create-db` or `-n`

Normally, when you run `mysqldump` with the `--all-databases` or `--databases` option, the program precedes the dump output for each database with a `CREATE DATABASE` statement to ensure that the database is created if it doesn't already exist. The `--no-create-db` option causes `CREATE DATABASE` statements not to be written. Note that their presence in the file is usually not a problem. They include an `IF NOT EXISTS` clause, so they're ignored when reloading the dump file for any database that does exist.

- `--no-create-info` or `-t`

This option suppresses the `CREATE TABLE` statement that normally precedes the `INSERT` statements containing a table's data. Use this option when you're interested in dumping only a table's data. The option is useful mostly when you plan to reload the data into tables that already exist.

- `--no-data` or `-d`

This option suppresses the `INSERT` statements containing table data. Use this option when you're interested in dumping only the `CREATE TABLE` statements that describe table structures. The `--no-data` option provides an easy way to get a dump file that can be processed to create empty tables with the same structure as the original tables.

- `--opt`

This option turns on a set of additional options to make the dump and reload operations more efficient. Specifically, it's equivalent to using the `--add-drop-table`, `--add-locks`, `--all`, `--quick`, `--extended-insert`, `--lock-tables`, and `--disable-keys` options together. Note that this option makes the output less portable and less likely to be understood by other database systems.

- `--quick`

This option tells `mysqldump` to write dump output as it reads each row from the server, which might be useful for large tables. By default, `mysqldump` reads all rows from a table into memory before writing the output; for large tables, this requires large amounts of memory, possibly causing the dump to fail.

3.4.2 Reloading `mysqldump` Output

To reload an SQL-format dump file produced by `mysqldump`, process it with `mysql`. For example, to make a copy of the `Country` table from the `world` database in the `test` database, you could issue these commands:

```
shell> mysqldump --opt world Country > dump.sql
shell> mysql test < dump.sql
```

`mysql` can read from a pipe, so you can combine the use of `mysqldump` and `mysql` into a single command. The preceding example can thus be written as one command:

```
shell> mysqldump --opt world Country | mysql test
```

This technique also can be used to copy databases or tables over the network to another server. For example, the following command uses a pipe to copy the `Country` table from the `world` database on the local host to the `world` database on the remote host `other.host.com`:

```
shell> mysqldump --opt world Country | mysql -h other.host.com world
```

3.5 Checking Tables with `mysqlcheck` and `myisamchk`

The `mysqlcheck` and `myisamchk` programs are used to check and repair tables (primarily MyISAM tables). They can help you keep your tables free from problems, or repair them if problems do occur—for example, if a MyISAM table becomes damaged as the result of a server crash.

`mysqlcheck` and `myisamchk` are similar in purpose, but they do have some differences. Here's a comparison of the two programs:

Both programs can check, repair, and analyze MyISAM tables. `mysqlcheck` also can optimize MyISAM tables, as well as check InnoDB tables and analyze BDB tables. There are certain operations that `myisamchk` can perform that `mysqlcheck` cannot, such as disabling or enabling indexes, although these operations aren't discussed in this study guide.

The two programs differ significantly in their mode of operation. `mysqlcheck` is a client program that communicates with the MySQL server over a network connection. It provides a command-line interface to the table-maintenance SQL statements supported by the server, such as `CHECK TABLE` and `REPAIR TABLE`. `mysqlcheck` determines which options were given on the command line, and then sends appropriate SQL statements to the MySQL server to perform the requested operation. This means that `mysqlcheck` requires the server to be running. It also means that you can use `mysqlcheck` to connect to remote servers. In contrast, `myisamchk` isn't a client program. It's a utility that operates directly on the files that represent MyISAM tables. This means that you must run `myisamchk` on the server host where those files are located. In addition, you need filesystem read privileges on those files for table checking operations, and write privileges for table repair operations.

The two programs also differ in their relationship with the server while they're running. With `mysqlcheck`, there's no problem of interaction with the server because `mysqlcheck` asks the server itself to do the work of checking and repairing the tables. With `myisamchk`, you

need to make sure that the server doesn't have the tables open and isn't using them at the same time. It's possible to get incorrect results or even to cause table damage if table files are used by `myisamchk` and the server simultaneously. The most certain way to avoid conflict is to bring the server down while running `myisamchk`. It's also possible to lock the tables while checking or repairing them with `myisamchk`, but the procedure is not described here. You can find the details in the *MySQL Reference Manual*.

The rest of this section describes how to use `mysqlcheck` and `myisamchk`.

`mysqlcheck` has three general modes of operation, depending on how it is invoked:

- By default, `mysqlcheck` interprets its first nonoption argument as a database name and checks all the tables in that database. If any other arguments follow the database name, `mysqlcheck` treats them as table names and checks just those tables. For example, the first of the following commands checks all the tables in the `world` database; the second checks just the `City` and `Country` tables in that database:

```
shell> mysqlcheck world
shell> mysqlcheck world City Country
```

- With the `--databases` (or `-B`) option, `mysqlcheck` interprets its nonoption arguments as database names and checks all the tables in each of the named databases. The following command checks the tables in both the `world` and `test` databases:

```
shell> mysqlcheck --databases world test
```

- With the `--all-databases` (or `-A`) option, `mysqlcheck` checks all tables in all databases:

```
shell> mysqlcheck --all-databases
```

The procedure for using `myisamchk` is quite different, due to the need to avoid using tables at the same time the server might be accessing them. Use `myisamchk` as follows:

1. Stop the server so that it cannot access the tables while you're working with them.
2. From a command prompt, change location into the database directory where the tables are located. This will be the subdirectory of the server's data directory that has the same name as the database containing the tables you would like to check. (The reason for changing location is to make it easier to refer to the table files. You can skip this step if you like, but you'll have to specify to `myisamchk` the directory where the tables are located.)
3. Invoke `myisamchk` with options indicating the operation you want performed, followed by arguments that name the tables on which `myisamchk` should operate. Each of these arguments can be either a table name or the name of the table's index file. An index filename is the same as the table name, plus an `.MYI` suffix.

The default `myisamchk` operation is to check tables. If that's what you want to do, no options are necessary and you need only name the table or tables to be checked. For example, to check a table named `City`, either of these commands will do:

```
shell> myisamchk City
shell> myisamchk City.MYI
```

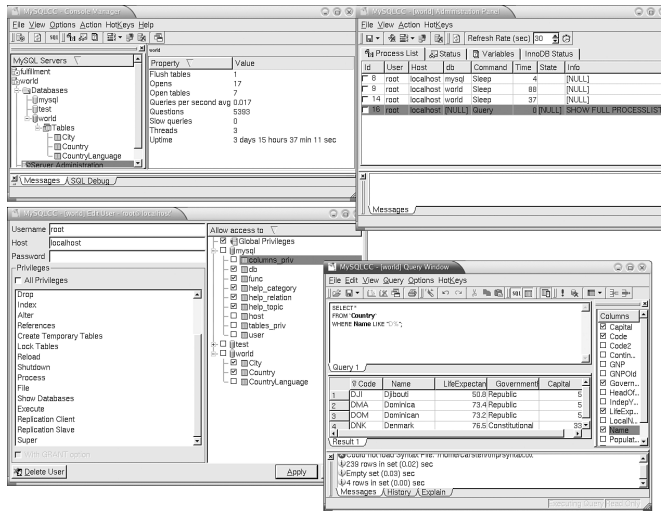
To repair a table, use the `--recover` option:

```
shell> myisamchk --recover City
```

`mysqlcheck` and `myisamchk` both take many options to control the type of table maintenance operation performed. The following list summarizes some of the more commonly used options. For the most part, the list contains options that are understood by both programs. Where this isn't the case, it's noted in the relevant option description.

- `--analyze` or `-a`
Analyze the distribution of key values in the table. This can improve performance of queries by speeding up index-based lookups.
- `--auto-repair` (`mysqlcheck` only)
Repair tables automatically if a check operation discovers problems.
- `--check` or `-c`
Check tables for problems. This is the default action if no other operation is specified.
- `--check-only-changed` or `-C`
Skip table checking except for tables that have been changed since they were last checked or tables that haven't been properly closed. The latter condition might occur if the server crashes while a table is open.
- `--fast` or `-F`
Skip table checking except for tables that haven't been properly closed.
- `--extended` (for `mysqlcheck`), `--extend-check` (for `myisamchk`), or `-e` (for both programs)
Run an extended table check. For `mysqlcheck`, when this option is given in conjunction with a repair option, a more thorough repair is performed than when the repair option is given alone. That is, the repair operation performed by `mysqlcheck --repair --extended` is more thorough than the operation performed by `mysqlcheck --repair`.
- `--medium-check` or `-m`
Run a medium table check.
- `--quick` or `-q`
For `mysqlcheck`, `--quick` without a repair option causes only the index file to be checked, leaving the datafile alone. For both programs, `--quick` in conjunction with a repair option causes the program to repair only the index file, leaving the datafile alone.
- `--repair` (for `mysqlcheck`), `--recover` (for `myisamchk`), or `-r` (for both programs)
Run a table repair operation.

3.6 Using MySQLCC



The MySQL Control Center (MySQLCC).

The MySQLCC (MySQL Control Center) program provides a graphical interface to the MySQL server. Precompiled binaries are currently available for Windows and Linux. MySQLCC can also be compiled from source.

The design goals of the MySQLCC graphical client are:

- To create a cross-platform client program that's as easy to use for novice users as the market-leading graphical database front ends.
- To create a client program that's as practical to use for experienced `mysql` client users as it is for novices, so that all users will prefer MySQLCC whenever the operating system provides them with a graphical user interface.
- To cover all common functions of the character-based clients while adding some frequently needed analysis tools.

MySQLCC is downloaded separately from MySQL Server and was at Version 0.9.4 (Beta) at the beginning of 2004.

The following list describes some of the features offered by MySQLCC:

- Interactive query entry and editing, including syntax highlighting and tab-completion. Syntax highlighting helps you see and understand the structure of queries more readily. With tab-completion, you can enter part of a keyword or identifier and complete it using the `Tab` key.
- A status window that provides easy access to the server's variables and status indicators. The window displays the output from the `SHOW VARIABLES` and `SHOW STATUS` statements.

- Capabilities for administering the server. For example, if you connect to the server using an account that has the appropriate privileges, you can flush logs, kill client connections, or tell the server to shut down.
- Capabilities for managing and administering user accounts.

3.7 MySQL Connectivity Drivers

MySQL AB provides drivers that aren't programs in themselves, but act as bridges to the MySQL server for client programs that speak a particular protocol. This section discusses Connector/ODBC and Connector/J, two drivers that provide connectivity to the MySQL server for programs that communicate using ODBC and JDBC, respectively.

The MySQL connectors are available for Windows and Unix. To use a connector, you must install it on the client host. It isn't necessary for the server to be running on the same machine or for the server to be running the same operating system as the client. This means that MySQL connectors are very useful for providing MySQL connectivity in heterogeneous environments. For example, people who use Windows machines can run ODBC or JDBC applications as clients that access MySQL databases located on a Linux server host.

3.7.1 MySQL Connector/ODBC

MySQL Connector/ODBC acts as a bridge between the MySQL server and client programs that use the ODBC standard. It provides a MySQL-specific driver for ODBC so that ODBC-based clients can access MySQL databases.

MySQL Connector/ODBC is based on the C client library. It converts ODBC calls made by the client program into C API operations that communicate with the server.

3.7.2 MySQL Connector/J

MySQL Connector/J is similar in spirit to Connector/ODBC, but is used by JDBC-based Java programs. It implements the client/server communication protocol directly and isn't based on the C client library. MySQL Connector/J converts JDBC calls made by the client program into the appropriate protocol operations.

3.8 Exercises

Question 1:

If you connect to a local server on a Unix machine using the hostname `localhost`, will the connection be made using TCP/IP or a Unix socket file? How will the connection be made if you use the local host's actual name?

Question 2:

Which connection parameters identify you to the MySQL server?

Question 3:

You want to execute a number of prewritten queries using the MySQL server running on the host `db.myexample.com`. Your `mysql` username is `juan` and you want to be prompted for your password. The prewritten queries are stored in the file `/tmp/queries.sql` and you want `world` to be the default database. What command do you issue?

Question 4:

Having connected to a server with the `mysql` client program, you want to run a query and display its output vertically. What statement terminator do you use to end the query?

Question 5:

Using `mysql`, you're about to enter a record into a table that has only one column. You've typed `INSERT INTO tbl VALUES ('teststring` and pressed Enter. Now the prompt looks like this: `'>`. What do you enter to perform the following operations?

- a. Send a valid query to the server
- b. Cancel the query

Question 6:

In an interactive session using `mysql`, you want to insert records into a table in the `test` database using a text file containing `INSERT` statements. The file is named `tbl_import.sql` and resides in `/tmp`. What command do you use to process the file in the following ways?

- a. Within the `mysql` session
- b. From the shell of your operating system

Question 7:

You're in the process of entering a long SQL statement in `mysql` and you decide not to send it to the server. What do you type to cancel the statement?

Question 8:

How do you back up the tables `City` and `Country` from the `world` database into the file `/tmp/world_backup.sql` using a single `mysqldump` command?

Question 9:

How do you back up the databases `db1` and `db2` into the file `/tmp/db1_db2_backup.sql` with a single `mysqldump` command that ensures existing tables will be overwritten when you restore them?

Question 10:

What are the advantages and disadvantages of the multiple-row INSERT statements `mysqldump` can produce?

Question 11:

You want to produce a backup of the `test` database into the file `/backups/structure.sql` using `mysqldump`. However, the purpose of the backup file is to create an empty copy of each table on another server. What command do you issue?

Question 12:

How do you restore a backup of tables in the `test` database from a backup file called `/backups/back.sql`? What options can you use with `mysqldump` to ensure existing tables will be overwritten when the tables are restored? If you don't use this `mysqldump` option, what option can you use when you restore the tables to ensure that the restore operation doesn't stop when it finds that a table already exists in the database?

Question 13:

Describe the main difference between `mysqlcheck` and `myisamchk`.

Question 14:

For which table types can you use `mysqlcheck`?

Question 15:

For which table types can you use `myisamchk`?

Question 16:

In addition to providing access to your database, `MySQLCC` serves as a graphical tool for server administration.

- What administrative actions does `MySQLCC` enable you to perform?
- What types of information about the server does `MySQLCC` provide?
- Besides the server, what else can you administer with `MySQLCC`?

Answers to Exercises

Answer 1:

- A Unix socket file will be used.
- When using the actual hostname, TCP/IP will be used.

Answer 2:

`--user` (or `-u`) and `--password` (or `-p`).

Answer 3:

```
shell> mysql -h db.example.com -p -u juan world < /tmp/queries.sql
```

Answer 4:

Use the \G sequence to display query output in vertical format.

Answer 5:

- a. One possibility is ');
- b. Another is to use the clear-query sequence: '\c

The '> prompt indicates that you began a single-quoted string but haven't finished it. Thus, you *must* enter the terminating single quote to finish the string regardless of whether you then want to enter the rest of the query or enter the cancel-query command.

Answer 6:

- a. To process the file within the `mysql` session, use the `SOURCE` command:

```
mysql> SOURCE /tmp/tbl_import.sql;
```

(Note that the semicolon is optional for the `SOURCE` command.) If `test` isn't the default database, you must first issue a `USE test` statement before processing the file.

- b. To process the file from the shell of the operating system, invoke `mysql` and direct it to read from the file:

```
shell> mysql test < /tmp/tbl_import.sql
```

You might also have to specify hostname, username, and password.

Answer 7:

To cancel a statement that you are entering, use the \c sequence.

Answer 8:

```
shell> mysqldump world City Country > /tmp/world_backup.sql
```

You might also have to specify hostname, username, and password.

Answer 9:

```
shell> mysqldump --add-drop-table --databases db1 db2 > /tmp/db1_db2_backup.sql
```

You might also have to specify hostname, username, and password.

Answer 10:

Advantages: Multiple-row statements can be reloaded more efficiently.

Disadvantages: Multiple-row statements are less readable and less easily ported to other database management systems.

Answer 11:

```
shell> mysqldump --no-data test > /backups/structure.sql
```

You might also have to specify hostname, username, and password.

Answer 12:

To restore tables, you can use this command:

```
shell> mysql test < /backups/back.sql
```

You might also have to specify hostname, username, and password.

To ensure that existing tables will be overwritten for the restore operation, you can use this command when dumping the tables:

```
shell> mysqldump --add-drop-table test > /backups/back.sql
```

To restore backups made without the `--add-drop-table` option, you can use the `mysql --force`, which will continue processing queries in batch mode even if errors occur.

Answer 13:

The main difference between `mysqlcheck` and `myisamchk` lies in how they access tables:

- `mysqlcheck` is a client program; it determines which options were given on the command line, and then sends appropriate SQL statements to the MySQL server to perform the requested operation.
- `myisamchk` is not a client program for the server. It performs operations on tables by accessing the table files directly.

Answer 14:

`mysqlcheck` can perform operations on MyISAM, InnoDB, and BDB tables.

Answer 15:

`myisamchk` works only for MyISAM tables.

Answer 16:

- a. MySQLCC enables you to kill client connections, issue flush commands, and shut down the server.
- b. MySQLCC has a server information window that provides single-click access to the same system and status variables that you get when issuing `SHOW VARIABLES` and `SHOW STATUS` statements manually.
- c. MySQLCC also enables you to administer user accounts.