Part 1. MySQL Developer I Exam

Table of Contents

Chapter 1. Client/Server Concepts

Almost all examples and exercises in this study guide use the *world database* as the sample data set. The accompanying CD-ROM contains the data for this database and instructions that describe how to create and populate the database for use with your own MySQL installation.

Question 1:

Is the following statement true or false?

All of the MySQL client programs and utilities communicate with the MySQL server.

Question 2:

Is the following statement true or false?

When running a MySQL server under Windows, client programs accessing that server also must run under Windows.

Question 3:

Is the following statement true?

All MySQL programs read startup options from plain text option files named `my.ini`, `my.cnf`, or `.my.cnf`.

Question 4:

Is the following statement true or false?

A command-line program commonly used to communicate with the server is called `mysqld`.

Question 5:

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 6:

Which connection parameters identify you to the MySQL server?

Question 7:

Suppose that you invoke `mysql` with the `-h localhost` option.

a. Will `mysql` establish a connection to a local server or a remote server?

b. Will `mysql` use a Unix socket file, a Windows named pipe, Windows shared memory, or TCP/IP?

c. Will `mysql` work only for a specific operating system?

Question 8:

Suppose that you invoke `mysql` with the `-h .` option.

a. Will `mysql` establish a connection to a local server or a remote server?

b. Will `mysql` use a Unix socket file, a Windows named pipe, Windows shared memory, or TCP/IP?

c. Will `mysql` work only for a specific operating system?

Question 9:

Suppose that you invoke `mysql` with the `-h 127.0.0.1` option.

a. Will `mysql` establish a connection to a local server or a remote server?

b. Will `mysql` use a Unix socket file, a Windows named pipe, Windows shared memory, or TCP/IP?

c. Will `mysql` work only for a specific operating system?

Question 10:

Suppose that you invoke `mysql` with the `-h 192.168.10.1` option.

a. Will `mysql` establish a connection to a local server or a remote server?

b. Will `mysql` use a Unix socket file, a Windows named pipe, Windows shared memory, or TCP/IP?

c. Will `mysql` work only for a specific operating system?

Question 11:

Suppose that you invoke `mysql` with no command-line options.

a. Will `mysql` establish a connection to a local server or a remote server?

b. Will `mysql` use a Unix socket file, a Windows named pipe, Windows shared memory, or TCP/IP?

c. Will `mysql` work only for a specific operating system?

Question 12:

Which of the following statements are true?

a. The SQL mode is set for the server, so it affects all clients that connect to the server.

b. If you want to set two SQL modes (for example, the `STRICT_ALL_TABLES` and `ERROR_FOR_DIVISION_BY_ZERO` modes), you must issue two `SET sql_mode` statements.

c. Unless explicitly declared as *global*, setting SQL modes affects only the client that sets the modes.

d. SQL modes affect the behavior of the server; for example, they influence the way in which the server handles invalid input data.

e. SQL modes affect the features that the server provides for a client; for example, you could turn `InnoDB` support on and off using SQL modes.

Answers to Exercises

Answer 1:

False. There are MySQL utilities that don't communicate with the server, but work directly on `MyISAM` table files. Examples include `myisamchk` and `myisampack`.

Answer 2:

False. MySQL can be used in heterogeneous environments. For example, a server running on a Unix host can be accessed by clients running on Windows machines.

Answer 3:

False. The MySQL server and the client programs and utilities provided by MySQL AB read options from the `my.ini`, `my.cnf`, and `.my.cnf` option files, but the MySQL GUI tools (MySQL Query Browser and MySQL Administrator) provide an interactive interface for specifying options and use their own XML files for storing information such as connection parameters.

Answer 4:

False. The most commonly used command-line program is called `mysql`, not `mysqld`. The latter is the MySQL server.

Answer 5:

1. A Unix socket file will be used.

2. When using the actual hostname, TCP/IP will be used.

Answer 6:

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

Answer 7:

`mysql -h localhost` establishes a local connection. On Unix-like operating systems, the connection is established through a Unix socket file. Under Windows, the connection is made using TCP/IP. `mysql -h localhost` can be used on any platform.

Answer 8:

`mysql -h .` tries to establish a local connection using shared memory, or, in case this fails, a named pipe. Shared memory and named pipes are available only under Windows, so the command will work only on Windows, and only if the server was started with the `--shared-memory` and `--enable-named-pipe` options.

Answer 9:

`mysql -h 127.0.0.1` establishes a local connection that uses TCP/IP on any operating system.

Answer 10:

A connection to a server running on a machine specified by an IP number will establish a connection via TCP/IP on any operating system. Whether this is a local or remote connection depends on whether `192.168.10.1` is the IP number of the local host.

Answer 11:

If no host is specified, the client will try to connect locally. Under Unix, this means it will use a Unix socket file. Under Windows, it will try to use shared memory first, then a named pipe, and TCP/IP if that fails. (Shared memory and named pipes will be tried only if the server has been started with these options.)

Answer 12:

a. False. That statement would be true only for SQL modes that are set globally, and for clients that connect after the SQL modes are set.

b. False. If you want to set two SQL modes (for example, set the `STRICT_ALL_TABLES` and `ERROR_FOR_DIVISION_BY_ZERO` modes), you have to set them using *one* statement, like this:

   ```
   SET sql_mode = 'STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
   ```

c. True. Unless explicitly declared as *global*, setting SQL modes affects only the client that sets the modes.

d. True. SQL modes affect the behavior of the server; for example, they influence the way in which the server handles invalid input data.

e. False. SQL modes don't affect the features that the server provides for a client. For example, you can turn `InnoDB` support on and off only at server startup.

Chapter 2. The `mysql` Client Program

Almost all examples and exercises in this study guide use the *world database* as the sample data set. The accompanying CD-ROM contains the data for this database and instructions that describe how to create and populate the database for use with your own MySQL installation.

Question 1:

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 2:

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 3:

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 4:

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 (command line) of your operating system

c. In batch mode so that the file is fully processed even if errors occur

Question 5:

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 6:

Which of the following commands will display help for the `SHOW` statement?

1. From the command line, issue `mysql -e 'help show'`.

2. From the command line, start `mysql`, and then issue `help show`.

3. From the command line, issue `mysqladmin -e 'help show'`.
4. From the command line, issue `mysqladmin --help show`.
5. From the command line, issue `mysqlshow show`..

Answers to Exercises

Answer 1:

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

Answer 2:

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

Answer 3:

a. One possibility is `');`
b. Another is to use the clear-statement 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 clear-statement command.

Answer 4:

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

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

   The semicolon is optional for this 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 options for hostname, username, and password.

c. To process a file completely even if errors occur, invoke `mysql` with the `--force` option:

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

Answer 5:

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

Answer 6:

The following commands will display help on the `SHOW` statement:

1. From the command line, issue `mysql -e 'help show'`.
2. From the command line, start `mysql`, and then issue `help show`.

Chapter 3. MySQL Query Browser

Almost all examples and exercises in this study guide use the *world database* as the sample data set. The accompanying CD-ROM contains the data for this database and instructions that describe how to create and populate the database for use with your own MySQL installation.

Question 1:

Which of these tasks can be performed by both MySQL Administrator and MySQL Query Browser?

a. Create databases (schemas)

b. Create tables

c. Modify table records

Question 2:

How do you create a view using MySQL Query Browser?

Question 3:

What methods can you use to create a query in MySQL Query Browser?

Question 4:

What is required for MySQL Query Browser to be able to join `InnoDB` tables automatically?

Question 5:

What is the default database (schema)?

Question 6:

How many ways can you execute a query?

Question 7:

What is required to edit a result set?

Question 8:

What different ways can you use to divide up the result area?

Question 9:

What does splitting the result area allow you to do?

Question 10:

In MySQL Query Browser, how can you search a result set?

Question 11:

Can you edit a table's structure (columns, indexes, and so forth) in MySQL Query Browser, or do you have to switch to MySQL Administrator to do this?

Question 12:

MySQL Query Browser stores connection profiles in a file named `mysqlx_user_connections.xml`. Which statements about this file are true?

a. The file stores connection profiles for MySQL Query Browser only.

b. The file is platform-specific, but you may copy it (for example, from one Windows machine to another Windows machine).

c. You can edit the file's contents only in the Connection dialog.

d. You can edit the file's contents in the connection editor.

Question 13:

Which statements about the Options dialog are true?

a. You can configure MySQL Query Browser-specific options using the Options dialog in MySQL Administrator, and the other way around.

b. You can manage connection profiles for MySQL Query Browser and for MySQL Administrator in the Options dialog of either tool.

c. You can configure the MySQL Table Editor that is used by both tools in the Options dialog of either tool.

Answers to Exercises

Answer 1:

Both tools can create databases (schemas) and tables, but only MySQL Query Browser can modify table records.

Answer 2:

To create a view, use either of these methods:

- Enter a `CREATE VIEW` statement and click `Execute`.
- Create a `SELECT` query, execute it, and click the `Create View` button.

Answer 3:

To create a query, you can use any of these methods:

- Enter the query directly into the query area.
- Double-click on a table to create a `SELECT *` query in the query area.
- Click and drag a table to the query area, selecting the type of statement to create (Select, Join, Left Join, Insert, Delete).

Answer 4:

The tables must have a foreign key relationship.

Answer 5:

It is the database (schema) against which all queries are applied.

Answer 6:

You can execute a query using any of these methods:

- Use keyboard shortcuts (for example, `Ctrl+E`).
- Click the `Execute` button.
- Choose an execution option from the Query menu.

Answer 7:

- The result set must have been retrieved from a single table, and the table must have a primary key.
- You must click the `Edit` button in the result area.

Answer 8:

- You can add a new tab to use for displaying query results.
- You can split an individual tab vertically.
- You can split an individual tab horizontally.

Answer 9:

You may compare results that are, for example, displayed in the left and right halves of the result area, by clicking the `Compare` button, or you could perform a master-detail analysis.

Answer 10:

The Search and Replace dialogs can be accessed by clicking the `Search` button below the result area. Any results can be searched, such as results selected from single tables, joined tables, or views.

Answer 11:

Both MySQL Administrator and MySQL Query Browser provide access to the MySQL Table Editor, which allows you to edit a table's structure. In MySQL Query Browser, right-click on a table name in the schema (database) browser and select `Edit Table`.

Answer 12:

Only the last statement is true. Connection profiles stored in the `mysqlx_user_connections.xml` file are shared among various MySQL tools; for example, MySQL Administrator uses the same information. The information is stored in plain text (in XML format) and can therefore be used by any MySQL Query Browser (or MySQL Administrator) installation, independent of the operating system. The file's contents can be edited using a plain text editor, in the Connection dialog, or in the connection editor.

Answer 13:

The Browser section appears only when you are running MySQL Query Browser, and the Administrator section appears only when you are running MySQL Administrator. Thus, you cannot configure MySQL Query Brower-specific options in MySQL Administrator, or the other way around. The other two statements are true.

Chapter 4. MySQL Connectors

Almost all examples and exercises in this study guide use the *world database* as the sample data set. The accompanying CD-ROM contains the data for this database and instructions that describe how to create and populate the database for use with your own MySQL installation.

Question 1:

Where does MySQL Connector/ODBC have to be installed?

a. Connector/ODBC must be installed on every client host where programs run that should use that connector.

b. Connector/ODBC has to be installed on the server host only. This will make it available for all clients connection from remote hosts.

c. Connector/ODBC has to be installed both on the server host and on all client hosts.

Question 2:

Which of the following statements is true?

a. MySQL Connectors are available for all operating systems that MySQL supports.

b. MySQL Connectors are shipped together with MySQL server distributions.

c. MySQL Connectors are shipped with the MySQL GUI tools.

d. MySQL Connectors are shipped separately from MySQL server distributions by MySQL AB.

e. MySQL Connectors are shipped separately from MySQL server distributions by third parties.

Question 3:

Is the following statement true? MySQL Connector/NET runs on Windows only.

Question 4:

Is the following statement true? All MySQL Connectors are based on MySQL's C API and are implemented using the MySQL C client library.

Question 5:

Is the following statement true? Like all other MySQL programs, MySQL Connectors are written in C.

Answers to Exercises

Answer 1:

MySQL Connector/ODBC must be installed on every *client host* where programs run that should use that connector. It must be installed on the server host only if there are programs running on the server host that should connect to the server using Connector/ODBC, because in that case the server host is also the client host.

Answer 2:

MySQL Connectors are shipped separately from MySQL server distributions by MySQL AB.

Answer 3:

MySQL Connector/NET runs on every platform that implements the .NET framework. Besides Windows, this includes Linux systems on which Mono is installed.

Answer 4:

False. MySQL Connector/NET and Connector/J are examples of connectors that do not use C client library but implement the client/server protocol directly.

Answer 5:

False. For example, MySQL Connector/NET is written in C#, and MySQL Connector/J is written in Java.

Chapter 5. Data Types

Almost all examples and exercises in this study guide use the *world database* as the sample data set. The accompanying CD-ROM contains the data for this database and instructions that describe how to create and populate the database for use with your own MySQL installation.

Question 1:

If you want to store monetary values (for example, values representing U.S. dollar-and-cent amounts such as $48.99), which data type should you use to avoid rounding errors?

Question 2:

Which data type is more space-efficient: `CHAR(100)` or `VARCHAR(100)`?

Question 3:

How do you make a `CHAR` or `VARCHAR` column case sensitive?

Question 4:

What's the difference between a string value that consists of characters and a string value that consists of bytes?

Question 5:

In a table `population`, you want to store the number of inhabitants of cities. Storage space is at a premium. You expect the maximum population to be 15,000,000 for a city. Which data type (and desired column attributes) would you use? What's the storage requirement for this data type?

Question 6:

In a table `user`, you have a comment column to store remarks. For each remark, you want to be able to store up to 2,000 characters. What data type would you use, and what's the storage requirement for each row if the average remark is 300 characters long?

Question 7:

You have a table in which you want to store birthdays of historical persons, and you decide to use the `DATE` data type to store the information. What's the earliest birthday you can store?

Question 8:

You perform the following `INSERT` operations on table `datetest`, which has a single `DATE` column called `d` with a default value of `NULL`:

```
INSERT INTO datetest VALUES ('10-02-08');

INSERT INTO datetest VALUES ('69-12-31');

INSERT INTO datetest VALUES ('70-01-01');
```

What data value will actually be stored in the table for each statement? Provide a short explanation. (Hint: This is independent from the SQL mode.)

Question 9:

You perform the following `INSERT` operation on table `datetest`, which has a single `DATE` column called `d` with a default value of `NULL`:

```
INSERT INTO datetest VALUES ('12:00:00');
```

What data value will actually be stored in the table? Provide a short explanation.

Question 10:

What are the advantages of floating-point over fixed-point data types?

a. Floating-point types can be processed using the processor's native binary format.

b. Floating-point types can be processed faster.

c. Floating-point types aren't subject to rounding errors, whereas fixed-point data types are.

d. Floating-point types always take less storage space than fixed-point data types.

Question 11:

Consider this column declaration:

```
bit_column BIT(4)
```

Which of the following statements are true?

a. The storage requirement is 4 bytes per value.

b. The storage requirement is approximately 1 byte per value.

c. The range of values is `0` to `15`.

d. the range of values is `0` to `4`.

Question 12:

What's the storage requirement of a `CHAR(10)` column?

Question 13:

The `utf8` character set has a variable storage requirement per character, depending on what character it is. Characters may take between one and three bytes. Does this mean that a `CHAR(10)` column has a variable length (in bytes) if its character set is `utf8`?

Question 14:

Are the following statements true? The character set of a column is determined by the character set of its table. The same holds for its collation.

Question 15:

If a column's collation is case sensitive, does this mean it's also accent sensitive?

Question 16:

Is the following statement true? Comparisons for binary strings are the same as comparisons for non-binary strings that use a binary collation.

Question 17:

What's the maximum length of a `VARCHAR` column?

a. 255 characters

b. 255 bytes

c. 65,535 characters (or a few less due to internal restrictions)

d. 65,535 bytes (or a few less due to internal restrictions)

Question 18:

Which statements are true?

a. `TIMESTAMP` values are displayed in the same format as `DATETIME` values.

b. `TIMESTAMP` values are displayed in `'YYYY-MM-DD hh:mm:ss'` format.

c. `TIMESTAMP` values are stored using the server's local time zone.

d. Legal `TIMESTAMP` values range from the beginning of the year 1970 (UTC) to a date in the year 2037.

Question 19:

`CREATE TABLE timestamptest1 (ts1 TIMESTAMP, i INT)` will create a table that has the same `TIMESTAMP` behavior for its first column as in MySQL versions prior to 4.1. What's that behavior, and how would you declare `ts1` explicitly to have that behavior for versions of MySQL from 4.1 on?

Question 20:

Assume that you've just created this table:

```
CREATE TABLE timestamptest (
 ts1 TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
 i INT
);
```

When you look at its structure, you will notice that the `TIMESTAMP` column is declared `NOT NULL`. What happens if you insert these records:

```
mysql> INSERT INTO timestamptest SET ts1=NULL, i=10;
mysql> INSERT INTO timestamptest SET ts1=0, i=11;
mysql> INSERT INTO timestamptest SET ts1='', i=12;
```

Question 21:

MySQL will make context-specific data type conversions not only when working with column values, but also when working with functions and operators that expect specific data types. For example, the `CONCAT()` function expects data of a string type, whereas the `+` operator expects data of a numeric type. What value will result from the following operation? Give a short explanation.

```
SELECT CONCAT(1,1+1);
```

Question 22:

What's the largest value you can store in a `TINYINT(2)` column?

Question 23:

What data types would you choose for a table that contains pictures with a maximum data length of 10MB, and remarks with a maximum length of 250 characters?

Question 24:

How much space is required to store a value that is 2,000 bytes long in a `BLOB` column?

Question 25:

How would you perform the following operations? Check what the server and the client time zones are, change the client's time zone to Berlin (`+1`), then check again what the client zone settings is.

Question 26:

Consider the following session listing:

```
mysql> CREATE TABLE ts_tz_test (ts TIMESTAMP);
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT NOW(); INSERT INTO ts_tz_test VALUES (NOW());
+---------------------+
| NOW()               |
+---------------------+
| 2005-05-25 16:30:34 |
+---------------------+
1 row in set (0.06 sec)

Query OK, 1 row affected (0.00 sec)

mysql> SELECT ts FROM ts_tz_test;
+---------------------+
| ts                  |
+---------------------+
| 2005-05-25 16:30:34 |
+---------------------+
1 row in set (0.00 sec)

mysql> SET time_zone = '+05:00';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT ts FROM ts_tz_test;
+---------------------+
| ts                  |
+---------------------+
| 2005-05-25 20:30:34 |
+---------------------+
1 row in set (0.00 sec)
```

What's the explanation for the change in the displayed value that occurs after changing the session time zone setting?

a. When changing the session time zone, the server changes the value stored in the `ts` column accordingly; this would also have happened if you had changed the global time zone.

b. When changing the session time zone, the server changes the value stored in the `ts` column accordingly; this would not have happened if you had changed the global time zone.

c. When changing the session time zone, the server converts the display (but not the value stored in the `ts` column) accordingly; this would also have happened if you had changed the global time zone.

d. When changing the session time zone, the server converts the display (but not the value stored in the `ts` column) accordingly; this would not have happened if you had changed the global time zone.

Question 27:

The table `myauto` looks like this:

```
mysql> DESCRIBE myauto;
+-------+---------+------+-----+---------+----------------+
| Field | Type    | Null | Key | Default | Extra          |
+-------+---------+------+-----+---------+----------------+
| id    | int(11) | NO   | PRI | NULL    | auto_increment |
+-------+---------+------+-----+---------+----------------+
```

No records have been inserted into the table so far. Now, a value is inserted like this:

```
mysql> INSERT INTO myauto (id) VALUES (NULL);
```

Which SQL function would you use to retrieve the last inserted value for `id` and what would be that value? When you invoke this function over and over again without inserting new values, and some other user on another connection inserts new rows into the table, what would your function call return?

Question 28:

The table `cliptest` has the following columns and rows:

```
mysql> DESCRIBE cliptest;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| number | int(11)      | YES  |     | NULL    |       |
| string | varchar(255) | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
mysql> SELECT * FROM cliptest;
+---------+---------------------------------------+
| number  | string                                |
+---------+---------------------------------------+
| 1000000 | The Hitchhiker's Guide to the Galaxy |
|    NULL | NULL                                  |
+---------+---------------------------------------+
```

The table structure is modified with this statement:

```
mysql> ALTER TABLE cliptest
    ->  MODIFY number TINYINT UNSIGNED NOT NULL,
    ->  MODIFY string TINYINT UNSIGNED NOT NULL
    -> ;
```

What will the table data look like afterward? Assume that strict SQL mode is not enabled.

Question 29:

Will the following statement work?

```
mysql> CREATE TABLE nulltest (
    ->  test1 INT NOT NULL,
    ->  test2 INT NULL,
    ->  test3 INT NOT NULL DEFAULT 123,
    ->  test4 INT NULL DEFAULT 456
    -> );
```

Question 30:

Is it possible to store `NULL` values in a `TIMESTAMP` column? If so, how do you define the column to allow this?

Question 31:

What is the effect of specifying a default value of `CURRENT_TIMESTAMP` for a `TIMESTAMP` column?

Question 32:

What is the effect of specifying the `ON UPDATE CURRENT_TIMESTAMP` attribute for a `TIMESTAMP` column?

Question 33:

Explain how you would go about creating and maintaining two `TIMESTAMP` columns named `created` and `updated` that meet the following requirements:

- Both the `created` and `updated` columns are set to the current time when a record is created.
- The `updated` column (but not the `created` column) is set to the current time whenever the record is updated.

Question 34:

John is sitting in London, and connects to the company's MySQL server. The server's global time zone is the same as John's, namely `'+00:00'` (UTC). John executes the following statement:

```
mysql> SELECT * FROM timestamps;
+---------------------+
| ts                  |
+---------------------+
| 2005-03-27 13:30:00 |
+---------------------+
1 row in set (0.00 sec)
```

Lydia is working in New York, where the time zone is five hours behind London. She wants to connect to the same server and see the contents of the same table, but adjusted for her own time zone. What statements does Lydia need to execute?

Question 35:

John is using a `time_zone` setting of `'+00:00'`. He retrieves a `TIMESTAMP` value with the following result:

```
mysql> SELECT * FROM timestamps;
+---------------------+
| ts                  |
+---------------------+
| 2005-04-02 06:45:00 |
+---------------------+
1 row in set (0.00 sec)
```

Lydia connects to the same server, adjusts her time zone, and retrieves the same `TIMESTAMP` value:

```
mysql> SET time_zone = '-05:00';
mysql> SELECT * FROM timestamps;
```

What value will Lydia see?

a. `'2005-04-02 01:45:00'`

b. `'2005-04-02 06:45:00'`

c. `'2005-04-02 11:45:00'`

Question 36:

Look at these multiple-row insert operations, where the first one fails but the second one succeeds with a warning when the data values are given in the opposite order:

```
mysql> INSERT INTO string_test VALUES
    -> ('far too long for this column'),('OK');
ERROR 1406 (22001): Data too long for column 't' at row 1
mysql> INSERT INTO string_test VALUES
    -> ('OK'),('far too long for this column');
Query OK, 2 rows affected, 1 warning (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 1
```

Which SQL mode has been enabled, and what kind of table type was used?

Question 37:

What will the contents of table `test_i` be after the following operations?

```
mysql> CREATE TABLE test_i
    -> (c1 INT(4) UNSIGNED, c2 INT(4) UNSIGNED ZEROFILL);
```

```
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO test_i (c1, c2)
    -> VALUES (1,1), (10000,10000);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Question 38:

Here's the structure of a table `typetest` with three columns (`number`, `string`, and `dates`). Assume the server is running in MySQL's “forgiving” SQL mode.

```
mysql> DESCRIBE typetest;
+--------+---------------------+------+-----+---------+-------+
| Field  | Type                | Null | Key | Default | Extra |
+--------+---------------------+------+-----+---------+-------+
| number | tinyint(3) unsigned | YES  |     | NULL    |       |
| string | char(5)             | YES  |     | NULL    |       |
| dates  | date                | YES  |     | NULL    |       |
+--------+---------------------+------+-----+---------+-------+
```

You perform the following `INSERT` operation on table `typetest`:

```
INSERT INTO typetest VALUES (1,22,333);
```

What data values will actually be stored in the table? Provide a short explanation.

Question 39:

Here's the structure of a table `typetest` with three columns (`number`, `string`, and `dates`). Assume the server is running in MySQL's “forgiving” SQL mode.

```
mysql> DESCRIBE typetest;
+--------+---------------------+------+-----+---------+-------+
| Field  | Type                | Null | Key | Default | Extra |
+--------+---------------------+------+-----+---------+-------+
| number | tinyint(3) unsigned | YES  |     | NULL    |       |
| string | char(5)             | YES  |     | NULL    |       |
| dates  | date                | YES  |     | NULL    |       |
+--------+---------------------+------+-----+---------+-------+
```

You perform the following `INSERT` operation on table `typetest`:

```
INSERT INTO typetest VALUES (1000,'yoodoo','999-12-31');
```

What data values will actually be stored in the table? Provide a short explanation.

Question 40:

Here's the structure of a table `typetest` with three columns (`number`, `string`, and `dates`). Assume the server is running in MySQL's “forgiving” SQL mode.

```
mysql> DESCRIBE typetest;
+--------+---------------------+------+-----+---------+-------+
| Field  | Type                | Null | Key | Default | Extra |
+--------+---------------------+------+-----+---------+-------+
| number | tinyint(3) unsigned | YES  |     | NULL    |       |
| string | char(5)             | YES  |     | NULL    |       |
| dates  | date                | YES  |     | NULL    |       |
+--------+---------------------+------+-----+---------+-------+
```

You perform the following `INSERT` operation on table `typetest`:

```
INSERT INTO typetest VALUES (NULL,NULL,NULL);
```

What data values will actually be stored in the table? Provide a short explanation.

Question 41:

Here's the structure of a table `typetest` with three columns (`number`, `string`, and `dates`). Assume the server is running in MySQL's “forgiving” SQL mode.

```
mysql> DESCRIBE typetest;
+--------+---------------------+------+-----+---------+-------+
| Field  | Type                | Null | Key | Default | Extra |
+--------+---------------------+------+-----+---------+-------+
| number | tinyint(3) unsigned | YES  |     | NULL    |       |
| string | char(5)             | YES  |     | NULL    |       |
| dates  | date                | YES  |     | NULL    |       |
+--------+---------------------+------+-----+---------+-------+
```

You perform the following `INSERT` operation on table `typetest`:

```
INSERT INTO typetest VALUES ('string',5+5,'string');
```

What data values will actually be stored in the table? Provide a short explanation.

Question 42:

Here's the structure of a table `typetest` with three columns (`number`, `string`, and `dates`). Assume the server is running in MySQL's “forgiving” SQL mode.

```
mysql> DESCRIBE typetest;
+--------+---------------------+------+-----+---------+-------+
| Field  | Type                | Null | Key | Default | Extra |
+--------+---------------------+------+-----+---------+-------+
| number | tinyint(3) unsigned | YES  |     | NULL    |       |
| string | char(5)             | YES  |     | NULL    |       |
| dates  | date                | YES  |     | NULL    |       |
+--------+---------------------+------+-----+---------+-------+
```

You perform the following `INSERT` operation on table `typetest`:

```
INSERT INTO typetest VALUES (-1,-1,'2000-02-32');
```

What data values will actually be stored in the table? Provide a short explanation.

Question 43:

Is the following statement true? Each character set has exactly one collation.

Question 44:

What's the maximum length of a `VARBINARY` column?

a. 255 characters

b. 255 bytes

c. 65,535 characters (or a few less due to internal restrictions)

d. 65,535 bytes (or a few less due to internal restrictions)

Question 45:

MySQL represents date values in '*YYYY-MM-DD*' format and time values in '*hh:mm:ss*' format. How can you change the display of date and time values?

Question 46:

Assume that you've just created this table:

```
CREATE TABLE timestamptest (
 ts1 TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
 i INT
);
```

When you look at its structure, you will notice that the `TIMESTAMP` column is declared `NOT NULL`. How can you insert `NULL` into the `TIMESTAMP` column?

Question 47:

Here's the structure of the table `datetimetest` with one column (`dt`), which will be used for the next six questions. Assume that the SQL mode has no input data restrictions enabled.

```
mysql> DESCRIBE datetimetest;
+-------+----------+------+
| Field | Type     | Null |
+-------+----------+------+
| dt    | datetime | YES  |
+-------+----------+------+
```

You perform the following `INSERT` operation on table `datetimetest`:

```
INSERT INTO datetimetest VALUES (NULL);
```

What data value will actually be stored in the `DATETIME` column? Provide a short explanation.

Question 48:

You perform the following `INSERT` operation on table `datetimetest`:

```
INSERT INTO datetimetest VALUES ('string');
```

What data value will actually be stored in the `DATETIME` column? Provide a short explanation. (Reminder: The table has one column. `dt` is a `DATETIME` column that allows `NULL` values.)

Question 49:

You perform the following `INSERT` operation on table `datetimetest`:

```
INSERT INTO datetimetest VALUES (200202082139);
```

What data value will actually be stored in the `DATETIME` columns? Provide a short explanation. (Reminder: The table has one column. `dt` is a `DATETIME` column that allows `NULL` values.)

Question 50:

You perform the following `INSERT` operation on table `datetimetest`:

```
INSERT INTO datetimetest VALUES (20020208213900);
```

What data value will actually be stored in the `DATETIME` columns? Provide a short explanation. (Reminder: The table has one column. `dt` is a `DATETIME` column that allows `NULL` values.)

Question 51:

You perform the following `INSERT` operation on table `datetimetest`:

```
INSERT INTO datetimetest VALUES ('2002-02-31 23:59:59');
```

What data value will actually be stored in the `DATETIME` columns? Provide a short explanation. (Reminder: The table has one column. `dt` is a `DATETIME` column that allows `NULL` values.)

Question 52:

You perform the following `INSERT` operation on table `datetimetest`:

```
INSERT INTO datetimetest VALUES ('2002-02-28 23:59:60');
```

What data value will actually be stored in the `DATETIME` columns? Provide a short explanation. (Reminder: The table has one column. `dt` is a `DATETIME` column that allows `NULL` values.)

Question 53:

Based on MySQL's capability for making context-specific data type conversions when working with functions and operators, what value will result from the following operation? Give a short explanation.

```
SELECT CONCAT(NULL,'Lennart');
```

Question 54:

Based on MySQL's capability for making context-specific data type conversions when working with functions and operators, what value will result from the following operation? Give a short explanation.

```
SELECT CONCAT(1,' plus ',1,' equals ',2);
```

Question 55:

Based on MySQL's capability for making context-specific data type conversions when working with functions and operators, what value will result from the following operation? Give a short explanation.

```
SELECT 1 + 1 + ' equals 2';
```

Question 56:

Based on MySQL's capability for making context-specific data type conversions when working with functions and operators, what value will result from the following operation? Give a short explanation.

```
SELECT 1 + 1 + '1.1 equals GUESS!';
```

Question 57:

Based on MySQL's capability for making context-specific data type conversions when working with functions and operators, what value will result from the following operation? Give a short explanation.

```
SELECT 1 + NULL;
```

Question 58:

Here's the structure of a table `continent` that has only one column (`name`, which stores names of continents). This table will be used for the next seven questions. Assume that the SQL mode has no input data restrictions enabled.

```
mysql> DESCRIBE continent\G
*************************** 1. row ***************************
  Field: name
   Type: enum('Africa','America','Antarctica','Asia','Australia','Europe')
   Null: YES
    Key:
Default: NULL
  Extra:
```

What string value will be stored by the following `INSERT` operation? What integer value will be stored internally?

```
INSERT INTO continent VALUES ('Africa');
```

Question 59:

Recall that table `continent` has only one column (`name`, with a data type of `ENUM`) that stores names of continents, with `NULL` values allowed. A `DESCRIBE` of the table shows the following (partial data only):

```
  Field: name
   Type: enum('Africa','America','Antarctica','Asia','Australia','Europe')
```

What string value will be stored by the following `INSERT` operation? What integer value will be stored internally?

```
INSERT INTO continent VALUES ('Europa');
```

Question 60:

Recall that table `continent` has only one column (`name`, with a data type of `ENUM`) that stores names of continents, with `NULL` values allowed. A `DESCRIBE` of the table shows the following (partial data only):

```
  Field: name
   Type: enum('Africa','America','Antarctica','Asia','Australia','Europe')
```

What string value will be stored by the following `INSERT` operation? What integer value will be stored internally?

```
INSERT INTO continent VALUES ('');
```

Question 61:

Recall that table `continent` has only one column (`name`, with a data type of `ENUM`) that stores names of continents, with `NULL` values allowed. A `DESCRIBE` of the table shows the following (partial data only):

```
  Field: name
   Type: enum('Africa','America','Antarctica','Asia','Australia','Europe')
```

What string value will be stored by the following `INSERT` operation? What integer value will be stored internally?

```
INSERT INTO continent VALUES (0);
```

Question 62:

Recall that table `continent` has only one column (`name`, with a data type of `ENUM`) that stores names of continents, with `NULL` values allowed. A `DESCRIBE` of the table shows the following (partial data only):

```
  Field: name
   Type: enum('Africa','America','Antarctica','Asia','Australia','Europe')
```

What string value will be stored by the following `INSERT` operation? What integer value will be stored internally?

```
INSERT INTO continent VALUES (1);
```

Question 63:

Recall that table `continent` has only one column (`name`, with a data type of `ENUM`) that stores names of continents, with `NULL` values allowed. A `DESCRIBE` of the table shows the following (partial data only):

```
  Field: name
   Type: enum('Africa','America','Antarctica','Asia','Australia','Europe')
```

What string value will be stored by the following `INSERT` operation? What integer value will be stored internally?

```
INSERT INTO continent VALUES ('1');
```

Question 64:

Recall that table `continent` has only one column (`name`, with a data type of `ENUM`) that stores names of continents, with `NULL` values allowed. A `DESCRIBE` of the table shows the following (partial data only):

```
  Field: name
   Type: enum('Africa','America','Antarctica','Asia','Australia','Europe')
```

What string value will be stored by the following `INSERT` operation? What integer value will be stored internally?

```
INSERT INTO continent VALUES (NULL);
```

Question 65:

The following `CREATE TABLE` statement shows the definition for table `defaults`, which will be used for the next seven questions.

```
mysql> CREATE TABLE defaults (
    ->  id INT UNSIGNED NOT NULL UNIQUE,
    ->  col1 INT NULL,
    ->  col2 INT NOT NULL,
    ->  col3 INT DEFAULT 42,
    ->  col4 CHAR(5) NULL,
    ->  col5 CHAR(5) NOT NULL,
    ->  col6 CHAR(5) DEFAULT 'yoo',
    ->  col7 TEXT NULL,
    ->  col8 TEXT NOT NULL,
    ->  col9 TIME NOT NULL,
    ->  col10 DATE NULL,
    ->  col11 DATE NOT NULL,
    ->  col12 DATE DEFAULT '2002-02-08',
    ->  col13 ENUM('doo','yoo'),
    ->  col14 SET('blabla','yooyoo'),
    ->  col15 ENUM('doo','yoo') NOT NULL,
    ->  col16 SET('blabla','yooyoo') NOT NULL
    -> );
```

What's the effect on the other columns with an `INT` data type if you issue the following `INSERT` statement? Why?

```
mysql> INSERT INTO defaults (id) VALUES (1);
```

Question 66:

Refer to the definition of the `defaults` table, shown in the previous question. What's the effect on the columns with a `CHAR` data type if you issue this `INSERT` statement? Why?

```
mysql> INSERT INTO defaults (id) VALUES (1);
```

Question 67:

Refer to the definition of the defaults table, shown two questions earlier. What's the effect on the columns with a TEXT data type if you issue this INSERT statement? Why?

```
mysql> INSERT INTO defaults (id) VALUES (1);
```

Reminder: Table defaults has two TEXT columns, shown in this partial table definition:

```
mysql> CREATE TABLE defaults (
    ->  id INT UNSIGNED NOT NULL UNIQUE,
    ->  . . .
    ->  col7 TEXT NULL,
    ->  col8 TEXT NOT NULL,
    ->  . . .
    -> );
```

Question 68:

Refer to the definition of the defaults table, shown three questions earlier. What's the effect on the columns with a TIME data type if you issue this INSERT statement? Why?

```
mysql> INSERT INTO defaults (id) VALUES (1);
```

Reminder: Table defaults has one TIME column, shown in this partial table definition:

```
mysql> CREATE TABLE defaults (
    ->  id INT UNSIGNED NOT NULL UNIQUE,
    ->  . . .
    ->  col9 TIME NOT NULL,
    ->  . . .
    -> );
```

Question 69:

Refer to the definition of the defaults table, shown four questions earlier. What's the effect on the columns with a DATE data type if you issue this INSERT statement? Why?

```
mysql> INSERT INTO defaults (id) VALUES (1);
```

Reminder: Table defaults has three DATE columns, shown in this partial table definition:

```
mysql> CREATE TABLE defaults (
    ->  id INT UNSIGNED NOT NULL UNIQUE,
    ->  . . .
    ->  col10 DATE NULL,
    ->  col11 DATE NOT NULL,
    ->  col12 DATE DEFAULT '2002-02-08',
    ->  . . .
    -> );
```

Question 70:

Refer to the definition of the defaults table, shown five questions earlier. What's the effect on the columns with an ENUM data type if you issue this INSERT statement? Why?

```
mysql> INSERT INTO defaults (id) VALUES (1);
```

Reminder: Table `defaults` has two `ENUM` columns, shown in this partial table definition:

```
mysql> CREATE TABLE defaults (
    ->  id INT UNSIGNED NOT NULL UNIQUE,
    ->  . . .
    ->  col13 ENUM('doo','yoo'),
    ->  col15 ENUM('doo','yoo') NOT NULL,
    ->  . . .
    -> );
```

Question 71:

Refer to the definition of the `defaults` table, shown six questions earlier. What's the effect on the columns with a `SET` data type if you issue this `INSERT` statement? Why?

```
mysql> INSERT INTO defaults (id) VALUES (1);
```

Reminder: Table `defaults` has two `SET` columns, shown in this partial table definition:

```
mysql> CREATE TABLE defaults (
    ->  id INT UNSIGNED NOT NULL UNIQUE,
    ->  . . .
    ->  col14 SET('blabla','yooyoo'),
    ->  col16 SET('blabla','yooyoo') NOT NULL
    ->  . . .
    -> );
```

Question 72:

The table `mytiny` has the following structure:

```
mysql> DESCRIBE mytiny;
+-------+------------+------+-----+---------+----------------+
| Field | Type       | Null | Key | Default | Extra          |
+-------+------------+------+-----+---------+----------------+
| id    | tinyint(4) | NO   | PRI | NULL    | auto_increment |
+-------+------------+------+-----+---------+----------------+
```

An application attempting to insert data with a program loop issues the following statement during every iteration of the loop:

```
INSERT INTO mytiny (id) VALUES (NULL);
```

How many times will this loop run without error? When an error occurs, what is the reason?

Answers to Exercises

Answer 1:

`DECIMAL`

Answer 2:

`CHAR(100)` stores 100 characters for every record, whereas `VARCHAR(100)` stores only the number of characters actually inserted, plus one byte to store the length of the entry. This means that `VARCHAR(100)` is normally more space-efficient. However, in the special case that you consistently insert 100-character values into the column, `CHAR(100)` is more space-efficient because the length byte used by `VARCHAR` to store the length of the entry is unneeded.

Answer 3:

By using a case-sensitive collation when specifying the column; for example, `codeName CHAR(10) COLLATE latin1_general_cs`.

Answer 4:

Non-binary values are sequences of characters that have a character set and collation. Binary values are sequences of arbitrary bytes that have no character set or collation. Characters might require one or more bytes each to store, whereas byte values require only a single byte each.

Answer 5:

`MEDIUMINT UNSIGNED` can hold numbers up to 16,777,215. The `UNSIGNED` attribute ensures that you don't store negative numbers by accident. Without `UNSIGNED`, the maximum positive number would only be 8,388,607. The storage requirement is 3 bytes per value.

Answer 6:

You could use either `VARCHAR(2000)` or `TEXT`. The latter can hold up to 65,535 characters; for a 300-character remark, the storage requirement is 300 characters plus 2 bytes to store the actual length of the entry. If you use `VARCHAR(2000)` instead, the storage requirement is the same, but if your table is a `MyISAM` table and contains many large columns, you might encounter a table type limitation, in which case you'll get an error like this:

```
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

As the error message indicates, the table type limitation does not apply to `TEXT` columns, so these are a bit more flexible.

Answer 7:

`'1000-01-01'` (January 1, 1000) is the earliest date that can be stored in a `DATE` column. You might be able to store earlier dates, but doing so isn't recommended because unexpected results from date operations might result.

Answer 8:

If a date is entered with a two-digit year value, MySQL converts it to a date between `'1970-01-01'` and `'2069-12-31'`. For each of the three examples, then, the results are as follows:

a. The value inserted is `'2010-02-08'`.

b. The value inserted is `'2069-12-31'`.

c. The value inserted is `'1970-01-01'`.

Answer 9:

The answer depends on the SQL mode which the server runs in. If the SQL mode is `TRADITIONAL`, an error occurs:

```
mysql> INSERT INTO datetest VALUES ('12:00:00');
ERROR 1292 (22007): Incorrect date value: '12:00:00' for column 'd' at row 1
```

This happens because `TRADITIONAL` mode includes the `NO_ZERO_IN_DATE` mode.

In MySQL's default forgiving SQL mode, the following value is inserted:

```
mysql> SET sql_mode = '';

mysql> INSERT INTO datetest VALUES ('12:00:00');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT d FROM datetest;
+------------+
| d          |
+------------+
| 2012-00-00 |
+------------+
```

MySQL interprets the inserted value as a 2-digit date (the year `12`), which it then converts into a 4-digit date (the year `2012`). The other parts of the `DATE` column (month, day) are set to zero.

Answer 10:

a. True. Floating-point types can be processed using the processor's native binary format.

b. True. Floating-point types can be processed faster. This is because they are using the processor's native binary format.

c. False. It's just the other way around: Floating-point types are subject to rounding errors, whereas fixed-point data types aren't.

d. It depends. Floating-point types may take less storage space than fixed-point data types. `FLOAT` takes 4 bytes, and `DOUBLE` takes 8 bytes, whereas `DECIMAL` takes approximately 4 bytes per 9 digits.

Answer 11:

For a `BIT(`*n*`)` column, the storage requirement can be calculated like this: `INT((`*n*`+7)/8)`, so it's `INT((4+7)/8)`, or 1 byte for our example.

The range of a `BIT(`*n*`)` column is `0` to `2`*n* `– 1`, so it's `0` to 2^4 `– 1`, or 0 to 15 for our example.

Answer 12:

The storage requirement depends on the character set used for that column. If it's a single-byte character set (for example, `latin1`), the storage requirement is 10 bytes. If it's a two-byte character set (for example, `ucs2`), the storage requirement is 20 bytes. If it's a character set that uses a variable number of bytes per character, the storage requirement is 10 times the largest number of bytes per character. `utf8` uses one to three bytes per character, so for `CHAR(10)`, the storage requirement is 30 bytes.

Answer 13:

No. The storage requirement of a `CHAR(10)` column that has `utf8` as its character set is determined by the largest possible character storage value (in bytes), which is three bytes in the case of `utf8`. The storage requirement is therefore always 30 bytes.

Answer 14:

The statements are not completely true. It's true that the character set of a column is determined by the table's default character set *if the column definition specifies no character set explicitly*. If the column definition also omits the collation, the table's default collation becomes the column's collation.

Answer 15:

No. It's possible for a column collation to be case sensitive and still treat accented and unaccented characters as the same character. This may also be true the other way around: A column collation may be accent sensitive but case insensitive.

Answer 16:

This is not true. In the case of binary collations, comparisons are performed per character, and characters might consist of multiple bytes. Binary strings are compared per byte.

Answer 17:

65,535 characters (or a few less due to internal restrictions imposed by storage engines).

Answer 18:

All of the statements are true, except that `TIMESTAMP` values are stored in UTC, not the server's local time zone.

Answer 19:

In older versions of MySQL (before 4.1), the first `TIMESTAMP` column in a table changes its value automatically when another table column is changed. It's initialized with the current date and time when a new record is inserted. That behavior can be explicitly declared like this:

```
CREATE TABLE timestamptest (
 ts1 TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
 i INT
);
```

Answer 20:

Only the first statement succeeds, and the `TIMESTAMP` column is set to the current date and time. The other two statements give an error:

```
mysql> INSERT INTO timestamptest SET ts1=NULL, i=10;
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO timestamptest SET ts1=0, i=11;
ERROR 1292 (22007): Incorrect datetime value: '0' for column 'ts1' at row 1
mysql> INSERT INTO timestamptest SET ts1='', i=12;
ERROR 1292 (22007): Incorrect datetime value: '' for column 'ts1' at row 1
mysql> SELECT ts1, i FROM timestamptest;
+---------------------+------+
| ts1                 | i    |
+---------------------+------+
| 2005-05-26 18:01:10 |   10 |
+---------------------+------+
```

Answer 21:

`'12'`. The second argument is evaluated to 2 and then the two arguments 1 and 2 are converted into strings before they're concatenated.

Answer 22:

`127`. The display width of `(2)` in the data type indicates only that values should be displayed in a two-digit format when they have only one digit. It doesn't restrict the range of values that can be stored in a `TINYINT` column.

Answer 23:

For the pictures, you would choose `MEDIUMBLOB`, which can store almost 16MB. For the remarks, you would choose `VARCHAR(250)`, which can store up to 250 characters and is more space-efficient than `CHAR(250)` if the values vary in length.

Answer 24:

2,002 bytes; 2,000 bytes for the value and 2 bytes to store the length of the value.

Answer 25:

The global variable contains the server's time zone, whereas the session (or local) variable holds the client's.

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
+--------------------+---------------------+
| @@global.time_zone | @@session.time_zone |
+--------------------+---------------------+
| SYSTEM             | SYSTEM              |
+--------------------+---------------------+

mysql> SET SESSION time_zone = '+01:00';

mysql> SELECT @@session.time_zone;
+---------------------+
| @@session.time_zone |
+---------------------+
| +01:00              |
+---------------------+
```

Answer 26:

When changing the session time zone, the server converts the display (but not the value stored in the `ts` column) accordingly; this would not have happened if you had changed the global time zone.

Answer 27:

You would use the `LAST_INSERT_ID()` SQL function and it would return 1 as the value inserted into the table. If you call `LAST_INSERT_ID()` repeatedly within the same connection, it will continue to return the same value (1), even if other connections insert new rows into the table.

Answer 28:

```
mysql> SELECT * FROM cliptest;
+--------+--------+
| number | string |
+--------+--------+
|    255 |      0 |
|      0 |      0 |
+--------+--------+
```

1000000 is clipped to the maximum value of TINYINT UNSIGNED. The string is converted to an integer number, and because it doesn't begin with an integer part, the result of the conversion is the number 0. The NULL entries are converted to values that match the new attribute NOT NULL, and because there are no DEFAULT values specified in the ALTER TABLE statement, MySQL uses the standard default values for integers, which is 0.

Answer 29:

The statement will create the table nulltest. All the column specifications are legal:

```
mysql> CREATE TABLE nulltest (
    ->  test1 INT NOT NULL,
    ->  test2 INT NULL,
    ->  test3 INT NOT NULL DEFAULT 123,
    ->  test4 INT NULL DEFAULT 456
    -> );
mysql> DESCRIBE nulltest;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| test1 | int(11) | NO   |     |         |       |
| test2 | int(11) | YES  |     | NULL    |       |
| test3 | int(11) | NO   |     | 123     |       |
| test4 | int(11) | YES  |     | 456     |       |
+-------+---------+------+-----+---------+-------+
```

Answer 30:

Yes, it is possible to store NULL values in a TIMESTAMP column.

However, unlike other data types, TIMESTAMP columns default to NOT NULL, and you must specifically include the NULL attribute in the column definition when you create or alter the TIMESTAMP column if you want to allow storage of NULL values.

Answer 31:

The effect of specifying a default value of CURRENT_TIMESTAMP for a TIMESTAMP column is that whenever a new record is created, that column is set to the record creation time.

Answer 32:

The effect of specifying the ON UPDATE CURRENT_TIMESTAMP attribute for a TIMESTAMP column is that the column is updated to the current timestamp value whenever any other column in the record is changed.

Answer 33:

A table with two TIMESTAMP columns that meet the requirements can be created as follows:

```
mysql> CREATE TABLE updates (
    ->   created TIMESTAMP DEFAULT 0,
    ->   updated TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    ->   data CHAR(30)
    -> );
```

```
Query OK, 0 rows affected (0.01 sec)
```

For record creation, to set *both* of the `TIMESTAMP` columns to the current time, we use the `INSERT` statement to insert `NULL` into *each* column:

```
mysql> INSERT INTO updates (created, updated, data)
    -> VALUES (NULL, NULL, 'original_value');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM updates;
+---------------------+---------------------+----------------+
| created             | updated             | data           |
+---------------------+---------------------+----------------+
| 2005-01-05 12:50:40 | 2005-01-05 12:50:40 | original_value |
+---------------------+---------------------+----------------+
1 row in set (0.00 sec)
```

For updates, we do not need to do anything special to update the `updated` column because it already has the `ON UPDATE CURRENT_TIMESTAMP` attribute:

```
mysql> UPDATE updates SET data='updated_value';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM updates;
+---------------------+---------------------+---------------+
| created             | updated             | data          |
+---------------------+---------------------+---------------+
| 2005-01-05 12:50:40 | 2005-01-05 12:50:46 | updated_value |
+---------------------+---------------------+---------------+
1 row in set (0.00 sec)
```

Answer 34:

For Lydia to see the data adjusted for her own time zone after establishing a session with the server, she must execute the statement `SET time_zone = '-05:00'` before she performs the `SELECT`:

```
mysql> SET time_zone = '-05:00';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM timestamps;
+---------------------+
| ts                  |
+---------------------+
| 2005-03-27 08:30:00 |
+---------------------+
1 row in set (0.00 sec)
```

Answer 35:

Lydia adjusted her time zone to five hours behind the time zone used by John, so the value displayed for the `TIMESTAMP` value will be five hours earlier the value John sees: `'2005-04-02 01:45:00'`.

Answer 36:

The SQL mode used during the session was `STRICT_TRANS_TABLES`, and the table type was `MyIS-AM` (or some other non-transactional type). This causes errors to be treated as warnings for errors that occur after the first row of a multiple-row `INSERT` statement.

Had the mode been `STRICT_ALL_TABLES`, we would have seen an error, not a warning, in both cases (although in the second attempt, the `'OK'` value would have been inserted):

```
mysql> INSERT INTO string_test VALUES
    -> ('OK'),('far too long for this column');
ERROR 1406 (22001): Data too long for column 't' at row 2
```

Had `string_test` been a transactional table type such as `InnoDB`, both insert operations would have failed.

Answer 37:

The display width of an integer doesn't affect its range (and the display width is evident only when the `ZEROFILL` attribute has been specified), so the table's contents will look like this:

```
mysql> SELECT c1, c2 FROM test_i;
+-------+-------+
| c1    | c2    |
+-------+-------+
|     1 |  0001 |
| 10000 | 10000 |
+-------+-------+
```

Answer 38:

```
+--------+--------+------------+
| number | string | dates      |
+--------+--------+------------+
|      1 | 22     | 0000-00-00 |
+--------+--------+------------+
```

`22` is converted to the string value `'22'`. The number `333` is interpreted as an invalid date, so the “zero” date is stored.

Answer 39:

```
+--------+--------+------------+
| number | string | dates      |
+--------+--------+------------+
|    255 | yoodo  | 0000-00-00 |
+--------+--------+------------+
```

The inserted number `1000` is too big to fit in the `TINYINT UNSIGNED` column, so the highest possible value (`255`) is inserted. `'yoodoo'` is too long for a `CHAR(5)` column and is thus truncated to five characters. `'999-12-31'` is a date that is earlier than the earliest possible `DATE` value (`'1000-01-01'`). This is interpreted as an invalid date, so the “zero” date is stored.

Answer 40:

```
+--------+--------+------------+
| number | string | dates      |
+--------+--------+------------+
|   NULL | NULL   | NULL       |
+--------+--------+------------+
```

All columns are declared `NULL` (by not specifying them as `NOT NULL`), so they accept `NULL` values.

Answer 41:

```
+--------+--------+------------+
| number | string | dates      |
+--------+--------+------------+
|      0 | 10     | 0000-00-00 |
+--------+--------+------------+
```

`'string'` is converted to a number for the `number` column; because there are no digit characters at the beginning of the string, the result is `0`. `5+5` is evaluated to `10`, which is converted to the string `'10'` before it is stored in the `string` column. `'string'` is converted to a date before it is stored in the `dates` column; because it is invalid as a date, the “zero” date is stored.

Answer 42:

```
+--------+--------+------------+
| number | string | dates      |
+--------+--------+------------+
|      0 | -1     | 0000-00-00 |
+--------+--------+------------+
```

`-1` is lower than the lowest possible value for any unsigned integer column, so it's converted to `0` before it's stored. `-1` is converted to the corresponding string value (`'-1'`) before it's stored. The inserted date has an invalid day portion (`32`); because this is interpreted as an invalid date, the “zero” date is stored.

Answer 43:

This is not true. A character set can have more than one collation, and most do. For example:

```
mysql> SHOW COLLATION LIKE 'hebrew%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| hebrew_general_ci | hebrew  | 16 | Yes     |          |       0 |
| hebrew_bin        | hebrew  | 71 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

Answer 44:

65,535 bytes (or a few less due to internal restrictions imposed by storage engines).

Answer 45:

You cannot change the default date or time display format. However, you can reformat values for display with the `DATE_FORMAT()` and `TIME_FORMAT()` functions.

Answer 46:

This is not possible without altering the table's structure first to explicitly declare the `TIMESTAMP` column `NULL`:

```
mysql> ALTER TABLE timestamptest
    ->  MODIFY ts1 TIMESTAMP NULL
    -> DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
    -> ;

mysql> INSERT INTO timestamptest (ts1, i) VALUES (NULL, 10);

mysql> SELECT ts1, i FROM timestamptest;
```

```
+------+------+
| ts1  | i    |
+------+------+
| NULL |   10 |
+------+------+
```

Answer 47:

```
+------+
| dt   |
+------+
| NULL |
+------+
```

Because `dt` isn't explicitly declared `NOT NULL`, it can hold `NULL` values.

Answer 48:

```
+---------------------+
| dt                  |
+---------------------+
| 0000-00-00 00:00:00 |
+---------------------+
```

`'string'` is an invalid `DATETIME` value, so the “zero” value is inserted instead.

Answer 49:

```
+---------------------+
| dt                  |
+---------------------+
| 2020-02-02 08:21:39 |
+---------------------+
```

The conversion of the 12-digit number `200202082139` produces a puzzling result. The value is interpreted as a `DATETIME`, with the leftmost two digits treated as the year portion of the `DATETIME`. `20` becomes the year value `2020`, and the rest of the digits (`0202082139`) are interpreted as month (`02`), day (`02`), hour (`08`), minute (`21`), and second (`39`).

Answer 50:

```
+---------------------+
| dt                  |
+---------------------+
| 2002-02-08 21:39:00 |
+---------------------+
```

`20020208213900` looks like a 14-digit `TIMESTAMP` value and is interpreted as a `DATETIME` on insertion, where the rightmost `00` is the seconds portion of the value.

Answer 51:

```
+---------------------+
| dt                  |
+---------------------+
| 0000-00-00 00:00:00 |
+---------------------+
```

`2002-02-31 23:59:59` is an invalid `DATETIME` value because February does not have 31 days.

Answer 52:

```
+---------------------+
| dt                  |
+---------------------+
| 0000-00-00 00:00:00 |
+---------------------+
```

`2002-02-28 23:59:60` isn't a valid `DATETIME` value because the seconds portion (`60`) isn't within the valid range. MySQL thus converts the value to the “zero” `DATETIME` value.

Answer 53:

`NULL`. If any argument to `CONCAT()` is `NULL`, the result is `NULL` as well.

Answer 54:

`'1 plus 1 equals 2'`. All arguments to `CONCAT()` are converted to strings before they're concatenated.

Answer 55:

`2`. The string `' equals 2'` is interpreted as a number. It evaluates to `0` because it has no leftmost numeric part. Thus, the operation performed is `1 + 1 + 0`.

Answer 56:

`3.1`. The leftmost part of the string `'1.1 equals GUESS'` contains the floating point number `1.1`. Thus, all numbers are converted to floats, so the operation performed is `1.0 + 1.0 + 1.1`.

Answer 57:

`NULL`. The result of an arithmetic operation is indeterminate with a `NULL` operand, so the result is `NULL`.

Answer 58:

String value: `'Africa'`. Internal number: `1` because `Africa` is the first member in the `ENUM` list.

Answer 59:

String value: `''` (the empty string). Internal number: `0`. Because `'Europa'` isn't a member of the `ENUM` list, the special error value of `''` (or `0` as the internal representation) is stored.

Answer 60:

String value: `''` (the empty string). Internal number: `0`. The empty string (internally, `0`) is the special error value that is stored if the inserted value isn't a member of the `ENUM` list, or if the empty string (or `0`) is explicitly stored, as in this case.

Answer 61:

String value: `''` (the empty string). Internal number: `0`. The empty string (internally, `0`) is the special error value that is stored if the inserted value isn't a member of the `ENUM` list, or if the empty string (or `0`) is explicitly stored, as in this case.

Answer 62:

String value: 'Africa'. Internal number: 1. 'Africa' is the first member in the ENUM list. The value can be inserted by giving the element number instead of the string value.

Answer 63:

String value: 'Africa'. Internal number: 1. 'Africa' is the first member in the ENUM list. MySQL first converts the string '1' to a number before it's inserted.

Answer 64:

NULL can be inserted into the name column because the column definition allows NULL values.

Answer 65:

These values will be inserted into the table's INT columns (partial listing only):

```
mysql> SELECT * FROM defaults\G
*********** 1. row
   id: 1
 col1: NULL
 col2: 0
 col3: 42
 ...
```

- col1: Because this column has no defined DEFAULT value and can accept NULL values, the value inserted is NULL.
- col2: This column is declared NOT NULL and has no defined DEFAULT value. Because the INSERT provides no explicit value for this column, MySQL assigns the implicit default value for integer columns, in this case 0.
- col3: This column was explicitly defined with a DEFAULT value (42), so this value is inserted.

Answer 66:

These values will be inserted into the table's CHAR columns (partial listing only):

```
mysql> SELECT * FROM defaults\G
*********** 1. row
   id: 1
 ...
 col4: NULL
 col5:
 col6: yoo
 ...
```

- col4: Because this column has no defined DEFAULT value and can accept NULL values, the value inserted is NULL.
- col5: This column is declared NOT NULL and has no defined DEFAULT value. Because the INSERT provides no explicit value for this column, MySQL assigns the implicit default value for string columns, in this case '' (the empty string).
- col6: This column was explicitly declared with a DEFAULT value ('yoo'), so this value is inserted.

Answer 67:

These values will be inserted into the table's `TEXT` columns (partial listing only):

```
mysql> SELECT * FROM defaults\G
*********** 1. row
   id: 1
 ...
 col7: NULL
 col8:
 ...
```

- `col7`: Because this column can accept `NULL` values, the value inserted is `NULL`.
- `col8`: This column is declared `NOT NULL`. Because the `INSERT` provides no explicit value for this column, MySQL assigns the implicit default value for string columns, in this case `''` (the empty string). (You cannot declare a `DEFAULT` value for a `TEXT` column.)

Answer 68:

This value will be inserted into the table's `TIME` column (partial listing only):

```
mysql> SELECT * FROM defaults\G
*********** 1. row
   id: 1
 ...
 col9: 00:00:00
 ...
```

`col9`: This column is declared `NOT NULL` and has no defined `DEFAULT` value. Because the `INSERT` provides no explicit value for this column, MySQL assigns the implicit default value for temporal columns, in this case `'00:00:00'`.

Answer 69:

These values will be inserted into the table's `DATE` columns (partial listing only):

```
mysql> SELECT * FROM defaults\G
*********** 1. row
   id: 1
 ...
col10: NULL
col11: 0000-00-00
col12: 2002-02-08
 ...
```

- `col10`: Because this column has no defined `DEFAULT` value and can accept `NULL` values, the value inserted is `NULL`.
- `col11`: This column is declared `NOT NULL` and has no defined `DEFAULT` value. Because the `INSERT` provides no explicit value for this column, MySQL assigns the implicit default value for temporal columns, in this case `'0000-00-00'`.
- `col12`: This column was explicitly defined with a `DEFAULT` value (`'2002-02-08'`), so this value is inserted.

Answer 70:

These values will be inserted into the table's `ENUM` columns (partial listing only):

```
mysql> SELECT * FROM defaults\G
*********** 1. row
   id: 1
  ...
col13: NULL
col15: doo
  ...
```

- `col13`: Because this column has no defined `DEFAULT` value and can accept `NULL` values, the value inserted is `NULL`.
- `col15`: The `ENUM` column is declared `NOT NULL` and has no defined `DEFAULT` value. Because the `INSERT` provides no explicit value for this column, MySQL uses the first list member as the implicit default value.

Answer 71:

These values will be inserted into the table's `SET` columns (partial listing only):

```
mysql> SELECT * FROM defaults\G
*********** 1. row
   id: 1
  ...
col14: NULL
col16:
  ...
```

- `col14`: Because this column has no defined `DEFAULT` value and can accept `NULL` values, the value inserted is `NULL`.
- `col16`: The `SET` column is declared `NOT NULL` and has no defined `DEFAULT` value. Because the `INSERT` provides no explicit value for this column, MySQL uses the empty string as the implicit default value.

Answer 72:

The loop will run 127 times without error. With the first loop run, `1` is inserted, with the second run, `2` is inserted, and so on. `127` is the maximum value for a `TINYINT` column. An error will occur when the application tries to insert `id` number `128`. This number will be clipped to `127`, and MySQL will try to insert this value once again. Because of the `PRIMARY KEY` restriction that requires unique values in the `id` column, this will result in a duplicate-key error. With the `mysql` client, the error would be displayed as follows:

```
ERROR 1062 (23000): Duplicate entry '127' for key 1
```

Chapter 6. Identifiers

Almost all examples and exercises in this study guide use the *world database* as the sample data set. The accompanying CD-ROM contains the data for this database and instructions that describe how to create and populate the database for use with your own MySQL installation.

Question 1:

Which of the following statements will produce an error?

```
SELECT
  Name AS 'City Name',
  Population AS c
FROM City AS c
WHERE Population > 1000000;
```

```
SELECT
  c.Name, c.Population
FROM City AS C;
```

```
SELECT
  ciudad.Name AS `City Name`,
  ciudad.Population AS "City Population"
FROM City AS ciudad;
```

Question 2:

Will the following statements succeed?

a. `SELECT "Name", "Population" FROM "City" LIMIT 1;`

b. `SELECT "Name", "Population" FROM City LIMIT 1;`

c. ``SELECT `Name`, `Population` FROM `City` LIMIT 1;``

d. ``SELECT 'Name', "Population" FROM `City` LIMIT 1;``

Question 3:

Will the following SQL statement succeed or result in an error? Assume that the table to be created doesn't already exist.

```
CREATE TABLE select (id INT);
```

Question 4:

Will the following SQL statement succeed or result in an error? Assume that the table to be created doesn't already exist.

```
CREATE TABLE `select` (id INT);
```

Question 5:

Will the following SQL statement succeed or result in an error? Assume that the table to be created doesn't already exist.

```
CREATE TABLE 'select' (id INT);
```

Question 6:

Will the following SQL statement succeed or result in an error? Assume that the table to be created doesn't already exist.

```
CREATE TABLE `select-me, please!` (id INT);
```

Question 7:

Will the following SQL statement succeed or result in an error? Assume that the table to be created doesn't already exist.

```
CREATE TABLE `select.me.please` (id INT);
```

Question 8:

Will the following SQL statement succeed or result in an error? Assume that the table to be created doesn't already exist.

```
CREATE TABLE MD5 (id INT);
```

Question 9:

Will the following SQL statement succeed or result in an error? Assume that the table to be created doesn't already exist.

```
CREATE TABLE `MD5()` (id INT);
```

Question 10:

Will the following SQL statement succeed or result in an error? Assume that the table to be created doesn't already exist.

```
CREATE TABLE `MD5('Lennart')` (id INT);
```

Question 11:

Will the following SQL statement succeed or result in an error? Assume that the table to be created doesn't already exist.

```
CREATE TABLE `COUNT(*)` (id INT);
```

Question 12:

Will the following SQL statement succeed or result in an error? Assume that the table to be created doesn't already exist.

```
CREATE TABLE `0123456789` (id INT);
```

Question 13:

Will the following SQL statement succeed or result in an error? Assume that the database to be created doesn't already exist.

```
CREATE DATABASE `0123456789`;
```

Question 14:

Using the client tool `mysql`, the following statements are issued:

```
mysql> CREATE DATABASE CaseTest;
Query OK, 1 row affected (0.00 sec)
mysql> USE casetest;
Database changed
```

Sometime later, the database is moved to another MySQL server. Trying to select the database as just shown results in an error message:

```
mysql> USE casetest;
ERROR 1049 (42000): Unknown database 'casetest'
```

What's the reason for this error message? How could this problem be solved? What could you do to prevent problems like this?

Answers to Exercises

Answer 1:

Aliases are not case sensitive except table aliases, and table aliases are case sensitive only if the `lower_case_table_names` system variable is 0. Assuming that `lower_case_table_names` is 0, only the second statement will result in an error because the table alias is `C` and it is referred to as `c` for the columns to be retrieved.

Answer 2:

You can surround any identifier with backticks, so the third statement will succeed. You can use double quotes for table names and other identifiers only if the `ANSI_QUOTES` SQL mode is enabled (or some other composite mode such as `ANSI` that includes `ANSI_QUOTES`). Therefore, the first statement will fail. The second statement will succeed even if `ANSI_QUOTES` isn't enabled, but in that case, the result is probably not what you would want if you're trying to select column values rather than strings:

```
mysql> SELECT @@SQL_MODE;
+------------+
| @@SQL_MODE |
+------------+
|            |
+------------+
mysql> SELECT "Name", "Population" FROM City LIMIT 1;
+------+------------+
| Name | Population |
+------+------------+
| Name | Population |
+------+------------+
```

The fourth statement will succeed in any case, but it won't yield the desired result (you should not quote identifiers with single quotes unless they're aliases):

```
mysql> SELECT 'Name', "Population" FROM `City` LIMIT 1;
+------+------------+
| Name | Population |
+------+------------+
| Name | Population |
+------+------------+
mysql> SET SQL_MODE = 'ANSI_QUOTES';
mysql> SELECT @@SQL_MODE;
+-------------+
| @@SQL_MODE  |
+-------------+
| ANSI_QUOTES |
+-------------+
mysql> SELECT 'Name', "Population" FROM `City` LIMIT 1;
+------+------------+
| Name | Population |
+------+------------+
| Name |    1780000 |
+------+------------+
```

Answer 3:

An error will result. It isn't possible to use a reserved word as an identifier without quoting the identifier.

```
mysql> CREATE TABLE select (id INT);
ERROR 1064 (42000): You have an error in your SQL syntax.
```

Answer 4:

The statement succeeds. When a reserved word is quoted, it can be used as an identifier. In this example, backticks are used; this works under all circumstances.

```
mysql> CREATE TABLE `select` (id INT);
Query OK, 0 rows affected
```

Answer 5:

An error will result. Single quotes can be used as quotes for aliases, but they cannot be used for identifiers such as table names.

```
mysql> CREATE TABLE 'select' (id INT);
ERROR 1064 (42000): You have an error in your SQL syntax.
```

Answer 6:

The statement succeeds. Almost every character, including dash, comma, space, and exclamation mark, is legal in a table identifier when the identifier is properly quoted.

```
mysql> CREATE TABLE `select-me, please!` (id INT);
Query OK, 0 rows affected
```

Answer 7:

An error will result. Periods cannot be used in a table identifier even when the identifier is quoted. Because periods are used to separate database names from table names, and table names from column names, they aren't acceptable within database or table identifiers.

```
mysql> CREATE TABLE `select.me.please` (id INT);
ERROR 1103 (42000): Incorrect table name 'select.me.please'
```

Answer 8:

The statement results in an error if the `IGNORE_SPACE` SQL mode is enabled. In that case, all function names become reserved words and the statement would return an error because `MD5` is a function name.

If the `IGNORE_SPACE` SQL mode is not enabled, the statement succeeds:

```
mysql> CREATE TABLE MD5 (id INT);
Query OK, 0 rows affected
```

Note that the `IGNORE_SPACE` SQL mode is part of the `ANSI` SQL mode.

Answer 9:

The statement succeeds. Parentheses are legal characters in an identifier as long as it is quoted.

```
mysql> CREATE TABLE `MD5()` (id INT);
Query OK, 0 rows affected
```

Answer 10:

The statement succeeds. Single quotes are legal characters in an identifier as long as it is quoted.

```
mysql> CREATE TABLE `MD5('Lennart')` (id INT);
Query OK, 0 rows affected
```

Answer 11:

Whether this statement succeeds is dependent on the operating system under which the MySQL server is running. If the operating system doesn't allow certain characters in filenames, the characters cannot be used for database and table names even when the identifier is quoted.

```
mysql> CREATE TABLE `COUNT(*)` (id INT);
ERROR 1 (HY000): Can't create/write to file '.\test\COUNT(*).frm'
(Errcode: 22)
```

Answer 12:

The statement succeeds. Identifiers consisting solely of numbers are accepted, provided that they're quoted.

```
mysql> CREATE TABLE `0123456789` (id INT);
Query OK, 0 rows affected
```

Answer 13:

The statement succeeds. Quoted identifiers consisting solely of numbers are accepted for database names.

```
mysql> CREATE DATABASE `0123456789`;
Query OK, 1 row affected
```

Answer 14:

The database `CaseTest` was created under an operating system that does not have case-sensitive filenames, such as Windows. The database was then moved to a machine running an operating system that has case-sensitive filenames, such as Linux. To solve this problem, the database should be selected as follows:

```
mysql> USE CaseTest;
Database changed
```

To prevent problems with database and table names under different operating systems, you could start the MySQL server on all operating systems you're using with the `lower_case_table_names` variable set. (However, you should do this before creating any databases or tables.)

Chapter 7. Databases

Almost all examples and exercises in this study guide use the *world database* as the sample data set. The accompanying CD-ROM contains the data for this database and instructions that describe how to create and populate the database for use with your own MySQL installation.

Question 1:

What does the `CREATE SCHEMA world` statement do?

a. Creates a nested database `world` inside the current database

b. Creates a database `world` that may contain other databases

c. Creates a database `world` if no such database exists already (if it does, an error occurs)

d. Creates a database `world`, overwriting any existing database of the same name

Question 2:

Each database directory contains a file named `db.opt`. What is stored in that file?

a. Information about the results of `CHECK TABLE` operations

b. The default character set and collation used when creating new tables in that database

c. The character set and collation that would be optimal regarding disk space efficiency when creating tables in that database

Question 3:

What properties of a database can you change with `ALTER DATABASE`?

a. The name of the database

b. The default character set used for creating new tables in that database

c. The character set of all tables that currently exist in that database

d. The default collation used for creating new tables in that database

Question 4:

What statement do you use to drop the `test` database? How can you undo, or cancel, this statement?

Question 5:

Is the following statement true or false?

A database must contain at least one table.

Question 6:

Which statements can you use to get information about the character set used for creating new tables in the `world` database?

Question 7:

Is the following statement true or false?

MySQL itself imposes no limit on the number of databases you can create on the server.

Question 8:

What clause can you add to a `CREATE DATABASE` statement to ensure that no error occurs if the database already exists?

Question 9:

What clause can you add to a `DROP DATABASE` statement to ensure that no error occurs if the database doesn't exist?

Answers to Exercises

Answer 1:

In MySQL, `CREATE SCHEMA world` is a synonym for `CREATE DATABASE world`. It creates a database with the name `world` unless such a database already exists.

Answer 2:

The `db.opt` file stores the default character set and collation used when creating new tables in that database, for example:

```
default-character-set=latin1
default-collation=latin1_swedish_ci
```

Answer 3:

`ALTER DATABASE` can be used to change the default character set and the default collation used for creating new tables in that database. Tables that exist when the statement is issued aren't changed. You cannot change the name of a database with `ALTER DATABASE`.

Answer 4:

`DROP DATABASE test`. This statement cannot be undone, so be careful with it.

Answer 5:

False. A database can be empty.

Answer 6:

There are two statements that can be used to obtain information about the character set of a database:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA
    -> WHERE SCHEMA_NAME = 'world'\G
*************************** 1. row ***************************
              CATALOG_NAME: NULL
               SCHEMA_NAME: world
DEFAULT_CHARACTER_SET_NAME: latin1
    DEFAULT_COLLATION_NAME: latin1_swedish_ci
```

```
                  SQL_PATH: NULL

mysql> SHOW CREATE DATABASE world\G
*************************** 1. row ***************************
       Database: world
Create Database: CREATE DATABASE `world`
                 /*!40100 DEFAULT CHARACTER SET latin1 */
```

Answer 7:

True. Such a limit could, however, be imposed by the operating system.

Answer 8:

```
IF NOT EXISTS
```

Answer 9:

```
IF EXISTS
```

Chapter 8. Tables and Indexes

Almost all examples and exercises in this study guide use the *world database* as the sample data set. The accompanying CD-ROM contains the data for this database and instructions that describe how to create and populate the database for use with your own MySQL installation.

Question 1:

Is the following statement true or false?

`InnoDB` imposes no limit on the number of tables that can be held in the `InnoDB` tablespace.

Question 2:

Is the following statement true or false?

In a MySQL database, every table has an `.frm` file in the appropriate database directory, regardless of the storage engine used.

Question 3:

Consider the following table:

```
mysql> DESCRIBE t;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| i     | int(11) | NO   | PRI |         |       |
+-------+---------+------+-----+---------+-------+
```

Which statements can you use to drop a `PRIMARY KEY`?

Question 4:

Name the different kinds of indexes that MySQL supports.

Question 5:

What must be true of the columns named in a `UNIQUE` index for the index to be functionally equivalent to a `PRIMARY KEY` on the same columns?

Question 6:

When you use `DROP TABLE` to remove a table, how do you tell MySQL not to report an error if the table doesn't exist? What's reported instead of an error?

Question 7:

Is the following statement true or false?

A table must contain at least one column.

Question 8:

Is the following statement true or false?

A table must contain at least one row.

Question 9:

Is the following statement true or false?

To create a table, you must first issue a statement to choose a default database in which to store the table.

Question 10:

Which clause can you add to a `CREATE TABLE` statement to ensure that no error occurs if the table already exists?

Question 11:

Why does the following SQL statement fail?

```
CREATE TABLE cats (
    id   INT       UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
    name CHAR(10)
);
```

Question 12:

There are two ways using SQL statements to create a copy of a table (structure and data). Assume that you want to create a copy of the `world.City` table in the `test` database. How would you accomplish that, using those two ways?

Question 13:

Is the following statement true or false?

You can add multiple columns to a table with a single `ALTER TABLE` statement.

Question 14:

Is the following statement true or false?

You can add one or more rows to a table with a single `ALTER TABLE` statement.

Question 15:

Is the following statement true or false?

You can create multiple indexes at a time with a single `ALTER TABLE` statement.

Question 16:

Is the following statement true or false?

You can drop all columns of a table (thus dropping the table itself) with a single `ALTER TABLE` statement.

Question 17:

There are two ways to rename table `tbl` to `tbl_new` with SQL statements. What statements can you use?

Question 18:

Name the two most common reasons to create an index on a table.

Question 19:

Table `mytable` has a composite `PRIMARY KEY` consisting of both `col1` and `col2`. Is it possible to declare one of the two columns as `NULL`, like this?

```
CREATE TABLE mytable (
    col1 CHAR(5) NOT NULL,
    col2 CHAR(5) NULL,
    PRIMARY KEY (col1,col2)
);
```

Question 20:

You have a table `mytable` that looks like this:

```
mysql> DESCRIBE mytable;
+-------+---------+
| Field | Type    |
+-------+---------+
| col1  | int(11) |
| col3  | int(11) |
+-------+---------+
```

You want to add three more columns: `col0` as the first column in the table, `col2` between `col1` and `col3`, and `col4` as the last column. All new columns should be of type `INT`. What SQL statement do you issue?

Question 21:

You want to see what indexes you have in table `tbl`, but `DESCRIBE tbl` does not show sufficient information. What other statement can you issue to obtain additional information about the table structure?

Question 22:

To declare a primary key on only one column (`col1`, with data type `INT`) of table `tbl` at creation time, you can use the following syntax:

```
mysql> CREATE TABLE tbl (col1 INT NOT NULL PRIMARY KEY);
```

What's the correct syntax if you want to declare a composite primary key for this table on two `INT` columns `col1` and `col2`?

Question 23:

How can you empty a table? What's the best SQL statement for this?

Question 24:

What `SHOW` statement will retrieve a list of all tables in the `test` database with a name that contains the string `'test'`? What's the corresponding `SELECT` statement that uses the `INFORMATION_SCHEMA` database?

Question 25:

What `SHOW` statement will retrieve a list of columns in the table `test.mytest`, where the column names begin with `id`?

Question 26:

What `SHOW` statement will retrieve a statement that could be used to re-create the table `test.mytest` in an arbitrary database? Assume that `test` is not the default database.

Question 27:

You know that `MyISAM` is the built-in default storage engine in MySQL. So, what is the explanation for the `Engine` value in the following `SHOW TABLE STATUS` output? What could you do to make sure that the table gets created as a `MyISAM` table?

```
mysql> CREATE TABLE defaulttype (id INT);
mysql> SHOW TABLE STATUS LIKE 'defaulttype';
+-------------+--------+-
| Name        | Engine | ...
+-------------+--------+-
| defaulttype | InnoDB | ...
+-------------+--------+-
```

Question 28:

How can you change the server's default storage engine for new tables from `MyISAM` to `InnoDB` if the server is already running?

Question 29:

Name three ways to work around a table size limitation that's imposed by the file size limitation of the operating system.

Question 30:

Which statements can you use to drop the index `idx_id` on table `tbl`? How can you recover the index?

Question 31:

List the differences between a `UNIQUE` index and a `PRIMARY KEY`.

Question 32:

Which type of index cannot be created with `CREATE INDEX`?

Question 33:

You want to create a table, but you want to decide later the database to which it should belong. How do you do accomplish this?

Question 34:

Provide the SQL statement that will create a table with the following structure:

```
mysql> DESCRIBE mytbl;
+-------+------------------+------+-----+---------+
| Field | Type             | Null | Key | Default |
+-------+------------------+------+-----+---------+
| col1  | int(10) unsigned | NO   | PRI | 0       |
| col2  | char(50)         | NO   | UNI |         |
| col3  | char(50)         | NO   | MUL |         |
+-------+------------------+------+-----+---------+
```

Question 35:

Is the following statement true or false?

You can drop multiple indexes at a time with a single `ALTER TABLE` statement.

Question 36:

Is the following statement true or false?

Using a single `ALTER TABLE` statement, you can add a new column as the first column in a table.

Question 37:

Suppose that you have the following table structure:

```
+-------+---------+
| Field | Type    |
+-------+---------+
| col   | int(11) |
+-------+---------+
```

You want to add another column with the name `COL` (all uppercase letters). How can you do this?

Question 38:

The table `mytable` has the following structure, with a `UNIQUE` index on its only column, `col`:

```
mysql> DESCRIBE mytable;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| col   | char(10) | YES  | MUL | NULL    |       |
+-------+----------+------+-----+---------+-------+
```

The table is empty. Will the following `INSERT` statement fail?

```
mysql> INSERT INTO mytable VALUES (NULL),(NULL),('data'),('test'),(NULL);
```

Question 39:

Table `mytable` contains the data shown in the following listing. The data should remain unchanged. Is it possible to add a `PRIMARY KEY` to table `mytable`? If it's possible, what SQL statement would you use to create a composite `PRIMARY KEY` for `col1` and `col2` on the table?

```
mysql> SELECT * FROM mytable;
+------+------+
| col1 | col2 |
+------+------+
| yoo  | doo  |
| doo  | yoo  |
| doo  | doo  |
| yoo  | yoo  |
+------+------+
```

Question 40:

What happens if you don't provide an index name when creating an index with `ALTER TABLE` or with `CREATE INDEX`?

Question 41:

Can you drop multiple indexes with a single `DROP INDEX` statement?

Question 42:

Consider the following tables:

```
mysql> SHOW TABLES LIKE 't\_%';
+-----------------------+
| Tables_in_world (t\_%) |
+-----------------------+
| t_active              |
| t_archive             |
| t_template            |
+-----------------------+
```

Assume that you want, *in one step*, to rename those tables like this:

1. `t_archive` should become `t_archive_lastyear`
2. `t_active` should become `t_archive`
3. `t_template` should become `t_active`

What's the SQL statement to use?

Question 43:

Assume that you've created a temporary table like this:

```
mysql> CREATE TEMPORARY TABLE t1 (i INT);
```

How can you rename that table to `t2`?

Question 44:

What `SHOW` statement will retrieve a list of tables in the database `test`, even if `test` isn't the current database?

Question 45:

What `SHOW` statement will retrieve a list of the columns in the table `mytest`, found in the `test` database?

Question 46:

What's the explanation for the following?

```
mysql> CREATE TABLE innodbtable (id INT) ENGINE=InnoDB;
mysql> SHOW TABLE STATUS LIKE 'innodbtable';
+-------------+--------+------------+------+-
| Name        | Engine | Row_format | Rows |
+-------------+--------+------------+------+-
| innodbtable | MyISAM | Fixed      |    0 |
+-------------+--------+------------+------+-
```

Question 47:

How can you change the server's default storage engine for new tables from `MyISAM` to `InnoDB` at server startup?

Question 48:

How can in individual client change its own default storage engine to `InnoDB`?

Answers to Exercises

Answer 1:

False. `InnoDB` allows for a maximum of two billion tables in its tablespace.

Answer 2:

True. However, depending on the storage engine, other files may also be present in the database directory.

Answer 3:

Either of the following approaches will work. Note that you'll have to specify the keyword `PRIMARY` as a quoted identifier if you use `DROP INDEX`.

1. Use `DROP INDEX`:

   ```
   mysql> DROP INDEX `PRIMARY` ON t;
   mysql> DESCRIBE t;
   +-------+---------+------+-----+---------+-------+
   | Field | Type    | Null | Key | Default | Extra |
   +-------+---------+------+-----+---------+-------+
   | i     | int(11) | NO   |     |         |       |
   +-------+---------+------+-----+---------+-------+
   ```

2. Use `ALTER TABLE`:

   ```
   mysql> ALTER TABLE t DROP PRIMARY KEY;
   mysql> DESCRIBE t;
   +-------+---------+------+-----+---------+-------+
   | Field | Type    | Null | Key | Default | Extra |
   +-------+---------+------+-----+---------+-------+
   | i     | int(11) | NO   |     |         |       |
   +-------+---------+------+-----+---------+-------+
   ```

Answer 4:

PRIMARY KEY, UNIQUE, INDEX (non-unique), FULLTEXT, and SPATIAL.

Answer 5:

The columns must be declared NOT NULL.

Answer 6:

By using the IF EXISTS clause; for example, DROP TABLE IF EXISTS *table_name*. This will generate a warning if the table doesn't exist, which you can view with SHOW WARNINGS.

Answer 7:

True. There cannot be a table with zero columns.

Answer 8:

False. Tables do not have to contain data, they may be empty.

Answer 9:

False. You can specify the database in which to create the table by using a fully qualified table name—that is, *db_name.table_name* (for example, mydb.mytable).

Answer 10:

IF NOT EXISTS

Answer 11:

Column specifications must be separated by commas. In this case, there must be a comma between the words KEY and name.

Answer 12:

1. The following single SQL statement creates a copy of the table:

   ```
   mysql> CREATE TABLE test.City SELECT * FROM world.City;
   ```

   The table definition of the target table looks like this:

   ```
   mysql> DESCRIBE test.City;
   +-------------+----------+------+-----+---------+-------+
   | Field       | Type     | Null | Key | Default | Extra |
   +-------------+----------+------+-----+---------+-------+
   | ID          | int(11)  | NO   |     | 0       |       |
   | Name        | char(35) | NO   |     |         |       |
   | CountryCode | char(3)  | NO   |     |         |       |
   | District    | char(20) | NO   |     |         |       |
   | Population  | int(11)  | NO   |     | 0       |       |
   +-------------+----------+------+-----+---------+-------+
   ```

2. The following two SQL statements create a copy of the table, too:

   ```
   mysql> CREATE TABLE test.City LIKE world.City;
   mysql> INSERT INTO test.City SELECT * FROM world.City;
   ```

```
Query OK, 4079 rows affected (0.05 sec)
Records: 4079  Duplicates: 0  Warnings: 0
```

The advantage of the latter two statements over the former solution is that the table definition (indexes, column properties, storage engine, and so forth) is copied one-to-one, too:

```
mysql> DESCRIBE test.City;
+-------------+----------+------+-----+---------+----------------+
| Field       | Type     | Null | Key | Default | Extra          |
+-------------+----------+------+-----+---------+----------------+
| ID          | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name        | char(35) | NO   |     |         |                |
| CountryCode | char(3)  | NO   |     |         |                |
| District    | char(20) | NO   |     |         |                |
| Population  | int(11)  | NO   |     | 0       |                |
+-------------+----------+------+-----+---------+----------------+
```

Answer 13:

True. MySQL supports multiple actions for a single `ALTER TABLE` statement.

Answer 14:

False. Rows cannot be added with an `ALTER TABLE` statement.

Answer 15:

True.

Answer 16:

False. MySQL will tell you to use the `DROP TABLE` statement for this action.

Answer 17:

Either of the following statements renames the table:

```
ALTER TABLE tbl RENAME TO tbl_new;
RENAME TABLE tbl TO tbl_new;
```

Answer 18:

Indexes can speed up retrievals, especially for large tables, and they can be used to place restrictions on columns to ensure that a column or a set of columns may contain only unique-valued entries.

Answer 19:

No, it isn't possible. A `PRIMARY KEY` can only contain columns that are specified as `NOT NULL`.

Answer 20:

```
mysql> ALTER TABLE mytable
    ->  ADD col0 INT FIRST,
    ->  ADD col2 INT AFTER col1,
    ->  ADD col4 INT
    -> ;
```

Answer 21:

`SHOW CREATE TABLE tbl` displays all index information for the table, including composite indexes. `SHOW INDEX FROM tbl` also shows index information, although the output might not be as easy to interpret.

Answer 22:

```
mysql> CREATE TABLE tbl (
    ->  col1 INT NOT NULL,
    ->  col2 INT NOT NULL,
    ->  PRIMARY KEY (col1, col2)
    -> );
```

Answer 23:

There are two main ways of emptying a table:

1. The `TRUNCATE` statement is fastest, but it doesn't tell you how many rows were deleted:

   ```
   mysql> TRUNCATE TABLE City;
   Query OK, 0 rows affected (0.02 sec)
   mysql> SELECT COUNT(*) FROM City;
   +----------+
   | COUNT(*) |
   +----------+
   |        0 |
   +----------+
   ```

2. The `DELETE` statement isn't as fast, but it tells you how many rows were deleted:

   ```
   mysql> DELETE FROM City;
   Query OK, 4079 rows affected (0.00 sec)
   ```

Answer 24:

Either of the following `SHOW` statements provides the desired information:

```
SHOW TABLES FROM test LIKE '%test%';
SHOW TABLE STATUS FROM test LIKE '%test%';
```

`SHOW TABLES` lists just the table names. `SHOW TABLE STATUS` displays the names and additional table information.

The corresponding statement that uses the `INFORMATION_SCHEMA` database to retrieve the desired data is:

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME LIKE '%test%';
```

Answer 25:

```
SHOW COLUMNS FROM mytest FROM test LIKE 'id%';
```

Answer 26:

```
SHOW CREATE TABLE test.mytest;
```

Answer 27:

The default storage engine has been changed to `InnoDB`. For example, this could have been done with the following statement:

```
SET storage_engine = InnoDB;
```

To make sure that the table gets created as a `MyISAM` table, you could do either of the following:

1. Specify the storage engine at table creation time:

   ```
   mysql> CREATE TABLE defaulttype (id INT) ENGINE=MyISAM;
   ```

2. Set the default storage engine to `MyISAM` and then create the table:

   ```
   mysql> SET storage_engine = MyISAM;
   mysql> CREATE TABLE defaulttype (id INT)
   ```

Those are just two out of many possibilities to achieve that result.

Answer 28:

A MySQL user with sufficient privileges can change the server's default storage engine to `InnoDB` while the server is running by using this statement:

```
SET GLOBAL storage_engine=InnoDB;
```

This setting affects all clients that connect after the statement executes.

Answer 29:

a. Use features of the table storage engine, such as by using `MERGE` tables to work around size limitations of the `MyISAM` storage engine.

b. Convert the table for use with a storage engine that allows larger tables. For example, convert `MyISAM` tables to `InnoDB` tables. The `InnoDB` tablespace can consist of several files and `InnoDB` can spread a table's contents over more than one of these files. This allows the table to be larger than any single file.

c. Use another filesystem or a newer version of the operating system that allows for larger files.

Answer 30:

`DROP INDEX idx_id ON tbl` or `ALTER TABLE tbl DROP INDEX idx_id`. You can recover the index by rebuilding it with a `CREATE INDEX` or `ALTER TABLE … ADD INDEX` statement.

Answer 31:

A `UNIQUE` index can contain `NULL` values; a `PRIMARY KEY` cannot. It's possible to have multiple `UNIQUE` indexes for a table, but there can be only one index defined as a `PRIMARY KEY` for each table.

Answer 32:

```
PRIMARY KEY
```

Answer 33:

It's not possible. All tables must belong to a database, and therefore must be created within a database.

Answer 34:

```
mysql> CREATE TABLE mytbl (
    ->  col1 INT UNSIGNED NOT NULL,
    ->  col2 CHAR(50) NOT NULL,
    ->  col3 CHAR(50) NOT NULL,
    ->  PRIMARY KEY(col1),
    ->  UNIQUE(col2),
    ->  INDEX(col3)
    -> );
```

Other variations are possible. For example, the indexes created by the `PRIMARY KEY` and `UNIQUE` clauses could be specified by adding `PRIMARY KEY` to the end of the `col1` definition and `UNIQUE` to the end of the `col2` definition.

Answer 35:

True.

Answer 36:

True. To do so, specify the keyword `FIRST` at the end of the `ADD` clause that provides the column definition.

Answer 37:

You cannot do this. Column names in a table must be unique regardless of lettercase.

Answer 38:

The `Key` value in `DESCRIBE` output for a `UNIQUE` index will be `UNI` or `PRI` if the index does not allow `NULL` values. The `Key` value is `MUL` if the index does allow `NULL` values because `NULL` in a `UNIQUE` index is a special case: Multiple `NULL` values are allowed, unlike any other value. For `mytable`, the `Key` value is `MUL`, which indicates that the `UNIQUE` index on `col` allows multiple `NULL` values. Consequently, the `INSERT` statement will not fail, even though it inserts several `NULL` values.

Answer 39:

Yes. You cannot add a primary key for either column individually because each contains duplicate non-`NULL` values. However, the combination of `col1` and `col2` has unique and non-`NULL` values only, so it's possible to create a `PRIMARY KEY` with this SQL statement:

```
ALTER TABLE mytable ADD PRIMARY KEY (col1, col2);
```

Answer 40:

With `ALTER TABLE … ADD INDEX`, if you don't explicitly provide a name for the index, MySQL creates an index name based on the name of the first indexed column. With `CREATE INDEX`, an error occurs if you don't provide a name for the index.

Answer 41:

No. If you want to drop more than one index at the same time, you must use `ALTER TABLE … DROP INDEX`.

Answer 42:

```
mysql> RENAME TABLE t_archive TO t_archive_lastyear,
    -> t_active TO t_archive, t_template TO t_active;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES LIKE 't\_%';
+---------------------+
| Tables_in_sg5 (t\_%) |
+---------------------+
| t_active            |
| t_archive           |
| t_archive_lastyear  |
+---------------------+
```

Answer 43:

You cannot use `RENAME TABLE` to rename a temporary table. You should use `ALTER TABLE` to do this.

```
mysql> RENAME TABLE t1 TO t2;
ERROR 1017 (HY000): Can't find file: './sg5/t1.frm' (errno: 2)
mysql> ALTER TABLE t1 RENAME TO t2;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Answer 44:

Either of the following statements provides the desired information:

```
SHOW TABLES FROM test;
SHOW TABLE STATUS FROM test;
```

Answer 45:

Either of the following statements will work:

```
SHOW COLUMNS FROM mytest FROM test;
SHOW COLUMNS FROM test.mytest;
```

Answer 46:

If you try to use a storage engine that is not compiled in or has been disabled, MySQL instead creates a table using the default engine. Because the request to create the table as a `InnoDB` table was not honored by the server, this means that the server either was not compiled with support for the `InnoDB` storage engine, or that `InnoDB` was disabled at server startup (using the `--skip-innodb` option). Because the table was created as a `MyISAM` table, the default storage engine must be `MyISAM`.

Answer 47:

A default storage engine of `InnoDB` can be specified at server startup by using the `--default-storage-engine=InnoDB` option, either on the command line or in an option file. For example, you could start the server like this:

```
mysqld --default-storage-engine=InnoDB
```

To use an option file, put these lines in the file:

```
[mysqld]
default-storage-engine=InnoDB
```

Answer 48:

Any client can change its connection-specific default storage engine by issuing this statement:

```
SET SESSION storage_engine=InnoDB;
```

This setting will not affect other client connections.

Chapter 9. Querying for Data

Almost all examples and exercises in this study guide use the *world database* as the sample data set. The accompanying CD-ROM contains the data for this database and instructions that describe how to create and populate the database for use with your own MySQL installation.

Some of the exercises can be done by using subqueries. However, because subqueries are covered in a later chapter, assume that your answers should not use them.

Question 1:

Assume that a table `t` contains a `DATE` column `d`. How would you find the oldest date? How would you find the most recent date?

Question 2:

Assume that a table `t` contains a date-valued column `d` as well as other columns. How would you find the row that contains the oldest date? How would you find the row containing the most recent date? (Note that these are different problems than finding the oldest and most recent date.)

Question 3:

Consider the `Country` table in the `world` database:

```
mysql> SELECT Code, Name, IndepYear FROM Country;
+------+-------------+-----------+
| Code | Name        | IndepYear |
+------+-------------+-----------+
| ABW  | Aruba       |      NULL |
| AFG  | Afghanistan |      1919 |
| AGO  | Angola      |      1975 |
| AIA  | Anguilla    |      NULL |
| ALB  | Albania     |      1912 |
...
```

Considering only those countries with a known (non-`NULL`) value in the `IndepYear` column, how can you determine the answers to the following questions?

a. Which country was the last to become independent?

b. Which country was the first to become independent?

Question 4:

Here's an alphabetical list of some basic clauses for the `SELECT` statement:

- `FROM`
- `GROUP BY`
- `HAVING`
- `LIMIT`
- `ORDER BY`

- `WHERE`

These clauses must be used in a specific order. What's this order?

Question 5:

In general, which of the `SELECT` statement clauses shown in the previous question are optional? Which of them are optional when retrieving data from a table?

Question 6:

You want to retrieve data from a table `City` in the `world` database. Your current database is `test`. How can you refer to the `City` table in the `FROM` clause of `SELECT` statements without changing the default database beforehand?

Question 7:

Consider the `City` table in the `world` database. You want to find cities whose names start with letters B to F and K to M, and that have more than a million inhabitants. The output should be sorted in ascending name order. What query will produce this result?

Question 8:

The table `petbirth` has the following structure and contents:

```
mysql> DESCRIBE petbirth;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| name  | char(20) | YES  |     | NULL    |       |
| birth | date     | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
mysql> SELECT * FROM petbirth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1995-07-29 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
| Lucy     | 1988-05-08 |
| Macie    | 1997-05-08 |
| Myra     | 1997-06-09 |
| Cheep    | 1998-05-08 |
+----------+------------+
```

Using the `MONTH()` and `YEAR()` SQL functions, you want to produce an ordered list that's sorted by year and month of birth:

```
+----------+-------+------+
| Pet      | Month | Year |
+----------+-------+------+
| Puffball |     3 | 1999 |
| Chirpy   |     9 | 1998 |
| Cheep    |     5 | 1998 |
| Whistler |    12 | 1997 |
| Myra     |     6 | 1997 |
| Macie    |     5 | 1997 |
| Slim     |     4 | 1996 |
| Bowser   |     7 | 1995 |
| Claws    |     3 | 1994 |
| Fluffy   |     2 | 1993 |
| Fang     |     8 | 1990 |
| Buffy    |     5 | 1989 |
| Lucy     |     5 | 1988 |
+----------+-------+------+
```

What's the appropriate SQL statement to produce this result?

Question 9:

`SELECT *` type queries are convenient when you want to retrieve all columns of a table. What is a reason not to use the `*` shorthand notation, though?

Question 10:

Consider the `Country` table in the `world` database. Which of the following statements will succeed?

1.
```
mysql> SELECT Name,
    -> GNP * 1.1 AS 'GNP raised by 10%'
    -> FROM Country LIMIT 1;
```

2.
```
mysql> SELECT Name,
    -> GNP * 1.1 GNP raised by 10%
    -> FROM Country LIMIT 1;
```

3.
```
mysql> SELECT Continent,
    -> SUM(GNP) * 1.1 AS 'New GNP'
    -> FROM Country GROUP BY Continent;
```

4.
```
mysql> SELECT Name,
    -> GNP AS 'Gross National Product'
    -> FROM Country WHERE 'Gross National Product' < 1000;
```

5.
```
mysql> SELECT Name,
    -> GNP AS 'Gross National Product'
    -> FROM Country WHERE GNP < 1000;
```

Question 11:

Consider the following table and list of collations:

```
mysql> SELECT * FROM t;
```

```
+------+
| c    |
+------+
| a    |
| A    |
| B    |
| A    |
| b    |
| a    |
+------+

mysql> SHOW COLLATION LIKE 'latin1_%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
...
| latin1_bin        | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci | latin1  | 48 |         |          |       0 |
| latin1_general_cs | latin1  | 49 |         |          |       0 |
...
```

Assume that the default collation is `latin1_general_ci`. What `SELECT` statements will yield the following results?

1.
   ```
   +------+
   | c    |
   +------+
   | a    |
   | A    |
   | A    |
   | a    |
   | B    |
   | b    |
   +------+
   ```

2.
   ```
   +------+
   | c    |
   +------+
   | A    |
   | A    |
   | a    |
   | a    |
   | B    |
   | b    |
   +------+
   ```

3.
   ```
   +------+
   | c    |
   +------+
   | A    |
   | A    |
   | B    |
   | a    |
   | a    |
   | b    |
   +------+
   ```

Question 12:

Consider this query:

```
mysql> SELECT DISTINCT CountryCode FROM CountryLanguage;
```

What's the corresponding `GROUP BY` query that would produce the same set of rows?

Question 13:

The table `petbirth` has the following structure and contents:

```
mysql> DESCRIBE petbirth;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| name  | char(20) | YES  |     | NULL    |       |
| birth | date     | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
mysql> SELECT * FROM petbirth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1995-07-29 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
| Lucy     | 1988-05-08 |
| Macie    | 1997-05-08 |
| Myra     | 1997-06-09 |
| Cheep    | 1998-05-08 |
+----------+------------+
```

You want to display name and birthday of the oldest pet. What's the appropriate SQL statement?

Question 14:

The table `pet` has the following structure and contents:

```
mysql> DESCRIBE pet;
+---------+----------+------+-----+---------+-------+
| Field   | Type     | Null | Key | Default | Extra |
+---------+----------+------+-----+---------+-------+
| name    | char(20) | YES  |     | NULL    |       |
| owner   | char(20) | YES  |     | NULL    |       |
| species | char(20) | YES  |     | NULL    |       |
| gender  | char(1)  | YES  |     | NULL    |       |
+---------+----------+------+-----+---------+-------+
mysql> SELECT * FROM pet;
+----------+--------+---------+--------+
| name     | owner  | species | gender |
+----------+--------+---------+--------+
| Fluffy   | Harold | cat     | f      |
| Claws    | Gwen   | cat     | m      |
| Buffy    | Harold | dog     | f      |
| Fang     | Benny  | dog     | m      |
| Bowser   | Diane  | dog     | m      |
| Chirpy   | Gwen   | bird    | f      |
| Whistler | Gwen   | bird    | NULL   |
| Slim     | Benny  | snake   | m      |
| Puffball | Diane  | hamster | f      |
+----------+--------+---------+--------+
```

What statements would you use to produce the following results?

a. The number of male and female pets (discarding the pets whose gender is unknown)

b. The number of pets of each species, with the species having the highest number of individuals appearing first

c. The number of dogs and cats, with the species that has the highest number of individuals to appear first, using a `WHERE` clause

d. The number of dogs and cats, with the species which has the highest number of individuals to appear first, using a `HAVING` clause

The column headings for the results should be `Kind`, `Gender`, and `Total`.

Question 15:

The table `personnel` has the following structure and contents:

```
mysql> DESCRIBE personnel;
+--------+----------------------+------+-----+---------+-------+
| Field  | Type                 | Null | Key | Default | Extra |
+--------+----------------------+------+-----+---------+-------+
| pid    | smallint(5) unsigned | NO   | PRI |         |       |
| unit   | tinyint(3) unsigned  | NO   |     |         |       |
| salary | decimal(9,2)         | NO   |     |         |       |
+--------+----------------------+------+-----+---------+-------+
mysql> SELECT * FROM personnel;
+-----+------+---------+
| pid | unit | salary  |
+-----+------+---------+
|   1 |   42 | 1500.00 |
|   2 |   42 | 1700.00 |
|   3 |   42 | 1950.00 |
|   4 |   42 | 2300.00 |
|   5 |   42 | 1900.00 |
|   6 |   23 |  850.00 |
|   7 |   23 | 1250.00 |
|   8 |   23 | 1450.00 |
|   9 |   23 | 1920.00 |
|  10 |   42 | 2200.00 |
|  11 |   23 | 2900.00 |
|  12 |   23 | 1000.00 |
|  13 |   42 | 2850.00 |
+-----+------+---------+
```

What statements would you use to retrieve the following information?

a. Find the number of employees, the salary total, and the average salary per employee for the two company units, with the highest total salary appearing first. The output should have column headings that should look like this:

```
+------+-----------+----------+------------+
| Unit | Employees | Total    | Average    |
+------+-----------+----------+------------+
```

b. Identify the highest and the lowest salary per unit. The output should have column headings that should look like this:

```
+------+---------+---------+
| Unit | High    | Low     |
+------+---------+---------+
```

Question 16:

Consider the `Country` table of the `world` database. Use the `GROUP_CONCAT()` function to produce one row per continent that displays a list of the countries in the continent that have a country population of more than 100 million.

Question 17:

Consider the `Country` table of the `world` database. Assume that you want to get a list of Dutch districts (`CountryCode = 'NLD'`) that looks like this:

```
Dutch Districts: Drenthe - Flevoland - Gelderland - Gelderland -
Gelderland - Gelderland - Groningen - Limburg - Limburg - Noord-Brabant -
Noord-Brabant - Noord-Brabant - Noord-Brabant - Noord-Holland -
Noord-Holland - Noord-Holland - Noord-Holland - Noord-Holland -
Overijssel - Overijssel - Utrecht - Utrecht - Zuid-Holland -
Zuid-Holland - Zuid-Holland - Zuid-Holland - Zuid-Holland - Zuid-Holland
```

What's the query that yields that result?

Question 18:

The table `pet` has the following structure and contents:

```
mysql> DESCRIBE pet;
+---------+----------+------+-----+---------+-------+
| Field   | Type     | Null | Key | Default | Extra |
+---------+----------+------+-----+---------+-------+
| name    | char(20) | YES  |     | NULL    |       |
| owner   | char(20) | YES  |     | NULL    |       |
| species | char(20) | YES  |     | NULL    |       |
| gender  | char(1)  | YES  |     | NULL    |       |
+---------+----------+------+-----+---------+-------+
mysql> SELECT * FROM pet;
+----------+--------+---------+--------+
| name     | owner  | species | gender |
+----------+--------+---------+--------+
| Fluffy   | Harold | cat     | f      |
| Claws    | Gwen   | cat     | m      |
| Buffy    | Harold | dog     | f      |
| Fang     | Benny  | dog     | m      |
| Bowser   | Diane  | dog     | m      |
| Chirpy   | Gwen   | bird    | f      |
| Whistler | Gwen   | bird    | NULL   |
| Slim     | Benny  | snake   | m      |
| Puffball | Diane  | hamster | f      |
+----------+--------+---------+--------+
```

What `COUNT()` values will the following statements return?

a. `SELECT COUNT(*) FROM pet;`

b. `SELECT COUNT(gender) FROM pet;`

c. `SELECT COUNT(DISTINCT gender) FROM pet;`

d. `SELECT COUNT(DISTINCT species) FROM pet;`

Question 19:

The table `pet` has the following structure and contents:

```
mysql> DESCRIBE pet;
+---------+----------+------+-----+---------+-------+
| Field   | Type     | Null | Key | Default | Extra |
+---------+----------+------+-----+---------+-------+
| name    | char(20) | YES  |     | NULL    |       |
| owner   | char(20) | YES  |     | NULL    |       |
| species | char(20) | YES  |     | NULL    |       |
| gender  | char(1)  | YES  |     | NULL    |       |
+---------+----------+------+-----+---------+-------+
mysql> SELECT * FROM pet;
+----------+--------+---------+--------+
| name     | owner  | species | gender |
+----------+--------+---------+--------+
| Fluffy   | Harold | cat     | f      |
| Claws    | Gwen   | cat     | m      |
| Buffy    | Harold | dog     | f      |
| Fang     | Benny  | dog     | m      |
| Bowser   | Diane  | dog     | m      |
| Chirpy   | Gwen   | bird    | f      |
| Whistler | Gwen   | bird    | NULL   |
| Slim     | Benny  | snake   | m      |
| Puffball | Diane  | hamster | f      |
+----------+--------+---------+--------+
```

Write an SQL statement to produce the following output:

```
+---------+--------+-------+
| Species | Gender | Total |
+---------+--------+-------+
| bird    | NULL   |     1 |
| bird    | f      |     1 |
| cat     | f      |     1 |
| cat     | m      |     1 |
| dog     | f      |     1 |
| dog     | m      |     2 |
| hamster | f      |     1 |
| snake   | m      |     1 |
+---------+--------+-------+
```

Question 20:

Using `WITH ROLLUP`, write a single statement that queries the `Country` table to produce counts of the number of countries in each continent, and a count of the total number of countries summed over all continents.

Question 21:

An e-commerce company has a different database for each project. For each project, it has a table that holds the team members. Unluckily, this information is not organized consistently, so there are three tables in three different databases that look like this:

```
mysql> SELECT * FROM project1.user;
+-------+-------------+
| name  | job         |
+-------+-------------+
| John  | Manager     |
| Steve | Programmer  |
| Andy  | Webdesigner |
+-------+-------------+
mysql> SELECT * FROM project2.users;
+-------+------------+
| nick  | task       |
+-------+------------+
| Jim   | Manager    |
| Steve | Programmer |
+-------+------------+
mysql> SELECT * FROM project3.members;
+--------+-------------+
| member | job         |
+--------+-------------+
| John   | Manager     |
| Steve  | Programmer  |
| Carol  | Webdesigner |
+--------+-------------+
```

Assume that you want output with column headings that look like this:

```
+------------+-------------+
| TeamMember | TeamTask    |
+------------+-------------+
```

What SQL statement will give you a list of all team members, sorted by their names? What would the statement be if you don't want team members to appear more than once in the list?

Question 22:

The table `personnel` has the following structure and contents:

```
mysql> DESCRIBE personnel;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| pid   | smallint(5) unsigned | NO   | PRI |         |       |
| unit  | tinyint(3) unsigned  | YES  |     | NULL    |       |
| grade | tinyint(3) unsigned  | YES  |     | NULL    |       |
+-------+----------------------+------+-----+---------+-------+
mysql> SELECT * FROM personnel;
```

```
+-----+------+-------+
| pid | unit | grade |
+-----+------+-------+
|   1 |   42 |     1 |
|   2 |   42 |     2 |
|   3 |   42 |  NULL |
|   4 |   42 |  NULL |
|   5 |   42 |  NULL |
|   6 |   23 |     1 |
|   7 |   23 |     1 |
|   8 |   23 |     1 |
|   9 |   23 |  NULL |
|  10 |   42 |  NULL |
|  11 |   23 |  NULL |
|  12 |   23 |     1 |
|  13 |   42 |  NULL |
+-----+------+-------+
```

What result will the following statement yield?

```
SELECT unit, COUNT(grade) FROM personnel GROUP BY unit;
```

Question 23:

Refer to the structure and contents shown for the `personnel` table in the previous question. What result will the following statement yield?

```
SELECT unit, SUM(grade) FROM personnel GROUP BY unit;
```

Question 24:

Refer to the structure and contents shown for the `personnel` table two questions earlier. What result will the following statement yield?

```
SELECT unit, AVG(grade) FROM personnel GROUP BY unit;
```

Question 25:

The table `personnel` has the following structure and contents:

```
mysql> DESCRIBE personnel;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| pid   | smallint(5) unsigned | NO   | PRI |         |       |
| unit  | tinyint(3) unsigned  | YES  |     | NULL    |       |
| grade | tinyint(3) unsigned  | YES  |     | NULL    |       |
+-------+----------------------+------+-----+---------+-------+
mysql> SELECT * FROM personnel;
+-----+------+-------+
| pid | unit | grade |
+-----+------+-------+
|   1 |   42 |     1 |
|   2 |   42 |     2 |
|   3 |   42 |  NULL |
|   4 |   42 |  NULL |
|   5 |   42 |  NULL |
|   6 |   23 |     1 |
|   7 |   23 |     1 |
|   8 |   23 |     1 |
|   9 |   23 |  NULL |
|  10 |   42 |  NULL |
|  11 |   23 |  NULL |
|  12 |   23 |     1 |
|  13 |   42 |  NULL |
+-----+------+-------+
```

What result will the following statement yield?

```
SELECT unit, COUNT(*) FROM personnel GROUP BY unit;
```

Question 26:

Refer to the structure and contents shown for the `personnel` table in the previous question. What result will the following statement yield?

```
SELECT unit, COUNT(DISTINCT grade) FROM personnel GROUP BY unit;
```

Answers to Exercises

Answer 1:

The following queries find the oldest and most recent dates in the column `d`:

```
SELECT MIN(d) FROM t;
SELECT MAX(d) FROM t;
```

Answer 2:

To find the rows containing the oldest and most recent dates in the column `d`, sort the dates in ascending or descending date order, and return the first row of the result:

```
SELECT * FROM t ORDER BY d LIMIT 1;
SELECT * FROM t ORDER BY d DESC LIMIT 1;
```

Answer 3:

a. Here's the last country that became independent:

```
mysql> SELECT Code, Name, IndepYear FROM Country
    -> WHERE IndepYear IS NOT NULL
    -> ORDER BY IndepYear DESC LIMIT 1;
+------+-------+-----------+
| Code | Name  | IndepYear |
+------+-------+-----------+
| PLW  | Palau |      1994 |
+------+-------+-----------+
```

b. Here's the first country that became independent:

```
mysql> SELECT Code, Name, IndepYear FROM Country
    -> WHERE IndepYear IS NOT NULL
    -> ORDER BY IndepYear ASC LIMIT 1;
+------+-------+-----------+
| Code | Name  | IndepYear |
+------+-------+-----------+
| CHN  | China |     -1523 |
+------+-------+-----------+
```

Answer 4:

The clauses must be used in this order:

1. `FROM`
2. `WHERE`
3. `GROUP BY`
4. `HAVING`
5. `ORDER BY`
6. `LIMIT`

Answer 5:

All the `SELECT` clauses shown in the previous question are optional. When you retrieve data from a table, the `FROM` clause is mandatory; all other clauses are optional.

Answer 6:

You can use a fully qualified table name. That is, precede the table name with the database name as a qualifier. The `City` table would be referred to as `world.City`.

Answer 7:

The following query retrieves cities having names that begin with B, C, D, E, F, K, L, or M and that have more than a million inhabitants:

```
mysql> SELECT Name, Population FROM City
    ->  WHERE (
    ->         (Name >= 'B' AND Name < 'G')
    ->         OR
    ->         (Name >= 'K' AND Name < 'N')
    ->        )
    ->         AND Population > 1000000
    ->  ORDER BY Name ASC
    -> ;
```

Other solutions are possible.

Answer 8:

Birthdays of pets (months and years only) in descending order:

```
mysql> SELECT
    ->  name         AS Pet,
    ->  MONTH(birth) AS Month,
    ->  YEAR(birth)  AS Year
    -> FROM petbirth
    -> ORDER BY Year DESC, Month DESC
    -> ;
```

Answer 9:

If you want to retrieve the table columns in a particular order, you cannot use `*`.

Answer 10:

1. The query will succeed.
2. The query will result in an error. Column aliases that contain spaces must be quoted.
3. The query will succeed.
4. The query will yield unpredictable results. You must not use a column alias in the `WHERE` clause, so the string in that clause is interpreted only as a string, not as a column reference.
5. The query will succeed.

Answer 11:

1. Either of the following statements will work:

   ```
   mysql> SELECT c FROM t ORDER BY c;
   mysql> SELECT c FROM t ORDER BY c COLLATE latin1_general_ci;
   ```

2.
   ```
   mysql> SELECT c FROM t ORDER BY c COLLATE latin1_general_cs;
   ```

3.
   ```
   mysql> SELECT c FROM t ORDER BY c COLLATE latin1_bin;
   ```

Answer 12:

```
mysql> SELECT CountryCode FROM CountryLanguage GROUP BY CountryCode;
```

Answer 13:

To retrieve information for the oldest animal, sort the table in birth order and limit the result to the first row:

```
mysql> SELECT
    ->  name, birth
    -> FROM petbirth
    -> ORDER BY birth ASC
    -> LIMIT 1
    -> ;
+------+------------+
| name | birth      |
+------+------------+
| Lucy | 1988-05-08 |
+------+------------+
```

Answer 14:

The following queries produce the desired results:

a. Number of pets by gender:

```
mysql> SELECT
    ->  gender AS Gender,
    ->  COUNT(*) AS Total
    -> FROM pet
    -> WHERE gender IS NOT NULL
    -> GROUP BY Gender;
+--------+-------+
| Gender | Total |
+--------+-------+
| f      |     4 |
| m      |     4 |
+--------+-------+
```

b. Number of pets by species:

```
mysql> SELECT
    ->  species AS Kind,
    ->  COUNT(*) AS Total
    -> FROM pet
    -> GROUP BY Kind
    -> ORDER BY Total DESC;
+---------+-------+
| Kind    | Total |
+---------+-------+
| dog     |     3 |
| cat     |     2 |
| bird    |     2 |
| snake   |     1 |
| hamster |     1 |
+---------+-------+
```

c. Number of dogs and cats, using a `WHERE` clause:

```
mysql> SELECT
    ->  species AS Kind,
    ->  COUNT(*) AS Total
    -> FROM pet
    -> WHERE species='dog' OR species='cat'
    -> GROUP BY Kind
    -> ORDER BY Total DESC;
+------+-------+
| Kind | Total |
+------+-------+
| dog  |     3 |
| cat  |     2 |
+------+-------+
```

d. Number of dogs and cats, using a `HAVING` clause:

```
mysql> SELECT
    ->  species AS Kind,
    ->  COUNT(*) AS Total
    -> FROM pet
```

```
        -> GROUP BY Kind
        -> HAVING Kind='dog' OR Kind='cat'
        -> ORDER BY Total DESC;
    +------+-------+
    | Kind | Total |
    +------+-------+
    | dog  |     3 |
    | cat  |     2 |
    +------+-------+
```

Answer 15:

a. Number of employees, total, and average salary per unit:

```
    mysql> SELECT
        ->  unit AS Unit,
        ->  COUNT(*) AS Employees,
        ->  SUM(salary) AS Total,
        ->  AVG(salary) AS Average
        -> FROM personnel
        -> GROUP BY Unit
        -> ORDER BY Total DESC
        -> ;
    +------+-----------+----------+-------------+
    | Unit | Employees | Total    | Average     |
    +------+-----------+----------+-------------+
    |   42 |         7 | 14400.00 | 2057.142857 |
    |   23 |         6 |  9370.00 | 1561.666667 |
    +------+-----------+----------+-------------+
```

b. Highest and lowest salary per unit:

```
    mysql> SELECT
        ->  unit AS Unit,
        ->  MAX(salary) AS High,
        ->  MIN(salary) AS Low
        -> FROM personnel
        -> GROUP BY Unit
        -> ;
    +------+---------+---------+
    | Unit | High    | Low     |
    +------+---------+---------+
    |   23 | 2900.00 |  850.00 |
    |   42 | 2850.00 | 1500.00 |
    +------+---------+---------+
```

Answer 16:

```
mysql> SELECT Continent, GROUP_CONCAT(Name)
    -> FROM Country
    -> WHERE Population > 100000000
    -> GROUP BY Continent;
+---------------+--------------------------------------------------+
| Continent     | GROUP_CONCAT(Name)                               |
+---------------+--------------------------------------------------+
| Asia          | Bangladesh,Pakistan,Japan,India,Indonesia,China |
| Europe        | Russian Federation                               |
| North America | United States                                  |
| Africa        | Nigeria                                        |
| South America | Brazil                                         |
+---------------+------------------------------------------------+
```

Answer 17:

```
mysql> SELECT GROUP_CONCAT(District ORDER BY District SEPARATOR " - ")
    -> AS 'Dutch Districts' FROM City
    -> WHERE CountryCode = 'NLD'\G
```

Answer 18:

a. This statement produces the total number of pets in the table:

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

b. `COUNT(gender)` counts only non-`NULL` values, so the statement counts only those rows containing a known gender. Because there's one `NULL` value in the `gender` column, that row isn't counted:

```
mysql> SELECT COUNT(gender) FROM pet;
+---------------+
| COUNT(gender) |
+---------------+
|             8 |
+---------------+
```

c. The statement counts the number of distinct known genders. The `NULL` gender of the bird isn't counted:

```
mysql> SELECT COUNT(DISTINCT gender) FROM pet;
+------------------------+
| COUNT(DISTINCT gender) |
+------------------------+
|                      2 |
+------------------------+
```

d. The statement counts the number of distinct known species:

```
mysql> SELECT COUNT(DISTINCT species) FROM pet;
+-------------------------+
| COUNT(DISTINCT species) |
+-------------------------+
|                       5 |
+-------------------------+
```

Answer 19:

The output counts the number of individuals for each combination of species and gender. It's displayed sorted by species and by gender within species. The following statement produces the output:

```
mysql> SELECT
    ->  species AS Species,
    ->  gender  AS Gender,
    ->  COUNT(*) AS Total
    -> FROM pet
    -> GROUP BY Species, Gender
    -> ;
```

Answer 20:

```
mysql> SELECT Continent, COUNT(*)
    -> FROM Country
    -> GROUP BY Continent WITH ROLLUP;
+---------------+----------+
| Continent     | COUNT(*) |
+---------------+----------+
| Asia          |       51 |
| Europe        |       46 |
| North America |       37 |
| Africa        |       58 |
| Oceania       |       28 |
| Antarctica    |        5 |
| South America |       14 |
| NULL          |      239 |
+---------------+----------+
```

Answer 21:

To display all team members, issue this statement:

```
mysql>  SELECT
    ->   name AS TeamMember,
    ->   job  AS TeamTask
    ->  FROM project1.user
    -> UNION ALL
    ->  SELECT * FROM project2.users
    -> UNION ALL
    ->  SELECT * FROM project3.members
    -> ORDER BY TeamMember
    -> ;
+------------+-------------+
| TeamMember | TeamTask    |
+------------+-------------+
| Andy       | Webdesigner |
| Carol      | Webdesigner |
| Jim        | Manager     |
| John       | Manager     |
| John       | Manager     |
| Steve      | Programmer  |
| Steve      | Programmer  |
| Steve      | Programmer  |
+------------+-------------+
```

To remove duplicates of team members, the statement is the same, with the exception that you omit the `ALL` keyword:

```
mysql>  SELECT
    ->   name AS TeamMember,
    ->   job  AS TeamTask
    ->  FROM project1.user
    -> UNION
    ->  SELECT * FROM project2.users
    -> UNION
    ->  SELECT * FROM project3.members
    -> ORDER BY TeamMember
    -> ;
+------------+-------------+
| TeamMember | TeamTask    |
+------------+-------------+
| Andy       | Webdesigner |
| Carol      | Webdesigner |
| Jim        | Manager     |
| John       | Manager     |
| Steve      | Programmer  |
+------------+-------------+
```

Answer 22:

The statement provides the number of grades assigned for each unit. It counts only assigned grades; that is, grades that are not `NULL`:

```
mysql> SELECT unit, COUNT(grade) FROM personnel GROUP BY unit;
+------+--------------+
| unit | COUNT(grade) |
+------+--------------+
|   23 |            4 |
|   42 |            2 |
+------+--------------+
```

Answer 23:

For each unit, the statement sums up grades that aren't `NULL`:

```
mysql> SELECT unit, SUM(grade) FROM personnel GROUP BY unit;
+------+------------+
| unit | SUM(grade) |
+------+------------+
|   23 |          4 |
|   42 |          3 |
+------+------------+
```

Answer 24:

For each unit, the statement calculates the average value of grades that aren't `NULL`:

```
mysql> SELECT unit, AVG(grade) FROM personnel GROUP BY unit;
+------+------------+
| unit | AVG(grade) |
+------+------------+
|   23 |     1.0000 |
|   42 |     1.5000 |
+------+------------+
```

Answer 25:

The statement counts the number of rows for each unit:

```
mysql> SELECT unit, COUNT(*) FROM personnel GROUP BY unit;
```

```
+------+----------+
| unit | COUNT(*) |
+------+----------+
|   23 |        6 |
|   42 |        7 |
+------+----------+
```

Answer 26:

The statement counts how many different non-`NULL` grades there are for each unit:

```
mysql> SELECT unit, COUNT(DISTINCT grade) FROM personnel GROUP BY unit;
+------+-----------------------+
| unit | COUNT(DISTINCT grade) |
+------+-----------------------+
|   23 |                     1 |
|   42 |                     2 |
+------+-----------------------+
```

Chapter 10. SQL Expressions

Almost all examples and exercises in this study guide use the *world database* as the sample data set. The accompanying CD-ROM contains the data for this database and instructions that describe how to create and populate the database for use with your own MySQL installation.

Question 1:

Consider the following query:

```
mysql> SELECT CONCAT(IF(IndepYear IS NOT NULL,
    -> CONCAT(Name, ' became independent in ', IndepYear),
    -> CONCAT(Name, ' is not independent')), '.')
    -> AS 'Independent or not?'
    -> FROM Country LIMIT 10;
+---------------------------------------------------+
| Independent or not?                               |
+---------------------------------------------------+
| Aruba is not independent.                         |
| Afghanistan became independent in 1919.           |
| Angola became independent in 1975.                |
| Anguilla is not independent.                      |
| Albania became independent in 1912.               |
| Andorra became independent in 1278.               |
| Netherlands Antilles is not independent.          |
| United Arab Emirates became independent in 1971.  |
| Argentina became independent in 1816.             |
| Armenia became independent in 1991.               |
+---------------------------------------------------+
```

What types of values are used in this query?

Question 2:

What's the explanation for the different results of the two comparisons in the following query?

```
mysql> SELECT 3.1415 + 0.9585 = 4.1, 3.1415E0 + 0.9585E0 = 4.1E0;
+-----------------------+-----------------------------+
| 3.1415 + 0.9585 = 4.1 | 3.1415E0 + 0.9585E0 = 4.1E0 |
+-----------------------+-----------------------------+
|                     1 |                           0 |
+-----------------------+-----------------------------+
```

Question 3:

Why does the first statement succeed, whereas the second one fails?

```
mysql> SELECT "Let's see whether this works.";
+-------------------------------+
| Let's see whether this works. |
+-------------------------------+
| Let's see whether this works. |
+-------------------------------+

mysql> SELECT "Let's see whether this works.";
ERROR 1054 (42S22): Unknown column 'Let's see whether this works.' in
'field list'
```

Question 4:

Consider the following statements. The first SELECT yields true, whereas the second SELECT yields false. What's the explanation for this?

```
mysql> SET @str1 = 'Hühner-Hugos Hähnchen sind schön.';
mysql> SET @str2 = 'Huehner-Hugos Haehnchen sind schoen.';
mysql> SELECT @str1 = @str2;
+---------------+
| @str1 = @str2 |
+---------------+
|             0 |
+---------------+
```

```
mysql> SET @str1 = 'Hühner-Hugos Hähnchen sind schön.';
mysql> SET @str2 = 'Huehner-Hugos Haehnchen sind schoen.';
mysql> SELECT @str1 = @str2;
+---------------+
| @str1 = @str2 |
+---------------+
|             1 |
+---------------+
```

Question 5:

What will the following statement yield, true or false?

```
mysql> SELECT 'Hello world!' = BINARY 'Hello world!';
```

- It will result in an error because it compares a non-binary string with a binary string, which is undefined.
- It will result in NULL because it compares a non-binary string with a binary string, which is undefined.
- It will result in true (1) because, if one string is binary, both strings are treated as binary in the comparison.
- It's not possible to answer the question without knowing the current character set and collation.

Question 6:

Consider the following statements. How should you rephrase the second statement so that it produces uppercase letters?

```
mysql> SELECT @var := BINARY 'Lennart';
+-------------------------+
| @var := BINARY 'Lennart' |
+-------------------------+
| Lennart                 |
+-------------------------+

mysql> SELECT UPPER(@var);
+-------------+
| UPPER(@var) |
+-------------+
| Lennart     |
+-------------+
```

Question 7:

Why does the first statement fail, whereas the second one succeeds?

```
mysql> SELECT NOW ();
ERROR 1305 (42000): FUNCTION world.NOW does not exist
```

```
mysql> SELECT NOW ();
+---------------------+
| NOW ()              |
+---------------------+
| 2005-06-02 10:38:46 |
+---------------------+
```

Question 8:

An SQL expression can consist of constants. What kind of constants are there? What else could an SQL expression consist of?

Question 9:

Give an example of a `SELECT` statement that contains an expression using numeric constants.

Question 10:

Give an example of a `SELECT` statement that contains an expression using string constants.

Question 11:

Give an example of a `SELECT` statement that contains an expression using date constants.

Question 12:

Give an example of a `SELECT` statement that contains an expression using time constants.

Question 13:

Give an example of a `SELECT` statement that contains an expression using a column reference.

Question 14:

Give an example of a `SELECT` statement that contains an expression using a function call.

Question 15:

Give an example of a `SELECT` statement that contains an expression using both a temporal value and a function call.

Question 16:

Suppose that the table `personnel` has the following data for employees in two organizational units:

```
mysql> SELECT * FROM personnel;
+-----+------+---------+
| pid | unit | salary  |
+-----+------+---------+
|   1 |   42 | 1500.00 |
|   2 |   42 | 1700.00 |
|   3 |   42 | 1950.00 |
|   4 |   42 | 2300.00 |
|   5 |   42 | 1900.00 |
|   6 |   23 |  850.00 |
|   7 |   23 | 1250.00 |
|   8 |   23 | 1450.00 |
|   9 |   23 | 1920.00 |
|  10 |   42 | 2200.00 |
|  11 |   23 | 2900.00 |
|  12 |   23 | 1000.00 |
|  13 |   42 | 2850.00 |
+-----+------+---------+
```

What SQL statement would you issue to retrieve a list showing tax deductions for each employee? Assume that the deduction is 40% of the salary. Gross salary, deduction, and net salary should be displayed with descriptive, instead of mathematical, headings (for example, use `Deduction` as a heading, not the expression used to calculate the deduction).

Question 17:

Refer to the data shown for the `personnel` table in the previous question. What SQL statement would you issue to retrieve a list showing tax deductions for each unit (displayed with descriptive headings instead of mathematical headings)? Assume that the deduction is 40% of the salary.

Question 18:

Refer to the data shown for the `personnel` table two questions earlier. What SQL statement would you issue to retrieve a list showing tax deductions for each employee? Assume that the deduction is 40% of the salary for employees with a salary of 2,000 or more, and 30% for those who earn less. The result should show descriptive headings.

Question 19:

Suppose that the table `personnel` has the following data for employees in two organizational units:

```
mysql> SELECT * FROM personnel;
+-----+------+---------+
| pid | unit | salary  |
+-----+------+---------+
|   1 |   42 | 1500.00 |
|   2 |   42 | 1700.00 |
|   3 |   42 | 1950.00 |
|   4 |   42 | 2300.00 |
|   5 |   42 | 1900.00 |
|   6 |   23 |  850.00 |
|   7 |   23 | 1250.00 |
|   8 |   23 | 1450.00 |
|   9 |   23 | 1920.00 |
|  10 |   42 | 2200.00 |
|  11 |   23 | 2900.00 |
|  12 |   23 | 1000.00 |
|  13 |   42 | 2850.00 |
+-----+------+---------+
```

What SQL statement would you issue to retrieve a list showing the cost rise for the organization if employee salaries are increased by 10%? The result should show descriptive headings instead of mathematical headings (for example, use `Cost Rise` as a heading, not the expression used to calculate the cost rise).

Question 20:

Refer to the data shown for the `personnel` table in the previous question. What SQL statement would you issue to retrieve a list showing the cost rise for the units if employee salaries are increased by 10% for unit 23 and by 5% for unit 42 (displayed with descriptive headings instead of mathematical headings)?

Question 21:

Suppose that the table `leonardo` has the following structure and contents:

```
mysql> DESCRIBE leonardo;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| name  | char(7) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
mysql> SELECT * FROM leonardo;
+---------+
| name    |
+---------+
| Lennart |
| lennart |
| LENNART |
| lEnNaRt |
+---------+
```

What output will the following statements yield?

```
SELECT DISTINCT name FROM leonardo;
```

```
SELECT name, COUNT(*) FROM leonardo GROUP BY name;
```

```
SELECT name, COUNT(*) FROM leonardo;
```

Question 22:

Suppose that the table `leonardo` has the following structure and contents:

```
mysql> DESCRIBE leonardo;
+-------+-----------+------+-----+---------+-------+
| Field | Type      | Null | Key | Default | Extra |
+-------+-----------+------+-----+---------+-------+
| name  | binary(7) | YES  |     | NULL    |       |
+-------+-----------+------+-----+---------+-------+
mysql> SELECT * FROM leonardo;
+---------+
| name    |
+---------+
| Lennart |
| lennart |
| LENNART |
| lEnNaRt |
+---------+
```

What output will the following statements yield?

```
SELECT DISTINCT name FROM leonardo;
SELECT name, COUNT(*) FROM leonardo GROUP BY name;
SELECT name, COUNT(*) FROM leonardo;
```

Question 23:

The SQL functions `LENGTH()` and `CHAR_LENGTH()` both return the length of a string. Why are there two functions for the same apparent functionality?

Question 24:

What's the result of the following string comparison?

```
SELECT MD5('lennart') = MD5('LENNART');
```

Question 25:

The `IF()` function returns the second argument if the condition in the first argument evaluates to true; otherwise, it returns the third argument. Knowing this, what do you expect `IF()` to return for the following comparisons?

a. `SELECT IF('ABC' = 'abc','TRUE','FALSE');`

b. `SELECT IF('ABC' = BINARY 'abc','TRUE','FALSE');`

c. `SELECT IF(BINARY 'ABC' = BINARY 'abc','TRUE','FALSE');`

Question 26:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`? (The pattern contains three underscores.)

```
SELECT 'abc' LIKE '%___%';
```

Question 27:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`? (The pattern contains five underscores.)

```
SELECT 'abc' LIKE '%_____%';
```

Question 28:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT '' LIKE '%';
```

Question 29:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT '' LIKE ' % ';
```

Question 30:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT '%' LIKE ' % ';
```

Question 31:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT ' % ' LIKE '%';
```

Question 32:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT 'Lennart' LIKE '_e%_t';
```

Question 33:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT 'Lennart' LIKE '_e%';
```

Question 34:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT NULL LIKE NULL;
```

Question 35:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT NULL LIKE '%';
```

Question 36:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT 'NULL' LIKE '%';
```

Question 37:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT BINARY 'NULL' LIKE 'null';
```

Question 38:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT '2002-02-08' LIKE '2002%';
```

Question 39:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT 23.42 LIKE '2%2';
```

Question 40:

Will the following query evaluate to true (`1`), false (`0`), or `NULL`?

```
SELECT 023.420 LIKE '2%2';
```

Question 41:

Suppose that the table `CityList` has the following structure:

```
mysql> DESCRIBE CityList;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| ID         | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name       | char(35) | NO   |     |         |                |
| Country    | char(3)  | NO   |     |         |                |
| District   | char(20) | NO   |     |         |                |
| Population | int(11)  | NO   |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
```

You want to retrieve the list of cities in the United States of America (`USA`), Denmark (`DNK`), and Germany (`DEU`) that have a number of inhabitants between 400,000 and 500,000, sorted from largest to smallest. Use the `IN()` operator for the country list to perform this query. The list should look like this:

```
+----------------+---------+------------+
| City           | Country | Population |
+----------------+---------+------------+
| Koebenhavn     | DNK     |     495699 |
| Leipzig        | DEU     |     489532 |
| Tucson         | USA     |     486699 |
| Nürnberg       | DEU     |     486628 |
| New Orleans    | USA     |     484674 |
| Las Vegas      | USA     |     478434 |
| Cleveland      | USA     |     478403 |
| Dresden        | DEU     |     476668 |
| Long Beach     | USA     |     461522 |
| Albuquerque    | USA     |     448607 |
| Kansas City    | USA     |     441545 |
| Fresno         | USA     |     427652 |
| Virginia Beach | USA     |     425257 |
| Atlanta        | USA     |     416474 |
| Sacramento     | USA     |     407018 |
+----------------+---------+------------+
```

Question 42:

Will the following statement evaluate to true (`1`), false (`0`), `NULL`, or result in an error?

```
SELECT NULL == NULL;
```

Question 43:

Will the following statement evaluate to true (`1`), false (`0`), `NULL`, or result in an error?

```
SELECT NULL = NULL = NULL;
```

Question 44:

Will the following statement evaluate to true (`1`), false (`0`), `NULL`, or result in an error?

```
SELECT NULL = NULL = 0;
```

Question 45:

Will the following statement evaluate to true (`1`), false (`0`), `NULL`, or result in an error?

```
SELECT NULL IS NOT 0;
```

Question 46:

Will the following statement evaluate to true (`1`), false (`0`), `NULL`, or result in an error?

```
SELECT 0 IS NOT NULL;
```

Question 47:

Will the following statement evaluate to true (`1`), false (`0`), `NULL`, or result in an error?

```
SELECT 0 <=> 1;
```

Question 48:

Is the comment in the following statement legal in MySQL?

```
INSERT /*! DELAYED */ INTO mytable VALUES (5);
```

Question 49:

Is the comment in the following statement legal in MySQL?

```
SELECT * FROM mytable WHERE id < 100 /*!40000 FOR UPDATE */;
```

Question 50:

Is the comment in the following statement legal in MySQL?

```
CREATE /*32303 TEMPORARY */ TABLE tbl (col INT);
```

Question 51:

Is the comment in the following statement legal in MySQL?

```
SELECT * FROM mytable; // * is not good style, though
```

Question 52:

Is the comment in the following statement legal in MySQL?

```
SELECT a, b, c FROM tbl; --different column list needed?
```

Question 53:

Which SQL functions could you use to store encrypted information? What functions could you use to retrieve the stored information unencrypted? Are there special prerequisites or requirements for using these functions?

Question 54:

What's the result of the following statement?

```
mysql> SELECT ROUND(1.5), ROUND(-1.5);
```

- `2` and `-1`.
- `1` and `-1`.
- `2` and `-2`.
- `2` in both cases.

Answers to Exercises

Answer 1:

The query uses these types of values:

- Table columns: `Name` and `IndepYear`.
- Literal values: `' became independent in '`, `' is not independent'`, `'.'`, and `'Independent or not?'`.
- Functions: The string function `CONCAT()` and the logical function `IF()`.

Answer 2:

In the first expression, exact values are used, whereas in the second expression, approximate values are used and rounding error is possible.

Answer 3:

The SQL modes were different for the two statements:

```
mysql> SELECT @@sql_mode;
+------------+
| @@sql_mode |
+------------+
|            |
+------------+

mysql> SELECT "Let's see whether this works.";
+-------------------------------+
| Let's see whether this works. |
+-------------------------------+
| Let's see whether this works. |
+-------------------------------+

mysql> SET sql_mode = 'ANSI_QUOTES';

mysql> SELECT @@sql_mode;
+-------------+
| @@sql_mode  |
+-------------+
| ANSI_QUOTES |
+-------------+

mysql> SELECT "Let's see whether this works.";
ERROR 1054 (42S22): Unknown column 'Let's see whether this works.' in
'field list'
```

It's a good idea to quote strings using single quotes to remain independent of the SQL mode.

Answer 4:

When the first `SELECT` statement was issued, the `collation_connection` was set to its default value (`latin1_swedish_ci`). For the second `SELECT` statement, the `collation_connection` was set to `latin1_german2_ci`:

```
mysql> SELECT @@collation_connection;
+------------------------+
| @@collation_connection |
+------------------------+
| latin1_swedish_ci      |
+------------------------+


mysql> SET @str1 = 'Hühner-Hugos Hähnchen sind schön.';
mysql> SET @str2 = 'Huehner-Hugos Haehnchen sind schoen.';
mysql> SELECT @str1 = @str2;
+---------------+
| @str1 = @str2 |
+---------------+
|             0 |
+---------------+

mysql> SET collation_connection = latin1_german2_ci;

mysql> SELECT @@collation_connection;
+------------------------+
| @@collation_connection |
+------------------------+
| latin1_german2_ci      |
+------------------------+


mysql> SET @str1 = 'Hühner-Hugos Hähnchen sind schön.';
mysql> SET @str2 = 'Huehner-Hugos Haehnchen sind schoen.';
mysql> SELECT @str1 = @str2;
+---------------+
| @str1 = @str2 |
+---------------+
|             1 |
+---------------+
```

Answer 5:

If either string in a comparison is binary, both strings are treated as binary. Because they're binary, the concept of character sets and collations doesn't apply. Therefore, we get true (`1`):

```
mysql> SELECT 'Hello world!' = BINARY 'Hello world!';
+-----------------------------------------+
| 'Hello world!' = BINARY 'Hello world!' |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
```

Answer 6:

You have to convert the binary string to a non-binary string using, for example, the `latin1` character set (other character sets will work, too):

```
mysql> SELECT UPPER(CONVERT(@var USING latin1));
+-----------------------------------+
| UPPER(CONVERT(@var USING latin1)) |
+-----------------------------------+
| LENNART                           |
+-----------------------------------+
```

Answer 7:

Normally, there must not be a space after the function name. However, you can set the SQL mode to allow spaces after function names:

```
mysql> SET sql_mode = 'IGNORE_SPACE';

mysql> SELECT @@sql_mode;
+--------------+
| @@sql_mode   |
+--------------+
| IGNORE_SPACE |
+--------------+

mysql> SELECT NOW ();
+---------------------+
| NOW ()              |
+---------------------+
| 2005-06-02 10:38:46 |
+---------------------+
```

Answer 8:

Constants are either literal numbers, strings, or temporal values. An SQL expression can consist of constants, `NULL` values, references to table columns, and function calls.

Answer 9:

```
SELECT 1 + 1;
```

Answer 10:

```
SELECT 'Hello ', 'world!';
```

Answer 11:

```
SELECT '2002-02-08', '2003-02-08';
```

Answer 12:

```
SELECT '21:39:00', '12:00:00';
```

Answer 13:

```
SELECT name, age FROM mytable;
```

Answer 14:

```
SELECT NOW();
```

Answer 15:

```
SELECT DATE_FORMAT(NOW(),'%m/%d/%Y');
```

Answer 16:

```
mysql> SELECT
    ->  pid AS PID,
    ->  salary AS 'Gross Salary',
    ->  salary * 0.4 AS Deduction,
    ->  salary * 0.6 AS 'Net Salary'
    -> FROM personnel
    -> ;
+-----+--------------+-----------+------------+
| PID | Gross Salary | Deduction | Net Salary |
+-----+--------------+-----------+------------+
|   1 |      1500.00 |    600.00 |     900.00 |
|   2 |      1700.00 |    680.00 |    1020.00 |
|   3 |      1950.00 |    780.00 |    1170.00 |
|   4 |      2300.00 |    920.00 |    1380.00 |
|   5 |      1900.00 |    760.00 |    1140.00 |
|   6 |       850.00 |    340.00 |     510.00 |
|   7 |      1250.00 |    500.00 |     750.00 |
|   8 |      1450.00 |    580.00 |     870.00 |
|   9 |      1920.00 |    768.00 |    1152.00 |
|  10 |      2200.00 |    880.00 |    1320.00 |
|  11 |      2900.00 |   1160.00 |    1740.00 |
|  12 |      1000.00 |    400.00 |     600.00 |
|  13 |      2850.00 |   1140.00 |    1710.00 |
+-----+--------------+-----------+------------+
```

Answer 17:

```
mysql> SELECT
    ->  unit AS Unit,
    ->  SUM(salary) AS 'Gross Salary',
    ->  SUM(salary) * 0.4 AS Deduction,
    ->  SUM(salary) * 0.6 AS 'Net Salary'
    -> FROM personnel
    -> GROUP BY unit
    -> ;
+------+--------------+-----------+------------+
| Unit | Gross Salary | Deduction | Net Salary |
+------+--------------+-----------+------------+
|   23 |      9370.00 |   3748.00 |    5622.00 |
|   42 |     14400.00 |   5760.00 |    8640.00 |
+------+--------------+-----------+------------+
```

Answer 18:

```
mysql> SELECT
    ->  pid AS PID,
    ->  salary as 'Gross Salary',
    ->  IF(salary < 2000, salary * 0.3, salary * 0.4) AS Deduction,
    ->  IF(salary < 2000, salary * 0.7, salary * 0.6) AS 'Net Salary'
    -> FROM personnel
    -> ;
+-----+--------------+-----------+------------+
| PID | Gross Salary | Deduction | Net Salary |
+-----+--------------+-----------+------------+
|   1 |      1500.00 |    450.00 |    1050.00 |
|   2 |      1700.00 |    510.00 |    1190.00 |
|   3 |      1950.00 |    585.00 |    1365.00 |
|   4 |      2300.00 |    920.00 |    1380.00 |
|   5 |      1900.00 |    570.00 |    1330.00 |
|   6 |       850.00 |    255.00 |     595.00 |
|   7 |      1250.00 |    375.00 |     875.00 |
|   8 |      1450.00 |    435.00 |    1015.00 |
|   9 |      1920.00 |    576.00 |    1344.00 |
|  10 |      2200.00 |    880.00 |    1320.00 |
|  11 |      2900.00 |   1160.00 |    1740.00 |
|  12 |      1000.00 |    300.00 |     700.00 |
|  13 |      2850.00 |   1140.00 |    1710.00 |
+-----+--------------+-----------+------------+
```

Answer 19:

```
mysql> SELECT
    ->  SUM(salary) AS Salary,
    ->  SUM(salary) * 1.1 AS 'New Salary',
    ->  SUM(salary) * 0.1 AS 'Cost Rise'
    -> FROM personnel
    -> ;
+----------+------------+-----------+
| Salary   | New Salary | Cost Rise |
+----------+------------+-----------+
| 23770.00 |   26147.00 |   2377.00 |
+----------+------------+-----------+
```

Answer 20:

```
mysql> SELECT
    ->  SUM(salary) AS Salary,
    ->  SUM(salary) * IF(unit = 23, 1.1, 1.05) AS 'New Salary',
    ->  SUM(salary) * IF(unit = 23, 0.1, 0.05) AS 'Cost Rise'
    -> FROM personnel
    -> GROUP BY unit
    -> ;
```

```
+----------+------------+-----------+
| Salary   | New Salary | Cost Rise |
+----------+------------+-----------+
|  9370.00 |   10307.00 |    937.00 |
| 14400.00 |   15120.00 |    720.00 |
+----------+------------+-----------+
```

Answer 21:

Each statement yields output as follows:

```
mysql> SELECT DISTINCT name FROM leonardo;
+---------+
| name    |
+---------+
| Lennart |
+---------+
```

Different lettercases aren't regarded as distinct in a non-binary context.

```
mysql> SELECT name, COUNT(*) FROM leonardo GROUP BY name;
+---------+----------+
| name    | COUNT(*) |
+---------+----------+
| Lennart |        4 |
+---------+----------+
```

Different lettercases aren't regarded as distinct in a non-binary context.

```
mysql> SELECT name, COUNT(*) FROM leonardo;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT()...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

As the error message indicates, this isn't a legal SQL statement.

Answer 22:

Each statement yields output as follows:

```
mysql> SELECT DISTINCT name FROM leonardo;
+---------+
| name    |
+---------+
| Lennart |
| lennart |
| LENNART |
| lEnNaRt |
+---------+
```

Letters in different cases have different byte values, so they are regarded as distinct for binary strings.

```
mysql> SELECT name, COUNT(*) FROM leonardo GROUP BY name;
+---------+----------+
| name    | COUNT(*) |
+---------+----------+
| LENNART |        1 |
| Lennart |        1 |
| lEnNaRt |        1 |
| lennart |        1 |
+---------+----------+
```

Letters in different cases have different byte values, so they are regarded as distinct for binary strings.

```
mysql> SELECT name, COUNT(*) FROM leonardo;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT()...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

As the error message indicates, this isn't a legal SQL statement.

Answer 23:

`LENGTH()` counts string length in bytes, `CHAR_LENGTH()` counts string length in characters. For strings that contain only single-byte characters, the two functions return identical results, but for strings that contain multi-byte characters, `LENGTH()` returns a different (higher) number than `CHAR_LENGTH()`.

Answer 24:

The `MD5()` function produces unequal values for the two strings because it treats arguments as binary strings:

```
+--------------------------------+
| MD5('lennart') = MD5('LENNART') |
+--------------------------------+
|                              0 |
+--------------------------------+
```

However, this result can be guessed only if you're able to anticipate the results of the `MD5()` function calls in advance (which is rather improbable):

```
mysql> SELECT MD5('lennart'), MD5('LENNART');
+----------------------------------+----------------------------------+
| MD5('lennart')                   | MD5('LENNART')                   |
+----------------------------------+----------------------------------+
| a6894e0c24b247a33b0de7e3fcd2b53f | d63445ebdc53cabe60ef708beafb39f0 |
+----------------------------------+----------------------------------+
```

Answer 25:

a. `TRUE`. A non-binary string comparison is performed in a case-insensitive manner (assuming that the default collation is case-insensitive).

b. `FALSE`. The keyword `BINARY` for either of the terms causes the strings to be compared as binary strings.

c. `FALSE`. The keyword `BINARY` for one (or both) of the terms causes the strings to be compared as binary strings.

Answer 26:

Three underscore metacharacters match any three characters, so they match the entire string `'abc'`. That leaves nothing left to be matched, but because `%` also matches nothing, the expression evaluates to true.

```
+-------------------+
| 'abc' LIKE '%___%' |
+--------------------+
|                  1 |
+--------------------+
```

Answer 27:

`'abc'` doesn't match the five underscore metacharacters, so the expression evaluates to false.

```
+----------------------+
| 'abc' LIKE '%_____%' |
+----------------------+
|                    0 |
+----------------------+
```

Answer 28:

The `'%'` metacharacter also matches the empty string, so the expression evaluates to true.

```
+-------------+
| '' LIKE '%' |
+-------------+
|           1 |
+-------------+
```

Answer 29:

The space characters surrounding `'%'` do not match the empty string, so the expression evaluates to false.

```
+---------------+
| '' LIKE ' % ' |
+---------------+
|             0 |
+---------------+
```

Answer 30:

The `'%'` character is matched by `'%'`, but not by the space characters surrounding it, so the expression evaluates to false.

```
+----------------+
| '%' LIKE ' % ' |
+----------------+
|              0 |
+----------------+
```

Answer 31:

Any non-`NULL` string is matched by the `'%'` metacharacter, so the expression evaluates to true.

```
+----------------+
| ' % ' LIKE '%' |
+----------------+
|              1 |
+----------------+
```

Answer 32:

The expression evaluates to true. `'_e'` at the start of the pattern matches `'Le'`, `'_t'` at the end matches `'rt'`, and the rest of the string is matched by the `'%'` metacharacter.

```
+-------------------------+
| 'Lennart' LIKE '_e%_t' |
+-------------------------+
|                       1 |
+-------------------------+
```

Answer 33:

The expression evaluates to true. `_e` matches `Le` and the rest of the string is matched by the `'%'` metacharacter.

```
+-----------------------+
| 'Lennart' LIKE '_e%' |
+-----------------------+
|                     1 |
+-----------------------+
```

Answer 34:

If either operand is `NULL`, the expression evaluates to `NULL`.

```
+----------------+
| NULL LIKE NULL |
+----------------+
|           NULL |
+----------------+
```

Answer 35:

If either operand is `NULL`, the expression evaluates to `NULL`. `'%'` matches anything (including the empty string) except `NULL`.

```
+---------------+
| NULL LIKE '%' |
+---------------+
|          NULL |
+---------------+
```

Answer 36:

In this case, `'NULL'` is just a string like any other string, not the `NULL` value, so the expression evaluates to true.

```
+-----------------+
| 'NULL' LIKE '%' |
+-----------------+
|               1 |
+-----------------+
```

Answer 37:

In both operands, `'NULL'` is just a string (in different lettercases). The strings are compared as binary strings due to the `BINARY` keyword, so the expression evaluates to false.

```
+---------------------------+
| BINARY 'NULL' LIKE 'null' |
+---------------------------+
|                         0 |
+---------------------------+
```

Answer 38:

Although it appears that the pattern match is performed on a date value rather than a string, the expression evaluates to true. The reason for this is that the “date” is actually a string value in this context.

```
+---------------------------+
| '2002-02-08' LIKE '2002%' |
+---------------------------+
|                         1 |
+---------------------------+
```

Answer 39:

Pattern-matching operations can be performed with numbers, too. Because `23.42` starts and ends with `2`, it's matched by `'2%2'`, so the expression evaluates to true.

```
+------------------+
| 23.42 LIKE '2%2' |
+------------------+
|                1 |
+------------------+
```

Answer 40:

Pattern-matching operations can be performed with numbers, too. However, because `023.420` does not start and end with `2`, it isn't matched by `'2%2'`. The expression evaluates to false.

```
+--------------------+
| 023.420 LIKE '2%2' |
+--------------------+
|                  0 |
+--------------------+
```

Answer 41:

```
mysql> SELECT Name AS City, Country, Population
    ->   FROM CityList
    ->  WHERE Country IN('DEU','DNK','USA')
    ->    AND POPULATION BETWEEN 400000 AND 500000
    ->  ORDER BY Population DESC
    -> ;
```

Answer 42:

The statement returns an error. It's a common error of programmers to mix up the == comparison operator used in many programming languages with the = comparison operator used in SQL.

```
mysql> SELECT NULL == NULL;
ERROR 1064 (42000): You have an error in your SQL syntax.
```

Answer 43:

The statement evaluates to `NULL`. If any operand in an equality comparison is `NULL`, the expression evaluates to `NULL`.

```
mysql> SELECT NULL = NULL = NULL;
+--------------------+
| NULL = NULL = NULL |
+--------------------+
|               NULL |
+--------------------+
```

Answer 44:

The statement evaluates to `NULL`. If any operand in an equality comparison is `NULL`, the expression evaluates to `NULL`.

```
mysql> SELECT NULL = NULL = 0;
+-----------------+
| NULL = NULL = 0 |
+-----------------+
|            NULL |
+-----------------+
```

Answer 45:

The statement returns an error. The `IS NULL` and `IS NOT NULL` operators must be followed by `NULL`.

```
mysql> SELECT NULL IS NOT 0;
ERROR 1064 (42000): You have an error in your SQL syntax.
```

Answer 46:

The statement evaluates to true (`1`) because `0` is not `NULL`.

```
mysql> SELECT 0 IS NOT NULL;
+---------------+
| 0 IS NOT NULL |
+---------------+
|             1 |
+---------------+
```

Answer 47:

The statement evaluates to false. The special `<=>` operator can be used for regular comparisons (exactly like the = operator). The only difference is that it's `NULL`-safe.

```
mysql> SELECT 0 <=> 1;
+---------+
| 0 <=> 1 |
+---------+
|       0 |
+---------+
```

Answer 48:

This is a legal comment in MySQL. The `/*!` part of the comment ensures that MySQL will not treat this as a comment, but other database servers will.

Answer 49:

This is a legal comment in MySQL. Other database servers will regard this as a comment; MySQL, however, will interpret the contents of the comment as part of the statement if the server version is 4.0.0 or higher.

Answer 50:

This is a legal comment in MySQL, but it is not a version-specific comment because it begins with `/*`, not `/*!`. For that purpose, it should be written as `/*!32303 TEMPORARY */`.

Answer 51:

This isn't a legal comment in MySQL.

Answer 52:

This SQL Standard-style comment isn't legal in MySQL because `--` must be followed by a space character.

Answer 53:

For encryption and decryption, you could use the following functions:

- `ENCODE()` and `DECODE()`; these have no special requirements.
- `DES_ENCRYPT()` and `DES_DECRYPT()`; these require SSL support to be enabled.
- `AES_ENCRYPT()` and `AES_DECRYPT()`; these have no special requirements.
- `PASSWORD()` can encrypt data, but has no corresponding decryption function. It should only be used for MySQL user account management.

Answer 54:

A fraction of `.5` or greater rounds away from zero:

```
mysql> SELECT ROUND(1.5), ROUND(-1.5);
+------------+-------------+
| ROUND(1.5) | ROUND(-1.5) |
+------------+-------------+
| 2          | -2          |
+------------+-------------+
```

Chapter 11. Updating Data

Almost all examples and exercises in this study guide use the *world database* as the sample data set. The accompanying CD-ROM contains the data for this database and instructions that describe how to create and populate the database for use with your own MySQL installation.

Question 1:

Consider the `City` table of the `world` database. Here is its `CREATE` statement:

```
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`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

Will the following statement succeed? If it does, what will it insert?

```
mysql> INSERT INTO City VALUES ();
```

Question 2:

Consider the `City` table of the `world` database. Here is its `CREATE` statement:

```
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`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

With MySQL running in *strict mode*, will the following statement succeed? If it does, what will it insert?

```
mysql> INSERT INTO City SET Name = NULL;
```

Question 3:

MySQL provides a number of ways to handle new rows that would cause duplicate-key errors for unique index values in a table. Specifically, (a) with the standard form of `INSERT`, the new rows can be rejected; (b) with `INSERT ... IGNORE`, you can force MySQL to discard the new rows that duplicate existing unique-key values; (c) with the `REPLACE` statement, you can force MySQL to delete the existing rows before inserting the new rows; and (d) you can use `ON DUPLICATE KEY UPDATE` to update specific columns of the existing row. For each of these options, give an example of the effect of its use on the following table:

```
mysql> DESCRIBE twounique;
+-------+---------------------+------+-----+---------+-------+
| Field | Type                | Null | Key | Default | Extra |
+-------+---------------------+------+-----+---------+-------+
| id1   | tinyint(3) unsigned | NO   | PRI | 0       |       |
| id2   | tinyint(3) unsigned | NO   | UNI | 0       |       |
+-------+---------------------+------+-----+---------+-------+
mysql> SELECT * FROM twounique;
+-----+-----+
| id1 | id2 |
+-----+-----+
|   1 |   2 |
+-----+-----+
```

Question 4:

Consider this table:

```
mysql> DESCRIBE twounique;
+-------+---------------------+------+-----+---------+-------+
| Field | Type                | Null | Key | Default | Extra |
+-------+---------------------+------+-----+---------+-------+
| id1   | tinyint(3) unsigned | NO   | PRI | 0       |       |
| id2   | tinyint(3) unsigned | NO   | UNI | 0       |       |
+-------+---------------------+------+-----+---------+-------+
mysql> SELECT * FROM twounique;
+-----+-----+
| id1 | id2 |
+-----+-----+
|   1 |   2 |
|   3 |   4 |
|   5 |   6 |
+-----+-----+
```

What are the contents of table `twounique` after executing each of the following SQL statements?

```
mysql> REPLACE INTO twounique VALUES (2,2);
```

```
mysql> REPLACE INTO twounique VALUES (2,6);
```

Question 5:

How do you add multiple records to a table with a single `INSERT` statement?

Question 6:

`INSERT` supports an `IGNORE` modifier, but `REPLACE` does not. Why is that?

Question 7:

The table `access_log` contains information on the number of times employees of a secured office open a door protected by personal ID number (PIN) codes. The structure of the table is:

```
mysql> DESCRIBE access_log;
+---------+------------------+------+-----+---------+-------+
| Field   | Type             | Null | Key | Default | Extra |
+---------+------------------+------+-----+---------+-------+
| PIN     | char(6)          | NO   | PRI |         |       |
| entries | int(10) unsigned | NO   |     | 0       |       |
+---------+------------------+------+-----+---------+-------+
```

The system was been put into use recently, and the table contains the following entries:

```
+--------+---------+
| PIN    | entries |
+--------+---------+
| 156734 |       6 |
| 578924 |       2 |
| 479645 |      10 |
| 356845 |       5 |
+--------+---------+
```

Now, two employees enter through the secured door using their PIN codes.

- The first employee uses the PIN code 578924.
- The second employee, who has not used the system before, uses the PIN code 687456 (which is a valid PIN for the door).

How can you log both entries in the `access_log` table using a statement that is the same for each entry (except for the PIN codes)?

Question 8:

To completely empty a table, what statement or statements can you use?

Question 9:

To partially empty a table, what statement or statements can you use?

Question 10:

What are the reasons why an `UPDATE` statement might have no effect (that is, not change any values)?

Question 11:

Why is the number of affected rows in the following `UPDATE` statement `0`, although the number of rows matched by the `WHERE` clause is `5`? Why is the number of matched rows `5` and not rather `10`?

```
mysql> SELECT pid, grade FROM personnel;
+-----+-------+
| pid | grade |
+-----+-------+
|   1 |     1 |
|   2 |     2 |
|   3 |  NULL |
|   4 |  NULL |
|   5 |  NULL |
|   6 |     1 |
|   7 |     1 |
|   8 |     1 |
|   9 |  NULL |
|  10 |  NULL |
|  11 |  NULL |
|  12 |     1 |
|  13 |  NULL |
+-----+-------+
13 rows in set (0.00 sec)

mysql> UPDATE personnel SET grade = 1 WHERE grade != 2;
Query OK, 0 rows affected (0.00 sec)
```

```
Rows matched: 5  Changed: 0  Warnings: 0
```

Question 12:

Is the following statement true or false?

To prevent accidental `UPDATE` statements that would change all rows in a table, you can start the `mysql` program with the `--safe-updates` option.

Question 13:

Is the following statement true or false?

To prevent accidental `UPDATE` statements that would change all rows in a table, you can start the `mysql` program with the `--safe-updates-and-deletes` option.

Question 14:

Is the following statement true or false?

To prevent accidental `UPDATE` statements that would change all rows in a table, you can start any client program with the `--safe-updates` option.

Question 15:

Consider the following listing of a table named `personnel` that has a primary key on the `pid` column:

```
mysql> SELECT * FROM personnel;
+-----+------+-------+
| pid | unit | grade |
+-----+------+-------+
|   1 |   42 |     1 |
|   2 |   42 |     2 |
|   3 |   42 |  NULL |
|   4 |   42 |  NULL |
|   5 |   42 |  NULL |
|   6 |   23 |     1 |
|   7 |   23 |     1 |
|   8 |   23 |     1 |
|   9 |   23 |  NULL |
|  10 |   42 |  NULL |
|  11 |   23 |  NULL |
|  12 |   23 |     1 |
|  13 |   42 |  NULL |
+-----+------+-------+
```

What single `UPDATE` statement would you use to set all rows with no grade to `3`?

Question 16:

Refer to the structure and contents shown for the `personnel` table in the previous question. What `REPLACE` statement would you use to set the `grade` column to `4` and the `unit` column to `45`, for all rows where the `pid` column has the value `10`?

Question 17:

The table `petnames` contains the following data:

```
mysql> SELECT * FROM petnames;
+--------+
| name   |
+--------+
| Lucy   |
| Macie  |
| Myra   |
| Cheep  |
| Lucy   |
| Myra   |
| Cheep  |
| Macie  |
| Pablo  |
| Stefan |
+--------+
```

Assume that you issue the following statement:

```
mysql> UPDATE petnames SET name = CONCAT(name, '_!!!') ORDER BY name LIMIT 1;
```

What will the table's contents be after the `UPDATE`?

Question 18:

Will the following statement delete all rows from the table `mytable`?

```
TRUNCATE TABLE mytable;
```

Question 19:

Will the following statement delete all rows from the table `mytable`? Will it reset the `AUTO_INCREMENT` counter if there is one in the table?

```
DELETE FROM mytable;
```

Question 20:

The `personnel` table has the following contents. Unfortunately, the unit numbers were interchanged for some reason. Unit 23 is supposed to be 42, and 42 is supposed to be 23. What statement would you use to resolve this problem?

```
mysql> SELECT * FROM personnel;
+-----+------+-------+
| pid | unit | grade |
+-----+------+-------+
|   1 |   42 |     1 |
|   2 |   42 |     2 |
|   3 |   42 |  NULL |
|   4 |   42 |  NULL |
|   5 |   42 |  NULL |
|   6 |   23 |     1 |
|   7 |   23 |     1 |
|   8 |   23 |     1 |
|   9 |   23 |  NULL |
|  10 |   42 |  NULL |
|  11 |   23 |  NULL |
|  12 |   23 |     1 |
|  13 |   42 |  NULL |
+-----+------+-------+
```

Answers to Exercises

Answer 1:

The statement will succeed. All columns are set to their default values (empty string or zero, respectively), with the exception of `ID` which has an `AUTO_INCREMENT` counter that is incremented:

```
mysql> INSERT INTO City VALUES ();
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM City WHERE ID = LAST_INSERT_ID();
+------+------+-------------+---------+------------+
| ID   | Name | CountryCode | District | Population |
+------+------+-------------+---------+------------+
| 4080 |      |             |         |          0 |
+------+------+-------------+---------+------------+
```

Answer 2:

The statement will result in an error:

```
mysql> SET SQL_MODE = 'STRICT_ALL_TABLES';

mysql> INSERT INTO City SET Name = NULL;
ERROR 1048 (23000): Column 'Name' cannot be null
```

Answer 3:

There are several ways to handle records that would cause duplicate-key errors for unique indexes:

a. Use the `IGNORE` option in `INSERT`. `IGNORE` silently ignores the attempt to insert a duplicate unique key value:

```
mysql> INSERT IGNORE INTO twounique VALUES (1,42)
    -> /* Note the number of affected rows */;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM twounique;
+-----+-----+
| id1 | id2 |
+-----+-----+
|   1 |   2 |
+-----+-----+
```

b. Use `REPLACE` instead of `INSERT`. `REPLACE` replaces any row that would otherwise duplicate a unique key value. The existing row will first be deleted, and then the new row will be inserted (with the result that the otherwise duplicated key value remains the same):

```
mysql> REPLACE INTO twounique VALUES (1,42)
    -> /* Note the number of affected rows */;
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT * FROM twounique;
```

```
+-----+-----+
| id1 | id2 |
+-----+-----+
|   1 |  42 |
+-----+-----+
```

c. Use an `INSERT` without the `IGNORE` option. An error occurs and the new record is not inserted:

```
mysql> INSERT INTO twounique VALUES (1,42);
ERROR 1062 (23000): Duplicate entry '1' for key 1
```

d. Use an `ON DUPLICATE KEY UPDATE` clause. The existing record is modified according to the specified column assignments:

```
mysql> INSERT INTO twounique VALUES (1,42)
    -> ON DUPLICATE KEY UPDATE id1 = id1+1, id2 = id2+1;
Query OK, 2 rows affected (0.01 sec)

mysql> SELECT * FROM twounique;
+-----+-----+
| id1 | id2 |
+-----+-----+
|   2 |   3 |
+-----+-----+
```

Answer 4:

The `REPLACE` statements change the table data as follows:

a. This statement replaces the first record rather than duplicating the existing unique value of `2` in the `id2` column:

```
mysql> REPLACE INTO twounique VALUES (2,2);
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT * FROM twounique;
+-----+-----+
| id1 | id2 |
+-----+-----+
|   2 |   2 |
|   3 |   4 |
|   5 |   6 |
+-----+-----+
```

Because the original record is deleted before the new row is inserted, the server reports `2 rows affected`.

b. This statement replaces the first record (containing the value set `2,2` prior to the change), rather than duplicating the value in the `id1` column:

```
mysql> REPLACE INTO twounique VALUES (2,6);
Query OK, 3 rows affected (0.00 sec)

mysql> SELECT * FROM twounique;
```

```
+-----+-----+
| id1 | id2 |
+-----+-----+
|   3 |   4 |
|   2 |   6 |
+-----+-----+
```

The result is that the first record then contains the value set `2,6`. The statement also replaces the third record (containing the value set `5,6` prior to the change), rather than duplicating the value in the `id2` column. The result is that the third record *also* contains the value set `2,6`. Because this result is identical to the value set of the first record, the first record is then deleted. The server optimizes the `REPLACE` statement to avoid doing more work than necessary. Thus, the first record is deleted but not re-inserted, and then the third record is deleted and then re-inserted. For that reason, the server reports `3 rows affected`.

Answer 5:

You can add multiple records with a single `INSERT` statement using multiple `VALUES` lists (this is an ANSI SQL-99 feature). The syntax is as follows:

```
INSERT INTO tbl [(col1, col2, ...)]
VALUES (value1, value2, ...), (value1, value2, ...), ...
```

Answer 6:

The point of `REPLACE` is to replace old records, not ignore new ones. If you specify the keyword `IGNORE` in an `INSERT` with multiple rows, any rows that duplicate an existing `PRIMARY KEY` or `UNIQUE` index in the table are ignored, which means they aren't inserted. If you don't specify `IGNORE`, the insert is aborted if there is any row that duplicates an existing unique key value. With `REPLACE`, you explicitly want to replace records that would violate a `PRIMARY KEY` or `UNIQUE` index constraint, so ignoring duplicates would make no sense.

Answer 7:

For the first employee, an `UPDATE` statement is needed to increase the count of the times PIN code 578924 has been used to open the door. For the second employee, a new record must be entered into the table, as PIN code 687456 is being used for the first time.

You can provide for both occurrences by utilizing the `ON DUPLICATE KEY UPDATE` clause of the `INSERT` statement:

```
INSERT INTO access_log (PIN, entries) VALUES ('578924', 1)
ON DUPLICATE KEY UPDATE entries = entries+1;

INSERT INTO access_log (PIN, entries) VALUES ('687456', 1)
ON DUPLICATE KEY UPDATE entries = entries+1;
```

After these two statements are executed, the `access_log` table has the following contents:

```
+--------+---------+
| PIN    | entries |
+--------+---------+
| 156734 |       6 |
| 578924 |       3 |
| 479645 |      10 |
| 356845 |       5 |
| 687456 |       1 |
+--------+---------+
```

Answer 8:

`TRUNCATE TABLE`, or `DELETE` with no `WHERE` clause, or `DELETE` with a `WHERE` clause that's always true. Assume that you have the following table:

```
mysql> SELECT * FROM tbl1;
+----+----------+
| id | sometext |
+----+----------+
|  1 | boo      |
|  2 | bar      |
|  3 | booboo   |
|  4 | barbar   |
+----+----------+
```

You could empty this table as follows:

- Use `TRUNCATE TABLE`:

  ```
  TRUNCATE TABLE tbl1;
  ```

- Use `DELETE` with no `WHERE` clause:

  ```
  DELETE FROM tbl1;
  ```

  This will not work when `mysql` has been started with the `--safe-updates` option, though.

- Use `DELETE` with a `WHERE` clause that's always true. The following statements use expressions that are true for every record in the table `tbl1`:

  ```
  DELETE FROM tbl1 WHERE id < 5;
  DELETE FROM tbl1 WHERE 1 = 1;
  ```

Answer 9:

Use a `DELETE` statement that includes a `WHERE` clause that selects only the records to be deleted. Assume that you have the following table:

```
mysql> SELECT * FROM tbl1;
+----+----------+
| id | sometext |
+----+----------+
|  1 | boo      |
|  2 | bar      |
|  3 | booboo   |
|  4 | barbar   |
+----+----------+
```

To delete records with `id` values of `2` and `3`, statements that accomplish that goal include the following:

```
mysql> DELETE FROM tbl1 WHERE id > 1 AND id < 4;
mysql> DELETE FROM tbl1 WHERE id BETWEEN 2 AND 3;
```

Answer 10:

An `UPDATE` statement changes no values if any of the following is true:

a. The table has no rows.

b. No rows match the conditions specified in the `WHERE` clause of the statement.

c. The updated columns are set to their current values.

Answer 11:

The number of affected rows is `0` because the `WHERE` clause matches only the rows where the `grade` column values are set to `1` already. Thus, the values are not changed. The number of matched rows is `5` rather than `10` because for `grade` values of `NULL`, the condition `grade != 2` is not true (the `!=` operator is never true for `NULL` values).

Answer 12:

True. To prevent accidental `UPDATE` (and `DELETE`) statements that don't contain a `WHERE` clause that uses a key, you can start the `mysql` command-line tool with the `--safe-updates` option:

```
C:\mysql\bin>mysql --safe-updates menagerie
Welcome to the MySQL monitor. Commands end with ; or \g.

mysql> DELETE FROM personnel;
ERROR 1175 (HY000): You are using safe update mode and you tried
to update a table without a WHERE that uses a KEY column
```

Answer 13:

False. There is no such option as `--safe-updates-and-deletes` to the `mysql` command-line tool. (In fact, there's no such option to any MySQL program.)

Answer 14:

False. The `--safe-updates` option is supported only by the `mysql` client program.

Answer 15:

The rows with no grade are those containing `NULL` in the `grade` column. The following `UPDATE` statement sets the `grade` column to `3` in all rows where `grade` is `NULL`:

```
mysql> UPDATE personnel SET grade = 3 WHERE grade IS NULL;
Query OK, 7 rows affected (0.02 sec)
Rows matched: 7  Changed: 7  Warnings: 0
```

Answer 16:

To make the required changes, you would issue this `REPLACE` statement:

```
mysql> REPLACE INTO personnel VALUES (10,45,4);
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT * FROM personnel WHERE pid=10;
+-----+------+-------+
| pid | unit | grade |
+-----+------+-------+
|  10 |   45 |     4 |
+-----+------+-------+
```

Answer 17:

The `UPDATE` changes the row that is first when they're sorted by `name`. The first name is `Cheep`. It occurs twice, but `LIMIT` constrains the change to include just one of the rows. The resulting table contents are as follows:

```
mysql> SELECT * FROM petnames;
+-----------+
| name      |
+-----------+
| Lucy      |
| Macie     |
| Myra      |
| Cheep_!!! |
| Lucy      |
| Myra      |
| Cheep     |
| Macie     |
| Pablo     |
| Stefan    |
+-----------+
```

Answer 18:

Yes. `TRUNCATE TABLE mytable` will delete all records.

Answer 19:

Yes. `DELETE FROM mytable` will delete all records. It will also return the exact number of deleted records. It may reset an existing `AUTO_INCREMENT` counter, too. If you want to avoid the latter, you should include a `WHERE` clause to `DELETE`, like this:

```
DELETE FROM mytable WHERE 1;
```

Answer 20:

To swap unit numbers 23 and 42, you would issue this statement:

```
mysql> UPDATE personnel
    ->  SET unit = IF(unit=23, 42, 23)
    -> ;
Query OK, 13 rows affected (0.02 sec)
Rows matched: 13  Changed: 13  Warnings: 0

mysql> SELECT * FROM personnel;
+-----+------+-------+
| pid | unit | grade |
+-----+------+-------+
|   1 |   23 |     1 |
|   2 |   23 |     2 |
|   3 |   23 |  NULL |
|   4 |   23 |  NULL |
|   5 |   23 |  NULL |
|   6 |   42 |     1 |
|   7 |   42 |     1 |
|   8 |   42 |     1 |
|   9 |   42 |  NULL |
|  10 |   23 |  NULL |
|  11 |   42 |  NULL |
|  12 |   42 |     1 |
|  13 |   23 |  NULL |
+-----+------+-------+
```