Part 2. MySQL Developer II Exam

Table of Contents

Chapter 12. Joins

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 kind of join can find matches (values that are present in both tables involved in the join)?

Question 2:

What kind of join or joins find mismatches (values that are present in only one of the tables involved in the join)?

Question 3:

Write an inner join using the comma operator that retrieves the names of countries and the names of languages that are the official languages in that country.

Question 4:

Write an inner join using `INNER JOIN` that retrieves the names of countries and the names of languages that are the official languages in that country. Can you use either the `ON` clause or the `USING` clause?

Question 5:

Write an inner join that displays country names, city names, and city population of all countries that are not independent (where the `IndepYear` column is `NULL`).

Question 6:

Write an inner join that shows the city population *as a total per country*, for countries that are not independent (where the `IndepYear` column is `NULL`).

Question 7:

What do you call a join that combines all rows in one table with all rows in another table? What's the syntax for joining the `City` and the `Country` table that way? Why would you typically want to avoid such joins, and how can you accomplish that?

Question 8:

When querying two tables with an outer join, is it *always* possible to rewrite a `LEFT JOIN` as a `RIGHT JOIN`, or are there exceptions? If there aren't any exceptions, what is it good for to have two kinds of outer joins?

Question 9:

In which cases is it necessary to qualify column names with the appropriate table name, and why would you want to qualify column names with table names even if that's not necessary?

Question 10:

In which cases do you have to qualify table names with database names?

Question 11:

In which case is it impossible to resolve name conflicts by qualifying column names with the appropriate table names, or table names with the appropriate database names? How would such a conflict be resolved? Give an example.

Question 12:

Which of the following statements are true?

1. Joins are restricted to `SELECT` statements.
2. Joins work for `INSERT`, `UPDATE`, and `DELETE` statements.
3. Joins work for `SELECT`, `UPDATE`, and `DELETE` statements.
4. Joins work for `UPDATE` statements, but it's not possible to change data in more than one table within a single join.
5. Joins work for `DELETE` statements, but it's not possible to delete data in more than one table within a single join.

Question 13:

Consider the following record from the `Country` table:

```
mysql> SELECT
    ->  Name, Region, Continent, SurfaceArea, Population
    ->  FROM Country
    ->  WHERE Name = 'Paraguay'
    -> ;
+----------+---------------+---------------+-------------+------------+
| Name     | Region        | Continent     | SurfaceArea | Population |
+----------+---------------+---------------+-------------+------------+
| Paraguay | South America | South America |   406752.00 |    5496000 |
+----------+---------------+---------------+-------------+------------+
```

What statement would you issue to retrieve a list of countries whose surface area is larger than that of Paraguay, when you consider countries on the same continent (South America) only? The column headings of the result should look like this:

```
+----------+-----------------+---------------+--------------+
| Country  | Other Countries | Continent     | Surface Area |
+----------+-----------------+---------------+--------------+
```

Question 14:

Consider the following record from the `Country` table:

```
mysql> SELECT
    ->  Name, Region, Continent, SurfaceArea, Population
    ->  FROM Country
    ->  WHERE Name = 'Germany'
    -> ;
+---------+----------------+-----------+-------------+------------+
| Name    | Region         | Continent | SurfaceArea | Population |
+---------+----------------+-----------+-------------+------------+
| Germany | Western Europe | Europe    |   357022.00 |   82164700 |
+---------+----------------+-----------+-------------+------------+
```

What statement would you issue to retrieve a list of the countries worldwide with a population at least as large as that of Germany? Germany should be included in the list, which should be sorted by descending population. The column headings of the result should look like this:

```
+---------+--------------------+------------+
| Country | Other Countries    | Population |
+---------+--------------------+------------+
```

Question 15:

Consider the following record from the `Country` table:

```
mysql> SELECT
    ->  Name, Region, Continent, SurfaceArea, Population
    ->  FROM Country
    ->  WHERE Name = 'Nepal'
    -> ;
+-------+---------------------------+-----------+-------------+------------+
| Name  | Region                    | Continent | SurfaceArea | Population |
+-------+---------------------------+-----------+-------------+------------+
| Nepal | Southern and Central Asia | Asia      |   147181.00 |   23930000 |
+-------+---------------------------+-----------+-------------+------------+
```

What statement would you issue to retrieve a list of countries in the same region with a population and surface area at least as large as that of Nepal? Nepal should be included in the list, which should be sorted by descending population. The column headings of the result should look like this, with all region names cut to a maximum length of 10 characters:

```
+---------+-----------------+----------------+------------+------------+
| Country | Other Countries | Region         | Population | Surface    |
+---------+-----------------+----------------+------------+------------+
```

Question 16:

Here's the structure and contents for two tables, `client` and `project`, which will be used for the next two questions.

```
mysql> DESCRIBE client;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| cid   | smallint(5) unsigned | NO   | PRI | 0       |       |
| name  | char(20)             | NO   |     |         |       |
+-------+----------------------+------+-----+---------+-------+
mysql> DESCRIBE project;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| pid   | int(10) unsigned     | NO   | PRI | 0       |       |
| cid   | smallint(5) unsigned | NO   |     | 0       |       |
| name  | char(30)             | NO   |     |         |       |
| start | date                 | YES  |     | NULL    |       |
| end   | date                 | YES  |     | NULL    |       |
+-------+----------------------+------+-----+---------+-------+
mysql> SELECT * FROM client;
+-----+---------------+
| cid | name          |
+-----+---------------+
| 101 | Seamen's       |
| 103 | Lennart AG     |
| 110 | MySQL AB       |
| 115 | Icoaten & Co.  |
| 125 | Nittboad Inc   |
+-----+--------------+
mysql> SELECT * FROM project;
+-------+-----+-------------+------------+------------+
| pid   | cid | name        | start      | end        |
+-------+-----+-------------+------------+------------+
| 10000 | 103 | New CMS     | 2003-01-00 | 2003-05-00 |
| 10010 | 110 | Texi2XML    | 2002-04-00 | 2003-09-00 |
| 10020 | 110 | Studyguides | 2002-09-00 | 2003-03-30 |
| 10030 | 115 | PDC Server  | 2003-01-00 | 2003-01-00 |
| 10040 | 103 | Intranet    | 2009-02-00 | NULL       |
| 10050 | 101 | Intranet    | NULL       | NULL       |
| 10060 | 115 | SMB Server  | 2003-05-00 | NULL       |
| 10070 | 115 | WLAN        | NULL       | 2003-07-00 |
+-------+-----+-------------+------------+------------+
```

How many rows will the following join statements return?

```
mysql> SELECT client.name, project.name, project.start, project.end
    ->  FROM client, project
    -> ;

mysql> SELECT client.name, project.name, project.start, project.end
    ->  FROM client, project
    ->  WHERE project.cid = client.cid
    -> ;
```

Question 17:

Refer to the `client` and `project` tables shown in the previous question. How many rows will the following join statements return?

```
mysql> SELECT client.name, project.name, project.start, project.end
    ->  FROM client, project
    ->  WHERE project.cid = client.cid
    ->  AND project.start IS NOT NULL
    -> ;

mysql> SELECT client.name, project.name, project.start, project.end
    ->  FROM client, project
    ->  WHERE project.cid = client.cid
    ->  AND project.start IS NOT NULL
    ->  AND project.end IS NOT NULL
    -> ;
```

Question 18:

Here's the structure and sample data for two tables, `client` and `project`, which will be used for the next three questions.

```
mysql> DESCRIBE client;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| cid   | smallint(5) unsigned | NO   | PRI | 0       |       |
| name  | char(20)             | NO   |     |         |       |
+-------+----------------------+------+-----+---------+-------+
mysql> DESCRIBE project;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| pid   | int(10) unsigned     | NO   | PRI | 0       |       |
| cid   | smallint(5) unsigned | NO   |     | 0       |       |
| name  | char(30)             | NO   |     |         |       |
| start | date                 | YES  |     | NULL    |       |
| end   | date                 | YES  |     | NULL    |       |
+-------+----------------------+------+-----+---------+-------+
mysql> SELECT * FROM client;
+-----+---------------+
| cid | name          |
+-----+---------------+
| 101 | Seamen's      |
| 103 | Lennart AG    |
| 110 | MySQL AB      |
| 115 | Icoaten & Co. |
| 125 | Nittboad Inc  |
+-----+---------------+
mysql> SELECT * FROM project;
+-------+-----+-------------+------------+------------+
| pid   | cid | name        | start      | end        |
+-------+-----+-------------+------------+------------+
| 10000 | 103 | New CMS     | 2003-01-00 | 2003-05-00 |
| 10010 | 110 | Texi2XML    | 2002-04-00 | 2003-09-00 |
| 10020 | 110 | Studyguides | 2002-09-00 | 2003-03-30 |
| 10030 | 115 | PDC Server  | 2003-01-00 | 2003-01-00 |
| 10040 | 103 | Intranet    | 2009-02-00 | NULL       |
| 10050 | 101 | Intranet    | NULL       | NULL       |
| 10060 | 115 | SMB Server  | 2003-05-00 | NULL       |
| 10070 | 115 | WLAN        | NULL       | 2003-07-00 |
+-------+-----+-------------+------------+------------+
```

Using the `client` and `project` tables, you want to retrieve a list of clients that have no projects for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue?

Question 19:

Refer to the structure and sample data for the `client` and `project` tables, shown in the previous question. Using these two tables, you want to retrieve a list of clients that have projects starting in the year 2003, sorted by start date, for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue?

Question 20:

Refer to the structure and sample data for the `client` and `project` tables, shown two questions earlier. Using these two tables, you want to retrieve a list of clients that have intranet projects, using the `LEFT JOIN` syntax, for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue?

Question 21:

Here's the structure and sample data for two tables, `client` and `project`, which will be used for the next three questions.

```
mysql> DESCRIBE client;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| cid   | smallint(5) unsigned | NO   | PRI | 0       |       |
| name  | char(20)             | NO   |     |         |       |
+-------+----------------------+------+-----+---------+-------+
mysql> DESCRIBE project;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| pid   | int(10) unsigned     | NO   | PRI | 0       |       |
| cid   | smallint(5) unsigned | NO   |     | 0       |       |
| name  | char(30)             | NO   |     |         |       |
| start | date                 | YES  |     | NULL    |       |
| end   | date                 | YES  |     | NULL    |       |
+-------+----------------------+------+-----+---------+-------+
mysql> SELECT * FROM client;
+-----+---------------+
| cid | name          |
+-----+---------------+
| 101 | Seamen's      |
| 103 | Lennart AG    |
| 110 | MySQL AB      |
| 115 | Icoaten & Co. |
| 125 | Nittboad Inc  |
+-----+---------------+
mysql> SELECT * FROM project;
+-------+-----+-------------+------------+------------+
| pid   | cid | name        | start      | end        |
+-------+-----+-------------+------------+------------+
| 10000 | 103 | New CMS     | 2003-01-00 | 2003-05-00 |
| 10010 | 110 | Texi2XML    | 2002-04-00 | 2003-09-00 |
| 10020 | 110 | Studyguides | 2002-09-00 | 2003-03-30 |
| 10030 | 115 | PDC Server  | 2003-01-00 | 2003-01-00 |
| 10040 | 103 | Intranet    | 2009-02-00 | NULL       |
| 10050 | 101 | Intranet    | NULL       | NULL       |
| 10060 | 115 | SMB Server  | 2003-05-00 | NULL       |
| 10070 | 115 | WLAN        | NULL       | 2003-07-00 |
+-------+-----+-------------+------------+------------+
```

Using the `client` and `project` tables, you want to retrieve a list of clients that have intranet projects, using the `INNER JOIN` syntax, for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue?

Question 22:

Refer to the structure and sample data for the `client` and `project` tables, shown in the previous question. Using these two tables, you want to retrieve a list of clients that have intranet projects, using an inner join with the `WHERE` clause, for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue?

Question 23:

Refer to the structure and sample data for the `client` and `project` tables, shown two questions earlier. Using these two tables, you want to retrieve a list of clients and their projects, sorted by client name, and within client name, sorted by start date, for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue? Clients without projects should also be displayed.

Question 24:

Here's the structure and sample data for two tables, `client` and `project`, which will be used for the next three questions.

```
mysql> DESCRIBE client;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| cid   | smallint(5) unsigned | NO   | PRI | 0       |       |
| name  | char(20)             | NO   |     |         |       |
+-------+----------------------+------+-----+---------+-------+
mysql> DESCRIBE project;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| pid   | int(10) unsigned     | NO   | PRI | 0       |       |
| cid   | smallint(5) unsigned | NO   |     | 0       |       |
| name  | char(30)             | NO   |     |         |       |
| start | date                 | YES  |     | NULL    |       |
| end   | date                 | YES  |     | NULL    |       |
+-------+----------------------+------+-----+---------+-------+
mysql> SELECT * FROM client;
+-----+---------------+
| cid | name          |
+-----+---------------+
| 101 | Seamen's      |
| 103 | Lennart AG    |
| 110 | MySQL AB      |
| 115 | Icoaten & Co. |
| 125 | Nittboad Inc  |
+-----+---------------+
mysql> SELECT * FROM project;
```

```
+-------+-----+-------------+------------+------------+
| pid   | cid | name        | start      | end        |
+-------+-----+-------------+------------+------------+
| 10000 | 103 | New CMS     | 2003-01-00 | 2003-05-00 |
| 10010 | 110 | Texi2XML    | 2002-04-00 | 2003-09-00 |
| 10020 | 110 | Studyguides | 2002-09-00 | 2003-03-30 |
| 10030 | 115 | PDC Server  | 2003-01-00 | 2003-01-00 |
| 10040 | 103 | Intranet    | 2009-02-00 | NULL       |
| 10050 | 101 | Intranet    | NULL       | NULL       |
| 10060 | 115 | SMB Server  | 2003-05-00 | NULL       |
| 10070 | 115 | WLAN        | NULL       | 2003-07-00 |
+-------+-----+-------------+------------+------------+
```

Using the `client` and `project` tables, you want to retrieve a list of clients that have no projects for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue?

Question 25:

Refer to the structure and sample data for the `client` and `project` tables, shown in the previous question. Using these two tables, you want to retrieve a list of clients that have projects starting in the year 2003, sorted by start date, for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue?

Question 26:

Refer to the structure and sample data for the `client` and `project` tables, shown two questions earlier. Using these two tables, you want to retrieve a list of clients that have intranet projects, using the `LEFT JOIN` syntax, for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue?

Question 27:

Here's the structure and sample data for two tables, `client` and `project`, which will be used for the next three questions.

```
mysql> DESCRIBE client;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| cid   | smallint(5) unsigned | NO   | PRI | 0       |       |
| name  | char(20)             | NO   |     |         |       |
+-------+----------------------+------+-----+---------+-------+
mysql> DESCRIBE project;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| pid   | int(10) unsigned     | NO   | PRI | 0       |       |
| cid   | smallint(5) unsigned | NO   |     | 0       |       |
| name  | char(30)             | NO   |     |         |       |
| start | date                 | YES  |     | NULL    |       |
| end   | date                 | YES  |     | NULL    |       |
+-------+----------------------+------+-----+---------+-------+
mysql> SELECT * FROM client;
+-----+---------------+
| cid | name          |
+-----+---------------+
| 101 | Seamen's      |
| 103 | Lennart AG    |
| 110 | MySQL AB      |
| 115 | Icoaten & Co. |
| 125 | Nittboad Inc  |
+-----+---------------+
mysql> SELECT * FROM project;
+-------+-----+-------------+------------+------------+
| pid   | cid | name        | start      | end        |
+-------+-----+-------------+------------+------------+
| 10000 | 103 | New CMS     | 2003-01-00 | 2003-05-00 |
| 10010 | 110 | Texi2XML    | 2002-04-00 | 2003-09-00 |
| 10020 | 110 | Studyguides | 2002-09-00 | 2003-03-30 |
| 10030 | 115 | PDC Server  | 2003-01-00 | 2003-01-00 |
| 10040 | 103 | Intranet    | 2009-02-00 | NULL       |
| 10050 | 101 | Intranet    | NULL       | NULL       |
| 10060 | 115 | SMB Server  | 2003-05-00 | NULL       |
| 10070 | 115 | WLAN        | NULL       | 2003-07-00 |
+-------+-----+-------------+------------+------------+
```

Using the `client` and `project` tables, you want to retrieve a list of clients that have intranet projects, using the `INNER JOIN` syntax, for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue?

Question 28:

Refer to the structure and sample data for the `client` and `project` tables, shown in the previous question. Using these two tables, you want to retrieve a list of clients that have intranet projects, using an inner join with the `WHERE` clause, for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue?

Question 29:

Refer to the structure and sample data for the `client` and `project` tables, shown two questions earlier. Using these two tables, you want to retrieve a list of clients and their projects, sorted by client name, and within client name, sorted by start date, for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue? Clients without projects should also be displayed.

Question 30:

Here's the structure and sample data for two tables, `client` and `project`.

```
mysql> DESCRIBE client;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| cid   | smallint(5) unsigned | NO   | PRI | 0       |       |
| name  | char(20)             | NO   |     |         |       |
+-------+----------------------+------+-----+---------+-------+
mysql> DESCRIBE project;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| pid   | int(10) unsigned     | NO   | PRI | 0       |       |
| cid   | smallint(5) unsigned | NO   |     | 0       |       |
| name  | char(30)             | NO   |     |         |       |
| start | date                 | YES  |     | NULL    |       |
| end   | date                 | YES  |     | NULL    |       |
+-------+----------------------+------+-----+---------+-------+
mysql> SELECT * FROM client;
+-----+---------------+
| cid | name          |
+-----+---------------+
| 101 | Seamen's      |
| 103 | Lennart AG    |
| 110 | MySQL AB      |
| 115 | Icoaten & Co. |
| 125 | Nittboad Inc  |
+-----+---------------+
mysql> SELECT * FROM project;
+-------+-----+-------------+------------+------------+
| pid   | cid | name        | start      | end        |
+-------+-----+-------------+------------+------------+
| 10000 | 103 | New CMS     | 2003-01-00 | 2003-05-00 |
| 10010 | 110 | Texi2XML    | 2002-04-00 | 2003-09-00 |
| 10020 | 110 | Studyguides | 2002-09-00 | 2003-03-30 |
| 10030 | 115 | PDC Server  | 2003-01-00 | 2003-01-00 |
| 10040 | 103 | Intranet    | 2009-02-00 | NULL       |
| 10050 | 101 | Intranet    | NULL       | NULL       |
| 10060 | 115 | SMB Server  | 2003-05-00 | NULL       |
| 10070 | 115 | WLAN        | NULL       | 2003-07-00 |
+-------+-----+-------------+------------+------------+
```

Using the `client` and `project` tables, you want to retrieve a list of clients and their projects, sorted by client name, and within client name, sorted by start date, for a report with these column headings:

```
+----------+---------+-------+------+
| CLIENT   | PROJECT | START | END  |
+----------+---------+-------+------+
```

What SQL statement will you issue? Clients without projects should be displayed, but projects that don't have a start date and projects that don't have an end date should not be displayed.

Answers to Exercises

Answer 1:

Any kind of join can find matches between tables.

Answer 2:

Outer joins, that is, `LEFT JOIN` and `RIGHT JOIN`.

Answer 3:

```
mysql> SELECT Name, Language
    -> FROM Country, CountryLanguage
    -> WHERE Code = CountryCode
    ->   AND IsOfficial = 'T';
+--------------------------------------+------------------+
| Name                                 | Language         |
+--------------------------------------+------------------+
| Afghanistan                          | Pashto           |
| Netherlands                          | Dutch            |
| Netherlands Antilles                 | Papiamento       |
| Albania                              | Albaniana        |
| Algeria                              | Arabic           |
| American Samoa                       | Samoan           |
...
| South Africa                         | English          |
| Luxembourg                           | German           |
| Nauru                                | English          |
| Pakistan                             | Urdu             |
| Northern Mariana Islands             | English          |
+--------------------------------------+------------------+
```

Answer 4:

```
mysql> SELECT Name, Language
    -> FROM Country INNER JOIN CountryLanguage
    -> ON Code = CountryCode
    -> WHERE IsOfficial = 'T';
+--------------------------------------+------------------+
| Name                                 | Language         |
+--------------------------------------+------------------+
| Afghanistan                          | Pashto           |
| Netherlands                          | Dutch            |
| Netherlands Antilles                 | Papiamento       |
| Albania                              | Albaniana        |
| Algeria                              | Arabic           |
| American Samoa                       | Samoan           |
...
| South Africa                         | English          |
| Luxembourg                           | German           |
| Nauru                                | English          |
| Pakistan                             | Urdu             |
| Northern Mariana Islands             | English          |
+--------------------------------------+------------------+
```

You can only use the `ON` clause because the column names of the join columns differ in the two tables

`Code` and `CountryCode`).

Answer 5:

```
mysql> SELECT Country.Name, City.Name, City.Population
    -> FROM Country
    -> INNER JOIN City
    -> ON CountryCode = Code
    -> WHERE IndepYear IS NULL;
+----------------------------+--------------------------+------------+
| Name                       | Name                     | Population |
+----------------------------+--------------------------+------------+
| Netherlands Antilles       | Willemstad               |       2345 |
| American Samoa             | Tafuna                   |       5200 |
| American Samoa             | Fagatogo                 |       2323 |
...
| Palestine                  | Jabaliya                 |     113901 |
| Palestine                  | Nablus                   |     100231 |
| Palestine                  | Rafah                    |      92020 |
+----------------------------+--------------------------+------------+
```

Answer 6:

```
mysql> SELECT Country.Name, SUM(City.Population)
    -> FROM Country
    -> INNER JOIN City
    -> ON CountryCode = Code
    -> WHERE IndepYear IS NULL
    -> GROUP BY Country.Name;
+----------------------------+----------------------+
| Name                       | SUM(City.Population) |
+----------------------------+----------------------+
| American Samoa             |                 7523 |
| Anguilla                   |                 1556 |
| Aruba                      |                29034 |
...
| Virgin Islands, U.S.       |                13000 |
| Wallis and Futuna          |                 1137 |
| Western Sahara             |               169000 |
+----------------------------+----------------------+
```

Answer 7:

A join that combines all rows in one table with all rows in another table is called a *Cartesian product*. You get a Cartesian product by not specifying a `WHERE` clause in a join that uses the comma operator, like this:

```
mysql> SELECT column, column, ... FROM City, Country;
```

This will result in a big result set. The number of rows this would produced can be calculated by multiplying the number of rows in both tables:

```
mysql> SET @CityCount = (SELECT COUNT(*) FROM City);

mysql> SET @CountryCount = (SELECT COUNT(*) FROM Country);

mysql> SELECT @CityCount * @CountryCount;
+----------------------------+
| @CityCount * @CountryCount |
+---------------------------+
|                    974881 |
+---------------------------+
```

You can avoid creating Cartesian products by accident by starting the `mysql` command-line client with the `--safe-updates` option.

Answer 8:

With two tables, it's always possible to rewrite a `LEFT JOIN` as a `RIGHT JOIN`. This might, however, not be possible for a join that involves more than two tables.

Answer 9:

It's necessary to qualify column names with table names whenever columns of joined tables have the same name. Even when that's not the case, it makes joins easier to understand when column names are qualified with table names, because that makes it instantly clear which tables the columns stem from.

Answer 10:

Whenever tables from more than one database are joined that have the same name.

Answer 11:

This is the case when a table is joined with itself (self-join). For example, to find out which countries have the same number of inhabitants, you would join the `Country` table with itself. To resolve name conflicts, you need to create aliases for the joined tables:

```
mysql> SELECT c1.Name, c2.Name, c1.Population
    -> FROM Country AS c1, Country AS c2
    -> WHERE c1.Population = c2.Population
    -> AND c1.Name != c2.Name AND c1.Population > 0;
+--------------------------+--------------------------+------------+
| Name                     | Name                     | Population |
+--------------------------+--------------------------+------------+
| Antigua and Barbuda      | American Samoa           |      68000 |
| Northern Mariana Islands | Andorra                  |      78000 |
| American Samoa           | Antigua and Barbuda      |      68000 |
| Saint Kitts and Nevis    | Cayman Islands           |      38000 |
| Niue                     | Falkland Islands         |       2000 |
| Norfolk Island           | Falkland Islands         |       2000 |
| Tokelau                  | Falkland Islands         |       2000 |
| Tuvalu                   | Nauru                    |      12000 |
| Falkland Islands         | Niue                     |       2000 |
| Norfolk Island           | Niue                     |       2000 |
| Tokelau                  | Niue                     |       2000 |
| Falkland Islands         | Norfolk Island           |       2000 |
| Niue                     | Norfolk Island           |       2000 |
| Tokelau                  | Norfolk Island           |       2000 |
| Andorra                  | Northern Mariana Islands |      78000 |
| Cayman Islands           | Saint Kitts and Nevis    |      38000 |
| Falkland Islands         | Tokelau                  |       2000 |
| Niue                     | Tokelau                  |       2000 |
| Norfolk Island           | Tokelau                  |       2000 |
| Nauru                    | Tuvalu                   |      12000 |
+--------------------------+--------------------------+------------+
20 rows in set (0.03 sec)
```

Note that `AND c1.Name != c2.Name` avoids comparing countries with themselves, and `AND c1.Population > 0` excludes the countries that have a zero population number.

Answer 12:

In MySQL, joins work for `INSERT`, `UPDATE`, and `DELETE` statements. It's possible to change data in more than one table when joining tables in an `UPDATE` or `DELETE` statement.

Answer 13:

This statement retrieves a list of the countries in South America that have a larger surface area than Paraguay:

```
mysql> SELECT
    ->   c1.Name AS 'Country',
    ->   c2.Name AS 'Other Countries',
    ->   c2.Continent AS 'Continent',
    ->   c2.SurfaceArea AS 'Surface Area'
    ->  FROM Country AS c1
    ->  INNER JOIN Country AS c2
    ->  USING (Continent)
    ->  WHERE c2.SurfaceArea > c1.SurfaceArea
    ->  AND c1.Name = 'Paraguay'
    -> ;
+----------+-----------------+---------------+--------------+
| Country  | Other Countries | Continent     | Surface Area |
+----------+-----------------+---------------+--------------+
| Paraguay | Argentina       | South America |   2780400.00 |
| Paraguay | Bolivia         | South America |   1098581.00 |
| Paraguay | Brazil          | South America |   8547403.00 |
| Paraguay | Chile           | South America |    756626.00 |
| Paraguay | Colombia        | South America |   1138914.00 |
| Paraguay | Peru            | South America |   1285216.00 |
| Paraguay | Venezuela       | South America |    912050.00 |
+----------+-----------------+---------------+--------------+
```

Answer 14:

This statement retrieves a list of all countries whose population is greater than or equal to that of Germany:

```
mysql> SELECT
    ->   c1.Name AS 'Country',
    ->   c2.Name AS 'Other Countries',
    ->   c2.Population AS 'Population'
    ->  FROM Country AS c1, Country AS c2
    ->  WHERE c2.Population >= c1.Population
    ->  AND c1.Name = 'Germany'
    ->  ORDER BY c2.Population DESC
    -> ;
+---------+--------------------+------------+
| Country | Other Countries    | Population |
+---------+--------------------+------------+
| Germany | China              | 1277558000 |
| Germany | India              | 1013662000 |
| Germany | United States      |  278357000 |
| Germany | Indonesia          |  212107000 |
| Germany | Brazil             |  170115000 |
| Germany | Pakistan           |  156483000 |
| Germany | Russian Federation |  146934000 |
| Germany | Bangladesh         |  129155000 |
| Germany | Japan              |  126714000 |
| Germany | Nigeria            |  111506000 |
| Germany | Mexico             |   98881000 |
| Germany | Germany            |   82164700 |
+---------+-------------------+-----------+
```

Answer 15:

This statement retrieves a list of all countries whose population and surface area are greater than or equal to that of Nepal:

```
mysql> SELECT
    ->   c1.Name AS 'Country',
    ->   c2.Name AS 'Other Countries',
    ->   LEFT(c2.Region,10) AS 'Region',
    ->   c2.Population AS 'Population',
    ->   c2.SurfaceArea AS 'Surface'
    ->  FROM Country AS c1, Country AS c2
    ->  WHERE c1.Region = c2.Region
    ->    AND c2.SurfaceArea >= c1.SurfaceArea
    ->    AND c2.Population  >= c1.Population
    ->    AND c1.Name = 'Nepal'
    ->  ORDER BY c2.Population DESC
    -> ;
+---------+-----------------+------------+------------+------------+
| Country | Other Countries | Region     | Population | Surface    |
+---------+-----------------+------------+------------+------------+
| Nepal   | India           | Southern a | 1013662000 | 3287263.00 |
| Nepal   | Pakistan        | Southern a |  156483000 |  796095.00 |
| Nepal   | Iran            | Southern a |   67702000 | 1648195.00 |
| Nepal   | Uzbekistan      | Southern a |   24318000 |  447400.00 |
| Nepal   | Nepal           | Southern a |   23930000 |  147181.00 |
+---------+-----------------+------------+------------+------------+
```

Answer 16:

The statements will return the following result sets:

```
+---------------+-------------+------------+------------+
| name          | name        | start      | end        |
+---------------+-------------+------------+------------+
| Seamen's      | New CMS     | 2003-01-00 | 2003-05-00 |
| Lennart AG    | New CMS     | 2003-01-00 | 2003-05-00 |
| MySQL AB      | New CMS     | 2003-01-00 | 2003-05-00 |
| Icoaten & Co. | New CMS     | 2003-01-00 | 2003-05-00 |
| Nittboad Inc  | New CMS     | 2003-01-00 | 2003-05-00 |
| Seamen's      | Texi2XML    | 2002-04-00 | 2003-09-00 |
| Lennart AG    | Texi2XML    | 2002-04-00 | 2003-09-00 |
| MySQL AB      | Texi2XML    | 2002-04-00 | 2003-09-00 |
| Icoaten & Co. | Texi2XML    | 2002-04-00 | 2003-09-00 |
| Nittboad Inc  | Texi2XML    | 2002-04-00 | 2003-09-00 |
| Seamen's      | Studyguides | 2002-09-00 | 2003-03-30 |
| Lennart AG    | Studyguides | 2002-09-00 | 2003-03-30 |
| MySQL AB      | Studyguides | 2002-09-00 | 2003-03-30 |
| Icoaten & Co. | Studyguides | 2002-09-00 | 2003-03-30 |
| Nittboad Inc  | Studyguides | 2002-09-00 | 2003-03-30 |
| Seamen's      | PDC Server  | 2003-01-00 | 2003-01-00 |
| Lennart AG    | PDC Server  | 2003-01-00 | 2003-01-00 |
| MySQL AB      | PDC Server  | 2003-01-00 | 2003-01-00 |
| Icoaten & Co. | PDC Server  | 2003-01-00 | 2003-01-00 |
| Nittboad Inc  | PDC Server  | 2003-01-00 | 2003-01-00 |
| Seamen's      | Intranet    | 2009-02-00 | NULL       |
| Lennart AG    | Intranet    | 2009-02-00 | NULL       |
| MySQL AB      | Intranet    | 2009-02-00 | NULL       |
| Icoaten & Co. | Intranet    | 2009-02-00 | NULL       |
| Nittboad Inc  | Intranet    | 2009-02-00 | NULL       |
| Seamen's       | Intranet    | NULL       | NULL       |
| Lennart AG     | Intranet    | NULL       | NULL       |
| MySQL AB       | Intranet    | NULL       | NULL       |
| Icoaten & Co.  | Intranet    | NULL       | NULL       |
| Nittboad Inc   | Intranet    | NULL       | NULL       |
| Seamen's       | SMB Server  | 2003-05-00 | NULL       |
| Lennart AG     | SMB Server  | 2003-05-00 | NULL       |
| MySQL AB       | SMB Server  | 2003-05-00 | NULL       |
| Icoaten & Co.  | SMB Server  | 2003-05-00 | NULL       |
| Nittboad Inc   | SMB Server  | 2003-05-00 | NULL       |
| Seamen's       | WLAN        | NULL       | 2003-07-00 |
| Lennart AG     | WLAN        | NULL       | 2003-07-00 |
| MySQL AB       | WLAN        | NULL       | 2003-07-00 |
| Icoaten & Co.  | WLAN        | NULL       | 2003-07-00 |
| Nittboad Inc   | WLAN        | NULL       | 2003-07-00 |
+----------------+-------------+------------+------------+
40 rows in set (0.00 sec)

+----------------+-------------+------------+------------+
| name           | name        | start      | end        |
+----------------+-------------+------------+------------+
| Seamen's       | Intranet    | NULL       | NULL       |
| Lennart AG     | New CMS     | 2003-01-00 | 2003-05-00 |
| Lennart AG     | Intranet    | 2009-02-00 | NULL       |
| MySQL AB       | Texi2XML    | 2002-04-00 | 2003-09-00 |
| MySQL AB       | Studyguides | 2002-09-00 | 2003-03-30 |
| Icoaten & Co.  | PDC Server  | 2003-01-00 | 2003-01-00 |
| Icoaten & Co.  | SMB Server  | 2003-05-00 | NULL       |
| Icoaten & Co.  | WLAN        | NULL       | 2003-07-00 |
+----------------+-------------+------------+------------+
8 rows in set (0.00 sec)
```

Answer 17:

The statements will return the following result sets:

```
+----------------+-------------+------------+------------+
| name           | name        | start      | end        |
+----------------+-------------+------------+------------+
| Lennart AG     | New CMS     | 2003-01-00 | 2003-05-00 |
| Lennart AG     | Intranet    | 2009-02-00 | NULL       |
| MySQL AB       | Texi2XML    | 2002-04-00 | 2003-09-00 |
| MySQL AB       | Studyguides | 2002-09-00 | 2003-03-30 |
| Icoaten & Co.  | PDC Server  | 2003-01-00 | 2003-01-00 |
| Icoaten & Co.  | SMB Server  | 2003-05-00 | NULL       |
+----------------+-------------+------------+------------+
6 rows in set (0.01 sec)

+----------------+-------------+------------+------------+
| name           | name        | start      | end        |
+----------------+-------------+------------+------------+
| Lennart AG     | New CMS     | 2003-01-00 | 2003-05-00 |
| MySQL AB       | Texi2XML    | 2002-04-00 | 2003-09-00 |
| MySQL AB       | Studyguides | 2002-09-00 | 2003-03-30 |
| Icoaten & Co.  | PDC Server  | 2003-01-00 | 2003-01-00 |
+----------------+-------------+------------+------------+
4 rows in set (0.00 sec)
```

Answer 18:

Here's one SQL statement that accomplishes the task:

```
mysql> SELECT
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c
    ->  LEFT JOIN project AS p
    ->  USING (cid)
    ->  WHERE p.cid IS NULL
    ->  ORDER BY CLIENT
    -> ;
+--------------+---------+-------+------+
| CLIENT       | PROJECT | START | END  |
+--------------+---------+-------+------+
| Nittboad Inc | NULL    | NULL  | NULL |
+--------------+---------+-------+------+
```

Answer 19:

Here's one SQL statement that accomplishes the task:

```
mysql> SELECT
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c
    ->  LEFT JOIN project AS p
    ->  USING (cid)
    ->  WHERE p.start BETWEEN '2003-01-00' AND '2003-12-31'
    ->  ORDER BY START
    -> ;
+---------------+------------+------------+------------+
| CLIENT        | PROJECT    | START      | END        |
+---------------+------------+------------+------------+
| Lennart AG    | New CMS    | 2003-01-00 | 2003-05-00 |
| Icoaten & Co. | PDC Server | 2003-01-00 | 2003-01-00 |
| Icoaten & Co. | SMB Server | 2003-05-00 | NULL       |
+---------------+------------+------------+------------+
```

Answer 20:

This statement accomplishes the task:

```
mysql> SELECT
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c
    ->  LEFT JOIN project AS p
    ->  USING (cid) /* or: ON c.cid = p.cid */
    ->  WHERE p.name = 'Intranet'
    -> ;
+------------+----------+------------+------+
| CLIENT     | PROJECT  | START      | END  |
+------------+----------+------------+------+
| Seamen's   | Intranet | NULL       | NULL |
| Lennart AG | Intranet | 2009-02-00 | NULL |
+------------+----------+------------+------+
```

Answer 21:

This statement accomplishes the task:

```
mysql> SELECT
```

```
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c
    ->  INNER JOIN project AS p
    ->  USING (cid) /* or: ON c.cid = p.cid */
    ->  WHERE p.name = 'Intranet'
    -> ;
+------------+----------+------------+------+
| CLIENT     | PROJECT  | START      | END  |
+------------+----------+------------+------+
| Seamen's   | Intranet | NULL       | NULL |
| Lennart AG | Intranet | 2009-02-00 | NULL |
+------------+----------+------------+------+
```

Answer 22:

This statement accomplishes the task:

```
mysql> SELECT
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c, project AS p
    ->  WHERE c.cid = p.cid
    ->  AND p.name = 'Intranet'
    -> ;
+------------+----------+------------+------+
| CLIENT     | PROJECT  | START      | END  |
+------------+----------+------------+------+
| Seamen's   | Intranet | NULL       | NULL |
| Lennart AG | Intranet | 2009-02-00 | NULL |
+------------+----------+------------+------+
```

Answer 23:

Here's one SQL statement that accomplishes the task:

```
mysql> SELECT
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c
    ->  LEFT JOIN project AS p
    ->  USING (cid) /* or: ON c.cid = p.cid */
    ->  ORDER BY c.name, p.start;
+---------------+-------------+------------+------------+
| CLIENT        | PROJECT     | START      | END        |
+---------------+-------------+------------+------------+
| Icoaten & Co. | WLAN        | NULL       | 2003-07-00 |
| Icoaten & Co. | PDC Server  | 2003-01-00 | 2003-01-00 |
| Icoaten & Co. | SMB Server  | 2003-05-00 | NULL       |
| Lennart AG    | New CMS     | 2003-01-00 | 2003-05-00 |
| Lennart AG    | Intranet    | 2009-02-00 | NULL       |
| MySQL AB      | Texi2XML    | 2002-04-00 | 2003-09-00 |
| MySQL AB      | Studyguides | 2002-09-00 | 2003-03-30 |
| Nittboad Inc  | NULL        | NULL       | NULL       |
| Seamen's      | Intranet    | NULL       | NULL       |
+---------------+-------------+------------+------------+
```

Answer 24:

Here's one SQL statement that accomplishes the task:

```
mysql> SELECT
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c
    ->  LEFT JOIN project AS p
    ->  USING (cid)
    ->  WHERE p.cid IS NULL
    -> ;
+-------------+---------+-------+------+
| CLIENT      | PROJECT | START | END  |
+-------------+---------+-------+------+
| Nittboad Inc | NULL    | NULL  | NULL |
+-------------+---------+-------+------+
```

Answer 25:

Here's one SQL statement that accomplishes the task:

```
mysql> SELECT
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c
    ->  LEFT JOIN project AS p
    ->  USING (cid)
    ->  WHERE p.start BETWEEN '2003-01-00' AND '2003-12-31'
    ->  ORDER BY START
    -> ;
+--------------+------------+------------+------------+
| CLIENT       | PROJECT    | START      | END        |
+--------------+------------+------------+------------+
| Lennart AG   | New CMS    | 2003-01-00 | 2003-05-00 |
| Icoaten & Co. | PDC Server | 2003-01-00 | 2003-01-00 |
| Icoaten & Co. | SMB Server | 2003-05-00 | NULL       |
+--------------+------------+------------+------------+
```

Answer 26:

This statement accomplishes the task:

```
mysql> SELECT
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c
    ->  LEFT JOIN project AS p
    ->  USING (cid) /* or: ON c.cid = p.cid */
    ->  WHERE p.name = 'Intranet'
    -> ;
+------------+----------+------------+------+
| CLIENT     | PROJECT  | START      | END  |
+------------+----------+------------+------+
| Seamen's   | Intranet | NULL       | NULL |
| Lennart AG | Intranet | 2009-02-00 | NULL |
+------------+----------+------------+------+
```

Answer 27:

This statement accomplishes the task:

```
mysql> SELECT
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c
    ->  INNER JOIN project AS p
    ->  USING (cid) /* or: ON c.cid = p.cid */
    ->  WHERE p.name = 'Intranet'
    -> ;
+------------+----------+------------+------+
| CLIENT     | PROJECT  | START      | END  |
+------------+----------+------------+------+
| Seamen's   | Intranet | NULL       | NULL |
| Lennart AG | Intranet | 2009-02-00 | NULL |
+------------+----------+------------+------+
```

Answer 28:

This statement accomplishes the task:

```
mysql> SELECT
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c, project AS p
    ->  WHERE c.cid = p.cid
    ->  AND p.name = 'Intranet'
    -> ;
+------------+----------+------------+------+
| CLIENT     | PROJECT  | START      | END  |
+------------+----------+------------+------+
| Seamen's   | Intranet | NULL       | NULL |
| Lennart AG | Intranet | 2009-02-00 | NULL |
+------------+----------+------------+------+
```

Answer 29:

Here's one SQL statement that accomplishes the task:

```
mysql> SELECT
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c
    ->  LEFT JOIN project AS p
    ->  USING (cid) /* or: ON c.cid = p.cid */
    ->  ORDER BY c.name, p.start;
+---------------+-------------+------------+------------+
| CLIENT        | PROJECT     | START      | END        |
+---------------+-------------+------------+------------+
| Icoaten & Co. | WLAN        | NULL       | 2003-07-00 |
| Icoaten & Co. | PDC Server  | 2003-01-00 | 2003-01-00 |
| Icoaten & Co. | SMB Server  | 2003-05-00 | NULL       |
| Lennart AG    | New CMS     | 2003-01-00 | 2003-05-00 |
| Lennart AG    | Intranet    | 2009-02-00 | NULL       |
| MySQL AB      | Texi2XML    | 2002-04-00 | 2003-09-00 |
| MySQL AB      | Studyguides | 2002-09-00 | 2003-03-30 |
| Nittboad Inc  | NULL        | NULL       | NULL       |
| Seamen's      | Intranet    | NULL       | NULL       |
+---------------+-------------+------------+------------+
```

Answer 30:

Here's one SQL statement that accomplishes the task:

```
mysql> SELECT
```

```
    ->  c.name AS CLIENT, p.name AS PROJECT,
    ->  p.start AS START, p.end AS END
    ->  FROM client AS c
    ->  LEFT JOIN project AS p
    ->  USING (cid) /* or: ON c.cid = p.cid */
    ->  WHERE p.start IS NOT NULL
    ->    AND p.end   IS NOT NULL
    ->  ORDER BY c.name ASC, p.start ASC
    -> ;
+---------------+-------------+------------+------------+
| CLIENT        | PROJECT     | START      | END        |
+---------------+-------------+------------+------------+
| Icoaten & Co. | PDC Server  | 2003-01-00 | 2003-01-00 |
| Lennart AG    | New CMS     | 2003-01-00 | 2003-05-00 |
| MySQL AB      | Texi2XML    | 2002-04-00 | 2003-09-00 |
| MySQL AB      | Studyguides | 2002-09-00 | 2003-03-30 |
+---------------+-------------+------------+------------+
```

Chapter 13. Subqueries

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 in an SQL statement may a scalar subquery be placed?

Question 2:

The following query selects those continents that have countries in which more than 50% of the population speak English. Is this an example of using a correlated subquery? Why or why not?

```
SELECT DISTINCT Continent
FROM Country
WHERE Code IN (SELECT CountryCode
               FROM CountryLanguage
               WHERE Language='English'
                 AND Percentage>50
              );
```

Question 3:

The following statement uses a non-correlated subquery to find the South American country with the smallest population:

```
SELECT * FROM Country
WHERE Continent = 'South America'
AND Population = (SELECT MIN(Population) FROM Country
                  WHERE Continent = 'South America');
```

Rewrite the statement to use a correlated subquery.

Question 4:

What is the effect of executing the following query?

```
SELECT Continent, Name
FROM Country c1
WHERE Population >= ALL (SELECT Population
                         FROM Country c2
                         WHERE c1.Continent=c2.Continent
                        );
```

Question 5:

What is the effect of executing the following query? (Compare to the previous exercise.)

```
SELECT Continent, Name
FROM Country
WHERE SurfaceArea > ANY (SELECT AVG(SurfaceArea)
                         FROM Country
                         GROUP BY Continent
                        );
```

Question 6:

How would you use a subquery to write a `SELECT` statement that answers the following question: What is the largest country (in terms of surface area) on each continent?

Question 7:

How would you use `IN` and a subquery to write a `SELECT` statement that answers the following question: What languages are spoken in countries where the form of government is a monarchy? Sort the languages lexically.

Question 8:

How would you use `EXISTS` and a subquery to write a `SELECT` statement that answers the following question: In what countries are people that speak German found? Order the country names lexically.

Question 9:

How would you use a row constructor to find the population of Houston, Texas, USA?

Question 10:

Using a `SELECT` statement with a subquery in the `FROM` clause, find the number of people in the region “Western Europe” that speak German as their mother tongue. The answer must be returned as a scalar.

Question 11:

Why is the following use of a subquery in the `FROM` clause not correct?

```
SELECT Name, Language
FROM Country AS c, (SELECT Language
                     FROM CountryLanguage
                     WHERE CountryCode = c.Code
                    ) AS tmp;
```

Question 12:

Joe wants to create a table of country capitals. He creates the `Capitals` table by copying the structure and data of the `City` table, but then by mistake copies *all* of the data from the `City` table into the `Capitals` table:

```
mysql> CREATE TABLE Capitals LIKE City;
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO Capitals SELECT * FROM City;
Query OK, 4079 rows affected (0.08 sec)
Records: 4079  Duplicates: 0  Warnings: 0
```

The city ID of a country's capital is stored in the `Capital` field of the `Country` table. Using a subquery, how can Joe remove all the non-capital cities from the `Capitals` table?

Question 13:

The following two tables, `client` and `project`, are used for the next three questions.

```
mysql> SELECT * FROM client;
+------+---------------+
| id   | name           |
+------+----------------+
|  101 | Seamen's       |
|  103 | Lennart AG     |
|  110 | MySQL AB       |
|  115 | Icoaten & Co.  |
|  125 | Nittboad Inc   |
+------+----------------+
mysql> SELECT * FROM project;
+-------+------+-------------+------------+------------+
| pid   | id   | name        | start      | end        |
+-------+------+-------------+------------+------------+
| 10000 |  103 | New CMS     | 2003-01-00 | 2003-05-00 |
| 10010 |  110 | Texi2XML    | 2002-04-00 | 2003-09-00 |
| 10020 |  110 | Studyguides | 2002-09-00 | 2003-03-30 |
| 10030 |  115 | PDC Server  | 2003-01-00 | 2003-01-00 |
| 10040 |  103 | Intranet    | 2009-02-00 | NULL       |
| 10050 |  101 | Intranet    | NULL       | NULL       |
| 10060 |  115 | SMB Server  | 2003-05-00 | NULL       |
| 10070 |  115 | WLAN        | NULL       | 2003-07-00 |
| 10080 |  135 | Intranet    | 2003-08-00 | NULL       |
| 10090 |  145 | PDC Server  | NULL       | NULL       |
+-------+------+-------------+------------+------------+
```

The `client` and `project` tables are related through their common column (`id`). Thus, the following statement could be used to determine which projects have clients (that is, which projects have an `id` whose value is the same as one of the `id` values found in the `client` table):

```
SELECT ... FROM project WHERE project.id IN
 (SELECT client.id FROM client)
```

How would you rewrite the statement using the `LEFT JOIN` operator? Your output should have column headings like this:

```
+------------+--------------+-----------+---------------+
| Project ID | Project Name | Client No | Client Name   |
+------------+--------------+-----------+---------------+
```

Question 14:

Recall that the `client` and `project` tables shown in the previous question are related through their common column (`id`). This SQL statement produces a list of projects that have an `id` whose value is the same as one of the `id` values found in the `client` table:

```
SELECT ... FROM project WHERE project.id IN
 (SELECT client.id FROM client)
```

How would you rewrite the statement using the `RIGHT JOIN` operator? Your output should have column headings like this:

```
+------------+--------------+-----------+---------------+
| Project ID | Project Name | Client No | Client Name   |
+------------+--------------+-----------+---------------+
```

Question 15:

Recall that the `client` and `project` tables shown two questions earlier are related through their common column (`id`). This SQL statement produces a list of projects that have an `id` whose value is the same as one of the `id` values found in the `client` table:

```
SELECT ... FROM project WHERE project.id IN
 (SELECT client.id FROM client)
```

How would you rewrite the statement using the `INNER JOIN` operator? Your output should have column headings like this:

```
+------------+-------------+----------+--------------+
| Project ID | Project Name | Client No | Client Name   |
+------------+-------------+----------+--------------+
```

Question 16:

The following two tables, `client` and `project`, are used for the next three questions.

```
mysql> SELECT * FROM client;
+------+---------------+
| id   | name          |
+------+---------------+
|  101 | Seamen's      |
|  103 | Lennart AG    |
|  110 | MySQL AB      |
|  115 | Icoaten & Co. |
|  125 | Nittboad Inc  |
+------+---------------+
mysql> SELECT * FROM project;
+-------+------+------------+------------+------------+
| pid   | id   | name       | start      | end        |
+-------+------+------------+------------+------------+
| 10000 |  103 | New CMS    | 2003-01-00 | 2003-05-00 |
| 10010 |  110 | Texi2XML   | 2002-04-00 | 2003-09-00 |
| 10020 |  110 | Studyguides | 2002-09-00 | 2003-03-30 |
| 10030 |  115 | PDC Server | 2003-01-00 | 2003-01-00 |
| 10040 |  103 | Intranet   | 2009-02-00 | NULL       |
| 10050 |  101 | Intranet   | NULL       | NULL       |
| 10060 |  115 | SMB Server | 2003-05-00 | NULL       |
| 10070 |  115 | WLAN       | NULL       | 2003-07-00 |
| 10080 |  135 | Intranet   | 2003-08-00 | NULL       |
| 10090 |  145 | PDC Server | NULL       | NULL       |
+-------+------+------------+------------+------------+
```

The `client` and `project` tables are related through their common column (`id`). Thus, the following statement could be used to determine which projects don't have clients (that is, which projects have an `id` whose value is not the same as one of the `id` values found in the `client` table):

```
SELECT ... FROM project WHERE project.id NOT IN
 (SELECT client.id FROM client)
```

How would you rewrite the statement using the `LEFT JOIN` operator? Your output should have column headings like this:

```
+------------+-------------+----------+--------------+
| Project ID | Project Name | Client No | Client Name   |
+------------+-------------+----------+--------------+
```

Question 17:

Recall that the `client` and `project` tables shown in the previous question are related through their common column (`id`). This SQL statement produces a list of projects that have an `id` whose value is not the same as one of the `id` values found in the `client` table:

```
SELECT ... FROM project WHERE project.id NOT IN
 (SELECT client.id FROM client)
```

How would you rewrite the statement using the `RIGHT JOIN` operator? Your output should have column headings like this:

```
+------------+-------------+-----------+--------------+
| Project ID | Project Name | Client No | Client Name   |
+------------+-------------+-----------+--------------+
```

Question 18:

Recall that the `client` and `project` tables shown two questions earlier are related through their common column (`id`). This SQL statement produces a list of projects that have an `id` whose value is not the same as one of the `id` values found in the `client` table:

```
SELECT ... FROM project WHERE project.id NOT IN
 (SELECT client.id FROM client)
```

How would you rewrite the statement using the `INNER JOIN` operator? Your output should have column headings like this:

```
+------------+-------------+-----------+--------------+
| Project ID | Project Name | Client No | Client Name   |
+------------+-------------+-----------+--------------+
```

Question 19:

We would like to execute a query on the `world` database that returns the name of a country with the highest number of official languages, the number of official languages in that country, and what those languages are. What should be inserted for each “`...`” in the following statement to accomplish this task?

```
SELECT CONCAT(
        'The country ',
        ... ,
        ' has ',
        ... ,
        ' official languages: ',
        ...
       );
```

Hint: This exercise covers more subjects discussed in the text than you might initially think. Do test your solution on the `world` database before looking at the answer.

Answers to Exercises

Answer 1:

A scalar subquery may be placed almost anywhere in an SQL statement that a scalar value, such as a true scalar, an argument to a function call, a term of a mathematical expression, and so forth, is expected. Exceptions include contexts in which a literal scalar is required, such as for arguments in `LIMIT` clauses.

Answer 2:

The example shown is *not* an example of a correlated subquery because the subquery can be resolved completely without regard to the outer query. In a correlated subquery, the inner SELECT is dependent on the outer query.

Answer 3:

```
SELECT * FROM Country c1
WHERE Continent = 'South America'
AND Population = (SELECT MIN(Population) FROM Country c2
                  WHERE c2.Continent = c1.Continent);
```

In the subquery, c1 depends on the outer query, because the c1 table alias is defined in the outer query.

Answer 4:

The query returns, for each continent, the country whose population is greater than or equal to the population of every country on the same continent. In other words, it returns the country with the greatest population on each continent.

Answer 5:

The query returns all the countries that have a surface area larger than the average surface area of the countries on any continent. The difference between this and the earlier query is that this is not a correlated subquery, so it is not restricted to comparing only to the surface area of the continent where the country is located. (For example, instead of AVG(Surface), you could use AVG(Population), although that would not yield in a meaningful result.)

Answer 6:

In the following query, the subquery returns the largest surface area found on each continent. This information is then used in the outer query to find the country on the same continent that has that surface area:

```
mysql> SELECT Continent C, Name, SurfaceArea
    -> FROM Country
    -> WHERE SurfaceArea = (
    ->  SELECT MAX(SurfaceArea)
    ->  FROM Country
    ->  WHERE Continent = C);
+---------------+--------------------+-------------+
| C             | Name               | SurfaceArea |
+---------------+--------------------+-------------+
| Oceania       | Australia          |  7741220.00 |
| South America | Brazil             |  8547403.00 |
| North America | Canada             |  9970610.00 |
| Asia          | China              |  9572900.00 |
| Africa        | Sudan              |  2505813.00 |
| Europe        | Russian Federation | 17075400.00 |
| Antarctica    | Antarctica         | 13120000.00 |
+---------------+--------------------+-------------+
```

Answer 7:

In the following query, the subquery returns the country code of countries where the government form is monarchy. This information is used in the outer query to find the languages spoken in those countries.

```
mysql> SELECT DISTINCT Language
    -> FROM CountryLanguage
    -> WHERE CountryCode IN (
    ->  SELECT Code
    ->  FROM Country
    ->  Where GovernmentForm = 'Monarchy')
    -> ORDER BY Language;
+----------+
| Language |
+----------+
| Arabic   |
| Asami    |
| Dzongkha |
| English  |
| Nepali   |
| Swazi    |
| Tongan   |
| Urdu     |
| Zulu     |
+----------+
```

Answer 8:

In the following query, the subquery returns all the languages spoken in a given country. This information is then used with the `EXISTS` predicate to determine if German is one of the languages spoken in that country.

```
mysql> SELECT Code c, Name
    -> FROM Country
    -> WHERE EXISTS (SELECT *
    ->               FROM CountryLanguage
    ->               WHERE CountryCode = c
    ->               AND Language = 'German')
    -> ORDER BY Name;
+-----+----------------+
| c   | Name           |
+-----+----------------+
| AUS | Australia      |
| AUT | Austria        |
| BEL | Belgium        |
| BRA | Brazil         |
| CAN | Canada         |
| CZE | Czech Republic |
| DNK | Denmark        |
| DEU | Germany        |
| HUN | Hungary        |
| ITA | Italy          |
| KAZ | Kazakstan      |
| LIE | Liechtenstein  |
| LUX | Luxembourg     |
| NAM | Namibia        |
| PRY | Paraguay       |
| POL | Poland         |
| ROM | Romania        |
| CHE | Switzerland    |
| USA | United States  |
+-----+----------------+
```

Answer 9:

The following query uses a row constructor to look up the population of Houston, Texas, USA in the

`City` table:

```
mysql> SELECT Population
    -> FROM City
    -> WHERE (Name, District, CountryCode) = ('Houston', 'Texas', 'USA');
+------------+
| Population |
+------------+
|    1953631 |
+------------+
```

Answer 10:

The answer must be returned as a scalar value; that is, as a result set with a single row and a single column. The following query accomplishes the task:

```
mysql> SELECT SUM(Speakers)
    -> FROM (SELECT (Population * Percentage) / 100 AS Speakers
    ->       FROM CountryLanguage cl, Country c
    ->       WHERE cl.CountryCode = c.Code
    ->         AND c.Region = 'Western Europe'
    ->         AND cl.Language = 'German'
    ->      ) AS tmp;
+---------------+
| SUM(Speakers) |
+---------------+
|  87156001.998 |
+---------------+
```

The subquery finds the number of German speakers for each country in Western Europe. The outer query sums up those numbers. The `tmp` alias is necessary because otherwise you would get this error:

```
ERROR 1248 (42000): Every derived table must have its own alias
```

Answer 11:

The server returns `ERROR 1109 (42S02): Unknown table 'c' in where clause` if you try to execute this query. Subqueries in the `FROM` clause of a query cannot be correlated with the outer query.

Answer 12:

The subquery in the following statement searches the `City` table to identify the city IDs of all capital cities. The `IS NOT NULL` clause is needed because a few countries don't have a capital. The outer statement deletes all rows in the table `Capitals` that are not found by the subquery:

```
mysql> DELETE FROM Capitals
    -> WHERE ID NOT IN (SELECT Capital
    ->                  FROM Country
    ->                  WHERE Capital IS NOT NULL);
Query OK, 3847 rows affected (1.19 sec)
```

Answer 13:

Subquery converted to a `LEFT JOIN`:

```
mysql> SELECT
    ->   p.pid AS `Project ID`,
    ->   p.name AS `Project Name`,
    ->   c.id AS `Client No`,
    ->   c.name AS `Client Name`
    ->  FROM project AS p
    ->  LEFT JOIN client AS c
    ->  USING (id) /* or: ON p.id = c.id */
    ->  WHERE c.name IS NOT NULL
    -> ;
+------------+--------------+-----------+---------------+
| Project ID | Project Name | Client No | Client Name   |
+------------+--------------+-----------+---------------+
|      10000 | New CMS      |       103 | Lennart AG    |
|      10010 | Texi2XML     |       110 | MySQL AB      |
|      10020 | Studyguides  |       110 | MySQL AB      |
|      10030 | PDC Server   |       115 | Icoaten & Co. |
|      10040 | Intranet     |       103 | Lennart AG    |
|      10050 | Intranet     |       101 | Seamen's      |
|      10060 | SMB Server   |       115 | Icoaten & Co. |
|      10070 | WLAN         |       115 | Icoaten & Co. |
+------------+--------------+-----------+---------------+
```

Answer 14:

Subquery converted to a `RIGHT JOIN`:

```
mysql> SELECT
    ->   p.pid AS `Project ID`,
    ->   p.name AS `Project Name`,
    ->   c.id AS `Client No`,
    ->   c.name AS `Client Name`
    ->  FROM client AS c
    ->  RIGHT JOIN project AS p
    ->  USING (id) /* or: ON c.id = p.id */
    ->  WHERE c.name IS NOT NULL
    -> ;
+------------+--------------+-----------+---------------+
| Project ID | Project Name | Client No | Client Name   |
+------------+--------------+-----------+---------------+
|      10000 | New CMS      |       103 | Lennart AG    |
|      10010 | Texi2XML     |       110 | MySQL AB      |
|      10020 | Studyguides  |       110 | MySQL AB      |
|      10030 | PDC Server   |       115 | Icoaten & Co. |
|      10040 | Intranet     |       103 | Lennart AG    |
|      10050 | Intranet     |       101 | Seamen's      |
|      10060 | SMB Server   |       115 | Icoaten & Co. |
|      10070 | WLAN         |       115 | Icoaten & Co. |
+------------+--------------+-----------+---------------+
```

Answer 15:

Subquery converted to an `INNER JOIN`:

```
mysql> SELECT
    ->   p.pid AS `Project ID`,
    ->   p.name AS `Project Name`,
    ->   c.id AS `Client No`,
    ->   c.name AS `Client Name`
    ->  FROM project AS p
    ->  INNER JOIN client AS c
    -> USING (id) /* or: ON p.id = c.id */
    -> ;
+------------+--------------+-----------+---------------+
| Project ID | Project Name | Client No | Client Name   |
+------------+-------------+-----------+---------------+
|      10000 | New CMS     |       103 | Lennart AG    |
|      10010 | Texi2XML    |       110 | MySQL AB      |
|      10020 | Studyguides |       110 | MySQL AB      |
|      10030 | PDC Server  |       115 | Icoaten & Co. |
|      10040 | Intranet    |       103 | Lennart AG    |
|      10050 | Intranet    |       101 | Seamen's      |
|      10060 | SMB Server  |       115 | Icoaten & Co. |
|      10070 | WLAN        |       115 | Icoaten & Co. |
+------------+-------------+-----------+---------------+
```

Answer 16:

Subquery converted to a `LEFT JOIN`:

```
mysql> SELECT
    ->   p.pid AS `Project ID`,
    ->   p.name AS `Project Name`,
    ->   c.id AS `Client No`,
    ->   c.name AS `Client Name`
    ->  FROM project AS p
    ->  LEFT JOIN client AS c
    ->  USING (id) /* or: ON p.id = c.id */
    ->  WHERE c.name IS NULL
    -> ;
+------------+--------------+-----------+-------------+
| Project ID | Project Name | Client No | Client Name |
+------------+--------------+-----------+-------------+
|      10080 | Intranet     |      NULL | NULL        |
|      10090 | PDC Server   |      NULL | NULL        |
+------------+--------------+-----------+-------------+
```

Answer 17:

Subquery converted to a `RIGHT JOIN`:

```
mysql> SELECT
    ->   p.pid AS `Project ID`,
    ->   p.name AS `Project Name`,
    ->   c.id AS `Client No`,
    ->   c.name AS `Client Name`
    ->  FROM client AS c
    ->  RIGHT JOIN project AS p
    ->  USING (id) /* or: ON c.id = p.id */
    ->  WHERE c.name IS NULL
    -> ;
+------------+--------------+-----------+-------------+
| Project ID | Project Name | Client No | Client Name |
+------------+--------------+-----------+-------------+
|      10080 | Intranet     |      NULL | NULL        |
|      10090 | PDC Server   |      NULL | NULL        |
+------------+--------------+-----------+-------------+
```

Answer 18:

This subquery cannot be converted into an inner join because an inner join will find only matching combinations, not rows that do not match.

Answer 19:

For each of the missing parameters in the `CONCAT()` function, we can insert a scalar subquery that returns the required data.

First, we consider how to find the name of the country with the highest number of official languages, and the number of official languages. The following query will give us the maximum number of official languages, and the country name that goes along with it:

```
SELECT Name, COUNT(*) AS nlanguages
FROM Country c, CountryLanguage cl
WHERE c.Code = cl.CountryCode
  AND cl.IsOfficial = 'T'
GROUP BY Name
ORDER BY nlanguages DESC
LIMIT 1;
```

However, there are a few problems with this approach that must be resolved. First of all, the preceding query does not give us a list of the official languages in the country, just the number of them. To find out what the languages are, we utilize the `GROUP_CONCAT()` function as part of the `SELECT` statement:

```
SELECT Name, COUNT(*) AS nlanguages,
       GROUP_CONCAT(Language) as languages
FROM Country c, CountryLanguage cl
WHERE c.Code = cl.CountryCode
  AND cl.IsOfficial = 'T'
GROUP BY Name
ORDER BY nlanguages DESC
LIMIT 1;
```

The second problem with our approach so far is that a scalar subquery may return only a single column. The query just presented returns three. We can solve this problem by nesting the subquery once again:

```
SELECT Name
FROM (SELECT Name, COUNT(*) AS nlanguages,
             GROUP_CONCAT(Language) as languages
      FROM Country c, CountryLanguage cl
      WHERE c.Code = cl.CountryCode
        AND cl.IsOfficial = 'T'
      GROUP BY Name
      ORDER BY nlanguages DESC
      LIMIT 1
     ) AS tmp;
```

For the number of languages and the list of languages, we can utilize the same method, of course replacing `Name` with `nlanguages` or `languages` in the outer `SELECT` as required.

The last problem we need to resolve is that, in the `CountryLanguage` table, there are *two* countries with the maximum number of official languages: South Africa and Switzerland both have four official languages.

Choosing an arbitrary one of these as input for our surrounding `CONCAT()` function will not invalidate the requirement in the question. The result for the `nlanguages` column will, in either case, always be correct. However, there is no guarantee that either Switzerland or South Africa will be the country selected every time the subquery is evaluated, so we run the risk of presenting the wrong combination of country name and list of languages. To resolve this, we place an extra condition on the `ORDER BY` clause, which (rather arbitrarily) does a secondary sort in default order (ascending) by the country name. This means that South Africa will appear, rather than Switzerland.

The complete statement thus looks like this:

```
SELECT
```

```
 CONCAT(
  'The country ',
  (SELECT Name FROM (
    SELECT Name, COUNT(*) AS nlanguages,
           GROUP_CONCAT(Language) as languages
    FROM Country c, CountryLanguage cl
    WHERE c.Code = cl.CountryCode
      AND cl.IsOfficial = 'T'
    GROUP BY Name
    ORDER BY nlanguages DESC, Name
    LIMIT 1
   ) AS tmp
  ),
  ' has ',
  (SELECT nlanguages FROM (
    SELECT Name, COUNT(*) AS nlanguages,
           GROUP_CONCAT(Language) as languages
    FROM Country c, CountryLanguage cl
    WHERE c.Code = cl.CountryCode
      AND cl.IsOfficial = 'T'
    GROUP BY Name
    ORDER BY nlanguages DESC, Name
    LIMIT 1
   ) AS tmp1
  ),
  ' official languages: ',
  (SELECT languages FROM  (
    SELECT Name, COUNT(*) AS nlanguages,
           GROUP_CONCAT(Language) as languages
    FROM Country c, CountryLanguage cl
    WHERE c.Code = cl.CountryCode
      AND cl.IsOfficial = 'T'
    GROUP BY Name
    ORDER BY nlanguages DESC, Name
    LIMIT 1
   ) AS tmp2
  )
 );
```

The result looks like this:

```
The country South Africa has 4 official languages:
Afrikaans,English,Xhosa,Zulu
```

Chapter 14. Views

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:

A view can be updatable but not insertable. Explain why this is so.

Question 2:

The following view `v1` is created based on the `City` and `Country` tables of the `world` database. Is it updatable, or even insertable?

```
CREATE VIEW v1 (CityName, CountryName)
AS SELECT City.Name, Country.Name FROM City, Country
WHERE City.CountryCode = Country.Code
AND City.CountryCode = 'DEU';
```

Question 3:

Create a view based on the `Country` table that displays total surface area of each continent, and the average surface of the countries on each continent. Is that view updatable?

Question 4:

Consider the following view:

```
mysql> CREATE VIEW vSurface
    -> (Name, ContinentSurface, CountryAvgSurface)
    -> AS SELECT Continent, SUM(SurfaceArea),
    -> AVG(SurfaceArea)
    -> FROM Country GROUP BY Continent;
```

That view could also be created as a summary table (either a temporary or a permanent one), like this:

```
mysql> CREATE TABLE stSurface
    -> AS SELECT Continent AS Name,
    -> SUM(SurfaceArea) AS ContinentSurface,
    -> AVG(SurfaceArea) AS CountryAvgSurface
    -> FROM Country GROUP BY Continent;
```

What is the advantage of using a summary table over using a view, and what is the disadvantage?

Question 5:

What happens if you try to specify `MERGE` with a view for which `MERGE` cannot be used?

Question 6:

With `CREATE OR REPLACE VIEW`, which of the following objects are replaced if they have the same name as the new view?

a. View

b. Table

c. Trigger

d. Stored routine

Question 7:

Which of the following methods for providing explicit names for the columns in a view work?

a. Include a column list

b. Provide column aliases in the view `SELECT` statement

c. Rename the columns when you select from the view

Question 8:

Which of these conditions make a view non-updatable?

a. Use of `ALGORITHM=TEMPTABLE` in the view definition

b. Use of `ALGORITHM=MERGE` in the view definition

c. Use of aggregate functions in the view definition

d. Use of `GROUP BY` or `HAVING` clauses in the view definition

e. Use of expressions like `col = col + 1` in the view definition

Question 9:

Is `WITH CHECK OPTION` allowed only for updatable views, only for non-updatable views, or both?

Question 10:

Why are `ALGORITHM = TEMPTABLE` and `WITH CHECK OPTION` mutually exclusive?

Answers to Exercises

Answer 1:

An updatable view is not insertable if any view columns consist of expressions such as `col+1` rather than simple table column references, or if any columns present in the base table and not named in the view or the `INSERT` statement have no default value.

Answer 2:

View `v1` is updatable as long as only one of its base tables is modified in an `UPDATE` statement. This is why the first two `UPDATE` statements succeed while the third one fails:

```
mysql> UPDATE v1 SET CityName = 'Werbelina'
```

```
    -> WHERE CityName = 'Berlin';
Rows matched: 239  Changed: 1  Warnings: 0
mysql> SELECT Name FROM City
    -> WHERE Name = 'Werbelina' OR Name = 'Berlin';
+-----------+
| Name      |
+-----------+
| Werbelina |
+-----------+
mysql> UPDATE v1 SET CountryName = 'Deutschland'
    -> WHERE CountryName = 'Germany';
Rows matched: 1  Changed: 1  Warnings: 0
mysql> SELECT Name FROM Country
    -> WHERE Name = 'Deutschland' OR Name = 'Germany';
+-------------+
| Name        |
+-------------+
| Deutschland |
+-------------+
mysql> UPDATE v1 SET CityName = 'Berlin',
    -> CountryName = 'Germany'
    -> WHERE CityName = 'Werbelina'
    -> AND CountryName = 'Deutschland';
ERROR 1393 (HY000): Can not modify more than one base table
through a join view 'world.v1'
```

For the same reason that the last UPDATE statement failed, the view is insertable only if a single table is affected:

```
mysql> SELECT Name FROM City WHERE Name = 'Neustadt';
+----------+
| Name     |
+----------+
| Neustadt |
+----------+
mysql> INSERT INTO v1 SET CityName = 'Neustadt',
    -> CountryName = 'Deutschland';
ERROR 1393 (HY000): Can not modify more than one base table
through a join view 'world.v1'
```

Answer 3:

This view shows the total surface area of each continent, and the average surface area of the countries on each continent. The view is not updatable because it uses aggregate functions and GROUP BY.

```
mysql> CREATE VIEW vSurface
    -> (Name, ContinentSurface, CountryAvgSurface)
    -> AS SELECT Continent, SUM(SurfaceArea), AVG(SurfaceArea)
    -> FROM Country GROUP BY Continent;
mysql> SELECT * FROM vSurface;
+---------------+------------------+-------------------+
| Name          | ContinentSurface | CountryAvgSurface |
+---------------+------------------+-------------------+
| Asia          |      31881005.00 |     625117.745098 |
| Europe        |      23049133.90 |     501068.128261 |
| North America |      24214470.00 |     654445.135135 |
| Africa        |      30250377.00 |     521558.224138 |
| Oceania       |       8564294.00 |     305867.642857 |
| Antarctica    |      13132101.00 |    2626420.200000 |
| South America |      17864926.00 |    1276066.142857 |
+---------------+------------------+-------------------+
```

Answer 4:

The advantage of using a summary table over using a view is that a summary table is often faster. The reason for this is that its result set is created only once (on table creation time), whereas the view's result set has to be calculated each time the view is selected. The disadvantage of a summary table is that its content is static and might become stale (the base table or tables might have been changed after creation of the summary table), whereas a view always provides the most recent data.

Answer 5:

MySQL issues a warning and resets the algorithm to `UNDEFINED`:

```
mysql> CREATE ALGORITHM=MERGE VIEW vSurfaceMERGE
    -> (Name, ContinentSurface, CountryAvgSurface)
    -> AS SELECT Continent,
    -> SUM(SurfaceArea), AVG(SurfaceArea)
    -> FROM Country GROUP BY Continent;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1354
Message: View merge algorithm can't be used here for now
         (assumed undefined algorithm)
1 row in set (0.03 sec)

mysql> SHOW CREATE VIEW vSurfaceMERGE\G
*************************** 1. row ***************************
       View: vSurfaceMERGE
Create View: CREATE ALGORITHM=UNDEFINED VIEW `world`.`vSurfaceMERGE`
AS select `world`.`Country`.`Continent` AS `Name`,
sum(`world`.`Country`.`SurfaceArea`) AS `ContinentSurface`,
avg(`world`.`Country`.`SurfaceArea`) AS `CountryAvgSurface`
from `world`.`Country` group by `world`.`Country`.`Continent`
```

Answer 6:

Only a view with the same name would be replaced.

Answer 7:

a. Works: Include a column list

b. Works: Provide column aliases in the view `SELECT` statement

c. Does not work: Rename the columns when you select from the view

Answer 8:

The following conditions make a view updatable or not:

a. Use of `ALGORITHM=TEMPTABLE` in the view definition: This makes a view non-updatable because updates would affect the temporary table rather than the base table.

b. Use of `ALGORITHM=MERGE` in the view definition: This does not affect the updatability of a view.

c. Use of aggregate functions in the view definition: These functions makes a view non-updatable.

d. Use of `GROUP BY` or `HAVING` clauses in the view definition: These clauses make a view non-updatable.

e. Use of expressions such as `col = col + 1` in the view definition: Use of expressions does not necessarily make a view non-updatable. The updatability depends on which columns are named. The following example demonstrates this:

```
mysql> CREATE VIEW vCountryPopCalc
    -> AS SELECT Name, Population * 1.05 AS PopNew
    -> FROM Country;
mysql> UPDATE vCountryPopCalc SET PopNew = 1000000000
    -> WHERE Name = 'United States';
ERROR 1348 (HY000): Column 'PopNew' is not updatable
mysql> UPDATE vCountryPopCalc SET Name = 'Vereinigte Staaten'
    -> WHERE Name = 'United States';
Rows matched: 1  Changed: 1  Warnings: 0
mysql> SELECT Name FROM Country
    -> WHERE Name = 'United States'
    -> OR Name = 'Vereinigte Staaten';
+--------------------+
| Name               |
+--------------------+
| Vereinigte Staaten |
+--------------------+
```

Answer 9:

`WITH CHECK OPTION` is allowed for updatable views only.

Answer 10:

Because `TEMPTABLE` makes a view non-updatable, while `WITH CHECK OPTION` requires the view to be updatable.

Chapter 15. Importing and Exporting 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:

Assume that you have a text file containing tab-separated data that you want to load into a table named `loadtest` that has the following structure:

```
mysql> DESCRIBE loadtest;
+---------+---------+------+-----+---------+-------+
| Field   | Type    | Null | Key | Default | Extra |
+---------+---------+------+-----+---------+-------+
| number1 | int(11) | YES  |     | NULL    |       |
| char1   | char(1) | YES  |     | NULL    |       |
| date1   | date    | YES  |     | NULL    |       |
+---------+---------+------+-----+---------+-------+
```

One line of the file looks like this, where whitespace between values represents tab characters:

```
NULL NULL NULL
```

If you use `LOAD DATA INFILE` to load the file, what values will be created in the table from the values in this line? What values should the line contain if you actually want a row of `NULL` values to be created in the table?

Question 2:

Trying to load data into a table you get this error:

```
mysql> LOAD DATA LOCAL INFILE
    -> 'C:\Dokumente und Einstellungen\All Users\data for t.txt'
    -> INTO TABLE t;
ERROR 2 (HY000): File 'C:Dokumente und EinstellungenAll Usersdata for t.txt'
not found (Errcode: 2)
```

What would the correct statement look like?

Question 3:

Consider the structure of the `Country` table:

```
mysql> SHOW CREATE TABLE Country\G
*************************** 1. row ***************************
       Table: Country
Create Table: CREATE TABLE `Country` (
  `Code` char(3) NOT NULL default '',
  `Name` char(52) NOT NULL default '',
  `Continent` enum('Asia','Europe','North America','Africa','Oceania',
  'Antarctica','South America') NOT NULL default 'Asia',
  `Region` char(26) NOT NULL default '',
  `SurfaceArea` float(10,2) NOT NULL default '0.00',
  `IndepYear` smallint(6) default NULL,
  `Population` int(11) NOT NULL default '0',
  `LifeExpectancy` float(3,1) default NULL,
  `GNP` float(10,2) default NULL,
  `GNPOld` float(10,2) default NULL,
  `LocalName` char(45) NOT NULL default '',
  `GovernmentForm` char(45) NOT NULL default '',
  `HeadOfState` char(60) default NULL,
  `Capital` int(11) default NULL,
  `Code2` char(2) NOT NULL default '',
  PRIMARY KEY  (`Code`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Assume that you want to populate that table with additional data that are stored in a file `more_countries.dat`. That file contains country codes and country names only.

a. What statement would you use to load the contents of that file into the `Country` table?

b. Considering that some columns are declared `NOT NULL`, will the statement succeed when it only loads the `Code` and `Name` columns?

Question 4:

Assume that you want to load data into a table that are in a file that is located on the server host. What privileges are necessary so that this succeeds?

Question 5:

Is there a MySQL client program that could be used for data import instead of the `LOAD DATA INFILE` statement? If so, what is it called?

Question 6:

The simplest form of the `LOAD DATA INFILE` statement looks like this:

```
LOAD DATA INFILE 'file_name' INTO TABLE table_name;
```

MySQL assumes a number of defaults for options that are omitted from the statement. What will MySQL use as the default separators for fields and lines?

Question 7:

The simplest form of the `LOAD DATA INFILE` statement looks like this:

```
LOAD DATA INFILE 'file_name' INTO TABLE table_name;
```

MySQL assumes a number of defaults for options that are omitted from the statement. What will MySQL use as the default behavior when there are lines in *`file_name`* that would duplicate unique-valued key entries? Does it make a difference if *`file_name`* is located on the client host rather than on the server host?

Question 8:

After loading data into a table, you see the following result from the `LOAD DATA INFILE` statement. What does it mean?

```
Query OK, 18 rows affected (0.00 sec)
Records: 9  Deleted: 9  Skipped: 0  Warnings: 2
```

Question 9:

After loading data into a table, you see the following result from the `LOAD DATA INFILE` statement. What does it mean?

```
Query OK, 0 rows affected (0.00 sec)
Records: 9  Deleted: 0  Skipped: 9  Warnings: 2
```

Question 10:

Which client programs can be used to populate the table `buildings` with information stored in the data file `/tmp/buildings.txt`? Assume that the contents of the file are in the default format expected by the program.

Question 11:

Assume that you've issued the following command to back up all table data in the `project` database:

```
shell> mysqldump --no-create-info --tab=/tmp
           --lines-terminated-by="\r\n" project
```

Could you use a similar command to back up all table data for multiple databases?

Question 12:

Assume that you've issued the following command to back up the data contained in the `test` database's `tbl1` and `tbl2` tables:

```
shell> mysqldump --tab=/backup --fields-terminated-by=,
           --lines-terminated-by="\r\n" test tbl1 tbl2
```

Can you use a similar, single, command to back up the data for any `tbl1` and `tbl2` tables in all databases on the server? If so, how?

Question 13:

Consider you want to load data from a file `even_more_countries.dat` into the `Country` table. The contents of that file look like this, where whitespace between values represents tab characters:

```
Unusable Data   Code    Name      PopCity  PopRural
Unusable_data   UDM     Udmurtia  235050   1430580
Unusable_data   BAV     Bavaria   3150025  2590245
```

You want to skip the column that contains unusable data, and you want to combine the values of the `PopCity` and `PopRural` columns into a single value that should go to the `Population` column of the table. Also, the first line of the file should be skipped. What's the correct statement to accomplish this?

Question 14:

The simplest form of the `LOAD DATA INFILE` statement looks like this:

```
LOAD DATA INFILE 'file_name' INTO TABLE table_name;
```

MySQL assumes a number of defaults for options that are omitted from the statement. Assuming that the filename has a single component, what will MySQL use as the default location for `file_name`?

Question 15:

The simplest form of the `LOAD DATA INFILE` statement looks like this:

```
LOAD DATA INFILE 'file_name' INTO TABLE table_name;
```

MySQL assumes a number of defaults for options that are omitted from the statement. What will MySQL use as the default table columns to load?

Question 16:

The simplest form of the `LOAD DATA INFILE` statement looks like this:

```
LOAD DATA INFILE 'file_name' INTO TABLE table_name;
```

MySQL assumes a number of defaults for options that are omitted from the statement. What will MySQL use as the default number of lines of `file_name` to skip?

Question 17:

Consider the following table data:

```
mysql> SELECT * FROM personnel;
+-----+------+-------+
| pid | unit | grade |
+-----+------+-------+
|  46 |   23 |    42 |
|  47 |   23 |    53 |
|  48 |   23 |   123 |
|  49 |   23 |   142 |
|  50 |   23 |   198 |
|  60 |   23 |   248 |
|  70 |   23 |   255 |
|  80 |   42 |   110 |
|  90 |   42 |   255 |
+-----+------+-------+
```

Assume that you want to export the `pid` and `unit` columns for the five highest grades to a file that has Windows-like line terminators (`\r\n`) and that looks like this:

```
"70";"23"
"90";"42"
"60";"23"
"50";"23"
"49";"23"
```

What statement would you issue?

Answers to Exercises

Answer 1:

The actual result of the `LOAD DATA` statement will look like this:

```
mysql> SELECT * FROM loadtest;
+---------+-------+------------+
| number1 | char1 | date1      |
+---------+-------+------------+
|       0 | N     | 0000-00-00 |
+---------+-------+------------+
```

The import file values are interpreted as the string `'NULL'`, not as `NULL` values. For non-string columns, these strings are converted first before they are inserted. This conversion results in values of `0` for integers and `'0000-00-00'` for dates. For the string column, the import string is clipped to the length of the column, resulting in a value of `'N'`. If you wanted to actually insert `NULL` values, the import file would have to contain the `\N` sequence that `LOAD DATA INFILE` interprets as representing `NULL`.

Answer 2:

In the correct statement, you would either use forward slashes or double backslashes in the path. You cannot use single backslashes because these are the escape character in MySQL. You don't need to treat space characters in the path in any special way.

```
mysql> LOAD DATA LOCAL INFILE
    -> 'C:\\Dokumente und Einstellungen\\All Users\\data for t.txt'
    -> INTO TABLE t;
```

Answer 3:

a. To load two columns of the table, use this statement:

   ```
   mysql> LOAD DATA LOCAL INFILE 'more_countries.dat'
       -> INTO TABLE Country (Code, Name);
   Query OK, 2 rows affected (0.01 sec)
   Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
   ```

   That statement assumes that the file `more_countries.dat` is located in the directory where `mysql` was invoked, and it further assumes that the file contains two new countries.

b. As just shown, the statement succeeds although it doesn't populate some of the columns that are declared `NOT NULL`. Those columns, however, all have default values (as can be seen in the `SHOW CREATE TABLE` statement).

Answer 4:

The server must have read access to that file, and you need the `FILE` privilege.

Answer 5:

Instead of using the SQL statement `LOAD DATA INFILE`, you could use the MySQL client program named `mysqlimport`. It takes arguments that correspond to various clauses of the `LOAD DATA INFILE` statement.

Answer 6:

The default separator for fields is the tab character (`\t`). The default separator for lines is the newline character (`\n`).

Answer 7:

If the data file is located on the server host, the default behavior in case of duplicate unique-valued key entries is to return an error. MySQL does not insert the duplicating line and skips the rest of the lines. If the data file is located on the client host, the default behavior changes. In this case, the default is to skip lines that would duplicate records; that is, MySQL operates as if the keyword `IGNORE` had been specified.

Answer 8:

The message provides four pieces of information.

- Nine lines were read from the data file (`Records: 9`).
- The keyword `REPLACE` was used in the `LOAD DATA INFILE` statement (because the `Deleted` value is non-zero).
- Each one of the nine lines in the data file duplicates an existing unique-valued key entry. Thus, each line causes a row to be deleted and replaces it (`Deleted: 9`, and `18 rows affected`).
- The warnings indicate that two problems were found in the import file (`Warnings: 2`).

Answer 9:

The message provides four pieces of information.

- Nine lines were read from the data file (`Records: 9`).
- The keyword `IGNORE` was used in the `LOAD DATA INFILE` statement (because the `Skipped` value is non-zero).
- Each one of the nine lines in the data file duplicates an existing unique-valued key entry. Thus, each line was skipped and not inserted (`Skipped: 9`, and `0 rows affected`).
- The warnings indicate that two problems were found in the import file (`Warnings: 2`).

Answer 10:

To populate the table from data stored in a text file, use `mysqlimport`. You must specify the database name. The table name is determined implicitly from the final component of the filename (excluding any filename extension):

```
shell> mysqlimport landmarks /tmp/buildings.txt
```

The same can be accomplished from with the `mysql` client program using the `LOAD DATA INFILE` statement. Here, you must specify the table name explicitly:

```
mysql> LOAD DATA INFILE '/tmp/buildings.txt' INTO landmarks.buildings;
```

Answer 11:

No, this is not possible. The `--tab` option can be used to dump only a single database. To back up multiple databases using this method, it's necessary to issue multiple `mysqldump` commands.

Answer 12:

`mysqldump` can back up multiple databases (or even all databases), but not when invoked with the `--tab` option. This option causes the output files for all tables to be written into a single directory. That would make it impossible to tell which database each table came from.

Answer 13:

You need to skip the first column of the `even_more_countries.dat` file, add the values of `PopCity` and `PopRural` and populate the `Population` columns of the table with the combined values, and you have to skip the first line:

```
mysql> LOAD DATA LOCAL INFILE 'even_more_countries.dat'
    -> INTO TABLE Country
    -> IGNORE 1 LINES
    -> (@unused, Code, Name, @PopCity, @PopRural)
    -> SET Population = @PopCity + @PopRural;
Query OK, 2 rows affected (0.02 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT Code, Name, Population FROM Country
    -> WHERE Code = 'UDM' OR Code = 'BAV';
+------+----------+------------+
| Code | Name     | Population |
+------+----------+------------+
| BAV  | Bavaria  |    5740270 |
| UDM  | Udmurtia |    1665630 |
+------+----------+------------+
2 rows in set (0.00 sec)
```

That statement assumes that the file `even_more_countries.dat` is located in the directory where `mysql` was invoked.

Answer 14:

The default location of *`file_name`* is the directory of the default database.

Answer 15:

All table columns are loaded by default. This means that *`file_name`* should specify a value for every table column. If lines don't contain values for all columns, each missing column is set to its default value.

Answer 16:

No lines of *`file_name`* are skipped by default.

Answer 17:

```
mysql> SELECT pid, unit
    ->  INTO OUTFILE 'highpers.dat'
    ->  FIELDS TERMINATED BY ';'
    ->         ENCLOSED BY '"'
    ->  LINES TERMINATED BY '\r\n'
    ->  FROM personnel
    ->  ORDER BY grade DESC
    ->  LIMIT 5
    -> ;
```

Chapter 16. User Variables

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 assignments will succeed?

1. `SET @a1 = 'test1'`
2. `SET @a2 := 'test2'`
3. `SET @a3 == 'test3'`
4. `SELECT @a4 = 'test4'`
5. `SELECT @a5 := 'test5'`

Question 2:

What result does the following statement yield?

```
SELECT IFNULL(@a, '@a is NULL');
```

Question 3:

1. Assume that you've issued the following statements. What's the result of the `SELECT` statement?

   ```
   mysql> SET @a = 1;
   mysql> SET @A = @a;
   mysql> SELECT @a, @A;
   ```

2. Assume that you're continuing the session by issuing these statements. What will the result of the `SELECT` statement be?

   ```
   mysql> SET @a = 2;
   mysql> SELECT @a, @A;
   ```

Question 4:

How can you store user variables so they're not lost when the session ends? How can you make them available for other threads?

Answers to Exercises

Answer 1:

In a `SET` statement, you may use either the `=` or the `:=` operator to assign a value to a user variable, so the first two assignments succeed. Trying to assign a value with `==` results in a syntax error. In a `SE-LECT` statement, you have to use the `:=` operator for assigning a value to a variable; if you use the `=` operator, the expression acts as a comparison rather than an assignment. In the fourth statement, the comparison is not true unless `@a4` happens to have the value `'test4'`.

Answer 2:

Assuming the user variable hasn't been assigned a value in a previous statement, you will get this result:

```
mysql> SELECT IFNULL(@a, '@a is NULL');
+--------------------------+
| IFNULL(@a, '@a is NULL') |
+--------------------------+
| @a is NULL               |
+--------------------------+
```

Answer 3:

User variable names are not case sensitive, so you will get the following results:

1.
   ```
   mysql> SELECT @a, @A;
   +------+------+
   | @a   | @A   |
   +------+------+
   | 1    | 1    |
   +------+------+
   ```

2.
   ```
   mysql> SELECT @a, @A;
   +------+------+
   | @a   | @A   |
   +------+------+
   | 2    | 2    |
   +------+------+
   ```

Answer 4:

Neither is possible. When the session ends, all its user variables are lost, and user variables are specific to the client connection.

Chapter 17. Prepared Statements

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 are the main reasons why using prepared statements is often more efficient than using ‘regular’ statements?

Question 2:

After you execute the following statements, how many prepared statements exist?

```
PREPARE s1 FROM 'SELECT 1';
PREPARE s2 FROM 'SELECT 2';
PREPARE s1 FROM 'SELECT (1+2';
```

Question 3:

John connects to the server and prepares a statement:

```
mysql> PREPARE s1 FROM 'SELECT Name, Population
    '> FROM Country WHERE Continent = ?';
Query OK, 0 rows affected (0.00 sec)
Statement prepared
```

Then Lydia connects to the same server and prepares a statement with the same name:

```
mysql> PREPARE s1 FROM 'SELECT NOW()';
Query OK, 0 rows affected (0.10 sec)
Statement prepared
```

What effect does this have on John's prepared statement?

Question 4:

Write a prepared statement that accepts a continent name and a population value as parameters, and uses the parameter values to select, from the `Country` table, the countries located in the given continent that have a population larger than the given population.

Use that prepared statement to determine which countries in Asia have a population of more than 100 million.

Question 5:

John connects to the server and prepares a statement:

```
mysql> PREPARE s1 FROM 'SELECT Name, Population
    '> FROM Country WHERE Continent = ?';
Query OK, 0 rows affected (0.00 sec)
Statement prepared
```

Then John disconnects, reconnects, and issues the following statements. What is the result of the EXECUTE statement?

```
mysql> SET @c = 'South America';
mysql> EXECUTE s1 USING @c;
```

Question 6:

How do you deallocate a prepared statement? Is it necessary to do so?

Question 7:

Which kinds of statements can you prepare?

a. All SQL statements

b. SELECT statements

c. INSERT, UPDATE, REPLACE, and DELETE statements

d. SET and DO statements

e. Many SHOW statements

f. CREATE TABLE statements

Question 8:

Does a prepared statement have to include the '?' parameter marker?

Answers to Exercises

Answer 1:

Prepared statements can be more efficient if the same statement is run several times because there is less network traffic and less time is spent parsing the same statement multiple times.

Answer 2:

One prepared statement exists. After the first two PREPARE statements, two prepared statements exist (s1 and s2). The third statement causes the original s1 to be discarded because it uses the same statement name. However, it does not result in a new prepared statement because the statement contains a syntax error. Only s2 exists after all three PREPARE statements have been executed.

Answer 3:

There is no effect. Prepared statements are specific to the session in which they are created. Statements prepared by one client do not affect those prepared by other clients.

Answer 4:

The statement would be prepared like this:

```
mysql> PREPARE spop FROM 'SELECT Name, Population
    '> FROM Country
    '> WHERE Continent = ? AND Population > ?';
Query OK, 0 rows affected (0.00 sec)
Statement prepared
```

It would be executed like this:

```
mysql> SET @c = 'Asia', @p = 100000000;
Query OK, 0 rows affected (0.00 sec)

mysql> EXECUTE spop USING @c, @p;
+------------+------------+
| Name       | Population |
+------------+------------+
| Bangladesh |  129155000 |
| Indonesia  |  212107000 |
| India      | 1013662000 |
| Japan      |  126714000 |
| China      | 1277558000 |
| Pakistan   |  156483000 |
+------------+------------+
6 rows in set (0.00 sec)
```

Answer 5:

An error occurs for the EXECUTE statement because prepared statement s1 does not exist. Statements prepared during a session are discarded when the session ends and are not available to later sessions.

Answer 6:

To deallocate a prepared statement, use DEALLOCATE PREPARE:

```
DEALLOCATE PREPARE my_stmt;
```

DROP PREPARE can also be used.

It is not necessary to deallocate a prepared statement created within a given session because the server discards the statement automatically when the session ends. However, deallocating the statement explicitly does allow the server to release resources earlier that are associated with the statement.

Answer 7:

You cannot prepare all SQL statements, but the rest of the list in the question contains the preparable statements.

Answer 8:

No. It will usually make sense to use that parameter marker, but otherwise prepared statements work without them, too:

```
mysql> PREPARE my_time FROM 'SELECT NOW()';
Query OK, 0 rows affected (0.02 sec)
Statement prepared

mysql> EXECUTE my_time;
+---------------------+
| NOW()               |
+---------------------+
| 2005-05-19 23:35:30 |
+---------------------+
```

Chapter 18. Stored Procedures and Functions

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:

Name some benefits of using stored routines.

Question 2:

Assume that you look at a stored routine that does not return a value and is invoked using a `CALL` statement. Is that routine a stored procedure or a stored function?

Question 3:

Assume that `test` is your default database. How can you create a `world_record_count()` procedure in the `world` database without changing the default database to `world` beforehand?

Question 4:

Assume that you want a stored procedure to run with the privileges of the person who calls it. How do you accomplish this?

Question 5:

Which of the following statements are true for the execution of stored routines?

1. The routine's environment is set so that the database that it belongs to becomes its default database for the duration of its execution.
2. The `sql_mode` system variable value in effect when the routine executes is the value that was current when it was defined.
3. The privileges of the routine are set to the privileges of its definer.

Question 6:

Which of the following block definitions are correct?

1.
   ```
   BEGIN my_block:
    (compound statements)
   END my_block;
   ```
2.
   ```
   my_block: BEGIN
    (compound statements)
   END my_block;
   ```

3.
```
myblock:
BEGIN my_block;
 (compound statements)
END my_block;
```

Question 7:

Are these statements legal if both appear in the same block?

```
DECLARE i INT;
DECLARE i INT;
```

Question 8:

Are these statements legal if both appear in the same block?

```
DECLARE c CHAR(10);
DECLARE c CONDITION FOR SQLSTATE '02000';
```

Question 9:

Does MySQL support use of cursors for updating tables?

Question 10:

If you want to catch the occurrence of a condition so that you can ignore it, how do you declare the handler?

Question 11:

Which of the `FOR`, `LOOP`, `REPEAT`, and `WHILE` loop constructs does MySQL support?

Question 12:

If the body of a loop must execute at least once, which is more appropriate, `REPEAT` or `WHILE`?

Question 13:

If the body of a loop need not necessarily execute even once, which is more appropriate, `REPEAT` or `WHILE`?

Question 14:

Suppose that an `EXIT` handler is declared in an outer block and the condition handled by the handler occurs within an inner block. Does control transfer to end the of the outer block or the inner block?

Question 15:

How can you retrieve the name and definition of the `world_record_count()` procedure in the `world` database?

Question 16:

Which properties of a stored routine can you change with the `ALTER PROCEDURE` and `ALTER FUNCTION` statements?

- The `SQL SECURITY` characteristic
- The name of the routine
- The routine definition
- The `COMMENT` characteristic
- All characteristics of the routine

Question 17:

The following two procedures are defined and then `loop_test()` is executed:

```
mysql> delimiter //
mysql> CREATE PROCEDURE incrementor (OUT i INT)
    -> BEGIN
    ->   REPEAT
    ->     SET i = i + 1;
    ->   UNTIL i > 9
    ->   END REPEAT;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE PROCEDURE loop_test ()
    -> BEGIN
    ->   DECLARE value INT default 0;
    ->   CALL incrementor(value);
    ->   SELECT value; /* What value is shown here? */
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL loop_test();
```

What is the result of the `SELECT value` statement?

a. 9

b. 10

c. `NULL`

d. The code enters an infinite loop and never reaches `SELECT value`.

e. `loop_test()` fails to call `incrementor()` because the declaration of this procedure failed; a space was needed in the `UNTIL` condition (`i > 9`).

Answers to Exercises

Answer 1:

- Because you can use compound statements and flow-control constructs, stored routines are more flexible regarding SQL syntax than “regular” SQL statements.

- Unlike “regular” SQL statements, a stored routine may use error handlers for exceptional conditions, thus error handling is more flexible.
- A collection of stored routines acts as a library of solutions to problems; it facilitates sharing of knowledge and experience.
- Stored routines may reduce the complexity of application code, making that code easier to read and maintain.
- Stored routines may make applications more consistent regarding how they perform particular operations.
- Stored routines reduce the need to maintain the same or similar code in multiple applications.

Answer 2:

It's a stored procedure. Functions always return a value, and they're invoked within an expression, rather than in a `CALL` statement.

Answer 3:

To do this, you need to qualify the procedure with the database name:

```
CREATE PROCEDURE world.world_record_count ...
```

Answer 4:

While creating the stored procedure, you'll have to use this characteristic in the definition:

```
SQL SECURITY INVOKER
```

If that characteristic is missing, the default is to run with the privileges of the person who defined the procedure.

Answer 5:

The first two statements are true. The third one is true only if the routine was defined either with the default value for the `SQL SECURITY` characteristic, or explicitly with the value of `DEFINER`.

Answer 6:

Only the second block definition is correct. The first block has the beginning label inside of the block. The third block has an extra label inside of the block.

Answer 7:

No. Variables in a block must have different names.

Answer 8:

Yes. Items of different types can have the same name. However, this might make the routine more difficult to understand.

Answer 9:

No. MySQL supports the use of cursors only for reading tables, not for updating them.

Answer 10:

The handler should be a `CONTINUE` handler and its statement should be an empty block.

Answer 11:

MySQL supports all of the mentioned loop constructs with the exception of `FOR`.

Answer 12:

`REPEAT` is more appropriate, although it is possible to use `WHILE`.

Answer 13:

`WHILE` is more appropriate.

Answer 14:

Control transfers to the end of the block in which the handler is declared. In this case, that is the outer block.

Answer 15:

There are two ways to accomplish this:

1. By querying the `INFORMATION_SCHEMA` database:

   ```
   mysql> SELECT ROUTINE_NAME, ROUTINE_DEFINITION
       -> FROM INFORMATION_SCHEMA.ROUTINES
       -> WHERE ROUTINE_SCHEMA = 'world'
       ->   AND ROUTINE_NAME = 'world_record_count'
       -> \G
   *************************** 1. row ***************************
         ROUTINE_NAME: world_record_count
   ROUTINE_DEFINITION: BEGIN
     SELECT 'Country', COUNT(*) FROM Country;
     SELECT 'City', COUNT(*) FROM City;
     SELECT 'CountryLanguage', COUNT(*) FROM CountryLanguage;
   END
   ```

2. By using the `SHOW CREATE PROCEDURE` statement:

   ```
   mysql> SHOW CREATE PROCEDURE world.world_record_count\G
   *************************** 1. row ***************************
          Procedure: world_record_count
           sql_mode:
   Create Procedure: CREATE PROCEDURE `world`.`world_record_count`()
   BEGIN
     SELECT 'Country', COUNT(*) FROM Country;
     SELECT 'City', COUNT(*) FROM City;
     SELECT 'CountryLanguage', COUNT(*) FROM CountryLanguage;
   END
   ```

You cannot use `SHOW PROCEDURE STATUS` because it doesn't return the definition:

```
mysql> SHOW PROCEDURE STATUS LIKE 'world_record_count'\G
```

```
*************************** 1. row ***************************
           Db: world
         Name: world_record_count
         Type: PROCEDURE
      Definer: wuser@localhost
     Modified: 2005-06-02 02:12:13
      Created: 2005-06-02 02:12:13
Security_type: DEFINER
      Comment:
```

Answer 16:

You can change only the `SQL SECURITY` and `COMMENT` characteristics.

Answer 17:

The parameter to `incrementor()` should have been defined as `INOUT`. Instead, `OUT` sets `i` to `NULL` and the code enters an infinite loop because the `UNTIL` condition never becomes true.

Chapter 19. Triggers

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 is the maximum number of triggers you can have per table?

Question 2:

Assume that you have 239 rows in the `Country` table, and you're just about to insert ten new rows using a multiple-row `INSERT` statement. There is a `BEFORE INSERT` trigger associated with that table. How often will it be activated?

Question 3:

Whenever data are modified in the `Country` table, you want a trigger to perform multiple operations before data actually get updated. How can you achieve that given that you cannot have more than one `BEFORE UPDATE` trigger per table?

Question 4:

Can you have a trigger named `abcde` that is a `BEFORE UPDATE` trigger for the `City` table, and another trigger also named `abcde` that is a `AFTER INSERT` trigger for the `Country` table, in the same database?

Question 5:

For which kind of triggers can you use the `OLD` prefix for column names, and for which can you use `NEW`? What restrictions are imposed on the use of `OLD` and `NEW`, respectively?

Answers to Exercises

Answer 1:

You can have a maximum of six triggers per table: Two `INSERT`, two `UPDATE`, and two `DELETE` triggers, where each type of statement has a `BEFORE` and an `AFTER` trigger. You cannot have more than one trigger of the same kind; for example, you cannot have more than one `BEFORE INSERT` trigger.

Answer 2:

The `BEFORE INSERT` trigger will be activated ten times, once for each row that is being inserted.

Answer 3:

You would use a compound statement; that is, a set of statements in the trigger body that is located in a block that starts with `BEGIN` and ends with `END`.

Answer 4:

No. Trigger names must be unique within a database.

Answer 5:

You can use `OLD` for `DELETE` and `UPDATE` triggers, and `NEW` for `INSERT` and `UPDATE` triggers. `OLD` must be used in a read-only fashion, whereas `NEW` can be used for reading or for changing data.

Chapter 20. Obtaining Database Metadata

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:

Using `mysqlshow`, what command would you issue to find out which tables in the `test` database have names starting with `my`?

Question 2:

Using `mysqlshow`, how can you see the indexes of a table named `mytable` in the `test` database? Can you retrieve information about the indexes of multiple tables issuing a single command?

Question 3:

Which client programs can be used display the structure of a table named `buildings`, including any indexes it might have?

Question 4:

Consider the `SHOW DATABASES LIKE 'w%'` SQL statement. What is the equivalent `mysqlshow` command to display the same information? What is the equivalent `SELECT` statement that uses the `INFORMATION_SCHEMA` database?

Question 5:

Consider the `SHOW TABLES FROM world` SQL statement. What is the equivalent `mysqlshow` command to display the same information? What is the equivalent `SELECT` statement that uses the `INFORMATION_SCHEMA` database?

Question 6:

Consider the `SHOW TABLES FROM world LIKE 'C%'` SQL statement. What is the equivalent `mysqlshow` command to display the same information? What is the equivalent `SELECT` statement that uses the `INFORMATION_SCHEMA` database?

Question 7:

Consider the `SHOW COLUMNS FROM City FROM world` SQL statement. What is the equivalent `mysqlshow` command to display the same information? What is the equivalent `SELECT` statement that uses the `INFORMATION_SCHEMA` database?

Question 8:

Consider the `SHOW KEYS FROM City FROM world` SQL statement. What is the equivalent `mysqlshow` command to display the same information? What is the equivalent `SELECT` statement that uses the `INFORMATION_SCHEMA` database?

Question 9:

What SQL statement provides information about the available character sets and their associated default collations?

Question 10:

What SQL statement lists which character set/collation combinations can be used?

Question 11:

What `SELECT` statement provides information about the tables available in the `INFORMATION_SCHEMA` database?

Answers to Exercises

Answer 1:

To list the tables in the `test` database starting with `my`, you would issue a `mysqlshow test "my%"` or `mysqlshow test "my*"` command.

Answer 2:

The `--keys` or `-k` option instructs `mysqlshow` to display table index information in addition to column information. For the `mytable` table in the `test` database, use `mysqlshow --keys test mytable`. You can show the indexes for only one table per command.

Answer 3:

You can display the structure of the table and its indexes using the `mysqlshow` command:

```
shell> mysqlshow --keys landmarks buildings
```

The same can be accomplished from within the `mysql` client by using the `DESCRIBE` (or `SHOW COLUMNS`) and `SHOW INDEX` statements:

```
mysql> DESCRIBE landmarks.buildings;
mysql> SHOW INDEX FROM landmarks.buildings;
```

Answer 4:

You could use `mysqlshow "w*"` to accomplish the same task. Using the `INFORMATION_SCHEMA` database, you could issue this statement:

```
SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
WHERE SCHEMA_NAME LIKE 'w%';
```

Answer 5:

You could use `mysqlshow world` to accomplish the same task. Using the `INFORMATION_SCHEMA` database, you could issue this statement:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'world';
```

Answer 6:

You could use `mysqlshow world "C*"` to accomplish the same task. Using the `INFORMATION_SCHEMA` database, you could issue this statement:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'world' AND TABLE_NAME LIKE 'C%';
```

Answer 7:

You could use `mysqlshow world City` to accomplish the same task. Here is a comparison between the `SHOW COLUMNS` statement and the `SELECT` statement using the `INFORMATION_SCHEMA` database:

```
mysql> SHOW COLUMNS FROM City FROM world\G
*************************** 1. row ***************************
  Field: ID
   Type: int(11)
   Null: NO
    Key: PRI
Default: NULL
  Extra: auto_increment
*************************** 2. row ***************************
  Field: Name
   Type: char(35)
   Null: NO
    Key:
Default:
  Extra:
*************************** 3. row ***************************
  Field: CountryCode
   Type: char(3)
   Null: NO
    Key:
Default:
  Extra:
*************************** 4. row ***************************
  Field: District
   Type: char(20)
   Null: NO
    Key:
Default:
  Extra:
*************************** 5. row ***************************
  Field: Population
   Type: int(11)
   Null: NO
    Key:
Default: 0
  Extra:

mysql> SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE,
    -> COLUMN_KEY, COLUMN_DEFAULT, EXTRA
    -> FROM INFORMATION_SCHEMA.COLUMNS
    -> WHERE TABLE_SCHEMA = 'world' AND TABLE_NAME = 'City'\G
*************************** 1. row ***************************
   COLUMN_NAME: ID
     DATA_TYPE: int
   IS_NULLABLE: NO
    COLUMN_KEY: PRI
COLUMN_DEFAULT: NULL
         EXTRA: auto_increment
*************************** 2. row ***************************
   COLUMN_NAME: Name
     DATA_TYPE: char
   IS_NULLABLE: NO
    COLUMN_KEY:
COLUMN_DEFAULT:
         EXTRA:
```

```
*************************** 3. row ***************************
   COLUMN_NAME: CountryCode
     DATA_TYPE: char
   IS_NULLABLE: NO
    COLUMN_KEY:
COLUMN_DEFAULT:
         EXTRA:
*************************** 4. row ***************************
   COLUMN_NAME: District
     DATA_TYPE: char
   IS_NULLABLE: NO
    COLUMN_KEY:
COLUMN_DEFAULT:
         EXTRA:
*************************** 5. row ***************************
   COLUMN_NAME: Population
     DATA_TYPE: int
   IS_NULLABLE: NO
    COLUMN_KEY:
COLUMN_DEFAULT: 0
         EXTRA:
```

Answer 8:

You could use `mysqlshow --keys world City` to accomplish the same task. (This command will display the table columns, too, so it's not exactly equivalent to `SHOW KEYS FROM City FROM world`. There's no way to display only the table's indexes using `mysqlshow`.)

There is no `SELECT` statement using the `INFORMATION_SCHEMA` database that would show exactly the same information as the `SHOW KEYS` statement. Here is what those two statements yield:

```
mysql> SHOW KEYS FROM City FROM world\G
*************************** 1. row ***************************
       Table: City
  Non_unique: 0
    Key_name: PRIMARY
Seq_in_index: 1
 Column_name: ID
   Collation: A
 Cardinality: 0
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:

mysql> SELECT * FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
    -> WHERE TABLE_SCHEMA = 'world' AND TABLE_NAME = 'City'\G
*************************** 1. row ***************************
           CONSTRAINT_CATALOG: NULL
            CONSTRAINT_SCHEMA: world
              CONSTRAINT_NAME: PRIMARY
                TABLE_CATALOG: NULL
                 TABLE_SCHEMA: world
                   TABLE_NAME: City
                  COLUMN_NAME: ID
             ORDINAL_POSITION: 1
POSITION_IN_UNIQUE_CONSTRAINT: NULL
      REFERENCED_TABLE_SCHEMA: NULL
        REFERENCED_TABLE_NAME: NULL
       REFERENCED_COLUMN_NAME: NULL
```

Answer 9:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS;
```

Answer 10:

```
SELECT * FROM INFORMATION_SCHEMA.COLLATION_CHARACTER_SET_APPLICABILITY;
```

Answer 11:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'INFORMATION_SCHEMA';
```

Chapter 21. Debugging MySQL Applications

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:

How could you find out what an error number labeled `errno` means (such as 13 in the following error message)? What kind of error is that?

```
Can't find file: './mysql/host.frm' (errno: 13)
```

Question 2:

MySQL has three levels of errors. What are they? What's the SQL statement to display messages at all three levels?

Question 3:

Which of the following statements are true?

a. `SHOW ERRORS` displays information about MySQL errors, MySQL warnings, and operating system-related errors.

b. `SHOW WARNINGS` displays warnings and notes, but not errors.

c. `SHOW ERRORS` lists all errors that have occurred since the client session was started.

d. The `perror` program gives the same kind of information as the `SHOW NOTES` SQL statement.

e. `SHOW COUNT(*) WARNINGS` shows how many errors, warnings, and notes have occurred for the previous SQL statement that could cause errors, warnings, or notes.

Question 4:

Which of the following statements is true?

a. `SHOW WARNINGS` displays information about errors, warnings, and notes that have occurred since the last time `SHOW WARNINGS` was issued

b. The output of `SHOW WARNINGS` can be limited to particular errors, warnings, and notes, like this:

   ```
   SHOW WARNINGS LIMIT 0, 1;
   ```

c. The output of `SHOW WARNINGS` can be limited to particular errors, warnings, and notes, like this:

   ```
   SHOW WARNINGS LIKE '1264';
   ```

d. The generation of errors and warnings can be suppressed by issuing this SQL statement:

```
SET sql_errors = false, sql_warnings = false;
```

e. The generation of notes can be suppressed by issuing this SQL statement:

```
SET sql_notes = 0;
```

Question 5:

Why is there a `SHOW ERRORS` statement if `SHOW WARNINGS` displays errors, too?

Answers to Exercises

Answer 1:

You could use the `perror` utility to find an error message for an error number, like this:

```
c:\mysql\bin>perror 13
Error code  13:  Permission denied
```

This is an operating system error.

Answer 2:

The MySQL error levels are:

- Error
- Warning
- Note

Messages at all three levels can be displayed with `SHOW WARNINGS`.

Answer 3:

a. False. `SHOW ERRORS` displays information about MySQL errors and MySQL warnings, but not about operating system-related errors.

b. False. `SHOW WARNINGS` displays warnings and notes, as well as errors.

c. False. `SHOW ERRORS` lists all errors that have occurred for the previous SQL statement that could cause errors.

d. False. The `perror` program gives information about operating system-related errors. There is no `SHOW NOTES` SQL statement.

e. True. `SHOW COUNT(*) WARNINGS` shows how many errors, warnings, and notes have occurred for the previous SQL statement that could cause errors, warnings, or notes.

Answer 4:

a. False. `SHOW WARNINGS` displays information about errors, warnings, and notes that have occurred for the previous SQL statement that could cause errors, warnings, or notes.

b. True. The output of `SHOW WARNINGS` can be limited to particular errors, warnings, and notes, like this:

   ```
   SHOW WARNINGS LIMIT 0, 1;
   ```

   In the preceding example, only the first error, warning, or note would be displayed.

c. False. The output of `SHOW WARNINGS` cannot be limited to particular errors, warnings, and notes.

d. False. The generation of errors and warnings cannot be suppressed.

e. True. The generation of notes can be suppressed by issuing this SQL statement:

   ```
   SET sql_notes = 0;
   ```

Answer 5:

`SHOW ERRORS` tends to produce less output than `SHOW WARNINGS` and thus makes it easier to focus on serious problems.

Chapter 22. Basic Optimizations

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 table with two indexes:

```
mysql> DESCRIBE fastindex;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| i1    | char(10) | NO   | MUL |         |       |
| i2    | char(10) | YES  | MUL | NULL    |       |
+-------+----------+------+-----+---------+-------+
```

With no other facts given, which of the following queries would you expect to run faster?

```
SELECT i1 FROM fastindex WHERE i1 LIKE 'mid%';
```

```
SELECT i2 FROM fastindex WHERE i2 LIKE 'mid%';
```

Question 2:

Consider the following table with indexes:

```
mysql> SHOW CREATE TABLE fastindex\G
*************************** 1. row ***************************
       Table: fastindex
Create Table: CREATE TABLE `fastindex` (
  `i1` char(10) NOT NULL default '',
  `i2` char(10) NOT NULL default '',
  KEY `i1` (`i1`(3)),
  KEY `i2` (`i2`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

With no other facts given, which of the following queries would you expect to run faster?

```
SELECT i1 FROM fastindex WHERE i1 LIKE 'mid%';
```

```
SELECT i2 FROM fastindex WHERE i2 LIKE 'mid%';
```

Question 3:

Under what circumstances can adding indexes to a table make table operations slower?

Question 4:

Indexing improves performance of `SELECT` queries only; it deteriorates performance of queries that change data. Is this true?

Question 5:

“For my purposes, it doesn't matter whether a query returns a result within one second or one minute, so I don't care about indexing.” Do you object to anything about that statement?

Question 6:

What are the main reasons to *not* index table columns?

Question 7:

Here's an excerpt of the `Country` table definition:

```
mysql> DESCRIBE Country\G
...
*************************** 3. row ***************************
  Field: Continent
   Type: enum('Asia','Europe','North America','Africa','Oceania',
        'Antarctica','South America')
   Null: NO
    Key:
Default: Asia
  Extra:
*************************** 4. row ***************************
...
```

If the `Continent` column had a `PRIMARY` key on it, how many rows could it have? Would the situation be different if it had a `UNIQUE` key on it?

Question 8:

Why is index processing better with short index values, rather than with long values?

Question 9:

What do you have to consider when making index values as short as possible?

Question 10:

What's the statement for creating a five character-long index on the `Name` column of the `City` table?

Question 11:

What are the main reasons why the use of the `LIMIT` clause can help improve the performance of queries?

Question 12:

What could you do to speed up data insertion operations?

Question 13:

Consider the following table:

```
mysql> DESCRIBE enumtest;
+-------+--------------------------------+------+-----+---------+-------+
| Field | Type                           | Null | Key | Default | Extra |
+-------+--------------------------------+------+-----+---------+-------+
| col   | enum('first','second','third') | NO   | PRI | first   |       |
+-------+--------------------------------+------+-----+---------+-------+
mysql> SELECT * FROM enumtest;
```

```
Empty set
```

Will the following statement fail or will it insert rows? What will the contents of the `enumtest` table be after executing the statement?

```
mysql> INSERT INTO enumtest VALUES
    -> ('first'),('second'),('third'),('false'),('fourth');
```

Answers to Exercises

Answer 1:

A column or index that can contain `NULL` values cannot be processed as fast as one that cannot contain `NULL`. `i1` and `i2` are identical except that `i1` cannot contain `NULL` values, so `i1` should be faster to process. Therefore, this query should be faster:

```
SELECT i1 FROM fastindex WHERE i1 LIKE 'mid%';
```

Answer 2:

```
SELECT i1 FROM fastindex WHERE i1 LIKE 'mid%';
```

would probably perform faster because `i1` is indexed with only the first three characters as subpart of that index. MySQL can look up that index faster because it contains only up to 3-character rows, as compared to the second index that could contain up to 10-character rows.

Answer 3:

Insert, delete, and update operations will become slower when the table has indexes, because those operations require the indexes to be updated, too.

Answer 4:

Indexing cannot only help speed up `SELECT` queries, but it can also improve `UPDATE` and `DELETE` statements. This is because indexing can help the server find the rows more quickly that should be updated or deleted. On the other hand, indexes will slow down `UPDATE` and `DELETE` statements because not only the original data have to be updated but also the indexes.

Answer 5:

Even if the speed of your own queries doesn't matter, indexes that help queries run faster reduce the time tables are locked and reduce the use of machine resources. This improves performance of queries for other clients.

Answer 6:

If you never refer to a column in comparisons (`WHERE`, `ORDER BY`, `GROUP BY`) there's no need to index that column. Furthermore, if a column contains only few distinct values, an index will likely not speed up queries.

Answer 7:

If the `ENUM` column `Continent` had a `PRIMARY` key on it, it could only contain 8 rows (the seven distinct continent names plus the empty string element that is inserted when an error occurs). The column is declared `NOT NULL`, so the situation isn't any different when a `UNIQUE` key is used. (If it wasn't, then the column could contain an arbitrary number of rows that would mostly be `NULL` values.)

Answer 8:

When short index values are retrieved, less information has to be read. Furthermore, they take less time to compare than long values, and more short index values fit into the key cache than do long values.

Answer 9:

The prefix values of an index should have about the same amount of uniqueness as the original values.

Answer 10:

You can use either of the following statements to create the index:

```
mysql> CREATE INDEX Name ON City(Name(5));
mysql> ALTER TABLE City ADD INDEX(Name(5));
```

Answer 11:

When using `LIMIT`, the server needs to return less information to the client. Another reason why `LIMIT` may speed up queries is that some row sorts terminate faster when combined with that clause.

Answer 12:

You could use multiple-row inserts, rather than single-row inserts. `LOAD DATA INFILE` will run even faster than any `INSERT` statement that inserts the same amount of rows. For `InnoDB` tables, you could group inserts within a transaction so that `InnoDB` will flush changes only when the transaction ends, rather than after every single `INSERT` statement. If you're planning to replace rows using `DELETE` and `INSERT`, you could as well use the MySQL extension `REPLACE` that runs faster.

Answer 13:

Table `enumtest` has a primary key on its only column `col`. Therefore, there can be only unique values in that column. Because of the `ENUM` data type, this means that there can be only four different values in the column (the three enumeration members and the empty string that is used for invalid values). `false` is an invalid value, so it is converted to `''` (the empty string). The last value (`fourth`) is not in the `ENUM` list, either, so it too is converted to the error value `''`. The primary key, however, prevents that same value from being stored again, which leads to a duplicate-key error:

```
mysql> INSERT INTO enumtest VALUES
    -> ('first'),('second'),('third'),('false'),('fourth');
ERROR 1062 (23000): Duplicate entry '' for key 1
```

The result of this statement depends on the storage engine. For a multiple-row `INSERT` statement into a `MyISAM` table, rows are inserted as long as no error occurs. If a row fails, that row and any following rows are not inserted. As a result, the table contents are:

```
mysql> SELECT * FROM enumtest;
+--------+
| col    |
+--------+
|        |
| first  |
| second |
| third  |
+--------+
4 rows in set
```

For an `InnoDB` table, the statement rolls back, leaving the table empty.