19

How to back up and restore a database

When you work with a database that stores important data, you should have a plan for backing up that database regularly. Then, you need to execute that plan. That way, if the hard drive that stores the database fails, you can restore the database and minimize the amount of data that's lost. In this chapter, you'll learn how to back up and restore a database.

You'll also learn some skills that are related to backing up and restoring a database. For example, you'll learn how to import data from a text file and export data to a text file. In addition, you'll learn how to check and repair tables, which can save you from having to restore a table or database.

Strategies for backing up and restoring a database

One important task of a database administrator is to regularly *back up* the database. Then, if the database ever becomes corrupted, the database administrator can use the backup files to *restore* the database.

In this chapter, you'll learn the backup skills that you can use regardless of whether you have the Community Edition or the Enterprise Edition of MySQL. However, you should know that the Enterprise Edition of MySQL includes a backup tool known as MySQL Enterprise Backup that's designed to back up InnoDB tables. As a result, if you're using this edition with InnoDB tables, you may want to use the Enterprise Backup tool instead of the backup strategy presented in this chapter.

A backup strategy

Figure 19-1 starts by presenting a strategy for backing up databases. MySQL provides for two types of backups. A *full backup* includes the structure and data of a database. To create a full backup, you can use the mysqldump program as described later in this chapter. This creates a SQL script file that can be used to recreate the database. You should create a full backup at regular intervals. For a medium-size database for a website, for example, you might want to create a full backup once a week.

When you use the mysqldump program, it locks all tables so other users can't update the database while it's being backed up. As a result, it's a good practice to schedule this backup at a time of low traffic for the database.

An *incremental backup* contains changes that have been made since the last full backup. With the backup strategy shown in this figure, the binary log must be enabled as described in chapter 17 to create the incremental backups.

When you use this strategy, you shouldn't store your backup files (SQL scripts or log files) on the same hard drive where the MySQL server is running. If you do, those backup files will be lost if that hard drive fails. As a result, it's a good practice to configure the binary log so it writes to a directory on a different hard drive. This has the added benefit of balancing the load between two hard drives.

When you create a backup strategy, don't forget that the database named mysql stores information about the users and privileges for all databases on the server. As a result, you typically want to include this database in your backups.

A restore strategy

The goal of backing up your databases is to allow you to restore them to their exact state at any specified point in time. This is known as a *point-in-time recovery* (*PITR*). To restore a database to a point in time, you can use the last full backup as described in figure 19-1. Then, you can use the binary log to restore the database from the time of the last full backup to the specified point in time.

A strategy for backing up databases

1. Use the mysqldump program to regularly create full backups of each database. These backups should be stored in one or more SQL script files.
2. If it's not already enabled, enable the binary log as described in chapter 17 to create incremental backups.

A strategy for restoring databases

1. Use the mysql program to run the SQL script file for the last full backup. If necessary, you can edit the SQL script file before you execute it.
2. Use the mysqlbinlog program to execute all statements in the binary log that occurred after the last full backup.

Description

- It's important for the database administrator to regularly *back up* the database. Then, if the database becomes corrupted, the database administrator can use the backup to *restore* the database.
- A *full backup* includes the structure and content of a database. You should perform full backups according to a regular schedule.
- An *incremental backup* only contains changes that have been made to the structure and content of a database since the last full backup.
- You often want to include the database named mysql in your backups, since this database stores information about the users and privileges for all databases on the server.
- You shouldn't store your backup files (SQL scripts or log files) on the same hard drive where the MySQL server is running. If you do, those backup files will be lost along with the databases if that hard drive fails.
- A *point-in-time recovery* (PITR) allows you to restore the data up to any specified point in time.

Figure 19-1 Strategies for backing up and restoring databases

How to back up a database

Now that you understand the basic strategies for backing up and restoring a database, you're ready to learn the details of backing up a database.

How to use mysqldump to back up a database

Figure 19-2 shows how to use the mysqldump program to back up, or *dump*, one or more databases into a SQL script file. To start, you display a command prompt and then use the cd command to change to MySQL's bin directory. This is the directory that stores the various MySQL command-line programs, including all of the programs described in this chapter. Just as in chapter 17, I use front slashes in this chapter to separate the directory and file names. However, you typically use backslashes on a Windows system.

After you change the directory, you can run the mysqldump program from the command prompt to perform a backup. To back up a single database using this program, you specify the name of the database on the mysqldump command. To backup multiple databases, you use the --databases or --all-databases option.

After you specify the databases to back up, you code a > character, followed by the path to the SQL script file where you want to store the backup. When you do that, it's generally considered a best practice to add a date to the end of your script file name. In this figure, for instance, all of the examples store the script file in the murach/mysql directory, and all of the SQL file names end with "2019-01-10".

Next, you code the -u option, followed by the name of a user with privileges to back up databases. In this figure, the program connects as the root user. Finally, you code the -p option so the program prompts for a password.

In addition to the --databases and --all-databases options, you often want to use the --single-transaction, --routines, and --events options when you back up a database. That way, your backup works correctly even if you're using transactions, and it includes stored routines and events. In addition, you may want to use the --flush-logs option so the server starts a new binary log file. This makes it easier to find the binary log file or files that you need if you restore the database later.

When you attempt to execute the mysqldump program, you may get an error that indicates that access is denied. To gain access, you can usually start the command prompt as an administrator. In Windows, for example, you can right-click on the icon that you use to start the Command Prompt window and select the "Run as administrator" command.

Similarly, if a firewall is running on your computer, it may attempt to block the mysqldump program. However, if you allow the mysqldump program to access the database, it should work properly.

Before I continue, you should know that you can also use MySQL Workbench to back up one or more databases to a SQL file. To do that, you select the Data Export option from the Administration tab of the Navigator window,

How to change to MySQL's bin directory

Using Windows

```
cd /program files/mysql/mysql server 8.0/bin
```

Using macOS or Unix/Linux

```
cd /usr/local/mysql/bin
```

How to run the mysqldump program

For a single database

```
mysqldump ap > /murach/mysql/ap-2019-01-10.sql -u root -p
```

For specified databases

```
mysqldump --databases ap ex om mysql > /murach/mysql/backup-2019-01-10.sql
-u root -p
```

For all databases

```
mysqldump --all-databases > /murach/mysql/all-db-2019-01-10.sql -u root -p
```

With additional options

```
mysqldump --databases ap ex om mysql --single-transaction --routines
--events --flush-logs > /murach/mysql/backup-2019-01-10.sql -u root -p
```

Common options for the mysqldump program

Option	Description
`--databases`	Identifies the databases to be backed up.
`--all-databases`	Indicates that all databases should be backed up.
`--single-transaction`	Guarantees that the data seen by mysqldump does not change. This option should be used for databases that use InnoDB tables and transactions.
`--routines`	Include stored procedures and functions.
`--events`	Include events.
`--flush-logs`	Causes MySQL to create a new binary log file using the next number in the sequence.

Description

- You can use the mysqldump program to back up, or *dump*, one or more databases into a SQL script file.
- If you get an error that indicates that access is denied, you may need to start the command prompt as an administrator. In Windows, you can do that by right-clicking on the icon that you use to start the Command Prompt window and selecting the "Run as administrator" command.
- If a firewall is running on your computer, it may attempt to block the mysqldump program. However, if you allow the mysqldump program to access the database, it should work properly.
- On a macOS system, you typically need to code a dot and slash (./) before the name of the mysqldump program to specify that it's in the current directory (the bin directory).

Figure 19-2 How to use mysqldump to back up a database

select the databases you want to back up on the Object Selection tab, select the "Export to Self-Contained File" option, enter the path and name for the backup file, and click the Start Export button. Note, however, that earlier releases of Workbench had some bugs that sometimes prevented this feature from working correctly. As a result, I recommend using the mysqldump program.

Another advantage of using the mysqldump program is that it's easier to automate. Although the details for doing this vary depending on the operating system, the same general principles apply to all operating systems. To start, you create a script file that executes the mysqldump command. Then, you use the operating system's task scheduler to execute that script file at a specified interval. For more information on using the task scheduler with your operating system, you can search the Internet.

A SQL script file for a database backup

Figure 19-3 shows an excerpt from a SQL script file for the last database backup you saw in the previous figure. This script starts with some comments that give some general information about this backup. For example, the first line includes information about the mysqldump program. Then, the third line identifies the host (localhost) and the database (AP). Unfortunately, only the first database in the backup is listed, which is probably a bug. Finally, the fifth line identifies the version of the MySQL server.

After the comments, this script includes several lines of code that are surrounded by the /*! and */ characters. These characters identify code that's specific to MySQL. As a result, a MySQL server uses this code, but another type of database server can ignore it. If, for example, you were porting a database from a MySQL server to an Oracle server, this would prevent Oracle from trying to execute these statements.

MySQL uses these statements to set some user variables, which are identified by a single at sign (@). These variables are set to the values of some system variables, which are identified by a double at sign (@@).

Immediately after the /*! characters, this script uses a number to indicate the minimum version of MySQL that's necessary to run the SET statement that follows. For example, 40101 indicates that MySQL 4.01.01 or later can run the statement. Similarly, 40014 indicates that MySQL 4.00.14 or later can run the statement.

Note that the line of code that contains the SET NAMES statement isn't surrounded by the /*! and */ characters. This statement identifies the character set that will be used by a connection between a client and a server, in this case, utf8mb4. For this script, it identifies the character set that will be used if the script is run to restore the databases.

After setting the user variables, this script includes a CREATE DATABASE statement that creates the AP database, followed by a USE statement that selects this database. The CREATE DATABASE statement includes an IF NOT EXISTS clause that's used if you're using MySQL 3.23.12 or later. Similarly, this statement includes a DEFAULT CHARACTER SET clause that's used if you're

Part of the SQL script file for a database backup — Page 1

```
-- MySQL dump 10.13  Distrib 8.0.13, for Win64 (x86_64)
--
-- Host: localhost    Database: ap
-- ------------------------------------------------------
-- Server version       8.0.13

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
 SET NAMES utf8mb4 ;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!50606 SET @OLD_INNODB_STATS_AUTO_RECALC=@@INNODB_STATS_AUTO_RECALC */;
/*!50606 SET GLOBAL INNODB_STATS_AUTO_RECALC=OFF */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Current Database: `ap`
--

CREATE DATABASE /*!32312 IF NOT EXISTS*/ `ap` /*!40100 DEFAULT CHARACTER SET
utf8mb4 COLLATE utf8mb4_0900_ai_ci */;


USE `ap`;
```

Description

- These scripts use two dashes (--) to identify comments.
- These scripts surround code in the /*! and */ characters to indicate that the code is specific to MySQL. As a result, a MySQL server can use that code, but another type of database server can ignore it.
- These scripts surround names with backticks (`). This allows for names that include spaces.

Figure 19-3 A SQL script file for a database backup (part 1 of 2)

using MySQL 4.01.00 or later. Finally, this script surrounds the name of the database with backticks (`). Although this isn't necessary for the AP database, it's required for names that include spaces or other special characters.

If you only generate a backup for a single database, the SQL file for that backup won't include the CREATE DATABASE and USE statements. As a result, if you want to recreate the entire database, you need to add these statements to the script.

The script continues with the statements necessary to create the structure and content for the database. That includes the tables of the database, as well as the views, stored procedures, functions, triggers, and events. In part 2 of figure 19-3, you can see the statements for creating the Terms table. To start, a DROP TABLE statement drops the table if it exists. Then, a CREATE TABLE statement recreates the table.

Next, the script uses an INSERT statement to reload all the data into the table. But first, it uses a LOCK TABLES statement to prevent other users from writing data to this table while the script is executing. In addition, it uses an ALTER TABLE statement to disable the indexes for the table. Then, after the data has been inserted into the table, the script enables the indexes for the table. This improves the performance of the insert operations. Finally, the script uses an UNLOCK TABLES statement to allow other users to update this table.

After the SQL statements that create the database and its objects, this script sets some system variables. To do that, it uses some of the user-defined variables that were defined at the beginning of the script. Again, this usually works the way you want, so you don't usually have to examine this code closely.

The last line of this script is a comment that indicates the point in time that the mysqldump program finished creating this SQL file. If you need to restore a database later, you can use this date/time value as the start time for the statements that are stored in your binary log files.

Part of the SQL script file for a database backup — Page 2

```
--
-- Table structure for table `terms`
--

DROP TABLE IF EXISTS `terms`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `terms` (
  `terms_id` int(11) NOT NULL AUTO_INCREMENT,
  `terms_description` varchar(50) NOT NULL,
  `terms_due_days` int(11) NOT NULL,
  PRIMARY KEY (`terms_id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `terms`
--

LOCK TABLES `terms` WRITE;
/*!40000 ALTER TABLE `terms` DISABLE KEYS */;
INSERT INTO `terms` VALUES (1,'Net due 10 days',10),(2,'Net due 20
days',20),(3,'Net due 30 days',30),(4,'Net due 60 days',60),(5,'Net due 90
days',90);
/*!40000 ALTER TABLE `terms` ENABLE KEYS */;
UNLOCK TABLES;

--
-- SQL statement for the table structure and data for all other tables
-- and any triggers associated with those tables
--


--
-- SQL statements for all views, stored procedures, functions, and events
--

/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET character_set_client = @saved_cs_client */;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
/*!50606 SET GLOBAL INNODB_STATS_AUTO_RECALC=@OLD_INNODB_STATS_AUTO_RECALC */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2019-01-10 15:36:10
```

Figure 19-3 A SQL script file for a database backup (part 2 of 2)

How to set advanced options for a database backup

In most cases, the options for the mysqldump program are set the way you want. As a result, you typically only use the options shown in figure 19-2 to create database backups. However, the mysqldump program provides many advanced options, such as the ones shown in figure 19-4 that let you change the way the mysqldump program works. These options let you customize the generated SQL script file so it contains comments and SQL statements that work exactly the way you want.

For example, if you want to delete all old binary log files after the backup is complete, you can specify the --delete-master-logs option. In most cases, you don't want to do this in case there's a problem with the backup file. And you definitely don't want to do this if you're using replication, since it might prevent statements from being relayed to other servers. However, if you're confident in the backup file and you're not using replication, you might want to use this option since it removes old files that are no longer needed.

By default, most of the options in this figure are enabled. For example, the first six options are all enabled by default. As a result, you don't need to use the mysqldump program to specify these options. However, if you want to disable any of these options, you can preface them with "skip-". For example, to disable the --add-locks option, you can use the --skip-add-locks option.

If you want to disable the first four options in this figure, you can specify the --compact option. Then, the SQL script includes only the statements needed to back up the database. This takes less disk space, and it's particularly useful if you're creating a new database and you know that other users won't attempt to access this database as you're creating it.

Some advanced options for the mysqldump program

Option	Description
`--add-drop-table`	For each table, add a statement that drops the table before the statement that creates the table.
`--add-locks`	For each table, surround the INSERT statements with statements that lock and unlock the table.
`--disable-keys`	For each table, surround the INSERT statements with statements that disable and enable keys.
`--comments`	Include comments in the script.
`--quote-names`	Enclose names with backtick (`) characters.
`--create-options`	Include all MySQL-specific options in CREATE TABLE statements.
`--compact`	Create a more compact SQL script that includes only the statements needed to back up the database. Using this option is the same as specifying the --skip-add-drop-table, --skip-add-locks, --skip-disable-keys, and --skip-comments options.
`--compress`	Use compression in server/client protocol.
`--delete-master-logs`	Deletes all binary log files after performing the dump.
`--force`	Continue even if the program encounters a SQL error.

How to use the "skip" prefix to disable an option

```
--skip-add-drop-table
```

Description

- The mysqldump program contains many advanced options that you can use to control how it works.
- Many options that enable features also include a corresponding option for disabling the option. These options usually begin with "skip-".
- For a complete list of options for the mysqldump program, look up this program in the MySQL Reference Manual.

Figure 19-4 How to set advanced options for a database backup

How to restore a database

A backup of a database is only helpful if you can use it to restore the database in the event of a hardware failure or other problem. So that's what you'll learn to do in this topic.

How to use a SQL script file to restore a full backup

Figure 19-5 shows how to use the mysql program to restore a full backup of one or more databases. To do that, you just use the mysql program to run the SQL script file that contains the full backup.

In this figure, the first example restores the AP database by running the backup file for the AP database that was created by the first example in figure 19-2. Here, the mysql program specifies that you should run this backup file against the AP database. As a result, for this example to work properly, the AP database must exist on the MySQL server. If it doesn't, you must create this database before you run the backup file, or you must edit the backup file so it includes statements to create and select the database. You might edit the backup file, for example, if you're copying the database to another server.

On the other hand, the second example assumes that the backup file includes the statements that create the database or databases that you're restoring. As a result, you don't need to specify the name of the database on the mysql command. This example runs the file that was created by the second example in figure 19-2, which backs up four databases.

Before you restore a database from a script file, it's generally considered a good idea to back up the existing database. That way, if the restore operation doesn't work correctly, you can restore the database back to its previous state. In addition, it's usually a good idea to open the SQL file for the backup and view it to make sure it does what you want. Then, if it doesn't, you can edit this file so it works the way you want it to work. For example, if you only want to restore a single table, you can delete all other statements in the backup file.

You can also use MySQL Workbench to restore databases by running a SQL file that contains a full backup of those databases. To do that, select the Data Import/Restore option in the Administration tab of the Navigator window, select the "Import from Self-Contained File" option on the Import from Disk tab, enter the path and name for the backup file, and click the Start Import button. In most cases, this feature works correctly. If it doesn't, though, you can use the mysql program as described in this figure.

How to use the mysql program to restore databases

A single database

```
mysql ap < /murach/mysql/ap-2019-01-10.sql -u root -p
```

Multiple databases

```
mysql < /murach/mysql/backup-2019-01-10.sql -u root -p
```

Description

- You can use the mysql program to restore one or more databases by running the SQL script file that contains the database backup.
- Before you restore a database from a script file, it's generally considered a good idea to back up the existing database in case the restore operation doesn't work correctly.
- Before you restore a database from a script file, you can open the SQL file and view it to make sure it does what you want. If it doesn't, you can edit this file.
- If you get an error that indicates that access is denied, you may need to start the command prompt as an administrator.
- If a firewall is running on your computer, it may attempt to block the mysql program. However, if you allow the mysql program to access the database, it should work properly.
- On a macOS system, you typically need to code a dot and slash (./) before the name of the mysql program to specify that it's in the current directory (the bin directory).

Figure 19-5 How to use a SQL script file to restore a full backup

How to execute statements in the binary log

Figure 19-6 begins by showing the files for a binary log named bin-log. Here, the bin-log.index file is a text file. This file contains a list of all of the numbered files that store the changes that have been made to the database. These numbered log files (bin-log.000001, bin-log.000002, etc.) are binary files.

Next, this figure shows how to use the mysqlbinlog program to execute all or some of the statements in the binary log. To start, the first example shows how to execute all statements stored in a single binary log. Here, the mysqlbinlog program begins by specifying the path and name of the file. Then, it specifies a pipe character (|) followed by "mysql". This indicates that the mysqlbinlog program uses the mysql program to execute the statements in the binary log. Finally, the -u option identifies a user with privileges to restore databases, and the -p option specifies that the program should prompt for a password.

When you restore data using a binary log as shown in the first example, every statement for every database is executed by default. But what if you only need to restore one database from the binary log? In that case, you can use the --database option to specify the name of that database as shown in the second example. Or, what if you only want to execute statements that fall within a specified date/time range? In that case, you can use one or both of the date/time options to specify a starting date/time, an ending date/time, or both a starting and an ending date/time.

Finally, what if your binary log has been split across multiple files due to server restarts or file size limits? In that case, you can specify a list of binary log files as shown in the fourth and fifth examples. In the fourth example, the names of the log files are separated by a space. This works for Windows, macOS, and Unix/Linux. For macOS and Unix/Linux, you can also use a regular expression to select all binary log files as shown in the fifth example. Although you might expect that you could use a wildcard character to select binary log files with Windows (bin-log.*), this doesn't currently work with the mysqlbinlog program. However, it might work with future versions of this program.

The files for a binary log named bin-log

```
bin-log.index
bin-log.000001
bin-log.000002
bin-log.000003
...
```

How to use the mysqlbinlog program to execute statements

For all databases

```
mysqlbinlog /murach/mysql/bin-log.000001 | mysql -u root -p
```

For a specific database

```
mysqlbinlog --database=ap /murach/mysql/bin-log.000001 | mysql -u root -p
```

For a specific time range

```
mysqlbinlog --start-datetime="2019-01-10 00:00:00"
/murach/mysql/bin-log.000001 | mysql -u root -p
```

For all databases using multiple binary log files

```
mysqlbinlog /murach/mysql/bin-log.000001 /murach/mysql/bin-log.000002 |
mysql -u root -p
```

For all databases using multiple binary log files (macOS and Unix/Linux only)

```
mysqlbinlog /murach/mysql/bin-log.[0-9]* | mysql -u root -p
```

Common options for the mysqlbinlog program

Option	Description
`--database=db_name`	Identifies the database.
`--start-datetime=datetime`	Identifies the starting date/time.
`--stop-datetime=datetime`	Identifies the ending date/time.

Description

- You can use the mysqlbinlog program to execute statements in the log file for all databases or for a specified database. You can also execute statements that fall within a specified date/time range.
- If the statements you want to execute are stored in multiple binary logs, you should specify all of them on the command prompt in sequence from the lowest numbered log file to the highest numbered log file.
- On a macOS system, you typically need to (1) begin by coding the sudo command, (2) code a dot and slash before the name of the mysqlbinlog program and the mysql program, and (3) specify a path to the data directory. For example, you can execute the first example shown above like this:

```
sudo ./mysqlbinlog ../data/binlog.000001 | ./mysql -u root -p
```

- In addition, on macOS, you may find it helpful to specify a password like this:

```
sudo ./mysqlbinlog ../data/binlog.000001 | ./mysql -u root
--password=sesame80
```

Figure 19-6 How to execute statements in the binary log

How to import and export data

When you back up a database as shown earlier in this chapter, you can use the backup script file to copy the database to another server. In that case, backing up the database can be referred to as *exporting a database*. Similarly, when you restore a database from a backup that was performed on another server, it can be referred to as *importing a database*.

In addition to exporting and importing an entire database, you may sometimes need to export data from a database to a file or import data from a file to a database. For example, you may need to load shipping rates that are stored in a text file into a table. Or, you may need to export data so it can be used by a spreadsheet program or imported by another database. Fortunately, MySQL makes it easy to import and export data.

How to export data to a file

Figure 19-7 shows how to export data to a file. To do that, you can add an INTO OUTFILE clause to a SELECT statement to save the result set in an output file. By default, this clause uses a tab character (\t) to separate, or *delimit*, columns. And it uses a new line character (\n) to separate, or delimit, rows. When you store this type of data in a file, the file is known as a *tab-delimited file*. This type of file is commonly used to store and transfer data.

The first example in this figure exports all data from the Vendor_Contacts table and stores it in a tab-delimited file named vendor_contacts.txt. Since this is a small table with just three columns and eight rows, the SELECT statement exports the entire table. If necessary, though, you can limit the amount of data by including a column list and a WHERE clause.

Since it's easy to export data to a tab-delimited file, and since this format can be read by most other programs, this is the type of file that you usually want to use. However, if you need to export your data to another format, you can include the optional FIELDS clause to specify the delimiters for the columns and rows. For example, it's also common to store data in a *comma-delimited file*. To export data to a comma-delimited file, you can include a FIELDS clause with a TERMINATED BY clause that indicates that every column should be terminated by a comma (,) and an ENCLOSED BY clause that indicates that each column should be enclosed by double quotes ("). In this figure, for instance, the second example exports the data in the Vendor_Contacts table to a comma-delimited file.

In addition, if your data might contain a double quote character ("), you also need to include an ESCAPED BY clause to specify an *escape character*. Then, MySQL uses the escape character to identify any double quote characters that are part of the data. In this figure, for instance, the second example uses a backslash character as the escape character. Since the backslash character is used to escape special characters such as tabs (\t), new lines (\n), and single quotes (\'), though, you must code two backslashes (\\) to use a backslash character as the escape character.

The syntax of the SELECT statement for exporting data to a file

```
SELECT column_list
INTO OUTFILE file_path
[FIELDS [TERMINATED BY string]
        [ENCLOSED BY char]
        [ESCAPED BY char]]
FROM table name
[WHERE search_condition]
[ORDER BY order_by_list]
```

A tab-delimited file

The statement

```
SELECT *
INTO OUTFILE '/ProgramData/MySQL/MySQL Server 8.0/Uploads/vendor_contacts.txt'
FROM vendor_contacts
```

The file contents when viewed in a text editor

```
5       Davison  Michelle
12      Mayteh   Kendall
17      Onandonga        Bruce
44      Antavius Anthony
76      Bradlee  Danny
94      Suscipe  Reynaldo
101     O'Sullivan       Geraldine
123     Bucket   Charles
```

A comma-delimited file

The statement

```
SELECT *
INTO OUTFILE '/ProgramData/MySQL/MySQL Server 8.0/Uploads/vendor_contacts.txt'
FIELDS TERMINATED BY ',' ENCLOSED BY '"' ESCAPED BY '\\'
FROM vendor_contacts
```

The file contents

```
"5","Davison","Michelle"
"12","Mayteh","Kendall"
"17","Onandonga","Bruce"
"44","Antavius","Anthony"
"76","Bradlee","Danny"
"94","Suscipe","Reynaldo"
"101","O'Sullivan","Geraldine"
"123","Bucket","Charles"
```

Description

- You can add an INTO OUTFILE clause to a SELECT statement to save the result set in an output file.
- You can use the FIELDS clause to identify the character that's used to *delimit* columns, the character that's used to delimit rows, and an *escape character*.
- On a Windows system, with MySQL 5.5 and later, you can usually store the output file in the Uploads directory shown above. On a macOS system, you can sometimes store the output file in the /tmp directory, but it's usually easier to use MySQL Workbench's export data feature.

Figure 19-7 How to export data to a file

Note that both of the examples in figure 19-7 store the output file in the MySQL Uploads directory. With MySQL 5.5 and later, this is the only directory that you can export a file to or import a file from by default. If you want to export to or import from a different directory, you can change the secure_file_priv system variable to that directory. Or, you can change this variable to an empty value so you can export to and import from any directory. This was the default with releases of MySQL before 5.5.

Note also that the directories in these examples are for a Windows system. For macOS, you can sometimes store the output file in the /tmp directory. However, later versions of MySQL often don't allow storing output files in any directory by default with macOS. In addition, it's difficult to enable this feature by setting the secure_file_priv system variable. As a result, you may want to look for other ways to export data, such as using MySQL Workbench's export data feature.

How to import data from a file

Figure 19-8 begins by showing how to use the LOAD DATA statement to load data from an input file into a table. Specifically, it shows how to use this statement to import the data that was exported by the examples in the previous figure.

To start, you code a LOAD DATA clause that identifies the path and name of the file. Then, you code an INTO TABLE clause that identifies the table that you want to import the data into as shown in the first example in this figure. If you're working with a tab-delimited file, that's all you need to do. If you're working with a comma-delimited file, though, you need to include a FIELDS clause that identifies the delimiters and the escape character. In this figure, for instance, the second example includes the correct delimiters and escape character for the comma-delimited file that was created in the previous figure.

For an import to work successfully, the columns in the input file must match the columns in the table. In this figure, for example, the Vendor_Contacts table has three required columns: an INT column followed by two VARCHAR(50) columns. As a result, MySQL must be able to convert the data that's stored in the vendor_contacts.txt file to the data types specified by the Vendor_Contacts table.

In addition, the data in the input file must not conflict with the values of any unique keys that are already stored in the rows of the table. If that happens, you'll get an error that indicates that you were attempting to make a duplicate entry. Usually, that's what you want. If it isn't, you can delete any duplicate entries from the table.

How to use the LOAD DATA statement to import data from a file

The syntax

```
LOAD DATA INFILE file_path
INTO TABLE table_name
[FIELDS [TERMINATED BY string]
        [ENCLOSED BY char]
        [ESCAPED BY char]]
```

A tab-delimited file

```
LOAD DATA INFILE
    '/ProgramData/MySQL/MySQL Server 8.0/Uploads/vendor_contacts.txt'
INTO TABLE vendor_contacts
```

A comma-delimited file

```
LOAD DATA INFILE
    '/ProgramData/MySQL/MySQL Server 8.0/Uploads/vendor_contacts.txt'
INTO TABLE vendor_contacts
FIELDS TERMINATED BY ','
       ENCLOSED BY '"'
       ESCAPED BY '\\'
```

Description

- You can use the LOAD DATA statement to load data from an input file into a table.
- The columns in the input file must match the columns in the table.
- The data in the input file must not conflict with the values of any unique keys that are already stored in the rows of the table.
- You can also use the mysqlimport program to load data from an input file into a table. For more information, see the MySQL Reference Manual.

Figure 19-8 How to import data from a file

How to check and repair tables

When the server or operating system shuts down unexpectedly, the tables in a database can become corrupted. When that happens, the users of the database won't be able to access the table data. Then, you can use the tools MySQL provides to determine which tables need to be repaired. In addition, you can use MySQL tools to repair MyISAM tables. However, since the InnoDB engine is typically able to recover from unexpected shutdowns on its own, MySQL doesn't provide tools for repairing InnoDB tables. If it can't recover, though, you can use the technique you'll learn in just a minute to restore the corrupted tables.

How to use the CHECK TABLE statement

Figure 19-9 shows how to use the CHECK TABLE statement to check tables. This statement works for both InnoDB and MyISAM tables. In addition, this statement works for views.

If the CHECK TABLE statement finds no problems with a table, it will mark the table as OK as shown in the Msg_text column in all three examples in this figure. This allows MySQL to begin using the table again. The CHECK TABLE statement might also give you a message of "Table is already up to date" if it wasn't necessary to check the table. If it doesn't return either of these messages, you should repair the table as described in the next figure.

In most cases, you'll use the CHECK TABLE statement to check a single table or view using the default options as shown in the first example. However, if you need to check multiple tables or views, you can separate the names of the tables or views with commas as shown in the second example.

If you don't specify any options, the CHECK TABLE statement uses the MEDIUM option to do its check. However, if you need to change the default options, you can specify them after the list of tables or views. For example, you can specify the EXTENDED option to perform a more thorough check that takes longer. Or, you can specify the QUICK option to perform a less thorough check that runs faster. To speed this check even further, you can specify the FAST or CHANGED options. These options automatically include the QUICK option, which is usually what you want. In this figure, for instance, the third example specifies the FAST option.

Before you specify any of these options, you should know that they are ignored by the InnoDB engine. As a result, if you're checking an InnoDB table, you don't need to code these options. The exception is the FOR UPGRADE option, which is used by both the InnoDB and MyISAM storage engines.

The syntax of the CHECK TABLE statement

```
CHECK TABLE table_list option_list
```

Options for the CHECK TABLE statement

Option	Description
EXTENDED	Does a full scan of each row. This ensures that the table is 100% consistent, but takes a long time.
MEDIUM	Does an average scan of each row. This is the default for a MyISAM table.
QUICK	Does a quick scan of the rows.
FAST	Checks only tables that have not been closed properly. Uses the QUICK option.
CHANGED	Checks only tables that have been changed since the last check or that have not been closed properly. Uses the QUICK option.
FOR UPGRADE	Checks whether the tables are compatible with the current release of MySQL.

A statement that checks a single table

```
CHECK TABLE vendors
```

	Table	Op	Msg_type	Msg_text
▶	ap.vendors	check	status	OK

A statement that checks multiple tables and views

```
CHECK TABLE vendors, invoices, terms, invoices_outstanding
```

	Table	Op	Msg_type	Msg_text
▶	ap.vendors	check	status	OK
	ap.invoices	check	status	OK
	ap.terms	check	status	OK
	ap.invoices_outstanding	check	status	OK

A statement that uses an option

```
CHECK TABLE vendors, invoices FAST
```

	Table	Op	Msg_type	Msg_text
▶	ap.vendors	check	status	OK
	ap.invoices	check	status	OK

Description

- The CHECK TABLE statement works for InnoDB and MyISAM tables and views.
- All of the options except for FOR UPGRADE are ignored by the InnoDB engine. This engine automatically performs a thorough check that detects most problems. If it finds a problem, the server shuts down to prevent the problem from getting worse.
- The FOR UPGRADE option is useful if you upgrade to a newer release of MySQL. In that case, a change in the new release might make data from the old release incompatible with the new release.
- The CHECK TABLE statement works only when the server is running.

Figure 19-9 How to use the CHECK TABLE statement

How to repair a MyISAM table

Figure 19-10 shows how to use the REPAIR TABLE statement to repair corrupted MyISAM tables. This statement works much like the CHECK TABLE statement. However, it has fewer options, so it's easier to use. In addition, the REPAIR TABLE statement doesn't work for InnoDB tables.

When you use the REPAIR TABLE statement, the repair operation can sometimes cause the table to lose data. As a result, it's generally considered a best practice to make a backup of a table before performing a repair. That's especially true if it's critical to retain all data.

As it repairs a table, the REPAIR TABLE statement checks whether an upgrade is required. If so, it automatically performs the same upgrade operation that's performed by the FOR UPGRADE option of the CHECK TABLE statement.

How to repair an InnoDB table

If the CHECK TABLE statement finds a problem with an InnoDB table, the server shuts down to prevent the problem from getting worse. Because of that, you have to use the procedure shown in figure 19-10 to repair an InnoDB table.

To start, you add the innodb_force_recovery system variable to the MySQL configuration file. Although you can code different values for this variable, a value of 4 is typically sufficient. To learn more about this value and the other values you can code, see the MySQL Reference Manual.

Once you've added the innodb_force_recovery system variable, you can restart the server. Then, you should be able to back up the database that contains the corrupted tables as described earlier in this chapter. Next, you remove the innodb_force_recovery variable from the configuration file and restart the server. Finally, you can restore the database from the backup you just created.

In most cases, this procedure will fix the corrupted tables and restore most of the data. If it doesn't, you can use your last full backup and your incremental backups to restore the database.

How to repair a MyISAM table

The syntax of the REPAIR TABLE statement

```
REPAIR TABLE table_list option_list
```

Common options for the REPAIR TABLE statement

Option	Description
`QUICK`	Performs a standard repair that fixes most common problems.
`EXTENDED`	Performs a more extended repair.

A statement that repairs a single table

```
REPAIR TABLE vendors
```

A statement that repairs two tables and uses an option

```
REPAIR TABLE vendors, invoices QUICK
```

How to repair an InnoDB table

1. Use a text editor as described in chapter 17 to add this system variable to the configuration file:

   ```
   innodb_force_recovery=4
   ```

2. Restart the server, and then use the mysqldump program to back up the database.
3. Remove the innodb_force_recovery variable from the configuration file, restart the server, and restore the database to fix the corrupted tables and restore as much data as possible.

Description

- To repair a MyISAM table, you can use the REPAIR TABLE statement. This statement works only when the server is running.
- The REPAIR TABLE statement checks the table to see whether an upgrade is required. If so, it automatically performs the same upgrade operation that's provided by the FOR UPGRADE option of the CHECK TABLE statement.
- It's generally considered a best practice to make a backup of a table before performing a table repair operation, since a table repair operation can sometimes cause you to lose data.
- To repair an InnoDB table, you use the procedure shown above. The innodb_force_recovery system variable allows the server to restart so you can back up and then restore the database that contains the corrupt tables.
- If restoring the database doesn't fix the corrupted tables, you can use your last full backup and your incremental backups to restore the database.

Figure 19-10 How to repair tables

How to use the mysqlcheck program

Figure 19-11 shows how to use the mysqlcheck program to perform the same kinds of checks and repairs that you can perform with the CHECK TABLE and REPAIR TABLE statements. If you understand how these statements work, you shouldn't have much trouble understanding how to use the mysqlcheck program. The advantage of using the mysqlcheck program is that it allows you to check all tables in a database without having to specify the name of each table.

As usual, if you're using a macOS system, you typically need to code a dot and slash (./) before the name of the mysqlcheck program. This indicates that this program is in the current directory (the bin directory).

How to use the mysqlcheck program to check tables

For a single database

```
mysqlcheck ap -u root -p
```

For multiple databases

```
mysqlcheck --databases ap ex om -u root -p
```

For all databases

```
mysqlcheck --all-databases -u root -p
```

For specified tables within a database

```
mysqlcheck ap vendors invoices -u root -p
```

For a quick check

```
mysqlcheck ap --quick -u root -p
```

For an extended check

```
mysqlcheck ap --extended -u root -p
```

Common options for checking tables

Option	Corresponding CHECK TABLE option
`--extended`	`EXTENDED`
`--medium-check`	`MEDIUM`
`--quick`	`QUICK`
`--fast`	`FAST`
`--check-only-changed`	`CHANGED`
`--check-upgrade`	`FOR UPGRADE`

How to use the mysqlcheck program to repair tables

For a standard repair

```
mysqlcheck ap --repair -u root -p
```

For an extended repair

```
mysqlcheck ap --repair --extended -u root -p
```

Common options for repairing tables

Option	Description
`--repair`	Performs a repair that fixes most common problems.
`--extended`	A more extended repair than the standard repair.
`--quick`	A faster repair than the standard repair.

Description

- The mysqlcheck program uses the CHECK TABLE and REPAIR TABLE statements to check and repair one or more tables.
- Most of the check and repair options are ignored by the InnoDB engine.
- You can only use the mysqlcheck program when the server is running.

Figure 19-11 How to use the mysqlcheck program

How to use the myisamchk program

Figure 19-12 shows how to use the myisamchk program to check and repair MyISAM tables. If you understand how to use the mysqlcheck program described in the previous figure, you shouldn't have much trouble understanding how this program works.

The advantage of the myisamchk program is that you can use it while the server is stopped. If, for example, the server won't start due to corrupted tables, you can use the myisamchk program to attempt to repair those tables. However, you should not attempt to use the myisamchk program if the server is running. That's because this program won't work if another program is using the same table. In that case, you should use one of the other techniques presented in this chapter.

When you use the myisamchk program, you need to point to the table in the file system. To do that, you begin by coding the path to the MySQL data directory. Then, you code the name of the database followed by the name of the table. In this figure, for instance, all examples specify a path to the Engine_Sample table in the EX database. The MySQL data directory shown in these examples is the default path on a Windows system. However, this path will be different on a macOS or Unix/Linux system.

Unlike the CHECK TABLE statement and the mysqlcheck program, you can both check a table and repair it using a single myisamchk command. To do that, you use the --force option as shown in the last example in this figure. Then, if the program finds a problem with the table, it automatically repairs it.

If you're using a macOS system, you typically need to begin with the sudo command, code a dot and slash (./) before the name of the myisamchk program, and specify the data directory. For example, you can execute the first example in this figure like this:

```
sudo ./myisamchk "../data/ex/engine_sample"
```

How to use the myisamchk program to check a table

For a standard check

```
myisamchk "/ProgramData/MySQL/MySQL Server 8.0/Data/ex/engine_sample"
```

For a medium check

```
myisamchk --medium-check
"/ProgramData/MySQL/MySQL Server 8.0/Data/ex/engine_sample"
```

For an extended check

```
myisamchk --extend-check
"/ProgramData/MySQL/MySQL Server 8.0/Data/ex/engine_sample"
```

Common options for checking a table

Option	Corresponding CHECK TABLE option
`--extend-check`	`EXTENDED`
`--medium-check`	`MEDIUM`
`--check`	`QUICK`
`--fast`	`FAST`
`--check-only-changed`	`CHANGED`
`--force`	None. Automatically repairs the table if errors are found. Uses the --recover option.

How to use the myisamchk program to repair a table

For a standard repair

```
myisamchk --recover
"/ProgramData/MySQL/MySQL Server 8.0/Data/ex/engine_sample"
```

For a quick repair

```
myisamchk --recover --quick
"/ProgramData/MySQL/MySQL Server 8.0/Data/ex/engine_sample"
```

For an extended repair

```
myisamchk --safe-recover
"/ProgramData/MySQL/MySQL Server 8.0/Data/ex/engine_sample"
```

Common options for repairing a table

Option	Description
`--recover`	Performs a standard repair that fixes most common problems.
`--quick`	A faster repair than the standard repair.
`--safe-recover`	A more extended repair than the standard repair.

A command that checks a table and repairs it if necessary

```
myisamchk --force
"/ProgramData/MySQL/MySQL Server 8.0/Data/ex/engine_sample"
```

Description

- The myisamchk program can check and repair MyISAM tables.
- You should only use the myisamchk program when the server is stopped.

Figure 19-12 How to use the myisamchk program

Perspective

In this chapter, you learned how to back up your databases and how to restore them if necessary. If you combine these skills with the skills you learned in the previous chapter for securing a database and working with user accounts, you are on your way to becoming a successful database administrator. Of course, there's much more to learn than what's presented here. If you're interested in learning more, I recommend you get a book specifically on database administration.

Terms

back up a database
restore a database
full backup
incremental backup
point-in-time recovery (PITR)
dump a database
export a database
import a database
delimit columns or rows
tab-delimited file
comma-delimited file
escape character