**Копіювання та відновлення БД Backup and Recovery**

**Table of Contents**

[Backup and Recovery Types 3](#_Toc58485008)

[Physical (Raw) Versus Logical Backups 3](#_Toc58485009)

[Online Versus Offline Backups 4](#_Toc58485010)

[Local Versus Remote Backups 5](#_Toc58485011)

[Snapshot Backups 5](#_Toc58485012)

[Full Versus Incremental Backups 5](#_Toc58485013)

[Full Versus Point-in-Time (Incremental) Recovery 5](#_Toc58485014)

[Table Maintenance 6](#_Toc58485015)

[Backup Scheduling, Compression, and Encryption 6](#_Toc58485016)

[Database Backup Methods 6](#_Toc58485017)

[Making a Hot Backup with MySQL Enterprise Backup 6](#_Toc58485018)

[Making Backups with mysqldump 6](#_Toc58485019)

[Making Backups by Copying Table Files 6](#_Toc58485020)

[Making Delimited-Text File Backups 7](#_Toc58485021)

[Making Incremental Backups by Enabling the Binary Log 7](#_Toc58485022)

[Making Backups Using Replicas 7](#_Toc58485023)

[Recovering Corrupt Tables 7](#_Toc58485024)

[Making Backups Using a File System Snapshot 7](#_Toc58485025)

[Example Backup and Recovery Strategy 8](#_Toc58485026)

[Establishing a Backup Policy 8](#_Toc58485027)

[Using Backups for Recovery 10](#_Toc58485028)

[Backup Strategy Summary 10](#_Toc58485029)

[Using mysqldump for Backups 11](#_Toc58485030)

[Dumping Data in SQL Format with mysqldump 11](#_Toc58485031)

[Reloading SQL-Format Backups 12](#_Toc58485032)

[Dumping Data in Delimited-Text Format with mysqldump 13](#_Toc58485033)

[Reloading Delimited-Text Format Backups 14](#_Toc58485034)

[mysqldump Tips 14](#_Toc58485035)

[Point-in-Time (Incremental) Recovery 16](#_Toc58485036)

[Point-in-Time Recovery Using Binary Log 16](#_Toc58485037)

[Point-in-Time Recovery Using Event Positions 17](#_Toc58485038)

[MyISAM Table Maintenance and Crash Recovery 19](#_Toc58485039)

[Using myisamchk for Crash Recovery 20](#_Toc58485040)

[How to Check MyISAM Tables for Errors 20](#_Toc58485041)

[How to Repair MyISAM Tables 21](#_Toc58485042)

[MyISAM Table Optimization 23](#_Toc58485043)

[Setting Up a MyISAM Table Maintenance Schedule 23](#_Toc58485044)

It is important to back up your databases so that you can recover your data and be up and running again in case problems occur, such as system crashes, hardware failures, or users deleting data by mistake. Backups are also essential as a safeguard before upgrading a MySQL installation, and they can be used to transfer a MySQL installation to another system or to set up replica servers.

MySQL offers a variety of backup strategies from which you can choose the methods that best suit the requirements for your installation. This chapter discusses several backup and recovery topics with which you should be familiar:

* Types of backups: Logical versus physical, full versus incremental, and so forth.
* Methods for creating backups.
* Recovery methods, including point-in-time recovery.
* Backup scheduling, compression, and encryption.
* Table maintenance, to enable recovery of corrupt tables.

## Backup and Recovery Types

This section describes the characteristics of different types of backups.

### Physical (Raw) Versus Logical Backups

Physical backups consist of raw copies of the directories and files that store database contents. This type of backup is suitable for large, important databases that need to be recovered quickly when problems occur.

Logical backups save information represented as logical database structure ([CREATE DATABASE](https://dev.mysql.com/doc/refman/8.0/en/create-database.html), [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) statements) and content ([INSERT](https://dev.mysql.com/doc/refman/8.0/en/insert.html) statements or delimited-text files). This type of backup is suitable for smaller amounts of data where you might edit the data values or table structure, or recreate the data on a different machine architecture.

Physical backup methods have these characteristics:

* The backup consists of exact copies of database directories and files. Typically this is a copy of all or part of the MySQL data directory.
* Physical backup methods are faster than logical because they involve only file copying without conversion.
* Output is more compact than for logical backup.
* Because backup speed and compactness are important for busy, important databases, the MySQL Enterprise Backup product performs physical backups. For an overview of the MySQL Enterprise Backup product, see [Section 30.2, “MySQL Enterprise Backup Overview”](https://dev.mysql.com/doc/refman/8.0/en/mysql-enterprise-backup.html).
* Backup and restore granularity ranges from the level of the entire data directory down to the level of individual files. This may or may not provide for table-level granularity, depending on storage engine. For example, InnoDB tables can each be in a separate file, or share file storage with other InnoDB tables; each MyISAM table corresponds uniquely to a set of files.
* In addition to databases, the backup can include any related files such as log or configuration files.
* Data from MEMORY tables is tricky to back up this way because their contents are not stored on disk. (The MySQL Enterprise Backup product has a feature where you can retrieve data from MEMORY tables during a backup.)
* Backups are portable only to other machines that have identical or similar hardware characteristics.
* Backups can be performed while the MySQL server is not running. If the server is running, it is necessary to perform appropriate locking so that the server does not change database contents during the backup. MySQL Enterprise Backup does this locking automatically for tables that require it.
* Physical backup tools include the **mysqlbackup** of MySQL Enterprise Backup for InnoDB or any other tables, or file system-level commands (such as **cp**, **scp**, **tar**, **rsync**) for MyISAM tables.
* For restore:
	+ MySQL Enterprise Backup restores InnoDB and other tables that it backed up.
	+ [**ndb\_restore**](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster-programs-ndb-restore.html) restores [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) tables.
	+ Files copied at the file system level can be copied back to their original locations with file system commands.

Logical backup methods have these characteristics:

* The backup is done by querying the MySQL server to obtain database structure and content information.
* Backup is slower than physical methods because the server must access database information and convert it to logical format. If the output is written on the client side, the server must also send it to the backup program.
* Output is larger than for physical backup, particularly when saved in text format.
* Backup and restore granularity is available at the server level (all databases), database level (all tables in a particular database), or table level. This is true regardless of storage engine.
* The backup does not include log or configuration files, or other database-related files that are not part of databases.
* Backups stored in logical format are machine independent and highly portable.
* Logical backups are performed with the MySQL server running. The server is not taken offline.
* Logical backup tools include the [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) program and the [SELECT ... INTO OUTFILE](https://dev.mysql.com/doc/refman/8.0/en/select.html) statement. These work for any storage engine, even MEMORY.
* To restore logical backups, SQL-format dump files can be processed using the [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) client. To load delimited-text files, use the [LOAD DATA](https://dev.mysql.com/doc/refman/8.0/en/load-data.html) statement or the [**mysqlimport**](https://dev.mysql.com/doc/refman/8.0/en/mysqlimport.html) client.

### Online Versus Offline Backups

Online backups take place while the MySQL server is running so that the database information can be obtained from the server. Offline backups take place while the server is stopped. This distinction can also be described as “hot” versus “cold” backups; a “warm” backup is one where the server remains running but locked against modifying data while you access database files externally.

Online backup methods have these characteristics:

* The backup is less intrusive to other clients, which can connect to the MySQL server during the backup and may be able to access data depending on what operations they need to perform.
* Care must be taken to impose appropriate locking so that data modifications do not take place that would compromise backup integrity. The MySQL Enterprise Backup product does such locking automatically.

Offline backup methods have these characteristics:

* Clients can be affected adversely because the server is unavailable during backup. For that reason, such backups are often taken from a replica that can be taken offline without harming availability.
* The backup procedure is simpler because there is no possibility of interference from client activity.

A similar distinction between online and offline applies for recovery operations, and similar characteristics apply. However, it is more likely for clients to be affected by online recovery than by online backup because recovery requires stronger locking. During backup, clients might be able to read data while it is being backed up. Recovery modifies data and does not just read it, so clients must be prevented from accessing data while it is being restored.

### Local Versus Remote Backups

A local backup is performed on the same host where the MySQL server runs, whereas a remote backup is done from a different host. For some types of backups, the backup can be initiated from a remote host even if the output is written locally on the server. host.

* [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) can connect to local or remote servers. For SQL output (CREATE and [INSERT](https://dev.mysql.com/doc/refman/8.0/en/insert.html) statements), local or remote dumps can be done and generate output on the client. For delimited-text output (with the [--tab](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_tab) option), data files are created on the server host.
* [SELECT ... INTO OUTFILE](https://dev.mysql.com/doc/refman/8.0/en/select-into.html) can be initiated from a local or remote client host, but the output file is created on the server host.
* Physical backup methods typically are initiated locally on the MySQL server host so that the server can be taken offline, although the destination for copied files might be remote.

### Snapshot Backups

Some file system implementations enable “snapshots” to be taken. These provide logical copies of the file system at a given point in time, without requiring a physical copy of the entire file system. (For example, the implementation may use copy-on-write techniques so that only parts of the file system modified after the snapshot time need be copied.) MySQL itself does not provide the capability for taking file system snapshots. It is available through third-party solutions such as Veritas, LVM, or ZFS.

### Full Versus Incremental Backups

A full backup includes all data managed by a MySQL server at a given point in time. An incremental backup consists of the changes made to the data during a given time span (from one point in time to another). MySQL has different ways to perform full backups, such as those described earlier in this section. Incremental backups are made possible by enabling the server's binary log, which the server uses to record data changes.

### Full Versus Point-in-Time (Incremental) Recovery

A full recovery restores all data from a full backup. This restores the server instance to the state that it had when the backup was made. If that state is not sufficiently current, a full recovery can be followed by recovery of incremental backups made since the full backup, to bring the server to a more up-to-date state.

Incremental recovery is recovery of changes made during a given time span. This is also called point-in-time recovery because it makes a server's state current up to a given time. Point-in-time recovery is based on the binary log and typically follows a full recovery from the backup files that restores the server to its state when the backup was made. Then the data changes written in the binary log files are applied as incremental recovery to redo data modifications and bring the server up to the desired point in time.

### Table Maintenance

Data integrity can be compromised if tables become corrupt. For [InnoDB](https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html) tables, this is not a typical issue. For programs to check [MyISAM](https://dev.mysql.com/doc/refman/8.0/en/myisam-storage-engine.html) tables and repair them if problems are found, see [Section 7.6, “MyISAM Table Maintenance and Crash Recovery”](https://dev.mysql.com/doc/refman/8.0/en/myisam-table-maintenance.html).

### Backup Scheduling, Compression, and Encryption

Backup scheduling is valuable for automating backup procedures. Compression of backup output reduces space requirements, and encryption of the output provides better security against unauthorized access of backed-up data. MySQL itself does not provide these capabilities. The MySQL Enterprise Backup product can compress InnoDB backups, and compression or encryption of backup output can be achieved using file system utilities. Other third-party solutions may be available.

## Database Backup Methods

This section summarizes some general methods for making backups.

### Making a Hot Backup with MySQL Enterprise Backup

Customers of MySQL Enterprise Edition can use the [MySQL Enterprise Backup](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_mysql_enterprise_backup) product to do [physical](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_physical) backups of entire instances or selected databases, tables, or both. This product includes features for [incremental](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_incremental_backup) and [compressed](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_compressed_backup) backups. Backing up the physical database files makes restore much faster than logical techniques such as the mysqldump command. InnoDB tables are copied using a [hot backup](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_hot_backup) mechanism. (Ideally, the InnoDB tables should represent a substantial majority of the data.) Tables from other storage engines are copied using a [warm backup](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_warm_backup) mechanism. For an overview of the MySQL Enterprise Backup product, see [Section 30.2, “MySQL Enterprise Backup Overview”](https://dev.mysql.com/doc/refman/8.0/en/mysql-enterprise-backup.html).

### Making Backups with mysqldump

The [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) program can make backups. It can back up all kinds of tables. (See [Section 7.4, “Using mysqldump for Backups”](https://dev.mysql.com/doc/refman/8.0/en/using-mysqldump.html).)

For InnoDB tables, it is possible to perform an online backup that takes no locks on tables using the [--single-transaction](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_single-transaction) option to [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html). See [Section 7.3.1, “Establishing a Backup Policy”](https://dev.mysql.com/doc/refman/8.0/en/backup-policy.html).

### Making Backups by Copying Table Files

MyISAM tables can be backed up by copying table files (\*.MYD, \*.MYI files, and associated \*.sdi files). To get a consistent backup, stop the server or lock and flush the relevant tables:

FLUSH TABLES tbl\_list WITH READ LOCK;

You need only a read lock; this enables other clients to continue to query the tables while you are making a copy of the files in the database directory. The flush is needed to ensure that the all active index pages are written to disk before you start the backup. See [Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Statements”](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html), and [Section 13.7.8.3, “FLUSH Statement”](https://dev.mysql.com/doc/refman/8.0/en/flush.html).

You can also create a binary backup simply by copying the table files, as long as the server isn't updating anything. (But note that table file copying methods do not work if your database contains InnoDB tables. Also, even if the server is not actively updating data, InnoDB may still have modified data cached in memory and not flushed to disk.)

For an example of this backup method, refer to the export and import example in [Section 13.2.5, “IMPORT TABLE Statement”](https://dev.mysql.com/doc/refman/8.0/en/import-table.html).

### Making Delimited-Text File Backups

To create a text file containing a table's data, you can use [SELECT \* INTO OUTFILE '***file\_name***' FROM ***tbl\_name***](https://dev.mysql.com/doc/refman/8.0/en/select-into.html). The file is created on the MySQL server host, not the client host. For this statement, the output file cannot already exist because permitting files to be overwritten constitutes a security risk. See [Section 13.2.10, “SELECT Statement”](https://dev.mysql.com/doc/refman/8.0/en/select.html). This method works for any kind of data file, but saves only table data, not the table structure.

Another way to create text data files (along with files containing [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) statements for the backed up tables) is to use [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) with the [--tab](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_tab) option. See [Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”](https://dev.mysql.com/doc/refman/8.0/en/mysqldump-delimited-text.html).

To reload a delimited-text data file, use [LOAD DATA](https://dev.mysql.com/doc/refman/8.0/en/load-data.html) or [**mysqlimport**](https://dev.mysql.com/doc/refman/8.0/en/mysqlimport.html).

### Making Incremental Backups by Enabling the Binary Log

MySQL supports incremental backups using the binary log. The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you performed a backup. Therefore, to allow a server to be restored to a point-in-time, binary logging must be enabled on it, which is the default setting for MySQL 8.0 ; see [Section 5.4.4, “The Binary Log”](https://dev.mysql.com/doc/refman/8.0/en/binary-log.html).

At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using [FLUSH LOGS](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-logs). This done, you need to copy to the backup location all binary logs which range from the one of the moment of the last full or incremental backup to the last but one. These binary logs are the incremental backup; at restore time, you apply them as explained in [Section 7.5, “Point-in-Time (Incremental) Recovery”](https://dev.mysql.com/doc/refman/8.0/en/point-in-time-recovery.html). The next time you do a full backup, you should also rotate the binary log using [FLUSH LOGS](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-logs) or [**mysqldump --flush-logs**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html). See [Section 4.5.4, “**mysqldump** — A Database Backup Program”](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html).

### Making Backups Using Replicas

If you have performance problems with a server while making backups, one strategy that can help is to set up replication and perform backups on the replica rather than on the source. See [Section 17.4.1, “Using Replication for Backups”](https://dev.mysql.com/doc/refman/8.0/en/replication-solutions-backups.html).

If you are backing up a replica, you should back up its connection metadata repository and applier metadata repository (see [Section 17.2.4, “Relay Log and Replication Metadata Repositories”](https://dev.mysql.com/doc/refman/8.0/en/replica-logs.html)) when you back up the replica's databases, regardless of the backup method you choose. This information is always needed to resume replication after you restore the replica's data. If your replica is replicating [LOAD DATA](https://dev.mysql.com/doc/refman/8.0/en/load-data.html) statements, you should also back up any SQL\_LOAD-\* files that exist in the directory that the replica uses for this purpose. The replica needs these files to resume replication of any interrupted [LOAD DATA](https://dev.mysql.com/doc/refman/8.0/en/load-data.html) operations. The location of this directory is the value of the [slave\_load\_tmpdir](https://dev.mysql.com/doc/refman/8.0/en/replication-options-replica.html#sysvar_slave_load_tmpdir) system variable. If the server was not started with that variable set, the directory location is the value of the [tmpdir](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_tmpdir) system variable.

### Recovering Corrupt Tables

If you have to restore MyISAM tables that have become corrupt, try to recover them using [REPAIR TABLE](https://dev.mysql.com/doc/refman/8.0/en/repair-table.html) or [**myisamchk -r**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) first. That should work in 99.9% of all cases. If [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) fails, see [Section 7.6, “MyISAM Table Maintenance and Crash Recovery”](https://dev.mysql.com/doc/refman/8.0/en/myisam-table-maintenance.html).

### Making Backups Using a File System Snapshot

If you are using a Veritas file system, you can make a backup like this:

1. From a client program, execute [FLUSH TABLES WITH READ LOCK](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-tables-with-read-lock).
2. From another shell, execute mount vxfs snapshot.
3. From the first client, execute [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html).
4. Copy files from the snapshot.
5. Unmount the snapshot.

Similar snapshot capabilities may be available in other file systems, such as LVM or ZFS.

Example Backup and Recovery Strategy

This section discusses a procedure for performing backups that enables you to recover data after several types of crashes:

* Operating system crash
* Power failure
* File system crash
* Hardware problem (hard drive, motherboard, and so forth)

The example commands do not include options such as [--user](https://dev.mysql.com/doc/refman/8.0/en/connection-options.html#option_general_user) and [--password](https://dev.mysql.com/doc/refman/8.0/en/connection-options.html#option_general_password) for the [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) and [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) client programs. You should include such options as necessary to enable client programs to connect to the MySQL server.

Assume that data is stored in the InnoDB storage engine, which has support for transactions and automatic crash recovery. Assume also that the MySQL server is under load at the time of the crash. If it were not, no recovery would ever be needed.

For cases of operating system crashes or power failures, we can assume that MySQL's disk data is available after a restart. The InnoDB data files might not contain consistent data due to the crash, but InnoDB reads its logs and finds in them the list of pending committed and noncommitted transactions that have not been flushed to the data files. InnoDB automatically rolls back those transactions that were not committed, and flushes to its data files those that were committed. Information about this recovery process is conveyed to the user through the MySQL error log. The following is an example log excerpt:

InnoDB: Database was not shut down normally.

InnoDB: Starting recovery from log files...

InnoDB: Starting log scan based on checkpoint at

InnoDB: log sequence number 0 13674004

InnoDB: Doing recovery: scanned up to log sequence number 0 13739520

InnoDB: Doing recovery: scanned up to log sequence number 0 13805056

InnoDB: Doing recovery: scanned up to log sequence number 0 13870592

InnoDB: Doing recovery: scanned up to log sequence number 0 13936128

...

InnoDB: Doing recovery: scanned up to log sequence number 0 20555264

InnoDB: Doing recovery: scanned up to log sequence number 0 20620800

InnoDB: Doing recovery: scanned up to log sequence number 0 20664692

InnoDB: 1 uncommitted transaction(s) which must be rolled back

InnoDB: Starting rollback of uncommitted transactions

InnoDB: Rolling back trx no 16745

InnoDB: Rolling back of trx no 16745 completed

InnoDB: Rollback of uncommitted transactions completed

InnoDB: Starting an apply batch of log records to the database...

InnoDB: Apply batch completed

InnoDB: Started

mysqld: ready for connections

For the cases of file system crashes or hardware problems, we can assume that the MySQL disk data is *not* available after a restart. This means that MySQL fails to start successfully because some blocks of disk data are no longer readable. In this case, it is necessary to reformat the disk, install a new one, or otherwise correct the underlying problem. Then it is necessary to recover our MySQL data from backups, which means that backups must already have been made. To make sure that is the case, design and implement a backup policy.

### Establishing a Backup Policy

To be useful, backups must be scheduled regularly. A full backup (a snapshot of the data at a point in time) can be done in MySQL with several tools. For example, [MySQL Enterprise Backup](https://dev.mysql.com/doc/refman/8.0/en/mysql-enterprise-backup.html) can perform a [physical backup](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_physical_backup) of an entire instance, with optimizations to minimize overhead and avoid disruption when backing up InnoDB data files; [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) provides online [logical backup](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_logical_backup). This discussion uses [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html).

Assume that we make a full backup of all our InnoDB tables in all databases using the following command on Sunday at 1 p.m., when load is low:

shell> mysqldump --all-databases --master-data --single-transaction > backup\_sunday\_1\_PM.sql

The resulting .sql file produced by [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) contains a set of SQL [INSERT](https://dev.mysql.com/doc/refman/8.0/en/insert.html) statements that can be used to reload the dumped tables at a later time.

This backup operation acquires a global read lock on all tables at the beginning of the dump (using [FLUSH TABLES WITH READ LOCK](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-tables-with-read-lock)). As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the [FLUSH](https://dev.mysql.com/doc/refman/8.0/en/flush.html) statement is issued, the backup operation may stall until those statements finish. After that, the dump becomes lock-free and does not disturb reads and writes on the tables.

It was assumed earlier that the tables to back up are InnoDB tables, so [--single-transaction](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_single-transaction) uses a consistent read and guarantees that data seen by [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) does not change. (Changes made by other clients to InnoDB tables are not seen by the [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) process.) If the backup operation includes nontransactional tables, consistency requires that they do not change during the backup. For example, for the MyISAM tables in the mysql database, there must be no administrative changes to MySQL accounts during the backup.

Full backups are necessary, but it is not always convenient to create them. They produce large backup files and take time to generate. They are not optimal in the sense that each successive full backup includes all data, even that part that has not changed since the previous full backup. It is more efficient to make an initial full backup, and then to make incremental backups. The incremental backups are smaller and take less time to produce. The tradeoff is that, at recovery time, you cannot restore your data just by reloading the full backup. You must also process the incremental backups to recover the incremental changes.

To make incremental backups, we need to save the incremental changes. In MySQL, these changes are represented in the binary log, so the MySQL server should always be started with the [--log-bin](https://dev.mysql.com/doc/refman/8.0/en/replication-options-binary-log.html#option_mysqld_log-bin) option to enable that log. With binary logging enabled, the server writes each data change into a file while it updates data. Looking at the data directory of a MySQL server that has been running for some days, we find these MySQL binary log files:

-rw-rw---- 1 guilhem guilhem 1277324 Nov 10 23:59 gbichot2-bin.000001

-rw-rw---- 1 guilhem guilhem 4 Nov 10 23:59 gbichot2-bin.000002

-rw-rw---- 1 guilhem guilhem 79 Nov 11 11:06 gbichot2-bin.000003

-rw-rw---- 1 guilhem guilhem 508 Nov 11 11:08 gbichot2-bin.000004

-rw-rw---- 1 guilhem guilhem 220047446 Nov 12 16:47 gbichot2-bin.000005

-rw-rw---- 1 guilhem guilhem 998412 Nov 14 10:08 gbichot2-bin.000006

-rw-rw---- 1 guilhem guilhem 361 Nov 14 10:07 gbichot2-bin.index

Each time it restarts, the MySQL server creates a new binary log file using the next number in the sequence. While the server is running, you can also tell it to close the current binary log file and begin a new one manually by issuing a [FLUSH LOGS](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-logs) SQL statement or with a [**mysqladmin flush-logs**](https://dev.mysql.com/doc/refman/8.0/en/mysqladmin.html) command. [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) also has an option to flush the logs. The .index file in the data directory contains the list of all MySQL binary logs in the directory.

The MySQL binary logs are important for recovery because they form the set of incremental backups. If you make sure to flush the logs when you make your full backup, the binary log files created afterward contain all the data changes made since the backup. Let's modify the previous [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) command a bit so that it flushes the MySQL binary logs at the moment of the full backup, and so that the dump file contains the name of the new current binary log:

shell> mysqldump --single-transaction --flush-logs --master-data=2 \

 --all-databases > backup\_sunday\_1\_PM.sql

After executing this command, the data directory contains a new binary log file, gbichot2-bin.000007, because the [--flush-logs](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_flush-logs) option causes the server to flush its logs. The [--master-data](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_master-data) option causes [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) to write binary log information to its output, so the resulting .sql dump file includes these lines:

-- Position to start replication or point-in-time recovery from

-- CHANGE MASTER TO MASTER\_LOG\_FILE='gbichot2-bin.000007',MASTER\_LOG\_POS=4;

Because the [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) command made a full backup, those lines mean two things:

* The dump file contains all changes made before any changes written to the gbichot2-bin.000007 binary log file or higher.
* All data changes logged after the backup are not present in the dump file, but are present in the gbichot2-bin.000007 binary log file or higher.

On Monday at 1 p.m., we can create an incremental backup by flushing the logs to begin a new binary log file. For example, executing a [**mysqladmin flush-logs**](https://dev.mysql.com/doc/refman/8.0/en/mysqladmin.html) command creates gbichot2-bin.000008. All changes between the Sunday 1 p.m. full backup and Monday 1 p.m. are written in gbichot2-bin.000007. This incremental backup is important, so it is a good idea to copy it to a safe place. (For example, back it up on tape or DVD, or copy it to another machine.) On Tuesday at 1 p.m., execute another [**mysqladmin flush-logs**](https://dev.mysql.com/doc/refman/8.0/en/mysqladmin.html) command. All changes between Monday 1 p.m. and Tuesday 1 p.m. are written in gbichot2-bin.000008 (which also should be copied somewhere safe).

The MySQL binary logs take up disk space. To free up space, purge them from time to time. One way to do this is by deleting the binary logs that are no longer needed, such as when we make a full backup:

shell> mysqldump --single-transaction --flush-logs --master-data=2 \

 --all-databases --delete-master-logs > backup\_sunday\_1\_PM.sql

**Note**

Deleting the MySQL binary logs with [**mysqldump --delete-master-logs**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) can be dangerous if your server is a replication source server, because replicas might not yet fully have processed the contents of the binary log. The description for the [PURGE BINARY LOGS](https://dev.mysql.com/doc/refman/8.0/en/purge-binary-logs.html) statement explains what should be verified before deleting the MySQL binary logs.

### Using Backups for Recovery

Now, suppose that we have a catastrophic unexpected exit on Wednesday at 8 a.m. that requires recovery from backups. To recover, first we restore the last full backup we have (the one from Sunday 1 p.m.). The full backup file is just a set of SQL statements, so restoring it is very easy:

shell> mysql < backup\_sunday\_1\_PM.sql

At this point, the data is restored to its state as of Sunday 1 p.m.. To restore the changes made since then, we must use the incremental backups; that is, the gbichot2-bin.000007 and gbichot2-bin.000008 binary log files. Fetch the files if necessary from where they were backed up, and then process their contents like this:

shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql

We now have recovered the data to its state as of Tuesday 1 p.m., but still are missing the changes from that date to the date of the crash. To not lose them, we would have needed to have the MySQL server store its MySQL binary logs into a safe location (RAID disks, SAN, ...) different from the place where it stores its data files, so that these logs were not on the destroyed disk. (That is, we can start the server with a [--log-bin](https://dev.mysql.com/doc/refman/8.0/en/replication-options-binary-log.html#option_mysqld_log-bin) option that specifies a location on a different physical device from the one on which the data directory resides. That way, the logs are safe even if the device containing the directory is lost.) If we had done this, we would have the gbichot2-bin.000009 file (and any subsequent files) at hand, and we could apply them using [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html) and [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) to restore the most recent data changes with no loss up to the moment of the crash:

shell> mysqlbinlog gbichot2-bin.000009 ... | mysql

### Backup Strategy Summary

In case of an operating system crash or power failure, InnoDB itself does all the job of recovering data. But to make sure that you can sleep well, observe the following guidelines:

* Always tun the MySQL server with binary logging enabled (that is the default setting for MySQL 8.0). If you have such safe media, this technique can also be good for disk load balancing (which results in a performance improvement).
* Make periodic full backups, using the [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) command shown earlier in [Section 7.3.1, “Establishing a Backup Policy”](https://dev.mysql.com/doc/refman/8.0/en/backup-policy.html), that makes an online, nonblocking backup.
* Make periodic incremental backups by flushing the logs with [FLUSH LOGS](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-logs) or [**mysqladmin flush-logs**](https://dev.mysql.com/doc/refman/8.0/en/mysqladmin.html).

Using mysqldump for Backups

**Tip**

Consider using the [MySQL Shell dump utilities](https://dev.mysql.com/doc/mysql-shell/8.0/en/mysql-shell-utilities-dump-instance-schema.html), which provide parallel dumping with multiple threads, file compression, and progress information display, as well as cloud features such as Oracle Cloud Infrastructure Object Storage streaming, and MySQL Database Service compatibility checks and modifications. Dumps can be easily imported into a MySQL Server instance or a MySQL Database Service DB System using the [MySQL Shell load dump utilities](https://dev.mysql.com/doc/mysql-shell/8.0/en/mysql-shell-utilities-load-dump.html). Installation instructions for MySQL Shell can be found [here](https://dev.mysql.com/doc/mysql-shell/8.0/en/mysql-shell-install.html).

This section describes how to use [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) to produce dump files, and how to reload dump files. A dump file can be used in several ways:

* As a backup to enable data recovery in case of data loss.
* As a source of data for setting up replicas.
* As a source of data for experimentation:
	+ To make a copy of a database that you can use without changing the original data.
	+ To test potential upgrade incompatibilities.

[**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) produces two types of output, depending on whether the [--tab](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_tab) option is given:

* Without [--tab](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_tab), [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) writes SQL statements to the standard output. This output consists of CREATE statements to create dumped objects (databases, tables, stored routines, and so forth), and INSERT statements to load data into tables. The output can be saved in a file and reloaded later using [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) to recreate the dumped objects. Options are available to modify the format of the SQL statements, and to control which objects are dumped.
* With [--tab](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_tab), [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) produces two output files for each dumped table. The server writes one file as tab-delimited text, one line per table row. This file is named ***tbl\_name***.txt in the output directory. The server also sends a [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) statement for the table to [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html), which writes it as a file named ***tbl\_name***.sql in the output directory.

### Dumping Data in SQL Format with mysqldump

This section describes how to use [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) to create SQL-format dump files. For information about reloading such dump files, see [Section 7.4.2, “Reloading SQL-Format Backups”](https://dev.mysql.com/doc/refman/8.0/en/reloading-sql-format-dumps.html).

By default, [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) writes information as SQL statements to the standard output. You can save the output in a file:

shell> mysqldump [arguments] > file\_name

To dump all databases, invoke [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) with the [--all-databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_all-databases) option:

shell> mysqldump --all-databases > dump.sql

To dump only specific databases, name them on the command line and use the [--databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_databases) option:

shell> mysqldump --databases db1 db2 db3 > dump.sql

The [--databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_databases) option causes all names on the command line to be treated as database names. Without this option, [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) treats the first name as a database name and those following as table names.

With [--all-databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_all-databases) or [--databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_databases), [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) writes [CREATE DATABASE](https://dev.mysql.com/doc/refman/8.0/en/create-database.html) and [USE](https://dev.mysql.com/doc/refman/8.0/en/use.html) statements prior to the dump output for each database. This ensures that when the dump file is reloaded, it creates each database if it does not exist and makes it the default database so database contents are loaded into the same database from which they came. If you want to cause the dump file to force a drop of each database before recreating it, use the [--add-drop-database](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_add-drop-database) option as well. In this case, [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) writes a [DROP DATABASE](https://dev.mysql.com/doc/refman/8.0/en/drop-database.html) statement preceding each [CREATE DATABASE](https://dev.mysql.com/doc/refman/8.0/en/create-database.html) statement.

To dump a single database, name it on the command line:

shell> mysqldump --databases test > dump.sql

In the single-database case, it is permissible to omit the [--databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_databases) option:

shell> mysqldump test > dump.sql

The difference between the two preceding commands is that without [--databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_databases), the dump output contains no [CREATE DATABASE](https://dev.mysql.com/doc/refman/8.0/en/create-database.html) or [USE](https://dev.mysql.com/doc/refman/8.0/en/use.html) statements. This has several implications:

* When you reload the dump file, you must specify a default database name so that the server knows which database to reload.
* For reloading, you can specify a database name different from the original name, which enables you to reload the data into a different database.
* If the database to be reloaded does not exist, you must create it first.
* Because the output contains no [CREATE DATABASE](https://dev.mysql.com/doc/refman/8.0/en/create-database.html) statement, the [--add-drop-database](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_add-drop-database) option has no effect. If you use it, it produces no [DROP DATABASE](https://dev.mysql.com/doc/refman/8.0/en/drop-database.html) statement.

To dump only specific tables from a database, name them on the command line following the database name:

shell> mysqldump test t1 t3 t7 > dump.sql

By default, if GTIDs are in use on the server where you create the dump file ([gtid\_mode=ON](https://dev.mysql.com/doc/refman/8.0/en/replication-options-gtids.html#sysvar_gtid_mode)), [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) includes a SET @@GLOBAL.gtid\_purged statement in the output to add the GTIDs from the [gtid\_executed](https://dev.mysql.com/doc/refman/8.0/en/replication-options-gtids.html#sysvar_gtid_executed) set on the source server to the [gtid\_purged](https://dev.mysql.com/doc/refman/8.0/en/replication-options-gtids.html#sysvar_gtid_purged) set on the target server. If you are dumping only specific databases or tables, it is important to note that the value that is included by [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) includes the GTIDs of all transactions in the [gtid\_executed](https://dev.mysql.com/doc/refman/8.0/en/replication-options-gtids.html#sysvar_gtid_executed) set on the source server, even those that changed suppressed parts of the database, or other databases on the server that were not included in the partial dump. If you only replay one partial dump file on the target server, the extra GTIDs do not cause any problems with the future operation of that server. However, if you replay a second dump file on the target server that contains the same GTIDs (for example, another partial dump from the same source server), any SET @@GLOBAL.gtid\_purged statement in the second dump file fails. To avoid this issue, either set the [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) option --set-gtid-purged to OFF or COMMENTED to output the second dump file without an active SET @@GLOBAL.gtid\_purged statement, or remove the statement manually before replaying the dump file.

### Reloading SQL-Format Backups

To reload a dump file written by [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) that consists of SQL statements, use it as input to the [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) client. If the dump file was created by [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) with the [--all-databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_all-databases) or [--databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_databases) option, it contains [CREATE DATABASE](https://dev.mysql.com/doc/refman/8.0/en/create-database.html) and [USE](https://dev.mysql.com/doc/refman/8.0/en/use.html) statements and it is not necessary to specify a default database into which to load the data:

shell> mysql < dump.sql

Alternatively, from within [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html), use a source command:

mysql> source dump.sql

If the file is a single-database dump not containing [CREATE DATABASE](https://dev.mysql.com/doc/refman/8.0/en/create-database.html) and [USE](https://dev.mysql.com/doc/refman/8.0/en/use.html) statements, create the database first (if necessary):

shell> mysqladmin create db1

Then specify the database name when you load the dump file:

shell> mysql db1 < dump.sql

Alternatively, from within [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html), create the database, select it as the default database, and load the dump file:

mysql> CREATE DATABASE IF NOT EXISTS db1;

mysql> USE db1;

mysql> source dump.sql

**Note**

For Windows PowerShell users: Because the "<" character is reserved for future use in PowerShell, an alternative approach is required, such as using quotes cmd.exe /c "mysql < dump.sql".

### Dumping Data in Delimited-Text Format with mysqldump

This section describes how to use [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) to create delimited-text dump files. For information about reloading such dump files, see [Section 7.4.4, “Reloading Delimited-Text Format Backups”](https://dev.mysql.com/doc/refman/8.0/en/reloading-delimited-text-dumps.html).

If you invoke [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) with the [--tab=***dir\_name***](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_tab) option, it uses ***dir\_name*** as the output directory and dumps tables individually in that directory using two files for each table. The table name is the base name for these files. For a table named t1, the files are named t1.sql and t1.txt. The .sql file contains a [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) statement for the table. The .txt file contains the table data, one line per table row.

The following command dumps the contents of the db1 database to files in the /tmp database:

shell> mysqldump --tab=/tmp db1

The .txt files containing table data are written by the server, so they are owned by the system account used for running the server. The server uses [SELECT ... INTO OUTFILE](https://dev.mysql.com/doc/refman/8.0/en/select-into.html) to write the files, so you must have the [FILE](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_file) privilege to perform this operation, and an error occurs if a given .txt file already exists.

The server sends the CREATE definitions for dumped tables to [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html), which writes them to .sql files. These files therefore are owned by the user who executes [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html).

It is best that [--tab](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_tab) be used only for dumping a local server. If you use it with a remote server, the [--tab](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_tab) directory must exist on both the local and remote hosts, and the .txt files are written by the server in the remote directory (on the server host), whereas the .sql files are written by [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) in the local directory (on the client host).

For [**mysqldump --tab**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html), the server by default writes table data to .txt files one line per row with tabs between column values, no quotation marks around column values, and newline as the line terminator. (These are the same defaults as for [SELECT ... INTO OUTFILE](https://dev.mysql.com/doc/refman/8.0/en/select-into.html).)

To enable data files to be written using a different format, [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) supports these options:

* [--fields-terminated-by=***str***](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_fields)

The string for separating column values (default: tab).

* [--fields-enclosed-by=***char***](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_fields)

The character within which to enclose column values (default: no character).

* [--fields-optionally-enclosed-by=***char***](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_fields)

The character within which to enclose non-numeric column values (default: no character).

* [--fields-escaped-by=***char***](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_fields)

The character for escaping special characters (default: no escaping).

* [--lines-terminated-by=***str***](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_lines-terminated-by)

The line-termination string (default: newline).

Depending on the value you specify for any of these options, it might be necessary on the command line to quote or escape the value appropriately for your command interpreter. Alternatively, specify the value using hex notation. Suppose that you want [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) to quote column values within double quotation marks. To do so, specify double quote as the value for the [--fields-enclosed-by](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_fields) option. But this character is often special to command interpreters and must be treated specially. For example, on Unix, you can quote the double quote like this:

--fields-enclosed-by='"'

On any platform, you can specify the value in hex:

--fields-enclosed-by=0x22

It is common to use several of the data-formatting options together. For example, to dump tables in comma-separated values format with lines terminated by carriage-return/newline pairs (\r\n), use this command (enter it on a single line):

shell> mysqldump --tab=/tmp --fields-terminated-by=,

 --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1

Should you use any of the data-formatting options to dump table data, you need to specify the same format when you reload data files later, to ensure proper interpretation of the file contents.

### Reloading Delimited-Text Format Backups

For backups produced with [**mysqldump --tab**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html), each table is represented in the output directory by an .sql file containing the [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) statement for the table, and a .txt file containing the table data. To reload a table, first change location into the output directory. Then process the .sql file with [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) to create an empty table and process the .txt file to load the data into the table:

shell> mysql db1 < t1.sql

shell> mysqlimport db1 t1.txt

An alternative to using [**mysqlimport**](https://dev.mysql.com/doc/refman/8.0/en/mysqlimport.html) to load the data file is to use the [LOAD DATA](https://dev.mysql.com/doc/refman/8.0/en/load-data.html) statement from within the [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) client:

mysql> USE db1;

mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1;

If you used any data-formatting options with [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) when you initially dumped the table, you must use the same options with [**mysqlimport**](https://dev.mysql.com/doc/refman/8.0/en/mysqlimport.html) or [LOAD DATA](https://dev.mysql.com/doc/refman/8.0/en/load-data.html) to ensure proper interpretation of the data file contents:

shell> mysqlimport --fields-terminated-by=,

 --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1 t1.txt

Or:

mysql> USE db1;

mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1

 FIELDS TERMINATED BY ',' FIELDS ENCLOSED BY '"'

 LINES TERMINATED BY '\r\n';

### mysqldump Tips

This section surveys techniques that enable you to use [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) to solve specific problems:

* How to make a copy a database
* How to copy a database from one server to another
* How to dump stored programs (stored procedures and functions, triggers, and events)
* How to dump definitions and data separately

#### Making a Copy of a Database

shell> mysqldump db1 > dump.sql

shell> mysqladmin create db2

shell> mysql db2 < dump.sql

Do not use [--databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_databases) on the [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) command line because that causes USE db1 to be included in the dump file, which overrides the effect of naming db2 on the [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) command line.

#### Copy a Database from one Server to Another

On Server 1:

shell> mysqldump --databases db1 > dump.sql

Copy the dump file from Server 1 to Server 2.

On Server 2:

shell> mysql < dump.sql

Use of [--databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_databases) with the [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) command line causes the dump file to include [CREATE DATABASE](https://dev.mysql.com/doc/refman/8.0/en/create-database.html) and [USE](https://dev.mysql.com/doc/refman/8.0/en/use.html) statements that create the database if it does exist and make it the default database for the reloaded data.

Alternatively, you can omit [--databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_databases) from the [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) command. Then you need to create the database on Server 2 (if necessary) and specify it as the default database when you reload the dump file.

On Server 1:

shell> mysqldump db1 > dump.sql

On Server 2:

shell> mysqladmin create db1

shell> mysql db1 < dump.sql

You can specify a different database name in this case, so omitting [--databases](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_databases) from the [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) command enables you to dump data from one database and load it into another.

#### Dumping Stored Programs

Several options control how [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) handles stored programs (stored procedures and functions, triggers, and events):

* [--events](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_events): Dump Event Scheduler events
* [--routines](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_routines): Dump stored procedures and functions
* [--triggers](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_triggers): Dump triggers for tables

The [--triggers](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_triggers) option is enabled by default so that when tables are dumped, they are accompanied by any triggers they have. The other options are disabled by default and must be specified explicitly to dump the corresponding objects. To disable any of these options explicitly, use its skip form: [--skip-events](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_events), [--skip-routines](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_routines), or [--skip-triggers](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_triggers).

#### Dumping Table Definitions and Content Separately

The [--no-data](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_no-data) option tells [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) not to dump table data, resulting in the dump file containing only statements to create the tables. Conversely, the [--no-create-info](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_no-create-info) option tells [**mysqldump**](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html) to suppress CREATE statements from the output, so that the dump file contains only table data.

For example, to dump table definitions and data separately for the test database, use these commands:

shell> mysqldump --no-data test > dump-defs.sql

shell> mysqldump --no-create-info test > dump-data.sql

For a definition-only dump, add the [--routines](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_routines) and [--events](https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html#option_mysqldump_events) options to also include stored routine and event definitions:

shell> mysqldump --no-data --routines --events test > dump-defs.sql

#### Using mysqldump to Test for Upgrade Incompatibilities

When contemplating a MySQL upgrade, it is prudent to install the newer version separately from your current production version. Then you can dump the database and database object definitions from the production server and load them into the new server to verify that they are handled properly. (This is also useful for testing downgrades.)

On the production server:

shell> mysqldump --all-databases --no-data --routines --events > dump-defs.sql

On the upgraded server:

shell> mysql < dump-defs.sql

Because the dump file does not contain table data, it can be processed quickly. This enables you to spot potential incompatibilities without waiting for lengthy data-loading operations. Look for warnings or errors while the dump file is being processed.

After you have verified that the definitions are handled properly, dump the data and try to load it into the upgraded server.

On the production server:

shell> mysqldump --all-databases --no-create-info > dump-data.sql

On the upgraded server:

shell> mysql < dump-data.sql

Now check the table contents and run some test queries.

## Point-in-Time (Incremental) Recovery

Point-in-time recovery refers to recovery of data changes up to a given point in time. Typically, this type of recovery is performed after restoring a full backup that brings the server to its state as of the time the backup was made. (The full backup can be made in several ways, such as those listed in [Section 7.2, “Database Backup Methods”](https://dev.mysql.com/doc/refman/8.0/en/backup-methods.html).) Point-in-time recovery then brings the server up to date incrementally from the time of the full backup to a more recent time.

### Point-in-Time Recovery Using Binary Log

This section explains the general idea of using the binary log to perform a point-in-time-recovery. The next section, [Section  “Point-in-Time Recovery Using Event Positions”](https://dev.mysql.com/doc/refman/8.0/en/point-in-time-recovery-positions.html), explains the operation in details with an example.

**Note**

Many of the examples in this and the next section use the [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) client to process binary log output produced by [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html). If your binary log contains \0 (null) characters, that output cannot be parsed by [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) unless you invoke it with the [--binary-mode](https://dev.mysql.com/doc/refman/8.0/en/mysql-command-options.html#option_mysql_binary-mode) option.

The source of information for point-in-time recovery is the set of binary log files generated subsequent to the full backup operation. Therefore, to allow a server to be restored to a point-in-time, binary logging must be enabled on it, which is the default setting for MySQL 8.0 (see [Section 5.4.4, “The Binary Log”](https://dev.mysql.com/doc/refman/8.0/en/binary-log.html)).

To restore data from the binary log, you must know the name and location of the current binary log files. By default, the server creates binary log files in the data directory, but a path name can be specified with the [--log-bin](https://dev.mysql.com/doc/refman/8.0/en/replication-options-binary-log.html#option_mysqld_log-bin) option to place the files in a different location. To see a listing of all binary log files, use this statement:

mysql> SHOW BINARY LOGS;

To determine the name of the current binary log file, issue the following statement:

mysql> SHOW MASTER STATUS;

The [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html) utility converts the events in the binary log files from binary format to text so that they can be viewed or applied. [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html) has options for selecting sections of the binary log based on event times or position of events within the log.

Applying events from the binary log causes the data modifications they represent to be reexecuted. This enables recovery of data changes for a given span of time. To apply events from the binary log, process [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html) output using the [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) client:

shell> mysqlbinlog binlog\_files | mysql -u root -p

If binary log files have been encrypted, which can be done from MySQL 8.0.14 onwards, [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html) cannot read them directly as in the above example, but can read them from the server using the [--read-from-remote-server](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html#option_mysqlbinlog_read-from-remote-server) (-R) option. For example:

shell> mysqlbinlog --read-from-remote-server --host=host\_name --port=3306 --user=root --password --ssl-mode=required binlog\_files | mysql -u root -p

Here, the option --ssl-mode=required has been used to ensure that the data from the binary log files is protected in transit, because it is sent to [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html) in an unencrypted format.

Viewing log contents can be useful when you need to determine event times or positions to select partial log contents prior to executing events. To view events from the log, send [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html) output into a paging program:

shell> mysqlbinlog binlog\_files | more

Alternatively, save the output in a file and view the file in a text editor:

shell> mysqlbinlog binlog\_files > tmpfile

shell> ... edit tmpfile ...

Saving the output in a file is useful as a preliminary to executing the log contents with certain events removed, such as an accidental [DROP TABLE](https://dev.mysql.com/doc/refman/8.0/en/drop-table.html). You can delete from the file any statements not to be executed before executing its contents. After editing the file, apply the contents as follows:

shell> mysql -u root -p < tmpfile

If you have more than one binary log to apply on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be unsafe:

shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!

shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!

Processing binary logs this way using different connections to the server causes problems if the first log file contains a [CREATE TEMPORARY TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) statement and the second log contains a statement that uses the temporary table. When the first [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) process terminates, the server drops the temporary table. When the second [**mysql**](https://dev.mysql.com/doc/refman/8.0/en/mysql.html) process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a single connection to apply the contents of all binary log files that you want to process. Here is one way to do so:

shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p

Another approach is to write the whole log to a single file and then process the file:

shell> mysqlbinlog binlog.000001 > /tmp/statements.sql

shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql

shell> mysql -u root -p -e "source /tmp/statements.sql"

When writing to a dump file while reading back from a binary log containing GTIDs (see [Section 17.1.3, “Replication with Global Transaction Identifiers”](https://dev.mysql.com/doc/refman/8.0/en/replication-gtids.html)), use the [--skip-gtids](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html#option_mysqlbinlog_skip-gtids) option with [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html), like this:

shell> mysqlbinlog --skip-gtids binlog.000001 > /tmp/dump.sql

shell> mysqlbinlog --skip-gtids binlog.000002 >> /tmp/dump.sql

shell> mysql -u root -p -e "source /tmp/dump.sql"

### Point-in-Time Recovery Using Event Positions

The last section, [Section “Point-in-Time Recovery Using Binary Log”](https://dev.mysql.com/doc/refman/8.0/en/point-in-time-recovery-binlog.html), explains the general idea of using the binary log to perform a point-in-time-recovery. The section explains the operation in details with an example.

As an example, suppose that around 20:06:00 on March 11, 2020, an SQL statement was executed that deleted a table. You can perform a point-in-time recovery to restore the server up to its state right before the table deletion. These are some sample steps to achieve that:

1. Restore the last full backup created before the point-in-time of interest (call it tp, which is 20:06:00 on March 11, 2020 in our example). When finished, note the binary log position up to which you have restored the server for later use, and restart the server.

**Note**

While the last binary log position recovered is also displayed by InnoDB after the restore and server restart, that is not a reliable means for obtaining the ending log position of your restore, as there could be DDL events and non-InnoDB changes that have taken place after the time reflected by the displayed position. Your backup and restore tool should provide you with the last binary log position for your recovery: for example, if you are using [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html) for the task, check the stop position of the binary log replay; if you are using MySQL Enterprise Backup, the last binary log position has been saved in your backup. See [Point-in-Time Recovery](https://dev.mysql.com/doc/mysql-enterprise-backup/8.0/en/advanced.point.html).

1. Find the precise binary log event position corresponding to the point in time up to which you want to restore your database. In our example, given that we know the rough time where the table deletion took place (tp), we can find the log position by checking the log contents around that time using the [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html) utility. Use the [--start-datetime](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html#option_mysqlbinlog_start-datetime) and [--stop-datetime](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html#option_mysqlbinlog_stop-datetime) options to specify a short time period around tp, and then look for the event in the output. For example:
2. shell> mysqlbinlog --start-datetime="2020-03-11 20:05:00" \
3. --stop-datetime="2020-03-11 20:08:00" --verbose \
4. /var/lib/mysql/bin.123456 | grep -C 15 "DROP TABLE"
5.
6. /\*!80014 SET @@session.original\_server\_version=80019\*//\*!\*/;
7. /\*!80014 SET @@session.immediate\_server\_version=80019\*//\*!\*/;
8. SET @@SESSION.GTID\_NEXT= 'ANONYMOUS'/\*!\*/;
9. # at 232
10. #200311 20:06:20 server id 1 end\_log\_pos 355 CRC32 0x2fc1e5ea Query thread\_id=16 exec\_time=0 error\_code=0
11. SET TIMESTAMP=1583971580/\*!\*/;
12. SET @@session.pseudo\_thread\_id=16/\*!\*/;
13. SET @@session.foreign\_key\_checks=1, @@session.sql\_auto\_is\_null=0, @@session.unique\_checks=1, @@session.autocommit=1/\*!\*/;
14. SET @@session.sql\_mode=1168113696/\*!\*/;
15. SET @@session.auto\_increment\_increment=1, @@session.auto\_increment\_offset=1/\*!\*/;
16. /\*!\C utf8mb4 \*//\*!\*/;
17. SET @@session.character\_set\_client=255,@@session.collation\_connection=255,@@session.collation\_server=255/\*!\*/;
18. SET @@session.lc\_time\_names=0/\*!\*/;
19. SET @@session.collation\_database=DEFAULT/\*!\*/;
20. /\*!80011 SET @@session.default\_collation\_for\_utf8mb4=255\*//\*!\*/;
21. DROP TABLE `pets`.`cats` /\* generated by server \*/
22. /\*!\*/;
23. # at 355
24. #200311 20:07:48 server id 1 end\_log\_pos 434 CRC32 0x123d65df Anonymous\_GTID last\_committed=1 sequence\_number=2 rbr\_only=no original\_committed\_timestamp=1583971668462467 immediate\_commit\_timestamp=1583971668462467 transaction\_length=473
25. # original\_commit\_timestamp=1583971668462467 (2020-03-11 20:07:48.462467 EDT)
26. # immediate\_commit\_timestamp=1583971668462467 (2020-03-11 20:07:48.462467 EDT)
27. /\*!80001 SET @@session.original\_commit\_timestamp=1583971668462467\*//\*!\*/;
28. /\*!80014 SET @@session.original\_server\_version=80019\*//\*!\*/;
29. /\*!80014 SET @@session.immediate\_server\_version=80019\*//\*!\*/;
30. SET @@SESSION.GTID\_NEXT= 'ANONYMOUS'/\*!\*/;
31. # at 434
32. #200311 20:07:48 server id 1 end\_log\_pos 828 CRC32 0x57fac9ac Query thread\_id=16 exec\_time=0 error\_code=0 Xid = 217
33. use `pets`/\*!\*/;
34. SET TIMESTAMP=1583971668/\*!\*/;
35. /\*!80013 SET @@session.sql\_require\_primary\_key=0\*//\*!\*/;

CREATE TABLE dogs

From the output of [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html), the DROP TABLE `pets`.`cats` statement can be found in the segment of the binary log between the line # at 232 and # at 355, which means the statement takes place after the log position 232, and the log is at position 355 after the DROP TABLE statement.

**Note**

Only use the [--start-datetime](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html#option_mysqlbinlog_start-datetime) and [--stop-datetime](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html#option_mysqlbinlog_stop-datetime) options to help you find the actual event positions of interest. Using the two options to specify the range of binary log segment to apply is not recommended: there is a higher risk of missing binary log events when using the options. Use [--start-position](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html#option_mysqlbinlog_start-position) and [--stop-position](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html#option_mysqlbinlog_stop-position) instead.

1. Apply the events in binary log file to the server, starting with the log position your found in step 1 (assume it is 155) and ending at the position you have found in step 2 that is before your point-in-time of interest (which is 232):
2. shell> mysqlbinlog --start-position=155 --stop-position=232 /var/lib/mysql/bin.123456 \

 | mysql -u root -p

The command recovers all the transactions from the starting position until just before the stop position. Because the output of [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html) includes SET TIMESTAMP statements before each SQL statement recorded, the recovered data and related MySQL logs reflect the original times at which the transactions were executed.

Your database has now been restored to the point-in-time of interest, tp, right before the table pets.cats was dropped.

1. Beyond the point-in-time recovery that has been finished, if you also want to reexecute all the statements after your point-in-time of interest, use [**mysqlbinlog**](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html) again to apply all the events after tp to the server. We noted in step 2 that after the statement we wanted to skip, the log is at position 355; we can use it for the [--start-position](https://dev.mysql.com/doc/refman/8.0/en/mysqlbinlog.html#option_mysqlbinlog_start-position) option, so that any statements after the position are included:
2. shell> mysqlbinlog --start-position=355 /var/lib/mysql/bin.123456 \

 | mysql -u root -p

Your database has been restored the latest statement recorded in the binary log file, but with the selected event skipped.

MyISAM Table Maintenance and Crash Recovery

This section discusses how to use [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to check or repair MyISAM tables (tables that have .MYD and .MYI files for storing data and indexes). For general [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) background, see [Section 4.6.4, “**myisamchk** — MyISAM Table-Maintenance Utility”](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html). Other table-repair information can be found at [Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”](https://dev.mysql.com/doc/refman/8.0/en/rebuilding-tables.html).

You can use [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to check, repair, or optimize database tables. The following sections describe how to perform these operations and how to set up a table maintenance schedule. For information about using [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to get information about your tables, see [Section 4.6.4.5, “Obtaining Table Information with myisamchk”](https://dev.mysql.com/doc/refman/8.0/en/myisamchk-table-info.html).

Even though table repair with [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) is quite secure, it is always a good idea to make a backup *before* doing a repair or any maintenance operation that could make a lot of changes to a table.

[**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) operations that affect indexes can cause MyISAM FULLTEXT indexes to be rebuilt with full-text parameters that are incompatible with the values used by the MySQL server. To avoid this problem, follow the guidelines in [Section 4.6.4.1, “myisamchk General Options”](https://dev.mysql.com/doc/refman/8.0/en/myisamchk-general-options.html).

MyISAM table maintenance can also be done using the SQL statements that perform operations similar to what [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) can do:

* To check MyISAM tables, use [CHECK TABLE](https://dev.mysql.com/doc/refman/8.0/en/check-table.html).
* To repair MyISAM tables, use [REPAIR TABLE](https://dev.mysql.com/doc/refman/8.0/en/repair-table.html).
* To optimize MyISAM tables, use [OPTIMIZE TABLE](https://dev.mysql.com/doc/refman/8.0/en/optimize-table.html).
* To analyze MyISAM tables, use [ANALYZE TABLE](https://dev.mysql.com/doc/refman/8.0/en/analyze-table.html).

For additional information about these statements, see [Section 13.7.3, “Table Maintenance Statements”](https://dev.mysql.com/doc/refman/8.0/en/table-maintenance-statements.html).

These statements can be used directly or by means of the [**mysqlcheck**](https://dev.mysql.com/doc/refman/8.0/en/mysqlcheck.html) client program. One advantage of these statements over [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) is that the server does all the work. With [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html), you must make sure that the server does not use the tables at the same time so that there is no unwanted interaction between [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) and the server.

### Using myisamchk for Crash Recovery

This section describes how to check for and deal with data corruption in MySQL databases. If your tables become corrupted frequently, you should try to find the reason why.

For an explanation of how MyISAM tables can become corrupted, see [Section 16.2.4, “MyISAM Table Problems”](https://dev.mysql.com/doc/refman/8.0/en/myisam-table-problems.html).

If you run [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html) with external locking disabled (which is the default), you cannot reliably use [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to check a table when [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html) is using the same table. If you can be certain that no one can access the tables using [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html) while you run [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html), you only have to execute [**mysqladmin flush-tables**](https://dev.mysql.com/doc/refman/8.0/en/mysqladmin.html) before you start checking the tables. If you cannot guarantee this, you must stop [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html) while you check the tables. If you run [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to check tables that [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html) is updating at the same time, you may get a warning that a table is corrupt even when it is not.

If the server is run with external locking enabled, you can use [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to check tables at any time. In this case, if the server tries to update a table that [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) is using, the server waits for [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to finish before it continues.

If you use [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to repair or optimize tables, you must always ensure that the [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html) server is not using the table (this also applies if external locking is disabled). If you do not stop [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html), you should at least do a [**mysqladmin flush-tables**](https://dev.mysql.com/doc/refman/8.0/en/mysqladmin.html) before you run [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html). Your tables may become corrupted if the server and [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) access the tables simultaneously.

When performing crash recovery, it is important to understand that each MyISAM table ***tbl\_name*** in a database corresponds to the three files in the database directory shown in the following table.

| **File** | **Purpose** |
| --- | --- |
| ***tbl\_name***.MYD | Data file |
| ***tbl\_name***.MYI | Index file |

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

[**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) works by creating a copy of the .MYD data file row by row. It ends the repair stage by removing the old .MYD file and renaming the new file to the original file name. If you use [--quick](https://dev.mysql.com/doc/refman/8.0/en/myisamchk-repair-options.html#option_myisamchk_quick), [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) does not create a temporary .MYD file, but instead assumes that the .MYD file is correct and generates only a new index file without touching the .MYD file. This is safe, because [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) automatically detects whether the .MYD file is corrupt and aborts the repair if it is. You can also specify the [--quick](https://dev.mysql.com/doc/refman/8.0/en/myisamchk-repair-options.html#option_myisamchk_quick) option twice to [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html). In this case, [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the .MYD file. Normally the use of two [--quick](https://dev.mysql.com/doc/refman/8.0/en/myisamchk-repair-options.html#option_myisamchk_quick) options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup of the table before running [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html).

### How to Check MyISAM Tables for Errors

To check a MyISAM table, use the following commands:

* [**myisamchk *tbl\_name***](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html)

This finds 99.99% of all errors. What it cannot find is corruption that involves only the data file (which is very unusual). If you want to check a table, you should normally run [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) without options or with the -s (silent) option.

* [**myisamchk -m *tbl\_name***](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html)

This finds 99.999% of all errors. It first checks all index entries for errors and then reads through all rows. It calculates a checksum for all key values in the rows and verifies that the checksum matches the checksum for the keys in the index tree.

* [**myisamchk -e *tbl\_name***](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html)

This does a complete and thorough check of all data (-e means “extended check”). It does a check-read of every key for each row to verify that they indeed point to the correct row. This may take a long time for a large table that has many indexes. Normally, [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) stops after the first error it finds. If you want to obtain more information, you can add the -v (verbose) option. This causes [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to keep going, up through a maximum of 20 errors.

* [**myisamchk -e -i *tbl\_name***](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html)

This is like the previous command, but the -i option tells [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to print additional statistical information.

In most cases, a simple [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) command with no arguments other than the table name is sufficient to check a table.

### How to Repair MyISAM Tables

The discussion in this section describes how to use [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) on MyISAM tables (extensions .MYI and .MYD).

You can also use the [CHECK TABLE](https://dev.mysql.com/doc/refman/8.0/en/check-table.html) and [REPAIR TABLE](https://dev.mysql.com/doc/refman/8.0/en/repair-table.html) statements to check and repair MyISAM tables.

Symptoms of corrupted tables include queries that abort unexpectedly and observable errors such as these:

* Can't find file ***tbl\_name***.MYI (Errcode: ***nnn***)
* Unexpected end of file
* Record file is crashed
* Got error ***nnn*** from table handler

To get more information about the error, run [**perror**](https://dev.mysql.com/doc/refman/8.0/en/perror.html) ***nnn***, where ***nnn*** is the error number. The following example shows how to use [**perror**](https://dev.mysql.com/doc/refman/8.0/en/perror.html) to find the meanings for the most common error numbers that indicate a problem with a table:

shell> perror 126 127 132 134 135 136 141 144 145

MySQL error code 126 = Index file is crashed

MySQL error code 127 = Record-file is crashed

MySQL error code 132 = Old database file

MySQL error code 134 = Record was already deleted (or record file crashed)

MySQL error code 135 = No more room in record file

MySQL error code 136 = No more room in index file

MySQL error code 141 = Duplicate unique key or constraint on write or update

MySQL error code 144 = Table is crashed and last repair failed

MySQL error code 145 = Table was marked as crashed and should be repaired

Note that error 135 (no more room in record file) and error 136 (no more room in index file) are not errors that can be fixed by a simple repair. In this case, you must use [ALTER TABLE](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html) to increase the MAX\_ROWS and AVG\_ROW\_LENGTH table option values:

ALTER TABLE tbl\_name MAX\_ROWS=xxx AVG\_ROW\_LENGTH=yyy;

If you do not know the current table option values, use [SHOW CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/show-create-table.html).

For the other errors, you must repair your tables. [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) can usually detect and fix most problems that occur.

The repair process involves up to three stages, described here. Before you begin, you should change location to the database directory and check the permissions of the table files. On Unix, make sure that they are readable by the user that [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html) runs as (and to you, because you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

This section is for the cases where a table check fails (such as those described in [Section 7.6.2, “How to Check MyISAM Tables for Errors”](https://dev.mysql.com/doc/refman/8.0/en/myisam-check.html)), or you want to use the extended features that [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) provides.

The [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) options used for table maintenance with are described in [Section 4.6.4, “**myisamchk** — MyISAM Table-Maintenance Utility”](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html). [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) also has variables that you can set to control memory allocation that may improve performance. See [Section 4.6.4.6, “myisamchk Memory Usage”](https://dev.mysql.com/doc/refman/8.0/en/myisamchk-memory.html).

If you are going to repair a table from the command line, you must first stop the [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html) server. Note that when you do [**mysqladmin shutdown**](https://dev.mysql.com/doc/refman/8.0/en/mysqladmin.html) on a remote server, the [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html) server is still available for a while after [**mysqladmin**](https://dev.mysql.com/doc/refman/8.0/en/mysqladmin.html) returns, until all statement-processing has stopped and all index changes have been flushed to disk.

***Stage 1: Checking your tables***

Run [**myisamchk \*.MYI**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) or [**myisamchk -e \*.MYI**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) if you have more time. Use the -s (silent) option to suppress unnecessary information.

If the [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html) server is stopped, you should use the [--update-state](https://dev.mysql.com/doc/refman/8.0/en/myisamchk-check-options.html#option_myisamchk_update-state) option to tell [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to mark the table as “checked.”

You have to repair only those tables for which [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) announces an error. For such tables, proceed to Stage 2.

If you get unexpected errors when checking (such as out of memory errors), or if [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) crashes, go to Stage 3.

***Stage 2: Easy safe repair***

First, try [**myisamchk -r -q *tbl\_name***](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) (-r -q means “quick recovery mode”). This attempts to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work, and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.
2. Use [**myisamchk -r *tbl\_name***](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) (-r means “recovery mode”). This removes incorrect rows and deleted rows from the data file and reconstructs the index file.
3. If the preceding step fails, use [**myisamchk --safe-recover *tbl\_name***](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html). Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode does not (but is slower).

**Note**

If you want a repair operation to go much faster, you should set the values of the [sort\_buffer\_size](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_sort_buffer_size) and [key\_buffer\_size](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_key_buffer_size) variables each to about 25% of your available memory when running [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html).

If you get unexpected errors when repairing (such as out of memory errors), or if [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) crashes, go to Stage 3.

***Stage 3: Difficult repair***

You should reach this stage only if the first 16KB block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it is necessary to create a new index file. Do so as follows:

1. Move the data file to a safe place.
2. Use the table description file to create new (empty) data and index files:

shell> mysql db\_name

mysql> SET autocommit=1;

mysql> TRUNCATE TABLE tbl\_name;

mysql> quit

1. Copy the old data file back onto the newly created data file. (Do not just move the old file back onto the new file. You want to retain a copy in case something goes wrong.)

**Important**

If you are using replication, you should stop it prior to performing the above procedure, since it involves file system operations, and these are not logged by MySQL.

Go back to Stage 2. [**myisamchk -r -q**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) should work. (This should not be an endless loop.)

You can also use the REPAIR TABLE ***tbl\_name*** USE\_FRM SQL statement, which performs the whole procedure automatically. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use [REPAIR TABLE](https://dev.mysql.com/doc/refman/8.0/en/repair-table.html).

### MyISAM Table Optimization

To coalesce fragmented rows and eliminate wasted space that results from deleting or updating rows, run [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) in recovery mode:

shell> myisamchk -r tbl\_name

You can optimize a table in the same way by using the [OPTIMIZE TABLE](https://dev.mysql.com/doc/refman/8.0/en/optimize-table.html) SQL statement. [OPTIMIZE TABLE](https://dev.mysql.com/doc/refman/8.0/en/optimize-table.html) does a table repair and a key analysis, and also sorts the index tree so that key lookups are faster. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use [OPTIMIZE TABLE](https://dev.mysql.com/doc/refman/8.0/en/optimize-table.html). See [Section 13.7.3.4, “OPTIMIZE TABLE Statement”](https://dev.mysql.com/doc/refman/8.0/en/optimize-table.html).

[**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) has a number of other options that you can use to improve the performance of a table:

* [--analyze](https://dev.mysql.com/doc/refman/8.0/en/myisamchk-other-options.html#option_myisamchk_analyze) or -a: Perform key distribution analysis. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use.
* [--sort-index](https://dev.mysql.com/doc/refman/8.0/en/myisamchk-other-options.html#option_myisamchk_sort-index) or -S: Sort the index blocks. This optimizes seeks and makes table scans that use indexes faster.
* [--sort-records=***index\_num***](https://dev.mysql.com/doc/refman/8.0/en/myisamchk-other-options.html#option_myisamchk_sort-records) or -R ***index\_num***: Sort data rows according to a given index. This makes your data much more localized and may speed up range-based [SELECT](https://dev.mysql.com/doc/refman/8.0/en/select.html) and ORDER BY operations that use this index.

### Setting Up a MyISAM Table Maintenance Schedule

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. One way to check and repair MyISAM tables is with the [CHECK TABLE](https://dev.mysql.com/doc/refman/8.0/en/check-table.html) and [REPAIR TABLE](https://dev.mysql.com/doc/refman/8.0/en/repair-table.html) statements. See [Section 13.7.3, “Table Maintenance Statements”](https://dev.mysql.com/doc/refman/8.0/en/table-maintenance-statements.html).

Another way to check tables is to use [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html). For maintenance purposes, you can use [**myisamchk -s**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html). The -s option (short for [--silent](https://dev.mysql.com/doc/refman/8.0/en/myisamchk-general-options.html#option_myisamchk_silent)) causes [**myisamchk**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) to run in silent mode, printing messages only when errors occur.

It is also a good idea to enable automatic MyISAM table checking. For example, whenever the machine has done a restart in the middle of an update, you usually need to check each table that could have been affected before it is used further. (These are “expected crashed tables.”) To cause the server to check MyISAM tables automatically, start it with the [myisam\_recover\_options](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_myisam_recover_options) system variable set. See [Section 5.1.8, “Server System Variables”](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html).

You should also check your tables regularly during normal system operation. For example, you can run a **cron** job to check important tables once a week, using a line like this in a crontab file:

35 0 \* \* 0 */*path*/*to*/*myisamchk --fast --silent */*path*/*to*/*datadir/\*/\*.MYI

This prints out information about crashed tables so that you can examine and repair them as necessary.

To start with, execute [**myisamchk -s**](https://dev.mysql.com/doc/refman/8.0/en/myisamchk.html) each night on all tables that have been updated during the last 24 hours. As you see that problems occur infrequently, you can back off the checking frequency to once a week or so.

Normally, MySQL tables need little maintenance. If you are performing many updates to MyISAM tables with dynamic-sized rows (tables with [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html), or [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) columns) or have tables with many deleted rows you may want to defragment/reclaim space from the tables from time to time. You can do this by using [OPTIMIZE TABLE](https://dev.mysql.com/doc/refman/8.0/en/optimize-table.html) on the tables in question. Alternatively, if you can stop the [**mysqld**](https://dev.mysql.com/doc/refman/8.0/en/mysqld.html) server for a while, change location into the data directory and use this command while the server is stopped:

shell> myisamchk -r -s --sort-index --myisam\_sort\_buffer\_size=16M \*/\*.MYI