[TCL (MySQL Transaction Control Language) 1](#_Toc57747159)

[Introduction on Transaction 1](#_Toc57747160)

[Understand the concept of a transaction 2](#_Toc57747161)

[MySQL and the ACID Model 2](#_Toc57747162)

[MySQL Transaction 4](#_Toc57747163)

[SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT 7](#_Toc57747164)

[LOCK and UNLOCK Tables 7](#_Toc57747165)

[SET TRANSACTION Syntax 8](#_Toc57747166)

[Transactional and Locking Statements 8](#_Toc57747167)

[START TRANSACTION, COMMIT, and ROLLBACK Statements 9](#_Toc57747168)

[Statements That Cannot Be Rolled Back 11](#_Toc57747169)

[Statements That Cause an Implicit Commit 11](#_Toc57747170)

[SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Statements 12](#_Toc57747171)

[LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Statements 13](#_Toc57747172)

[LOCK TABLES and UNLOCK TABLES Statements 13](#_Toc57747173)

[Table Lock Acquisition 14](#_Toc57747174)

[Table Lock Release 15](#_Toc57747175)

[Interaction of Table Locking and Transactions 16](#_Toc57747176)

[LOCK TABLES and Triggers 16](#_Toc57747177)

[Table-Locking Restrictions and Conditions 17](#_Toc57747178)

[SET TRANSACTION Statement 18](#_Toc57747179)

[Transaction Isolation Levels 18](#_Toc57747180)

[Transaction Access Mode 18](#_Toc57747181)

[Transaction Characteristic Scope 19](#_Toc57747182)

[XA Transactions 20](#_Toc57747183)

# TCL (MySQL Transaction Control Language)

## Introduction on Transaction

A transaction is a logical unit of work that contains one or more SQL statements. Transactions are atomic units of work that can be committed or rolled back. When a transaction makes multiple changes to the database, either all the changes succeed when the transaction is committed, or all the changes are undone when the transaction is rolled back.

A transaction begins with the first executable SQL statement. A transaction ends when it is committed or rolled back, either explicitly with a COMMIT or ROLLBACK statement or implicitly when a DDL (Data Definition Language (DDL) is used to manage table and index structure and CREATE, ALTER, RENAME, DROP and TRUNCATE statements are to name a few data definition elements) statement is issued.

### ****Understand the concept of a transaction****

To understand the concept of a transaction, consider a banking database. Suppose a bank customer transfers money from his savings account (SB a/c) to his current account (CA a/c), the statement will be divided into four blocks :

* Debit SB a/c.
* Credit CA a/c.
* Record in Transaction Journal
* End Transaction

The SQL statement to debit SB a/c is as follows:

UPDATE sb\_accounts SET balance = balance – 1000 WHERE account\_no = 932656 ;

The SQL statement to credit OD a/c is as follows:

UPDATE ca\_accounts SET balance = balance + 1000 WHERE account\_no = 933456 ;

The SQL statement for record in transaction journal is as follows:

INSERT INTO journal VALUES  
(100896, 'Tansaction on Benjamin Hampshair a/c', '26-AUG-08' 932656, 933456, 1000);

The SQL statement for End Transaction is as follows :

COMMIT WORK;

## ****MySQL and the ACID Model****

ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably. In MySQL, InnoDB storage engine supports ACID-compliant features. The following sections discuss how MySQL features, in particular, the InnoDB storage engine, interact with the categories of the ACID model:

**Atomicity:**The atomicity aspect of the ACID model mainly involves InnoDB transactions. Related MySQL features include :

* Autocommit setting.
* COMMIT statement.
* ROLLBACK statement.
* Operational data from the INFORMATION\_SCHEMA tables.

**Consistency:**The consistency aspect of the ACID model mainly involves internal InnoDB processing to protect data from crashes. Related MySQL features include :

* InnoDB doublewrite buffer.
* InnoDB crash recovery.

**Isolation:**The isolation aspect of the ACID model mainly involves InnoDB transactions, in particular, the isolation level that applies to each transaction. Related MySQL features include :

* Autocommit setting.
* SET ISOLATION LEVEL statement.
* The low-level details of InnoDB locking. During performance tuning, you see these details through INFORMATION\_SCHEMA tables.

**Durability:**The durability aspect of the ACID model involves MySQL software features interacting with your particular hardware configuration. Because of the many possibilities depending on the capabilities of your CPU, network, and storage devices, this aspect is the most complicated to provide concrete guidelines for. Related MySQL features include:

* InnoDB doublewrite buffer turned on and off by the innodb\_doublewrite configuration option.
* Configuration option innodb\_flush\_log\_at\_trx\_commit.
* Configuration option sync\_binlog.
* Configuration option innodb\_file\_per\_table.
* Write buffer in a storage device, such as a disk drive, SSD, or RAID array.
* Battery-backed cache in a storage device.
* The operating system used to run MySQL, in particular, its support for the fsync() system call.
* Uninterruptible power supply (UPS) protecting the electrical power to all computer servers and storage devices that run MySQL servers and store MySQL data.
* Your backup strategy, such as frequency and types of backups, and backup retention periods.
* For distributed or hosted data applications, the particular characteristics of the data centers where the hardware for the MySQL servers is located, and network connections between the data centers.

## ****MySQL Transaction****

MySQL (here we maintain version 5.6) supports local transactions (within a given client session) through statements such as SET autocommit, START TRANSACTION, COMMIT, and ROLLBACK. Here is the syntax of START TRANSACTION, COMMIT, and ROLLBACK:

START TRANSACTION

transaction\_characteristic [, transaction\_characteristic] ...]

transaction\_characteristic:

WITH CONSISTENT SNAPSHOT

| READ WRITE

| READ ONLY

BEGIN [WORK]

COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

SET autocommit = {0 | 1}

These statements provide control over use of transactions :

* The START TRANSACTION or BEGIN statement begins a new transaction.
* COMMIT commits the current transaction, making its changes permanent.
* ROLLBACK rolls back the current transaction, canceling its changes.
* The SETautocommit statement disables or enables the default autocommit mode for the current session.

By default, MySQL runs with autocommit mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MySQL stores the update on disk to make it permanent. The change cannot be rolled back.

Currently (by default), MySQL runs with autocommit mode enabled.

mysql> select \* from student\_mast;

+------------+------------------+----------+

| STUDENT\_ID | NAME | ST\_CLASS |

+------------+------------------+----------+

| 2 | Neena Kochhar | 9 |

| 3 | Lex De Haan | 9 |

| 4 | Alexander Hunold | 11 |

+------------+------------------+----------+

3 rows in set (0.09 sec)

Let execute an update command:

mysql>mysql> UPDATE STUDENT\_MAST SET ST\_CLASS=8 WHERE STUDENT\_ID=2;

Query OK, 1 row affected (0.08 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql>mysql> select \* from student\_mast;

+------------+------------------+----------+

| STUDENT\_ID | NAME | ST\_CLASS |

+------------+------------------+----------+

| 2 | Neena Kochhar | 8 |

| 3 | Lex De Haan | 9 |

| 4 | Alexander Hunold | 11 |

+------------+------------------+----------+

3 rows in set (0.00 sec)

Now execute the ROLLBACK command to return in the previous stage :

mysql>mysql> ROLLBACK;

Query OK, 0 rows affected (0.03 sec)

mysql>mysql> select \* from student\_mast;

+------------+------------------+----------+

| STUDENT\_ID | NAME | ST\_CLASS |

+------------+------------------+----------+

| 2 | Neena Kochhar | 8 |

| 3 | Lex De Haan | 9 |

| 4 | Alexander Hunold | 11 |

+------------+------------------+----------+

3 rows in set (0.00 sec)

There is no roll back as MySQL runs with autocommit mode enabled.

To disable autocommit mode, use the START TRANSACTION statement. See the following example :

mysql>mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE STUDENT\_MAST SET ST\_CLASS=10 WHERE STUDENT\_ID=2;

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> select \* from student\_mast;

+------------+------------------+----------+

| STUDENT\_ID | NAME | ST\_CLASS |

+------------+------------------+----------+

| 2 | Neena Kochhar | 10 |

| 3 | Lex De Haan | 9 |

| 4 | Alexander Hunold | 11 |

+------------+------------------+----------+

3 rows in set (0.00 sec)

mysql> ROLLBACK;

Query OK, 0 rows affected (0.07 sec)

mysql> select \* from student\_mast;

+------------+------------------+----------+

| STUDENT\_ID | NAME | ST\_CLASS |

+------------+------------------+----------+

| 2 | Neena Kochhar | 8 |

| 3 | Lex De Haan | 9 |

| 4 | Alexander Hunold | 11 |

+------------+------------------+----------+

3 rows in set (0.00 sec)

**MySQL statements that cannot be Rolled Back and statements that cause an implicit Commit**

In MySQL, some statements cannot be rolled back. DDL statements such as CREATE or DROP databases, CREATE, ALTER or DROP tables or stored routines. You should design a transaction without these statements.

The statements listed in this section (and any synonyms for them) implicitly end any transaction active in the current session, as if you had done a COMMIT before executing the statement.

* Data definition language (DDL) statements that define or modify database objects. ALTER DATABASE ... UPGRADE DATA DIRECTORY NAME, ALTER EVENT, ALTER PROCEDURE, ALTER SERVER, ALTER TABLE, ALTER VIEW, CREATE DATABASE, CREATE EVENT, CREATE INDEX, CREATE PROCEDURE, CREATE SERVER, CREATE TABLE, CREATE TRIGGER, CREATE VIEW, DROP DATABASE, DROP EVENT, DROP INDEX, DROP PROCEDURE, DROP SERVER, DROP TABLE, DROP TRIGGER, DROP VIEW, RENAME TABLE, TRUNCATE TABLE.
* ALTER FUNCTION, CREATE FUNCTION, and DROP FUNCTION also cause an implicit commit when used with stored functions, but not with UDFs. (ALTER FUNCTION can only be used with stored functions.)
* ALTER TABLE, CREATE TABLE, and DROP TABLE do not commit a transaction if the TEMPORARY keyword is used.
* Statements that implicitly use or modify tables in the MySQL database. CREATE USER, DROP USER, GRANT, RENAME USER, REVOKE, SET PASSWORD.
* Transaction-control and locking statements. BEGIN, LOCK TABLES, SET autocommit = 1 (if the value is not already 1), START TRANSACTION, UNLOCK TABLES.
* Data loading statements. LOAD DATA INFILE. LOAD DATA INFILE causes an implicit commit only for tables using the NDB storage engine.
* Administrative statements. ANALYZE TABLE, CACHE INDEX, CHECK TABLE, LOAD INDEX INTO CACHE, OPTIMIZE TABLE, REPAIR TABLE.
* Replication control statements. Beginning with MySQL 5.6.7: START SLAVE, STOP SLAVE, RESET SLAVE, CHANGE MASTER TO.

## ****SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT****

InnoDB supports the SQL statements SAVEPOINT, ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT and the optional WORK keyword for ROLLBACK.

The SAVEPOINT statement sets a named transaction savepoint with a name of the identifier. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The ROLLBACK TO SAVEPOINT statement rolls back a transaction to the named savepoint without terminating the transaction. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but InnoDB does not release the row locks that were stored in memory after the savepoint.

Here is the syntax:

SAVEPOINT identifier

ROLLBACK [WORK] TO [SAVEPOINT] identifier

RELEASE SAVEPOINT identifier

## ****LOCK and UNLOCK Tables****

MySQL enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

LOCK TABLES explicitly acquires table locks for the current client session. Table locks can be acquired for base tables or views. You must have the LOCK TABLES privilege, and the SELECT privilege for each object to be locked.

UNLOCK TABLES explicitly releases any table locks held by the current session. LOCK TABLES implicitly releases any table locks held by the current session before acquiring new locks.

Here is the syntax:

LOCK TABLES

tbl\_name [[AS] alias] lock\_type

[, tbl\_name [[AS] alias] lock\_type] ...

lock\_type:

READ [LOCAL]

| [LOW\_PRIORITY] WRITE

UNLOCK TABLES

### ****SET TRANSACTION Syntax****

SET [GLOBAL | SESSION] TRANSACTION

transaction\_characteristic [, transaction\_characteristic] ...

transaction\_characteristic:

ISOLATION LEVEL level

| READ WRITE

| READ ONLY

level:

REPEATABLE READ

| READ COMMITTED

| READ UNCOMMITTED

| SERIALIZABLE

* With the GLOBAL keyword, the statement applies globally for all subsequent sessions. Existing sessions are unaffected.
* With the SESSION keyword, the statement applies to all subsequent transactions performed within the current session.
* Without any SESSION or GLOBAL keyword, the statement applies to the next (not started) transaction performed within the current session.

## Transactional and Locking Statements

MySQL supports local transactions (within a given client session) through statements such as [SET autocommit](https://dev.mysql.com/doc/refman/8.0/en/commit.html), [START TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/commit.html), [COMMIT](https://dev.mysql.com/doc/refman/8.0/en/commit.html), and [ROLLBACK](https://dev.mysql.com/doc/refman/8.0/en/commit.html). See [Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”](https://dev.mysql.com/doc/refman/8.0/en/commit.html). XA transaction support enables MySQL to participate in distributed transactions as well. See [Section 13.3.8, “XA Transactions”](https://dev.mysql.com/doc/refman/8.0/en/xa.html).

### START TRANSACTION, COMMIT, and ROLLBACK Statements

START TRANSACTION

[transaction\_characteristic [, transaction\_characteristic] ...]

transaction\_characteristic: {

WITH CONSISTENT SNAPSHOT

| READ WRITE

| READ ONLY

}

BEGIN [WORK]

COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

SET autocommit = {0 | 1}

These statements provide control over use of [transactions](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_transaction):

* START TRANSACTION or BEGIN start a new transaction.
* COMMIT commits the current transaction, making its changes permanent.
* ROLLBACK rolls back the current transaction, canceling its changes.
* SET autocommit disables or enables the default autocommit mode for the current session.

By default, MySQL runs with [autocommit](https://dev.mysql.com/doc/refman/8.0/en/glossary.html" \l "glos_autocommit" \o "autocommit) mode enabled. This means that, when not otherwise inside a transaction, each statement is atomic, as if it were surrounded by START TRANSACTION and COMMIT. You cannot use ROLLBACK to undo the effect; however, if an error occurs during statement execution, the statement is rolled back.

To disable autocommit mode implicitly for a single series of statements, use the START TRANSACTION statement:

START TRANSACTION;

SELECT @A:=SUM(salary) FROM table1 WHERE type=1;

UPDATE table2 SET summary=@A WHERE type=1;

COMMIT;

With START TRANSACTION, autocommit remains disabled until you end the transaction with COMMIT or ROLLBACK. The autocommit mode then reverts to its previous state.

START TRANSACTION permits several modifiers that control transaction characteristics. To specify multiple modifiers, separate them by commas.

* The WITH CONSISTENT SNAPSHOT modifier starts a [consistent read](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_consistent_read) for storage engines that are capable of it. This applies only to InnoDB. The effect is the same as issuing a START TRANSACTION followed by a [SELECT](https://dev.mysql.com/doc/refman/8.0/en/select.html) from any InnoDB table. See [Section 15.7.2.3, “Consistent Nonlocking Reads”](https://dev.mysql.com/doc/refman/8.0/en/innodb-consistent-read.html). The WITH CONSISTENT SNAPSHOT modifier does not change the current transaction [isolation level](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_isolation_level), so it provides a consistent snapshot only if the current isolation level is one that permits a consistent read. The only isolation level that permits a consistent read is [REPEATABLE READ](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_repeatable-read). For all other isolation levels, the WITH CONSISTENT SNAPSHOT clause is ignored. A warning is generated when the WITH CONSISTENT SNAPSHOT clause is ignored.
* The READ WRITE and READ ONLY modifiers set the transaction access mode. They permit or prohibit changes to tables used in the transaction. The READ ONLY restriction prevents the transaction from modifying or locking both transactional and nontransactional tables that are visible to other transactions; the transaction can still modify or lock temporary tables.

MySQL enables extra optimizations for queries on InnoDB tables when the transaction is known to be read-only. Specifying READ ONLY ensures these optimizations are applied in cases where the read-only status cannot be determined automatically. See [Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”](https://dev.mysql.com/doc/refman/8.0/en/innodb-performance-ro-txn.html) for more information.

If no access mode is specified, the default mode applies. Unless the default has been changed, it is read/write. It is not permitted to specify both READ WRITE and READ ONLY in the same statement.

In read-only mode, it remains possible to change tables created with the TEMPORARY keyword using DML statements. Changes made with DDL statements are not permitted, just as with permanent tables.

For additional information about transaction access mode, including ways to change the default mode, see [Section 13.3.7, “SET TRANSACTION Statement”](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html).

If the [read\_only](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_read_only) system variable is enabled, explicitly starting a transaction with START TRANSACTION READ WRITE requires the [CONNECTION\_ADMIN](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_connection-admin) privilege (or the deprecated [SUPER](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_super) privilege).

**Important**

Many APIs used for writing MySQL client applications (such as JDBC) provide their own methods for starting transactions that can (and sometimes should) be used instead of sending a START TRANSACTION statement from the client. See [Chapter 29, *Connectors and APIs*](https://dev.mysql.com/doc/refman/8.0/en/connectors-apis.html), or the documentation for your API, for more information.

To disable autocommit mode explicitly, use the following statement:

SET autocommit=0;

After disabling autocommit mode by setting the [autocommit](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_autocommit) variable to zero, changes to transaction-safe tables (such as those for [InnoDB](https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html" \o "Chapter 15 The InnoDB Storage Engine) or [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html)) are not made permanent immediately. You must use [COMMIT](https://dev.mysql.com/doc/refman/8.0/en/commit.html) to store your changes to disk or ROLLBACK to ignore the changes.

[autocommit](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_autocommit) is a session variable and must be set for each session. To disable autocommit mode for each new connection, see the description of the [autocommit](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_autocommit) system variable at [Section 5.1.8, “Server System Variables”](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html).

BEGIN and BEGIN WORK are supported as aliases of START TRANSACTION for initiating a transaction. START TRANSACTION is standard SQL syntax, is the recommended way to start an ad-hoc transaction, and permits modifiers that BEGIN does not.

The BEGIN statement differs from the use of the BEGIN keyword that starts a [BEGIN ... END](https://dev.mysql.com/doc/refman/8.0/en/begin-end.html) compound statement. The latter does not begin a transaction. See [Section 13.6.1, “BEGIN ... END Compound Statement”](https://dev.mysql.com/doc/refman/8.0/en/begin-end.html).

**Note**

Within all stored programs (stored procedures and functions, triggers, and events), the parser treats BEGIN [WORK] as the beginning of a [BEGIN ... END](https://dev.mysql.com/doc/refman/8.0/en/begin-end.html) block. Begin a transaction in this context with [START TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/commit.html) instead.

The optional WORK keyword is supported for COMMIT and ROLLBACK, as are the CHAIN and RELEASE clauses. CHAIN and RELEASE can be used for additional control over transaction completion. The value of the [completion\_type](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_completion_type) system variable determines the default completion behavior. See [Section 5.1.8, “Server System Variables”](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html).

The AND CHAIN clause causes a new transaction to begin as soon as the current one ends, and the new transaction has the same isolation level as the just-terminated transaction. The new transaction also uses the same access mode (READ WRITE or READ ONLY) as the just-terminated transaction. The RELEASE clause causes the server to disconnect the current client session after terminating the current transaction. Including the NO keyword suppresses CHAIN or RELEASE completion, which can be useful if the [completion\_type](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_completion_type) system variable is set to cause chaining or release completion by default.

Beginning a transaction causes any pending transaction to be committed. See [Section 13.3.3, “Statements That Cause an Implicit Commit”](https://dev.mysql.com/doc/refman/8.0/en/implicit-commit.html), for more information.

Beginning a transaction also causes table locks acquired with [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) to be released, as though you had executed [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html). Beginning a transaction does not release a global read lock acquired with [FLUSH TABLES WITH READ LOCK](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-tables-with-read-lock).

For best results, transactions should be performed using only tables managed by a single transaction-safe storage engine. Otherwise, the following problems can occur:

* If you use tables from more than one transaction-safe storage engine (such as InnoDB), and the transaction isolation level is not [SERIALIZABLE](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_serializable), it is possible that when one transaction commits, another ongoing transaction that uses the same tables sees only some of the changes made by the first transaction. That is, the atomicity of transactions is not guaranteed with mixed engines and inconsistencies can result. (If mixed-engine transactions are infrequent, you can use [SET TRANSACTION ISOLATION LEVEL](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html) to set the isolation level to [SERIALIZABLE](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_serializable) on a per-transaction basis as necessary.)
* If you use tables that are not transaction-safe within a transaction, changes to those tables are stored at once, regardless of the status of autocommit mode.
* If you issue a [ROLLBACK](https://dev.mysql.com/doc/refman/8.0/en/commit.html) statement after updating a nontransactional table within a transaction, an [ER\_WARNING\_NOT\_COMPLETE\_ROLLBACK](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_warning_not_complete_rollback) warning occurs. Changes to transaction-safe tables are rolled back, but not changes to nontransaction-safe tables.

Each transaction is stored in the binary log in one chunk, upon [COMMIT](https://dev.mysql.com/doc/refman/8.0/en/commit.html). Transactions that are rolled back are not logged. (***Exception***: Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a [ROLLBACK](https://dev.mysql.com/doc/refman/8.0/en/commit.html) statement at the end to ensure that modifications to the nontransactional tables are replicated.) See [Section 5.4.4, “The Binary Log”](https://dev.mysql.com/doc/refman/8.0/en/binary-log.html).

You can change the isolation level or access mode for transactions with the [SET TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html) statement. See [Section 13.3.7, “SET TRANSACTION Statement”](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html).

Rolling back can be a slow operation that may occur implicitly without the user having explicitly asked for it (for example, when an error occurs). Because of this, [SHOW PROCESSLIST](https://dev.mysql.com/doc/refman/8.0/en/show-processlist.html) displays Rolling back in the State column for the session, not only for explicit rollbacks performed with the [ROLLBACK](https://dev.mysql.com/doc/refman/8.0/en/commit.html) statement but also for implicit rollbacks.

**Note**

In MySQL 8.0, BEGIN, COMMIT, and ROLLBACK are not affected by [--replicate-do-db](https://dev.mysql.com/doc/refman/8.0/en/replication-options-replica.html#option_mysqld_replicate-do-db) or [--replicate-ignore-db](https://dev.mysql.com/doc/refman/8.0/en/replication-options-replica.html#option_mysqld_replicate-ignore-db) rules.

When InnoDB performs a complete rollback of a transaction, all locks set by the transaction are released. If a single SQL statement within a transaction rolls back as a result of an error, such as a duplicate key error, locks set by the statement are preserved while the transaction remains active. This happens because InnoDB stores row locks in a format such that it cannot know afterward which lock was set by which statement.

If a [SELECT](https://dev.mysql.com/doc/refman/8.0/en/select.html) statement within a transaction calls a stored function, and a statement within the stored function fails, that statement rolls back. If [ROLLBACK](https://dev.mysql.com/doc/refman/8.0/en/commit.html) is executed for the transaction subsequently, the entire transaction rolls back.

### Statements That Cannot Be Rolled Back

Some statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases, those that create, drop, or alter tables or stored routines.

You should design your transactions not to include such statements. If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails, the full effect of the transaction cannot be rolled back in such cases by issuing a [ROLLBACK](https://dev.mysql.com/doc/refman/8.0/en/commit.html) statement.

### Statements That Cause an Implicit Commit

The statements listed in this section (and any synonyms for them) implicitly end any transaction active in the current session, as if you had done a [COMMIT](https://dev.mysql.com/doc/refman/8.0/en/commit.html) before executing the statement.

Most of these statements also cause an implicit commit after executing. The intent is to handle each such statement in its own special transaction. Transaction-control and locking statements are exceptions: If an implicit commit occurs before execution, another does not occur after.

* ***Data definition language (DDL) statements that define or modify database objects.*** [ALTER EVENT](https://dev.mysql.com/doc/refman/8.0/en/alter-event.html), [ALTER FUNCTION](https://dev.mysql.com/doc/refman/8.0/en/alter-function.html), [ALTER PROCEDURE](https://dev.mysql.com/doc/refman/8.0/en/alter-procedure.html), [ALTER SERVER](https://dev.mysql.com/doc/refman/8.0/en/alter-server.html), [ALTER TABLE](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html), [ALTER VIEW](https://dev.mysql.com/doc/refman/8.0/en/alter-view.html), [CREATE DATABASE](https://dev.mysql.com/doc/refman/8.0/en/create-database.html), [CREATE EVENT](https://dev.mysql.com/doc/refman/8.0/en/create-event.html), [CREATE FUNCTION](https://dev.mysql.com/doc/refman/8.0/en/create-function.html), [CREATE INDEX](https://dev.mysql.com/doc/refman/8.0/en/create-index.html), [CREATE PROCEDURE](https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html), [CREATE ROLE](https://dev.mysql.com/doc/refman/8.0/en/create-role.html), [CREATE SERVER](https://dev.mysql.com/doc/refman/8.0/en/create-server.html), [CREATE SPATIAL REFERENCE SYSTEM](https://dev.mysql.com/doc/refman/8.0/en/create-spatial-reference-system.html), [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html), [CREATE TRIGGER](https://dev.mysql.com/doc/refman/8.0/en/create-trigger.html), [CREATE VIEW](https://dev.mysql.com/doc/refman/8.0/en/create-view.html), [DROP DATABASE](https://dev.mysql.com/doc/refman/8.0/en/drop-database.html), [DROP EVENT](https://dev.mysql.com/doc/refman/8.0/en/drop-event.html), [DROP FUNCTION](https://dev.mysql.com/doc/refman/8.0/en/drop-function.html), [DROP INDEX](https://dev.mysql.com/doc/refman/8.0/en/drop-index.html), [DROP PROCEDURE](https://dev.mysql.com/doc/refman/8.0/en/drop-procedure.html), [DROP ROLE](https://dev.mysql.com/doc/refman/8.0/en/drop-role.html), [DROP SERVER](https://dev.mysql.com/doc/refman/8.0/en/drop-server.html), [DROP SPATIAL REFERENCE SYSTEM](https://dev.mysql.com/doc/refman/8.0/en/drop-spatial-reference-system.html), [DROP TABLE](https://dev.mysql.com/doc/refman/8.0/en/drop-table.html), [DROP TRIGGER](https://dev.mysql.com/doc/refman/8.0/en/drop-trigger.html), [DROP VIEW](https://dev.mysql.com/doc/refman/8.0/en/drop-view.html), [INSTALL PLUGIN](https://dev.mysql.com/doc/refman/8.0/en/install-plugin.html), [RENAME TABLE](https://dev.mysql.com/doc/refman/8.0/en/rename-table.html), [TRUNCATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/truncate-table.html), [UNINSTALL PLUGIN](https://dev.mysql.com/doc/refman/8.0/en/uninstall-plugin.html).

[CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) and [DROP TABLE](https://dev.mysql.com/doc/refman/8.0/en/drop-table.html) statements do not commit a transaction if the TEMPORARY keyword is used. (This does not apply to other operations on temporary tables such as [ALTER TABLE](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html) and [CREATE INDEX](https://dev.mysql.com/doc/refman/8.0/en/create-index.html), which do cause a commit.) However, although no implicit commit occurs, neither can the statement be rolled back, which means that the use of such statements causes transactional atomicity to be violated. For example, if you use [CREATE TEMPORARY TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) and then roll back the transaction, the table remains in existence.

The [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) statement in InnoDB is processed as a single transaction. This means that a [ROLLBACK](https://dev.mysql.com/doc/refman/8.0/en/commit.html) from the user does not undo [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) statements the user made during that transaction.

[CREATE TABLE ... SELECT](https://dev.mysql.com/doc/refman/8.0/en/create-table.html) causes an implicit commit before and after the statement is executed when you are creating nontemporary tables. (No commit occurs for CREATE TEMPORARY TABLE ... SELECT.)

* ***Statements that implicitly use or modify tables in the****mysql****database.*** [ALTER USER](https://dev.mysql.com/doc/refman/8.0/en/alter-user.html), [CREATE USER](https://dev.mysql.com/doc/refman/8.0/en/create-user.html), [DROP USER](https://dev.mysql.com/doc/refman/8.0/en/drop-user.html), [GRANT](https://dev.mysql.com/doc/refman/8.0/en/grant.html), [RENAME USER](https://dev.mysql.com/doc/refman/8.0/en/rename-user.html), [REVOKE](https://dev.mysql.com/doc/refman/8.0/en/revoke.html), [SET PASSWORD](https://dev.mysql.com/doc/refman/8.0/en/set-password.html).
* ***Transaction-control and locking statements.*** [BEGIN](https://dev.mysql.com/doc/refman/8.0/en/commit.html), [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html), SET autocommit = 1 (if the value is not already 1), [START TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/commit.html), [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html).

[UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) commits a transaction only if any tables currently have been locked with [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) to acquire nontransactional table locks. A commit does not occur for [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) following [FLUSH TABLES WITH READ LOCK](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-tables-with-read-lock) because the latter statement does not acquire table-level locks.

Transactions cannot be nested. This is a consequence of the implicit commit performed for any current transaction when you issue a [START TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/commit.html) statement or one of its synonyms.

Statements that cause an implicit commit cannot be used in an XA transaction while the transaction is in an ACTIVE state.

The [BEGIN](https://dev.mysql.com/doc/refman/8.0/en/commit.html) statement differs from the use of the BEGIN keyword that starts a [BEGIN ... END](https://dev.mysql.com/doc/refman/8.0/en/begin-end.html) compound statement. The latter does not cause an implicit commit. See [Section 13.6.1, “BEGIN ... END Compound Statement”](https://dev.mysql.com/doc/refman/8.0/en/begin-end.html).

* ***Data loading statements.*** [LOAD DATA](https://dev.mysql.com/doc/refman/8.0/en/load-data.html). [LOAD DATA](https://dev.mysql.com/doc/refman/8.0/en/load-data.html) causes an implicit commit only for tables using the [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) storage engine.
* ***Administrative statements.*** [ANALYZE TABLE](https://dev.mysql.com/doc/refman/8.0/en/analyze-table.html), [CACHE INDEX](https://dev.mysql.com/doc/refman/8.0/en/cache-index.html), [CHECK TABLE](https://dev.mysql.com/doc/refman/8.0/en/check-table.html), [FLUSH](https://dev.mysql.com/doc/refman/8.0/en/flush.html), [LOAD INDEX INTO CACHE](https://dev.mysql.com/doc/refman/8.0/en/load-index.html), [OPTIMIZE TABLE](https://dev.mysql.com/doc/refman/8.0/en/optimize-table.html), [REPAIR TABLE](https://dev.mysql.com/doc/refman/8.0/en/repair-table.html), [RESET](https://dev.mysql.com/doc/refman/8.0/en/reset.html) (but not [RESET PERSIST](https://dev.mysql.com/doc/refman/8.0/en/reset-persist.html)).
* ***Replication control statements***. [START REPLICA | SLAVE](https://dev.mysql.com/doc/refman/8.0/en/start-replica.html), [STOP REPLICA | SLAVE](https://dev.mysql.com/doc/refman/8.0/en/stop-replica.html), [RESET REPLICA | SLAVE](https://dev.mysql.com/doc/refman/8.0/en/reset-replica.html), [CHANGE MASTER TO](https://dev.mysql.com/doc/refman/8.0/en/change-master-to.html).

### SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Statements

SAVEPOINT identifier

ROLLBACK [WORK] TO [SAVEPOINT] identifier

RELEASE SAVEPOINT identifier

InnoDB supports the SQL statements [SAVEPOINT](https://dev.mysql.com/doc/refman/8.0/en/savepoint.html), [ROLLBACK TO SAVEPOINT](https://dev.mysql.com/doc/refman/8.0/en/savepoint.html), [RELEASE SAVEPOINT](https://dev.mysql.com/doc/refman/8.0/en/savepoint.html) and the optional WORK keyword for [ROLLBACK](https://dev.mysql.com/doc/refman/8.0/en/commit.html).

The [SAVEPOINT](https://dev.mysql.com/doc/refman/8.0/en/savepoint.html) statement sets a named transaction savepoint with a name of ***identifier***. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The [ROLLBACK TO SAVEPOINT](https://dev.mysql.com/doc/refman/8.0/en/savepoint.html) statement rolls back a transaction to the named savepoint without terminating the transaction. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but InnoDB does not release the row locks that were stored in memory after the savepoint. (For a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

If the [ROLLBACK TO SAVEPOINT](https://dev.mysql.com/doc/refman/8.0/en/savepoint.html) statement returns the following error, it means that no savepoint with the specified name exists:

ERROR 1305 (42000): SAVEPOINT identifier does not exist

The [RELEASE SAVEPOINT](https://dev.mysql.com/doc/refman/8.0/en/savepoint.html) statement removes the named savepoint from the set of savepoints of the current transaction. No commit or rollback occurs. It is an error if the savepoint does not exist.

All savepoints of the current transaction are deleted if you execute a [COMMIT](https://dev.mysql.com/doc/refman/8.0/en/commit.html), or a [ROLLBACK](https://dev.mysql.com/doc/refman/8.0/en/commit.html) that does not name a savepoint.

A new savepoint level is created when a stored function is invoked or a trigger is activated. The savepoints on previous levels become unavailable and thus do not conflict with savepoints on the new level. When the function or trigger terminates, any savepoints it created are released and the previous savepoint level is restored.

### LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Statements

LOCK INSTANCE FOR BACKUP

UNLOCK INSTANCE

LOCK INSTANCE FOR BACKUP acquires an instance-level backup lock that permits DML during an online backup while preventing operations that could result in an inconsistent snapshot.

Executing the LOCK INSTANCE FOR BACKUP statement requires the [BACKUP\_ADMIN](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_backup-admin) privilege. The [BACKUP\_ADMIN](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_backup-admin) privilege is automatically granted to users with the [RELOAD](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_reload) privilege when performing an in-place upgrade to MySQL 8.0 from an earlier version.

Multiple sessions can hold a backup lock simultaneously.

UNLOCK INSTANCE releases a backup lock held by the current session. A backup lock held by a session is also released if the session is terminated.

LOCK INSTANCE FOR BACKUP prevents files from being created, renamed, or removed. [REPAIR TABLE](https://dev.mysql.com/doc/refman/8.0/en/repair-table.html) [TRUNCATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/truncate-table.html), [OPTIMIZE TABLE](https://dev.mysql.com/doc/refman/8.0/en/optimize-table.html), and account management statements are blocked. See [Section 13.7.1, “Account Management Statements”](https://dev.mysql.com/doc/refman/8.0/en/account-management-statements.html). Operations that modify InnoDB files that are not recorded in the InnoDB redo log are also blocked.

LOCK INSTANCE FOR BACKUP permits DDL operations that only affect user-created temporary tables. In effect, files that belong to user-created temporary tables can be created, renamed, or removed while a backup lock is held. Creation of binary log files is also permitted.

A backup lock acquired by LOCK INSTANCE FOR BACKUP is independent of transactional locks and locks taken by [FLUSH TABLES ***tbl\_name*** [, ***tbl\_name***] ... WITH READ LOCK](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-tables-with-read-lock-with-list), and the following sequences of statements are permitted:

LOCK INSTANCE FOR BACKUP;

FLUSH TABLES tbl\_name [, tbl\_name] ... WITH READ LOCK;

UNLOCK TABLES;

UNLOCK INSTANCE;

FLUSH TABLES tbl\_name [, tbl\_name] ... WITH READ LOCK;

LOCK INSTANCE FOR BACKUP;

UNLOCK INSTANCE;

UNLOCK TABLES;

The [lock\_wait\_timeout](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_lock_wait_timeout) setting defines the amount of time that a LOCK INSTANCE FOR BACKUP statement waits to acquire a lock before giving up.

### LOCK TABLES and UNLOCK TABLES Statements

LOCK TABLES

tbl\_name [[AS] alias] lock\_type

[, tbl\_name [[AS] alias] lock\_type] ...

lock\_type: {

READ [LOCAL]

| [LOW\_PRIORITY] WRITE

}

UNLOCK TABLES

MySQL enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

Locks may be used to emulate transactions or to get more speed when updating tables. This is explained in more detail in [Table-Locking Restrictions and Conditions](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html#lock-tables-restrictions).

[LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) explicitly acquires table locks for the current client session. Table locks can be acquired for base tables or views. You must have the [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_lock-tables) privilege, and the [SELECT](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_select) privilege for each object to be locked.

For view locking, [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) adds all base tables used in the view to the set of tables to be locked and locks them automatically. For tables underlying any view being locked, [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) checks that the view definer (for SQL SECURITY DEFINER views) or invoker (for all views) has the proper privileges on the tables.

If you lock a table explicitly with [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html), any tables used in triggers are also locked implicitly, as described in [LOCK TABLES and Triggers](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html#lock-tables-and-triggers).

If you lock a table explicitly with [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html), any tables related by a foreign key constraint are opened and locked implicitly. For foreign key checks, a shared read-only lock ([LOCK TABLES READ](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html)) is taken on related tables. For cascading updates, a shared-nothing write lock ([LOCK TABLES WRITE](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html)) is taken on related tables that are involved in the operation.

[UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) explicitly releases any table locks held by the current session. [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) implicitly releases any table locks held by the current session before acquiring new locks.

Another use for [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) is to release the global read lock acquired with the [FLUSH TABLES WITH READ LOCK](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-tables-with-read-lock) statement, which enables you to lock all tables in all databases. See [Section 13.7.8.3, “FLUSH Statement”](https://dev.mysql.com/doc/refman/8.0/en/flush.html). (This is a very convenient way to get backups if you have a file system such as Veritas that can take snapshots in time.)

A table lock protects only against inappropriate reads or writes by other sessions. A session holding a WRITE lock can perform table-level operations such as [DROP TABLE](https://dev.mysql.com/doc/refman/8.0/en/drop-table.html) or [TRUNCATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/truncate-table.html). For sessions holding a READ lock, [DROP TABLE](https://dev.mysql.com/doc/refman/8.0/en/drop-table.html) and [TRUNCATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/truncate-table.html) operations are not permitted.

The following discussion applies only to non-TEMPORARY tables. [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) is permitted (but ignored) for a TEMPORARY table. The table can be accessed freely by the session within which it was created, regardless of what other locking may be in effect. No lock is necessary because no other session can see the table.

#### Table Lock Acquisition

To acquire table locks within the current session, use the [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) statement, which acquires metadata locks .

The following lock types are available:

READ [LOCAL] lock:

* The session that holds the lock can read the table (but not write it).
* Multiple sessions can acquire a READ lock for the table at the same time.
* Other sessions can read the table without explicitly acquiring a READ lock.
* The LOCAL modifier enables nonconflicting [INSERT](https://dev.mysql.com/doc/refman/8.0/en/insert.html) statements (concurrent inserts) by other sessions to execute while the lock is held. (See [Section 8.11.3, “Concurrent Inserts”](https://dev.mysql.com/doc/refman/8.0/en/concurrent-inserts.html).) However, READ LOCAL cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock. For InnoDB tables, READ LOCAL is the same as READ.

[LOW\_PRIORITY] WRITE lock:

* The session that holds the lock can read and write the table.
* Only the session that holds the lock can access the table. No other session can access it until the lock is released.
* Lock requests for the table by other sessions block while the WRITE lock is held.
* The LOW\_PRIORITY modifier has no effect. In previous versions of MySQL, it affected locking behavior, but this is no longer true. It is now deprecated and its use produces a warning. Use WRITE without LOW\_PRIORITY instead.

WRITE locks normally have higher priority than READ locks to ensure that updates are processed as soon as possible. This means that if one session obtains a READ lock and then another session requests a WRITE lock, subsequent READ lock requests wait until the session that requested the WRITE lock has obtained the lock and released it. (An exception to this policy can occur for small values of the [max\_write\_lock\_count](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_max_write_lock_count) system variable; see [Section 8.11.4, “Metadata Locking”](https://dev.mysql.com/doc/refman/8.0/en/metadata-locking.html).)

If the [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) statement must wait due to locks held by other sessions on any of the tables, it blocks until all locks can be acquired.

A session that requires locks must acquire all the locks that it needs in a single [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) statement. While the locks thus obtained are held, the session can access only the locked tables. For example, in the following sequence of statements, an error occurs for the attempt to access t2 because it was not locked in the [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) statement:

mysql> LOCK TABLES t1 READ;

mysql> SELECT COUNT(\*) FROM t1;

+----------+

| COUNT(\*) |

+----------+

| 3 |

+----------+

mysql> SELECT COUNT(\*) FROM t2;

ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES

Tables in the INFORMATION\_SCHEMA database are an exception. They can be accessed without being locked explicitly even while a session holds table locks obtained with [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html).

You cannot refer to a locked table multiple times in a single query using the same name. Use aliases instead, and obtain a separate lock for the table and each alias:

mysql> LOCK TABLE t WRITE, t AS t1 READ;

mysql> INSERT INTO t SELECT \* FROM t;

ERROR 1100: Table 't' was not locked with LOCK TABLES

mysql> INSERT INTO t SELECT \* FROM t AS t1;

The error occurs for the first [INSERT](https://dev.mysql.com/doc/refman/8.0/en/insert.html) because there are two references to the same name for a locked table. The second [INSERT](https://dev.mysql.com/doc/refman/8.0/en/insert.html) succeeds because the references to the table use different names.

If your statements refer to a table by means of an alias, you must lock the table using that same alias. It does not work to lock the table without specifying the alias:

mysql> LOCK TABLE t READ;

mysql> SELECT \* FROM t AS myalias;

ERROR 1100: Table 'myalias' was not locked with LOCK TABLES

Conversely, if you lock a table using an alias, you must refer to it in your statements using that alias:

mysql> LOCK TABLE t AS myalias READ;

mysql> SELECT \* FROM t;

ERROR 1100: Table 't' was not locked with LOCK TABLES

mysql> SELECT \* FROM t AS myalias;

#### Table Lock Release

When the table locks held by a session are released, they are all released at the same time. A session can release its locks explicitly, or locks may be released implicitly under certain conditions.

* A session can release its locks explicitly with [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html).
* If a session issues a [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) statement to acquire a lock while already holding locks, its existing locks are released implicitly before the new locks are granted.
* If a session begins a transaction (for example, with [START TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/commit.html)), an implicit [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) is performed, which causes existing locks to be released. (For additional information about the interaction between table locking and transactions, see [Interaction of Table Locking and Transactions](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html#lock-tables-and-transactions).)

If the connection for a client session terminates, whether normally or abnormally, the server implicitly releases all table locks held by the session (transactional and nontransactional). If the client reconnects, the locks are no longer in effect. In addition, if the client had an active transaction, the server rolls back the transaction upon disconnect, and if reconnect occurs, the new session begins with autocommit enabled. For this reason, clients may wish to disable auto-reconnect. With auto-reconnect in effect, the client is not notified if reconnect occurs but any table locks or current transaction are lost. With auto-reconnect disabled, if the connection drops, an error occurs for the next statement issued. The client can detect the error and take appropriate action such as reacquiring the locks or redoing the transaction. See [C API Automatic Reconnection Control](https://dev.mysql.com/doc/c-api/8.0/en/c-api-auto-reconnect.html).

**Note**

If you use [ALTER TABLE](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html) on a locked table, it may become unlocked. For example, if you attempt a second [ALTER TABLE](https://dev.mysql.com/doc/refman/8.0/en/alter-table.html) operation, the result may be an error Table '***tbl\_name***' was not locked with LOCK TABLES. To handle this, lock the table again prior to the second alteration. See also [Section B.3.6.1, “Problems with ALTER TABLE”](https://dev.mysql.com/doc/refman/8.0/en/alter-table-problems.html).

#### Interaction of Table Locking and Transactions

[LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html" \o "13.3.6 LOCK TABLES and UNLOCK TABLES Statements) and [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) interact with the use of transactions as follows:

* [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) is not transaction-safe and implicitly commits any active transaction before attempting to lock the tables.
* [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) implicitly commits any active transaction, but only if [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) has been used to acquire table locks. For example, in the following set of statements, [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) releases the global read lock but does not commit the transaction because no table locks are in effect:
* FLUSH TABLES WITH READ LOCK;
* START TRANSACTION;
* SELECT ... ;
* UNLOCK TABLES
* Beginning a transaction (for example, with [START TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/commit.html)) implicitly commits any current transaction and releases existing table locks.
* [FLUSH TABLES WITH READ LOCK](https://dev.mysql.com/doc/refman/8.0/en/flush.html#flush-tables-with-read-lock) acquires a global read lock and not table locks, so it is not subject to the same behavior as [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) and [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) with respect to table locking and implicit commits. For example, [START TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/commit.html) does not release the global read lock. See [Section 13.7.8.3, “FLUSH Statement”](https://dev.mysql.com/doc/refman/8.0/en/flush.html" \o "13.7.8.3 FLUSH Statement).
* Other statements that implicitly cause transactions to be committed do not release existing table locks..
* The correct way to use [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) and [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) with transactional tables, such as InnoDB tables, is to begin a transaction with SET autocommit = 0 (not [START TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/commit.html)) followed by [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html), and to not call [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) until you commit the transaction explicitly. For example, if you need to write to table t1 and read from table t2, you can do this:
* SET autocommit=0;
* LOCK TABLES t1 WRITE, t2 READ, ...;
* *...* *do* something *with* *tables* t1 *and* t2 here *...*
* COMMIT;

UNLOCK TABLES;

When you call [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html), InnoDB internally takes its own table lock, and MySQL takes its own table lock. InnoDB releases its internal table lock at the next commit, but for MySQL to release its table lock, you have to call [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html). You should not have [autocommit = 1](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_autocommit), because then InnoDB releases its internal table lock immediately after the call of [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html), and deadlocks can very easily happen. InnoDB does not acquire the internal table lock at all if [autocommit = 1](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_autocommit), to help old applications avoid unnecessary deadlocks.

* [ROLLBACK](https://dev.mysql.com/doc/refman/8.0/en/commit.html) does not release table locks.

#### LOCK TABLES and Triggers

If you lock a table explicitly with [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html), any tables used in triggers are also locked implicitly:

* The locks are taken as the same time as those acquired explicitly with the [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) statement.
* The lock on a table used in a trigger depends on whether the table is used only for reading. If so, a read lock suffices. Otherwise, a write lock is used.
* If a table is locked explicitly for reading with [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html), but needs to be locked for writing because it might be modified within a trigger, a write lock is taken rather than a read lock. (That is, an implicit write lock needed due to the table's appearance within a trigger causes an explicit read lock request for the table to be converted to a write lock request.)

Suppose that you lock two tables, t1 and t2, using this statement:

LOCK TABLES t1 WRITE, t2 READ;

If t1 or t2 have any triggers, tables used within the triggers are also locked. Suppose that t1 has a trigger defined like this:

CREATE TRIGGER t1\_a\_ins AFTER INSERT ON t1 FOR EACH ROW

BEGIN

UPDATE t4 SET count = count+1

WHERE id = NEW.id AND EXISTS (SELECT a FROM t3);

INSERT INTO t2 VALUES(1, 2);

END;

The result of the [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) statement is that t1 and t2 are locked because they appear in the statement, and t3 and t4 are locked because they are used within the trigger:

* t1 is locked for writing per the WRITE lock request.
* t2 is locked for writing, even though the request is for a READ lock. This occurs because t2 is inserted into within the trigger, so the READ request is converted to a WRITE request.
* t3 is locked for reading because it is only read from within the trigger.
* t4 is locked for writing because it might be updated within the trigger.

#### Table-Locking Restrictions and Conditions

You can safely use [KILL](https://dev.mysql.com/doc/refman/8.0/en/kill.html) to terminate a session that is waiting for a table lock. See [Section 13.7.8.4, “KILL Statement”](https://dev.mysql.com/doc/refman/8.0/en/kill.html).

[LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) and [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) cannot be used within stored programs.

Tables in the performance\_schema database cannot be locked with [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html), except the setup\_***xxx*** tables.

The following statements are prohibited while a [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) statement is in effect: [CREATE TABLE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html), [CREATE TABLE ... LIKE](https://dev.mysql.com/doc/refman/8.0/en/create-table.html), [CREATE VIEW](https://dev.mysql.com/doc/refman/8.0/en/create-view.html), [DROP VIEW](https://dev.mysql.com/doc/refman/8.0/en/drop-view.html), and DDL statements on stored functions and procedures and events.

For some operations, system tables in the mysql database must be accessed. For example, the [HELP](https://dev.mysql.com/doc/refman/8.0/en/help.html) statement requires the contents of the server-side help tables, and [CONVERT\_TZ()](https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html#function_convert-tz) might need to read the time zone tables. The server implicitly locks the system tables for reading as necessary so that you need not lock them explicitly. These tables are treated as just described:

mysql.help\_category

mysql.help\_keyword

mysql.help\_relation

mysql.help\_topic

mysql.time\_zone

mysql.time\_zone\_leap\_second

mysql.time\_zone\_name

mysql.time\_zone\_transition

mysql.time\_zone\_transition\_type

If you want to explicitly place a WRITE lock on any of those tables with a [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) statement, the table must be the only one locked; no other table can be locked with the same statement.

Normally, you do not need to lock tables, because all single [UPDATE](https://dev.mysql.com/doc/refman/8.0/en/update.html) statements are atomic; no other session can interfere with any other currently executing SQL statement. However, there are a few cases when locking tables may provide an advantage:

* If you are going to run many operations on a set of MyISAM tables, it is much faster to lock the tables you are going to use. Locking MyISAM tables speeds up inserting, updating, or deleting on them because MySQL does not flush the key cache for the locked tables until [UNLOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) is called. Normally, the key cache is flushed after each SQL statement.

The downside to locking the tables is that no session can update a READ-locked table (including the one holding the lock) and no session can access a WRITE-locked table other than the one holding the lock.

* If you are using tables for a nontransactional storage engine, you must use [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) if you want to ensure that no other session modifies the tables between a [SELECT](https://dev.mysql.com/doc/refman/8.0/en/select.html) and an [UPDATE](https://dev.mysql.com/doc/refman/8.0/en/update.html). The example shown here requires [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) to execute safely:
* LOCK TABLES trans READ, customer WRITE;
* SELECT SUM(value) FROM trans WHERE customer\_id=some\_id;
* UPDATE customer
* SET total\_value=sum\_from\_previous\_statement
* WHERE customer\_id=some\_id;

UNLOCK TABLES;

Without [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html), it is possible that another session might insert a new row in the trans table between execution of the [SELECT](https://dev.mysql.com/doc/refman/8.0/en/select.html) and [UPDATE](https://dev.mysql.com/doc/refman/8.0/en/update.html) statements.

You can avoid using [LOCK TABLES](https://dev.mysql.com/doc/refman/8.0/en/lock-tables.html) in many cases by using relative updates (UPDATE customer SET ***value***=***value***+***new\_value***) or the [LAST\_INSERT\_ID()](https://dev.mysql.com/doc/refman/8.0/en/information-functions.html#function_last-insert-id) function.

You can also avoid locking tables in some cases by using the user-level advisory lock functions [GET\_LOCK()](https://dev.mysql.com/doc/refman/8.0/en/locking-functions.html#function_get-lock) and [RELEASE\_LOCK()](https://dev.mysql.com/doc/refman/8.0/en/locking-functions.html#function_release-lock). These locks are saved in a hash table in the server and implemented with pthread\_mutex\_lock() and pthread\_mutex\_unlock() for high speed.

### SET TRANSACTION Statement

SET [GLOBAL | SESSION] TRANSACTION

transaction\_characteristic [, transaction\_characteristic] ...

transaction\_characteristic: {

ISOLATION LEVEL level

| access\_mode

}

level: {

REPEATABLE READ

| READ COMMITTED

| READ UNCOMMITTED

| SERIALIZABLE

}

access\_mode: {

READ WRITE

| READ ONLY

}

This statement specifies [transaction](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_transaction) characteristics. It takes a list of one or more characteristic values separated by commas. Each characteristic value sets the transaction [isolation level](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_isolation_level) or access mode. The isolation level is used for operations on [InnoDB](https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html" \o "Chapter 15 The InnoDB Storage Engine) tables. The access mode specifies whether transactions operate in read/write or read-only mode.

In addition, [SET TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html) can include an optional GLOBAL or SESSION keyword to indicate the scope of the statement.

* [Transaction Isolation Levels](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html#set-transaction-isolation-level)
* [Transaction Access Mode](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html#set-transaction-access-mode)
* [Transaction Characteristic Scope](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html#set-transaction-scope)

#### Transaction Isolation Levels

To set the transaction isolation level, use an ISOLATION LEVEL ***level*** clause. It is not permitted to specify multiple ISOLATION LEVEL clauses in the same [SET TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html) statement.

The default isolation level is [REPEATABLE READ](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_repeatable-read). Other permitted values are [READ COMMITTED](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_read-committed), [READ UNCOMMITTED](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_read-uncommitted), and [SERIALIZABLE](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_serializable)..

#### Transaction Access Mode

To set the transaction access mode, use a READ WRITE or READ ONLY clause. It is not permitted to specify multiple access-mode clauses in the same [SET TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html) statement.

By default, a transaction takes place in read/write mode, with both reads and writes permitted to tables used in the transaction. This mode may be specified explicitly using [SET TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html) with an access mode of READ WRITE.

If the transaction access mode is set to READ ONLY, changes to tables are prohibited. This may enable storage engines to make performance improvements that are possible when writes are not permitted.

In read-only mode, it remains possible to change tables created with the TEMPORARY keyword using DML statements. Changes made with DDL statements are not permitted, just as with permanent tables.

The READ WRITE and READ ONLY access modes also may be specified for an individual transaction using the [START TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/commit.html) statement.

#### Transaction Characteristic Scope

You can set transaction characteristics globally, for the current session, or for the next transaction only:

* With the GLOBAL keyword:
  + The statement applies globally for all subsequent sessions.
  + Existing sessions are unaffected.
* With the SESSION keyword:
  + The statement applies to all subsequent transactions performed within the current session.
  + The statement is permitted within transactions, but does not affect the current ongoing transaction.
  + If executed between transactions, the statement overrides any preceding statement that sets the next-transaction value of the named characteristics.
* Without any SESSION or GLOBAL keyword:
  + The statement applies only to the next single transaction performed within the session.
  + Subsequent transactions revert to using the session value of the named characteristics.
  + The statement is not permitted within transactions:
  + mysql> START TRANSACTION;
  + Query OK, 0 rows affected (0.02 sec)
  + mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
  + ERROR 1568 (25001): Transaction characteristics can't be changed

while a transaction is in progress

A change to global transaction characteristics requires the [CONNECTION\_ADMIN](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_connection-admin) privilege (or the deprecated [SUPER](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_super) privilege). Any session is free to change its session characteristics (even in the middle of a transaction), or the characteristics for its next transaction (prior to the start of that transaction).

To set the global isolation level at server startup, use the [--transaction-isolation=***level***](https://dev.mysql.com/doc/refman/8.0/en/server-options.html#option_mysqld_transaction-isolation) option on the command line or in an option file. Values of ***level*** for this option use dashes rather than spaces, so the permissible values are [READ-UNCOMMITTED](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_read-uncommitted), [READ-COMMITTED](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_read-committed), [REPEATABLE-READ](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_repeatable-read), or [SERIALIZABLE](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_serializable).

Similarly, to set the global transaction access mode at server startup, use the [--transaction-read-only](https://dev.mysql.com/doc/refman/8.0/en/server-options.html#option_mysqld_transaction-read-only) option. The default is OFF (read/write mode) but the value can be set to ON for a mode of read only.

For example, to set the isolation level to [REPEATABLE READ](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_repeatable-read) and the access mode to READ WRITE, use these lines in the [mysqld] section of an option file:

[mysqld]

transaction-isolation = REPEATABLE-READ

transaction-read-only = OFF

At runtime, characteristics at the global, session, and next-transaction scope levels can be set indirectly using the [SET TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html) statement, as described previously. They can also be set directly using the [SET](https://dev.mysql.com/doc/refman/8.0/en/set-variable.html) statement to assign values to the [transaction\_isolation](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_transaction_isolation) and [transaction\_read\_only](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_transaction_read_only) system variables:

* [SET TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html) permits optional GLOBAL and SESSION keywords for setting transaction characteristics at different scope levels.
* The [SET](https://dev.mysql.com/doc/refman/8.0/en/set-variable.html) statement for assigning values to the [transaction\_isolation](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_transaction_isolation) and [transaction\_read\_only](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html" \l "sysvar_transaction_read_only) system variables has syntaxes for setting these variables at different scope levels.

The following tables show the characteristic scope level set by each [SET TRANSACTION](https://dev.mysql.com/doc/refman/8.0/en/set-transaction.html) and variable-assignment syntax.

**Table 13.9 SET TRANSACTION Syntax for Transaction Characteristics**

| **Syntax** | **Affected Characteristic Scope** |
| --- | --- |
| SET GLOBAL TRANSACTION ***transaction\_characteristic*** | Global |
| SET SESSION TRANSACTION ***transaction\_characteristic*** | Session |
| SET TRANSACTION ***transaction\_characteristic*** | Next transaction only |

**Table 13.10 SET Syntax for Transaction Characteristics**

| **Syntax** | **Affected Characteristic Scope** |
| --- | --- |
| SET GLOBAL ***var\_name*** = ***value*** | Global |
| SET @@GLOBAL.***var\_name*** = ***value*** | Global |
| SET PERSIST ***var\_name*** = ***value*** | Global |
| SET @@PERSIST.***var\_name*** = ***value*** | Global |
| SET PERSIST\_ONLY ***var\_name*** = ***value*** | No runtime effect |
| SET @@PERSIST\_ONLY.***var\_name*** = ***value*** | No runtime effect |
| SET SESSION ***var\_name*** = ***value*** | Session |
| SET @@SESSION.***var\_name*** = ***value*** | Session |
| SET ***var\_name*** = ***value*** | Session |
| SET @@***var\_name*** = ***value*** | Next transaction only |
| **Syntax** | **Affected Characteristic Scope** |

It is possible to check the global and session values of transaction characteristics at runtime:

SELECT @@GLOBAL.transaction\_isolation, @@GLOBAL.transaction\_read\_only;

SELECT @@SESSION.transaction\_isolation, @@SESSION.transaction\_read\_only;

### XA Transactions

[**XA Transaction SQL Statements**](https://dev.mysql.com/doc/refman/8.0/en/xa-statements.html)

[**XA Transaction States**](https://dev.mysql.com/doc/refman/8.0/en/xa-states.html)

[**Restrictions on XA Transactions**](https://dev.mysql.com/doc/refman/8.0/en/xa-restrictions.html)

Support for [XA](https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_xa) transactions is available for the [InnoDB](https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html) storage engine. The MySQL XA implementation is based on the X/Open CAE document Distributed Transaction Processing: The XA Specification. This document is published by The Open Group and available at <http://www.opengroup.org/public/pubs/catalog/c193.htm>. Limitations of the current XA implementation are described in [Section 13.3.8.3, “Restrictions on XA Transactions”](https://dev.mysql.com/doc/refman/8.0/en/xa-restrictions.html).

On the client side, there are no special requirements. The XA interface to a MySQL server consists of SQL statements that begin with the XA keyword. MySQL client programs must be able to send SQL statements and to understand the semantics of the XA statement interface. They do not need be linked against a recent client library. Older client libraries also work.

Among the MySQL Connectors, MySQL Connector/J 5.0.0 and higher supports XA directly, by means of a class interface that handles the XA SQL statement interface for you.

XA supports distributed transactions, that is, the ability to permit multiple separate transactional resources to participate in a global transaction. Transactional resources often are RDBMSs but may be other kinds of resources.

A global transaction involves several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends ACID properties “up a level” so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties. (As with nondistributed transactions, [SERIALIZABLE](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_serializable) may be preferred if your applications are sensitive to read phenomena. [REPEATABLE READ](https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_repeatable-read) may not be sufficient for distributed transactions.)

Some examples of distributed transactions:

* An application may act as an integration tool that combines a messaging service with an RDBMS. The application makes sure that transactions dealing with message sending, retrieval, and processing that also involve a transactional database all happen in a global transaction. You can think of this as “transactional email.”
* An application performs actions that involve different database servers, such as a MySQL server and an Oracle server (or multiple MySQL servers), where actions that involve multiple servers must happen as part of a global transaction, rather than as separate transactions local to each server.
* A bank keeps account information in an RDBMS and distributes and receives money through automated teller machines (ATMs). It is necessary to ensure that ATM actions are correctly reflected in the accounts, but this cannot be done with the RDBMS alone. A global transaction manager integrates the ATM and database resources to ensure overall consistency of financial transactions.

Applications that use global transactions involve one or more Resource Managers and a Transaction Manager:

* A Resource Manager (RM) provides access to transactional resources. A database server is one kind of resource manager. It must be possible to either commit or roll back transactions managed by the RM.
* A Transaction Manager (TM) coordinates the transactions that are part of a global transaction. It communicates with the RMs that handle each of these transactions. The individual transactions within a global transaction are “branches” of the global transaction. Global transactions and their branches are identified by a naming scheme described later.

The MySQL implementation of XA enables a MySQL server to act as a Resource Manager that handles XA transactions within a global transaction. A client program that connects to the MySQL server acts as the Transaction Manager.

To carry out a global transaction, it is necessary to know which components are involved, and bring each component to a point when it can be committed or rolled back. Depending on what each component reports about its ability to succeed, they must all commit or roll back as an atomic group. That is, either all components must commit, or all components must roll back. To manage a global transaction, it is necessary to take into account that any component or the connecting network might fail.

The process for executing a global transaction uses two-phase commit (2PC). This takes place after the actions performed by the branches of the global transaction have been executed.

1. In the first phase, all branches are prepared. That is, they are told by the TM to get ready to commit. Typically, this means each RM that manages a branch records the actions for the branch in stable storage. The branches indicate whether they are able to do this, and these results are used for the second phase.
2. In the second phase, the TM tells the RMs whether to commit or roll back. If all branches indicated when they were prepared that they were able to commit, all branches are told to commit. If any branch indicated when it was prepared that it was not able to commit, all branches are told to roll back.

In some cases, a global transaction might use one-phase commit (1PC). For example, when a Transaction Manager finds that a global transaction consists of only one transactional resource (that is, a single branch), that resource can be told to prepare and commit at the same time.