* Índice:

[1.1 Transacciones 1](#_Toc503515706)

[1.1.1 Transacciones teoría 1](#_Toc503515707)

[1.1.2 Transferencia bancaria: Ejemplo de operación en la que se requiere una transacción: 2](#_Toc503515708)

[1.1.3 START TRANSACTION, COMMIT, and ROLLBACK Syntax 2](#_Toc503515709)

[**1.1.3.1** Instrucciones relacionadas con las transacciones: 3](#_Toc503515710)

[**1.1.3.2** Desactivar / Activar el AUTOCOMMIT en MySQL 3](#_Toc503515711)

[1.1.3.2.1 Averiguar si se ha iniciado una transacción en la sesión en la que trabajamos: NO SE PUEDE: 4](#_Toc503515712)

[1.1.4 Comandos SQL que llevan implícito un COMMIT antes de ejecutarse, finalizando la transacción activa en la sesión actual: 5](#_Toc503515713)

[1.1.5 Concurrencia de operaciones de manipulación de datos 6](#_Toc503515714)

[1.1.6 Isolation level 6](#_Toc503515715)

[**1.1.6.1** ACID 7](#_Toc503515716)

[**1.1.6.2** page 7](#_Toc503515717)

[**1.1.6.3** data files 7](#_Toc503515718)

## Transacciones

### Transacciones teoría

Qué son las transacciones:

<http://es.wikipedia.org/wiki/Transacci%C3%B3n_%28base_de_datos%29>

<https://technet.microsoft.com/es-es/library/ms190612%28v=sql.105%29.aspx>

Comandos en MySQL para usar las transacciones:

<http://mysql.conclase.net/curso/?sqlsen=COMMIT>

Más información:

<http://mysql.conclase.net/curso/?sqlsen=COMMIT>

<http://programacion.net/articulo/transacciones_en_mysql_242>

<https://basmoxo.files.wordpress.com/2011/10/manejo-de-concurrencia-en-mysql.pdf>

Una transacción **agrupa un conjunto de operaciones** y garantiza que:

* + **o bien se realizan todas** las operaciones,
  + **o si surge algún problema no se ejecuta ninguna**: si alguna de las operaciones de la transacción no ha podido realizarse por algún problema, entonces se anulan todas las operaciones, con lo que la BD no sufre ninguna modificación.

1. Para conseguirlo la transacción va realizando las modificaciones, no sobre los datos reales, sino sobre datos temporalesen memoria RAM que son copia de los reales, y que sólo se convierten en reales y definitivos en el momento en que se confirma la transacción.

Normalmente en un SGBD cada comando SQL es una transacción (es independiente de los demás comandos), de este modo si ejecutamos el comando:

**UPDATE empleados SET salario = salario \*1.05;**

Supongamos que el SGBD debe actualizar el salario de los 2500 empleados de la tabla empleados, la actualización se hará fila a fila (empleado por empleado).

Como ese comando es una transacción, el SGBD nos asegura que el comando se ejecutará completamente: aunque el SGBD se apague/cuelgue durante la operación, por ejemplo cuando sólo había actualizado el salario de 1233 empleados (cuando se reinicie el servidor completará las operaciones que quedaron inacabadas, para así dejar la BD en un estado consistente). No sería admisible que se actualizara el salario de sólo algunos de los empleados cuando deberían ser todos.

Como logra este requisito el SGBD:

Cuando se ejecuta un comando de modificación de datos (UPDATE, DELETE, INSERT), esa operación se hace sobre una copia de la tabla original, y cuando el comando finaliza de modificar todas las filas necesarias, esa tabla temporal sustituye a la tabla original.

Además de esa manera es posible que el SGBD pueda atender consultas SELECT sobre la BD sin esperar a que finalice el comando de modificación de datos.

### Transferencia bancaria: Ejemplo de operación en la que se requiere una transacción:

Realizar una transferencia bancaria de 500€ de una cuenta bancaria donante a otra receptora

se realizan 3 operaciones en la BD:

1. OPERACIÓN 1: disminuir el balance del donante en 500€
2. OPERACIÓN 2: aumentar el balance del receptor en €500.

Si se apagase el servidor después de realizar la operación 1 y antes de realizar la operación 2, entonces el donante habría perdido 500€, pero el receptor no habría recibido ningún dinero.

**START TRANSACTION;**

# **OPERACIÓN 1**: disminuir el balance del donante en 500€

**UPDATE cuenta**

**SET balance = balance – 500**

**WHERE id\_cuenta = 12233789289374897**

# **OPERACIÓN 2**: aumentar el balance del receptor en €500.

**UPDATE cuenta**

**SET balance = balance + 500**

**WHERE id\_cuenta = 98765737367467464**

#**confirmar la transacción o deshacerla**

**COMMIT;** # también valdría **START TRANSACTION;**

# si quisiera deshacer la transacción escribiría en su lugar: **ROLLBACK;**

**OPERACIÓN 3**

**UPDATE cuenta**

**SET balance = balance – 9990**

**WHERE id\_cuenta = 56233789289374888**

# **OPERACIÓN 4**: aumentar el balance del receptor en €500.

**UPDATE cuenta**

**SET balance = balance + 9990**

**WHERE id\_cuenta = 44765737367467411**

Las transacciones son especialmente importantes cuando el cliente se conecta al servidor **remotamente (a través de la red):** Un fallo de red podría cortar la conexión cuando todavía no se han transmitido todas las órdenes que deben ejecutarse en el servidor; si eso aconteciese y no nos protegiésemos utilizando transacciones, la base de datos tendría datos erróneos.

### START TRANSACTION, COMMIT, and ROLLBACK Syntax

<https://dev.mysql.com/doc/refman/5.1/en/commit.html>

START TRANSACTION [WITH CONSISTENT SNAPSHOT]

BEGIN [WORK]

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

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

SET autocommit = {0 | 1}

#### Instrucciones relacionadas con las transacciones:

* + **START TRANSACTION** or **BEGIN**or **BEGIN WORK**: start a new transaction.
  + **COMMIT**:commits the current transaction, making its changes permanent.
  + **ROLLBACK**:rolls back the current transaction, canceling its changes.
  + SET autocommit= 0 disables autocommit mode.
  + SET autocommit= 1 enables autocommit mode.

BEGIN and BEGIN WORK are supported as aliases of START TRANSACTION for initiating a transaction. START TRANSACTION is standard SQL syntax and is the recommended way to start an ad-hoc transaction.

#### Desactivar/ Activar el AUTOCOMMIT en MySQL

MySQL tiene activado por defecto el **autocommit**. Esto quiere decir que cada operación de modificación de datos se hace definitiva inmediatamente en el disco duro, sin posibilidad de deshacerla.

1. Maneras de modificar el **AUTOCOMMIT** desde el cliente (Workbench / consola):
   * **Workbench***(posterior a 8.0):* Query – Auto-CommitTransactions (desmarcar la opción). Al reiniciar Workbench vuelve a marcarse.
   * **Workbench***(anteriora 8.0)*:*Preferencias - SQL Execution - Leave autocommit mode enabled by default (ModoAutocommithabilitadopordefecto)*.
   * **Consola**: SET AUTOCOMMIT = 0; SET AUTOCOMMIT=1;
2. Modificar el **AUTOCOMMIT** en la configuración del servidor:

Modificar el comportamiento predeterminado que el servidor tiene ante cualquier conexión desde un cliente:

* + iniciar el servidor con el comando servicemysqldstartinlcuyecto el parámetro que modifica el valor de autocommit:

autocommit=0 # autocommit deshabilitado

autocommit=1 # autocommit habilitado

* + modificar el fichero de configuración del servidor MySQL:

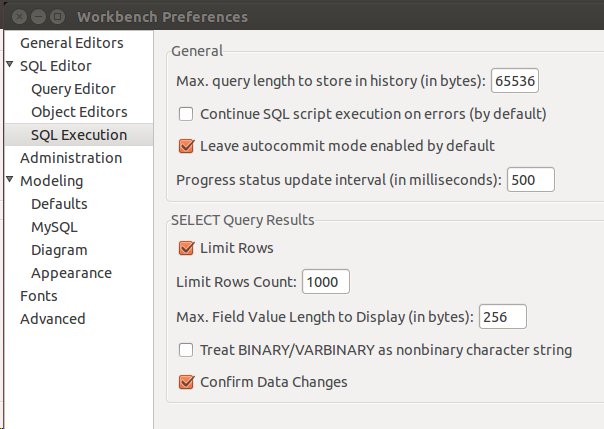
[mysqld]

autocommit=0 #si queremos autocommit deshabilitado

* Aclaraciones y ejemplo de AUTOCOMMIT

El comando **START TRANSACTION** deshabilita automáticamente el **AUTOCOMMIT**, y permanecerá deshabilitado hasta que finalicemos la transacción con **COMMIT** o **ROLLBACK**, tras lo cual se vuelve a restaurar el **AUTOCOMMIT** al estado anterior al START TRANSACTION. Si tras iniciar una transacción con START TRANSACTION decidimos finalizar la transacción mediante otro comando **START TRANSACTION**, entonces el modo **AUTOCOMMIT** sigue deshabilitado hasta que finalice la transacción.

Si AUTOCOMMIT vale 0 y desde el cliente cambiamos su valor a 1, el servidor MySQL realiza un COMMIT implícito de manera automática y de ese modo hace definitivas las modificaciones efectuadas hasta ese momento.



El **comando para habilitar o deshabilitar el modo autocommit** es:

SET autocommit= 1;

SET autocommit = 0;

**AUTOCOMMIT** es una variable de sesión y puede ser modificada durante la sesión.

Para modificar la variable AUTOCOMMIT y que esa modificación afecte a todas las nuevas conexiones de clientes que se realicen el futuro, deberemos de modificar la variable global de sistema AUTOCOMMIT. ([Maneras de modificar el autocommit en la configuración del servidor](#Maneras_de_modificar_el_autocommit_servi)).

En el siguiente ejemplo el Segundo START TRANSACTION ejecuta implícitamente un COMMIT del anterior START TRANSACTION:

USE enterprisedb2;

START TRANSACTION;

DELETE FROM emp

WHERE empno = 7934;

START TRANSACTION;

DELETE FROM emp

WHERE empno = 8888;

##### Averiguar si se ha iniciado una transacción en la sesión en la que trabajamos: NO SE PUEDE:

el problema de este método es que averiguas si alguna sesión ha iniciado una TRANSACTION en el servidor, pero no sabes si es tu sesión u otra.

De hecho hasta que no ejecutas un comando de lectura o modificación de datos no se informa de la existencia de la transacción (así que si acabas de ejecutar START TRANSACTION y todavía no has ejecutado ningún otro comando, no te informa de que se ha iniciado una transacción).

show engineinnodb status;

Buscar este apartado y confirma si estamos en una transacción:

Comprobamos que en este caso Sí estamos en una transacción:

------------

TRANSACTIONS

------------

Trxid counter 29195

Purge done for trx's n:o< 28700 undo n:o < 0 state: running but idle

History list length 356

LIST OF TRANSACTIONS FOR EACH SESSION:

---TRANSACTION 29194, ACTIVE 35 sec

2 lock struct(s), heap size 360, 15 row lock(s), undo log entries 14

MySQL thread id 4, OS thread handle 0x7f69011a6700, query id 73 localhost root init

showengineinnodb status

Sin embargo, en este caso No estamos en una transacción:

------------

TRANSACTIONS

------------

Trxid counter 29196

Purge done for trx's n:o< 29196 undo n:o < 0 state: running but idle

History list length 357

LIST OF TRANSACTIONS FOR EACH SESSION:

---TRANSACTION 0, not started

MySQL thread id 5, OS thread handle 0x7f69011a6700, query id 83 localhost root init

showengineinnodb status

### Comandos SQL que llevan implícito un COMMIT antes de ejecutarse, finalizando la transacción activa en la sesión actual:

<http://dev.mysql.com/doc/refman/5.6/en/implicit-commit.html>

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](http://dev.mysql.com/doc/refman/5.6/en/commit.html) 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 user-defined functions. (ALTER FUNCTION can only be used with stored functions.)

**Statements that implicitly use or modify tables in the mysql database**. ALTER USER, 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.

### Concurrencia de operaciones de manipulación de datos

La concurrencia sucede cuando varios usuarios intentan acceder a los mismos datos simultáneamente (varios usuarios consultan el precio del mismo producto, varios usuarios compran el mismo producto (reduciendo el stock)). Hay 2 clases de transacciones y admiten distintos tipos de concurrencia:

* + las **transacciones de lectura** pueden realizarse simultáneamente (utilizan bloqueo compartido/no exclusivo). Por tanto, las operaciones de lectura se pueden ejecutar concurrentemente, porque no interfieren entre sí.
  + Las **transacciones de escritura** utilizan bloqueo exclusivo/no compartido, por lo que el SGBD no puede ejecutar simultáneamente varias transacciones de escritura (el SGBD coloca al resto de transacciones de escritura en una cola de espera, cuando finalice la transacción de escritura en curso pasará a ejecutarse la siguiente transacción de escritura encolada), el motivo es que pueden interferir entre sí (una orden de poner a 0 el stock de un producto y otra orden intenta ponerlo a 100, la última que se ejecuta es la que deja el valor definitivo).

Sin embargo, las transacciones de lectura no tienen que esperar a que finalice la transacción de escritura, sino que se pueden ejecutar en cada momento ya que acceden a los datos reales (mientras la transacción de escritura en curso modifica los datos en una tabla temporal).

Un mismo usuario puede tener varias conexiones abiertas con el SGBD, y esas conexiones son distintas entre sí. Si el usuario realiza transacciones desde ambas conexiones, son consideradas como independientes.

### Isolation level

<https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_isolation_level>

Una de las tareas fundamentales de un sistema gestor de bases de datos es el aislamiento (**I**solation) de las operaciones. **I**solation es la **I** del acrónimo [ACID](#_ACID).

El aislamiento es el parámetro que regula el equilibrio entre el rendimiento, fiabilidad, consistencia y repetitividad de los resultados cuando se realizan simultáneamente múltiples transacciones que modifican y consultan datos.

Los niveles de aislamiento soportados por InnoDB son, del más alto al más bajo:

**SERIALIZABLE**, **REPEATABLE READ**, **READ COMMITTED**, y **READ UNCOMMITTED**

El modo por defecto es: **REPEATABLE READ**

En la mayoría de ocasiones se mantiene el nivel de aislamiento por defecto (**REPEATABLE READ**).

Para un uso avanzado, los usuarios expertos podría utilizar el nivel **READ COMMITTED**, dado que aumenta el nivel de escalabilidad en operaciones OLTP o en operaciones de *data warehousing*, dado que en este tipo de operaciones la existencia de pequeñas inconsistencias no afectan al resultado final que se obtiene al analizar inmensas cantidades de datos.

Los niveles **SERIALIZABLE** y **READ UNCOMMITTED** modifican los algoritmos de procesado de manera tan extrema que rara vez son utilizados.

#### ACID

<https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_acid>

An acronym standing for **Atomicity, Consistency, Isolation, and Durability**. These properties are all desirable in a database system, and are all closely tied to the notion of a **transaction**. The transactional features of InnoDB adhere to the ACID principles.

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.

The database remains in a **consistent** state at all times -- after each commit or rollback, and while transactions are in progress. If related data is being updated across multiple tables, queries see either all old values or all new values, not a mix of old and new values.

Transactions are protected (**isolated**) from each other while they are in progress; they cannot interfere with each other or see each other's uncommitted data. This isolation is achieved through the **locking** mechanism. Experienced users can adjust the **isolation level**, trading off less protection in favor of increased performance and **concurrency**, when they can be sure that the transactions really do not interfere with each other.

The results of transactions are **durable**: once a commit operation succeeds, the changes made by that transaction are safe from power failures, system crashes, race conditions, or other potential dangers that many non-database applications are vulnerable to. Durability typically involves writing to disk storage, with a certain amount of redundancy to protect against power failures or software crashes during write operations. (In InnoDB, the **doublewrite buffer** assists with durability.)

See Also [atomic](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_atomic), [commit](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_commit), [concurrency](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_concurrency), [doublewrite buffer](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_doublewrite_buffer), [isolation level](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_isolation_level), [locking](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_locking), [rollback](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_rollback), [transaction](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_transaction).

#### page

<https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_page>

A unit representing how much data InnoDB transfers at any one time between disk (the **data files**) and memory (the **buffer pool**). A page can contain one or more **rows**, depending on how much data is in each row. If a row does not fit entirely into a single page, InnoDB sets up additional pointer-style data structures so that the information about the row can be stored in one page.

One way to fit more data in each page is to use **compressed row format**. For tables that use BLOBs or large text fields, **compact row format** allows those large columns to be stored separately from the rest of the row, reducing I/O overhead and memory usage for queries that do not reference those columns.

When InnoDB reads or writes sets of pages as a batch to increase I/O throughput, it reads or writes an **extent** at a time.

All the InnoDB disk data structures within a MySQL instance share the same **page size**.

See Also [buffer pool](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_buffer_pool), [compact row format](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_compact_row_format), [compressed row format](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_compressed_row_format), [data files](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_data_files), [extent](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_extent), [page size](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_page_size), [row](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_row).

#### data files

<https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_data_files>

The files that physically contain the InnoDB**table** and **index** data.

The **system tablespace**, which holds the **data dictionary** and is capable of holding data for multiple InnoDB tables, is represented by one or more .ibdata data files.

File-per-table tablespaces, which hold data for a single InnoDB table, are represented by a.ibd data file.

General tablespaces (introduced in MySQL 5.7.6), which can hold data for multiple InnoDB tables, are also represented by a .ibd data file.

See Also [data dictionary](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_data_dictionary), [file-per-table](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_file_per_table), [general tablespace](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_general_tablespace), [.ibd file](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_ibd_file), [ibdata file](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_ibdata_file), [index](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_index), [system tablespace](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_system_tablespace), [table](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_table), [tablespace](https://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_tablespace).