**START TRANSACTION, COMMIT, ROLLBACK**

Una transacción es un bloque de instrucciones SQL que queremos que se ejecute como si fuese una única instrucción. El objetivo es que si una de las instrucciones que componen la transacción falla la base de datos siga estando el estado coherente e integro en el que estaba antes de lanzar la transacción.

Para ello deberemos controlar cualquier excepción que se pueda producir durante la ejecución de la transacción.

Características de las transacciones:

* [**Atomicidad**](https://es.wikipedia.org/wiki/Atomicidad)**:** Si cuando una operación consiste en una serie de pasos, de los que o bien se ejecutan todos o ninguno, es decir, las transacciones son completas.
* [**Consistencia**](https://es.wikipedia.org/wiki/Consistencia_de_datos)**:** (*Integridad*). Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de *Integridad* de la base de datos. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido. "La Integridad de la Base de Datos nos permite asegurar que los datos son exactos y consistentes, es decir que estén siempre intactos, sean siempre los esperados y que de ninguna manera cambian ni se deformen. De esta manera podemos garantizar que la información que se presenta al usuario será siempre la misma."
* [**Aislamiento**](https://es.wikipedia.org/wiki/Aislamiento_(ACID))**:** Esta propiedad asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error. Esta propiedad define cómo y cuándo los cambios producidos por una operación se hacen visibles para las demás operaciones concurrentes. El aislamiento puede alcanzarse en distintos niveles, siendo el parámetro esencial a la hora de seleccionar SGBDs.
* [**Durabilidad**](https://es.wikipedia.org/wiki/Datos_persistentes)**:** (*Persistencia*). Esta propiedad asegura que una vez realizada la operación, esta persistirá y no se podrá deshacer aunque falle el sistema y que de esta forma los datos sobrevivan de alguna manera.

Para utilizar transacciones en MySQL, debemos utilizar el motor de almacenamiento InnoDB.

Por defecto, MySQL se ejecuta en modo autocommit (tanto en innodb como en MyIsam). Esto significa que tan pronto como se ejecuta una sentencia se actualiza (modifica) la tabla, MySQL almacenará la actualización en disco.

**NO SE PUEDEN USAR CON SENTENCIAS DDL DE CREACION DE TABLAS (el rollback, no deshace los cambios)**

Si se están usando tablas de transacción segura (como InnoDB o BDB), se puede poner MySQL en modo no-autocommit con el comando siguiente:

SET AUTOCOMMIT=0

Después de desconectar el modo autocommit asignando cero a la variable AUTOCOMMIT, se debe usar COMMIT para almecenar los cambios en disco o ROLLBACK si se quieren ignorar los cambios hechos desde el principio de la transacción. Una transacción por tanto tiene dos finales posibles, COMMIT y ROLLBACK.

Si se quiere desactivar el modo autocommit para una serie de sentencias, por tanto, se puede usar la sentencia START TRANSACTION:

START TRANSACTION;

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

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

COMMIT;

También se puede usar BEGIN y BEGIN WORK en lugar de START TRANSACTION para iniciar una transacción.

Si no se están usando tablas de transacción segura, cualquier cambio será almacenado inmediatamente, independientemente del estado del modo autocommit.

Si se usa una sentencia ROLLBACK después de actualizar una tabla no transaccional, se obtendrá un error (ER\_WARNING\_NOT\_COMPLETE\_ROLLBACK) como un aviso. Todas las tablas de transacción segura serán restauradas, pero cualquier tabla de transacción no segura no cambiará.

Veremos un ejemplo completo, extraído del articulo fuente de esta publicación, donde utilizaremos START TRANSACTION (no es necesario AUTOCOMMIT en 0)

CREATE TABLE `departamentos` (

`CODIGO` INTEGER(11) NOT NULL DEFAULT ‘0’,

`NOMBRE` VARCHAR(100),

`PRESUPUESTO` INTEGER(11) DEFAULT NULL,

PRIMARY KEY (`CODIGO`) )ENGINE=InnoDB

Ahora, insertaremos registros de la tabla departamentos\_externos a departamentos mediante una transacción:

START TRANSACTION;

SELECT @A := presupuesto FROM departamentos\_externos WHERE codigo =11;

INSERT INTO departamentos( codigo, nombre, presupuesto ) VALUES ( 11, ‘Departamento test’, @A );

COMMIT;

En el ejemplo anterior se guardó el presupuesto del departamento externo 11 en la variable @A y luego fue asignado al presupuesto en la tabla departamentos.

Otro ejemplo:

START TRANSACTION;

SELECT @A := presupuesto, @B := codigo, @C := nombre FROM departamentos\_externos WHERE codigo=33;

INSERT INTO departamentos( codigodep, nombredep, presupuesto ) VALUES (@B , @C , @A );

COMMIT ;

Otro ejemplo más:

START TRANSACTION;

SELECT @A:=PRESUPUESTO FROM departamentos\_externos WHERE codigo=11;

UPDATE departamentos SET PRESUPUESTO = PRESUPUESTO + @A WHERE codigo=33;

COMMIT;

Al realizar una transacción SQL hay que tener en cuenta que apenas se realice un INSERT, UPDATE o DELETE se genera un bloqueo sobre la tabla y que otros clientes no pueden acceder para escribir esta tabla. Otros clientes podrán realizar SELECTs sobre la tabla, pero no podrán ver los datos del primer cliente hasta que los mismos sean confirmados.