



UD3

CONSULTA Y MODIFICACIÓN DE BASES DE DATOS

MP_0484
Bases de Datos

3.7 Transacciones

Introducción

Una transacción es un conjunto de sentencias SQL que se ejecutan en la BBDD como una única operación, indivisible, confirmándose o deshaciéndose todo el conjunto de sentencias SQL. La transacción queda finalizada con las sentencias apropiadas o implícitamente terminando la sesión.

Supongamos que queremos borrar una fila de una tabla, pero al teclear la orden SQL se nos olvida la cláusula WHERE y borramos todas las filas de la tabla. Pues bien, esto no es un problema, porque los SGBD permiten dar marcha atrás a un trabajo realizado con las órdenes oportunas.

Transacciones

Cuando hacemos transacciones/operaciones sobre la BD, es decir, cuando insertamos, actualizamos y eliminamos datos en las tablas, los cambios no se aplicarán a la BD hasta que los confirmemos. Esto significa que, si durante el tiempo que hemos estado realizando transacciones no hemos hecho ninguna confirmación y de pronto se va la luz, ¿todo el trabajo se habrá perdido? ¿Las tablas estarán en la situación de partida? Todo ello depende de cuál sea el estado de la variable @@autocommit.

COMMIT

La instrucción COMMIT hace que los cambios realizados por la transacción sean definitivos, irrevocables. Solo se debe utilizar si estamos de acuerdo con los cambios, conviene asegurarse mucho antes de realizar el COMMIT, ya que las instrucciones ejecutadas pueden afectar a miles de registros. Además, el cierre correcto de la sesión da lugar a un COMMIT, aunque siempre podemos ejecutar explícitamente esta instrucción a fin de asegurarnos de lo que hacemos (figura 1).

```

1  use prueba_crear;
2
3
4  • DELETE FROM juego50
5    where nombreJuego like '%fifa%';
6  • COMMIT;
-

```

✓	8	01:04:34	DELETE FROM juego50 where nombreJuego like "%fifa%"	1 row(s) affected
✓	9	01:04:51	COMMIT	0 row(s) affected

Figura 1. Ejemplo de uso de COMMIT

Existe la posibilidad de validar automáticamente las transacciones sin tener que indicarlo de manera explícita. Para eso sirve el parámetro AUTOCOMMIT. El valor por defecto es ON (1), de manera que INSERT, UPDATE y DELETE son instrucciones definitivas. Hay dos comandos que nos permiten establecer este funcionamiento:

- **SET AUTOCOMMIT 0|1** nos permite establecer el valor a 1, para que, por defecto, todas las transacciones se validen automáticamente sin necesidad de COMMIT.
- **SELECT @@autocommit** se visualiza en estado de la variable autocommit para ver el funcionamiento.

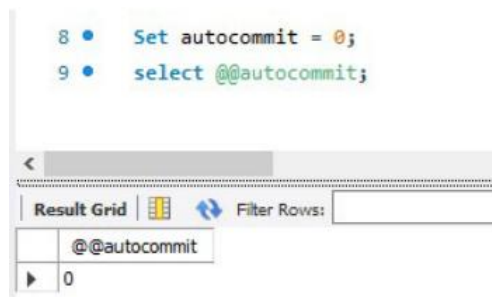


Figura 2. Ajuste de la variable @@autocommit

ROLLBACK

Este comando vuelve a la instrucción anterior al inicio de la transacción, normalmente el último COMMIT; la última instrucción DDL o DCL, o al inicio de sesión. Anula definitivamente los cambios, por lo que conviene también asegurarse de esta operación. Un abandono de sesión incorrecto o un problema de comunicación o de caída del sistema dan lugar a un ROLLBACK implícito. En la figura 3 se puede ver un ejemplo de uso del comando, en el que se realiza un DELETE y tal y como se muestra se llega a borrar el registro. Esto se revierte en la última captura al utilizar el comando ROLLBACK.

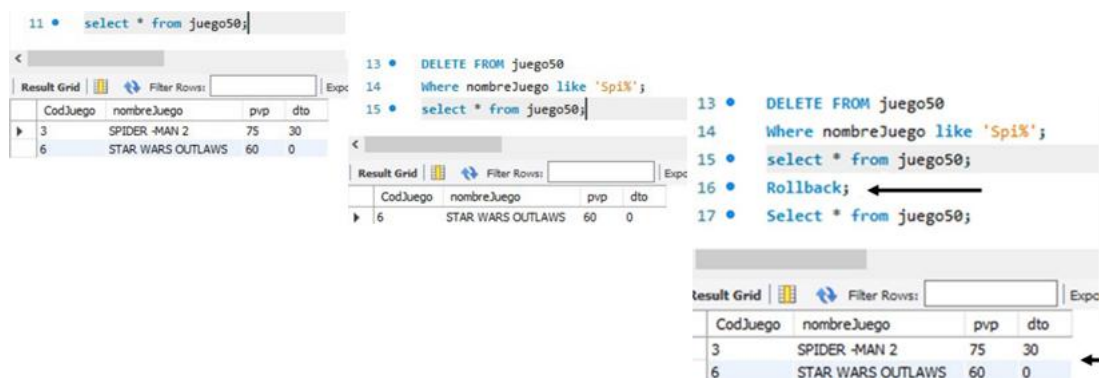


Figura 3. Ejemplo de uso del comando ROLLBACK

Grupo de acciones como una transacción

Se pueden definir transacciones para que un conjunto de acciones se tome como un bloque, de forma que el COMMIT se realiza si todas las acciones del bloque se han llevado a cabo con éxito. Este tipo de acciones tiene mucha importancia para situaciones en las que una acción no tiene sentido por sí sola (es más, podría producir una inconsistencia en la BD) y es necesario que o se produzcan dos o más acciones o ninguna. Por ejemplo, primero se calcula el sumatorio de un campo y después se actualiza la modificación de otra tabla en función del resultado de dicho sumatorio. Esto sería el típico caso de realizar las dos secuencias juntas o no tiene sentido.

START TRANSACTION;

Sentencia1;

Sentencia2;

...

Sentencia;

COMMIT;

Políticas de bloqueo de registros

Las BD están preparadas para ser usadas por varios usuarios accediendo a los mismos datos y actualizándolos. El uso de los SGBD fue un gran avance respecto de los sistemas de ficheros, ya que se encargan de controlar y permitir los accesos simultáneos de forma eficiente y sin errores por varios usuarios simultáneamente.

El SGBD asegura la integridad y la consistencia de la información almacenada en las BD. Se pueden establecer ciertas acciones a nivel de usuario avanzado para bloquear o no objetos para uso exclusivo durante un tiempo.

LOCK {TABLE|TABLES} nombre_tabla1, nombre2, READ|WRITE;

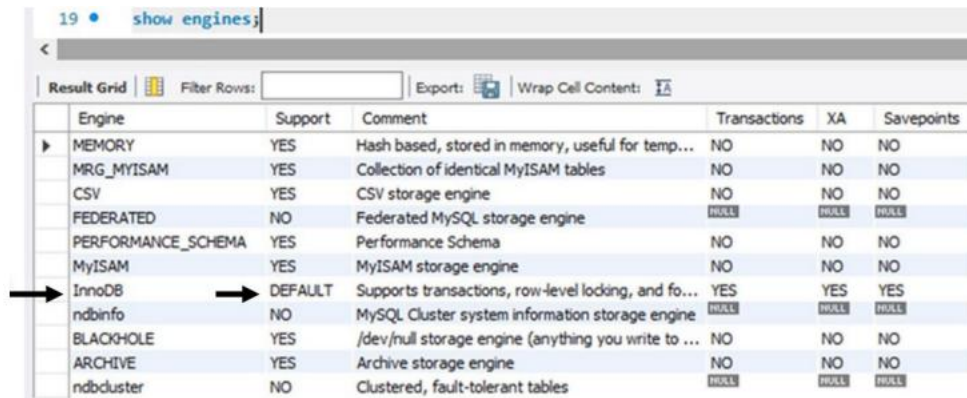
UNLOCK {TABLE | TABLES};

El motor de BD de MySQL Workbench es por defecto InnoDB. En este caso no es necesario bloquear y desbloquear objetos, es el propio motor de BD el que realiza estas acciones.

Motores de almacenamiento en MySQL WorkBench

Son el elemento principal en la gestión de tablas del SGBD. El motor de almacenamiento es el que se encarga de almacenar, recuperar, preparar y gestionar toda la información de una tabla.

Según el motor de BBDD con el que estemos trabajando, el bloqueo de registros se estará realizando de una u otra forma. Tal y como vemos en la figura 4, podemos consultar que motor está utilizando MySQL.



Engine	Support	Comment	Transactions	XA	Savepoints
MEMORY	YES	Hash based, stored in memory, useful for temp...	NO	NO	NO
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
InnoDB	DEFAULT	Supports transactions, row-level locking, and fo...	YES	YES	YES
ndbinfo	NO	MySQL Cluster system information storage engine	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine (anything you write to ...	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
ndbcluster	NO	Clustered, fault-tolerant tables	NO	NO	NO

Figura 4. Consulta del motor de almacenamiento

MySQL utiliza habitualmente dos motores de BD:

- **InnoDB** tiene como característica la seguridad en la ejecución de las transacciones. Garantiza que, si no se realizan de forma correcta, se pueden revertir los cambios. También realiza bloqueo total sobre las tablas cuando ejecuta sentencias DML: INSERT, UPDATE, DELETE, etc.
- **MyISAM** tiene como característica la gran velocidad al recuperar la información. Es la mejor opción cuando las sentencias predominantes sean consultas y, sobre todo, si se necesitan consultas sobre FULLTEXT. El inconveniente es que no realiza comprobaciones en la inserción de datos respecto a la integridad referencial y podemos perder fiabilidad.