**Clase 20**

**Sublenguaje TCL**

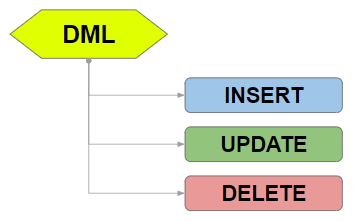
*FUNDAMENTOS DE TCL*

Se conoce como **Transaction Control Language** (o TCL) al grupo de sentencias del sub-lenguaje de **Control de Transacciones** que se utilizan para administrar transacciones en la DB.

Se utilizan para **gestionar los cambios** realizados por las sentencias DML y **agruparlas en transacciones lógicas.El papel de TCL es fundamental** ya que, a través del mismo, controlamos las cláusulas u operaciones DML, agrupandolas de manera tal que se establezca una lógica transaccional cuando realizamos múltiples operaciones que afectan a una o más tablas.

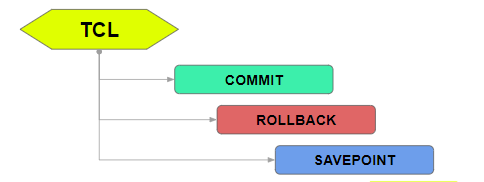
Básicamente **nos ayuda a mantener la integridad de los datos manipulados**.

Cuando mencionamos **operaciones DML**, nos estamos refiriendo a los tres tipos de operaciones más importantes para la manipulación de datos:

****

*COMANDOS TCL PARA EL CONTROL DE TRANSACCIONES*

**Transaction Control Language** incluye tres sentencias claves que se integran a las cláusulas SQL, para poder controlar cada una de las operaciones del DML durante el proceso de ejecución de las mismas.



*COMANDO COMMIT*

**COMMIT** es un comando que sirve para “*confirmar*” las operaciones realizadas sobre una o más tablas.

Cuando es ejecutado, se ocupa de **confirmar** o guardar los cambios realizados en la o las tabla(s), haciéndolos permanentes y poniendo esos cambios a disposición del resto de los usuarios.Durante la ejecución de algunas de las sentencias DML, la o las modificaciones realizadas se guardan de forma “*temporal*” en un archivo especial del Servidor de la BD denominado **log de transacciones.**

Al ejecutarse el comando **COMMIT**, dichas modificaciones impactan de forma **permanente** en la DB, reflejando los cambios en la tabla o tablas en cuestión.

*COMANDO ROLLBACK*

**ROLLBACK** se ocupa de “*deshacer*”, o volver al estado permanente anterior a él o los cambios realizado(s) sobre la(s) tabla(s) en cuestión.

Básicamente funciona como el comando **UNDO** (*Deshacer*), o **CTRL+Z**, que utilizamos de manera frecuente en cualquier programa de computadora.Previo a utilizar Rollback debemos tener presente que, el mismo, solo funciona si no se ejecutó antes el comando **COMMIT.**

En el caso de realizar una operación **DML** y luego haber ejecutado el comando **COMMIT**, la operación ya habrá impactado en la DB y no podrá deshacerse con **ROLLBACK.**

*COMANDO SAVEPOINT*

**SAVEPOINT** nos permite **establecer** un **punto de recuperación** dentro de la transacción, utilizando un identificador.

Podemos hacer un **ROLLBACK** deshaciendo sólo las instrucciones que se han ejecutado hasta un determinado **SAVEPOINT** que se indique.Ideal para implementarlo en **modificaciones masivas de registros**, estableciendo una marca específica cada cierto bloque de registros modificados.

Si en algún punto de la modificación debemos ejecutar **ROLLBACK**, podemos hacerlo definiendo alguno de los **SAVEPOINT**, para no tener que perder todo el bloque de registros modificados.

Ten presente también que, al ejecutar el **COMMIT**, todo **SAVEPOINT** que hayamos establecido se perderá dado que en este punto, **ROLLBACK**, no puede volver a ejecutarse.

*DEFINIR EL INICIO DE UNA TRANSACCIÓN*

*LÍMITES DE UNA TRANSACCIÓN*

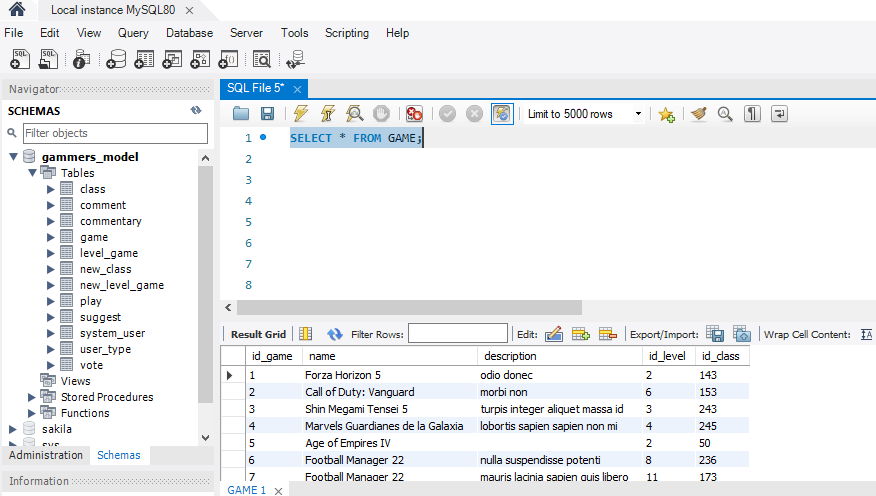
Los límites de una transacción son el **ámbito de espacio temporal** para cualquier tipo de modificación que se realice, invocando una o más cláusulas DML.

Para aplicar los comandos hasta aquí mencionados una transacción se inicia son siempre ejecuta, previo cualquier operación de modificación de datos, el comando **START TRANSACTION**.

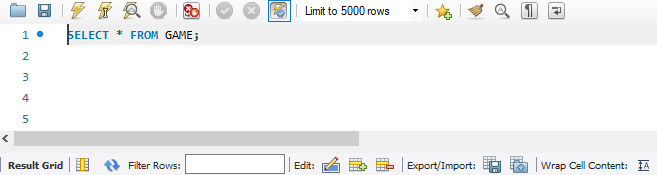
Por defecto MySQL se ejecuta en modo **autocommit**, esto significa que si una sentencia de **DML** no está dentro de una transacción, cada una de ellas es atómica, como si estuviera entre un START TRANSACTION y un COMMIT.

*INICIO DE UNA TRANSACCIÓN*

Antes de ingresar en las prácticas con código SQL, veamos cómo **Mysql Workbench** ayuda a entender las transacciones, a través de sus herramientas gráficas, para manipular operaciones DML.

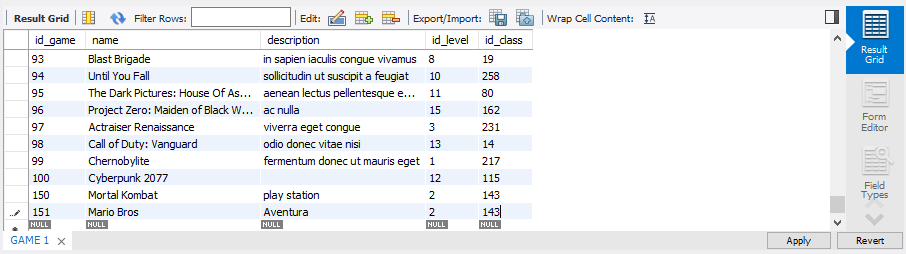
**

Ejecuta una consulta sobre alguna tabla. Verás que, al cargar la misma, existe un apartado para **agregar**, **editar**, o **eliminar** registros. Pulsa, ahora, alguno de estos botones y realiza en la tabla, la operación con el registro que has elegido.

**

Los **botones de edición** funcionan como **inicio de una transacción**:Puedes eliminar uno o más registros, agregar, o modificar uno existente pero, si no pulsas el botón **Apply**, los cambios no se harán efectivos.

Por lo tanto, **Apply** funciona como **COMMIT**, y **Revert** como **ROLLBACK**.



*AUTOCOMMIT*

Para definir el autocommit, MySQL cuenta con la variable llamada **autocommit**, seteada por defecto en **1** para que cada operación DML impacte automáticamente en la tabla.

**Para comenzar a trabajar con transacciones debemos desactivar previamente esta variable.**

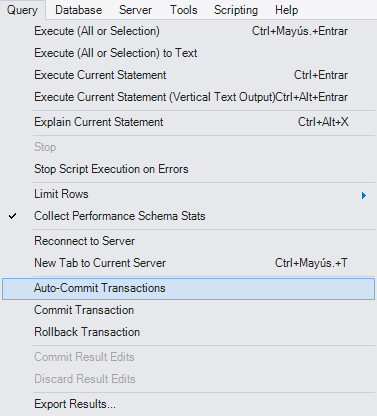
Escribamos en una ventana de script, lo siguiente:



Si su valor es **1** debemos pasarlo a **0**, ejecutando el siguiente comando en la pestaña de script:



También podemos verificar en el menú **Query > Auto-Commit Transactions** que no tenga el check. De tenerlo, haz clic sobre el punto de menú para desactivarlo.



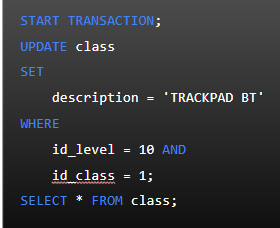
*START TRANSACTION*

Llevemos ejemplos con código a cada comando relacionado a transacciones que vimos en la primera parte de esta clase.

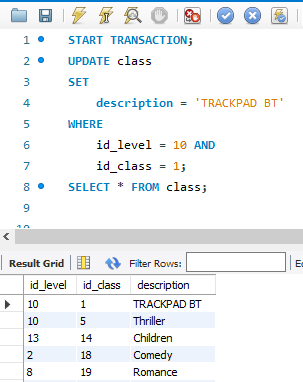
Comenzamos por el principal: **START TRANSACTION**.

*Veamos cómo entra en acción este comando y cómo se comporta al momento de ejecutar operaciones DML.*

Ejecutando una sentencia **UPDATE**, iniciando previamente la sentencia **START TRANSACTION**, veremos que el registro afectado se modificará sin problema alguno.

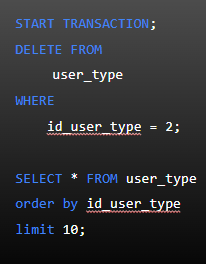
**

Si sales de la sesión **Mysql Workbench** y vuelves a ingresar, verás que el cambio solicitado al registro de la tabla en cuestión, no se asentó. Faltó finalizar el inicio de la transacción con **Commit** para reflejar dicho cambio.

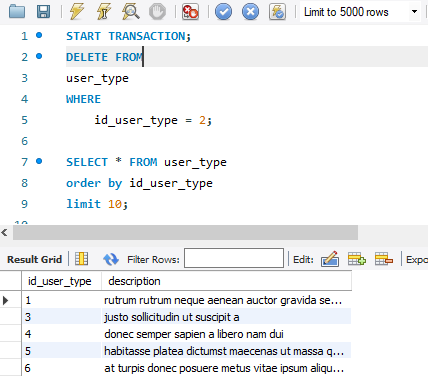


*ROLLBACK TRANSACTION*

Ejecutemos a continuación una **consulta de eliminación de un registro**. Iniciamos nuevamente la transacción, luego la **operación DML** y finalmente la consulta **SELECT** para visualizar el resultado.

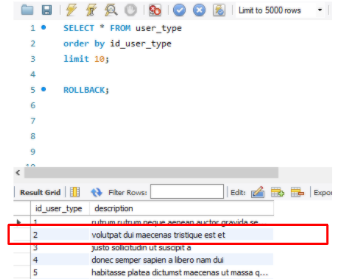


A través de la consulta **SELECT**, veremos que los registros son eliminados correctamente de la tabla. Ahora, en una nueva línea de la ventana de script, ejecutamos la sentencia rollback:



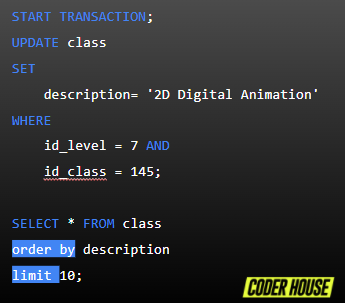
El registro que eliminamos anteriormente, volverá a su estado original.

Así es como funciona la combinación de **START TRANSACTION** y **ROLLBACK.**

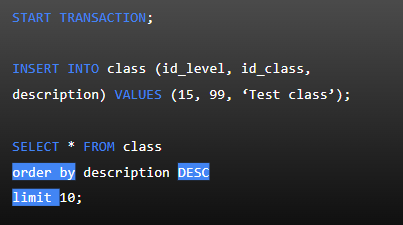
****

*COMMIT TRANSACTION*

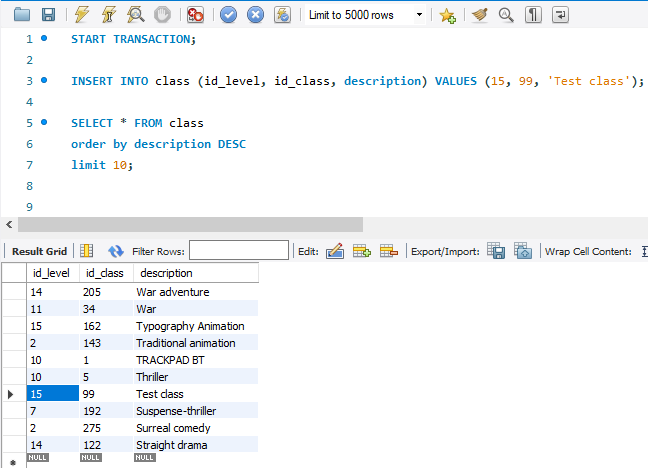
Ejecutando una sentencia **UPDATE** e iniciando previamente la sentencia **START TRANSACTION**, veremos que el registro afectado se modificará sin problema alguno.

****

Vamos con una nueva operación del tipo DML. En este caso, ejecutamos una inserción y luego, visualizamos los resultados con **SELECT**.

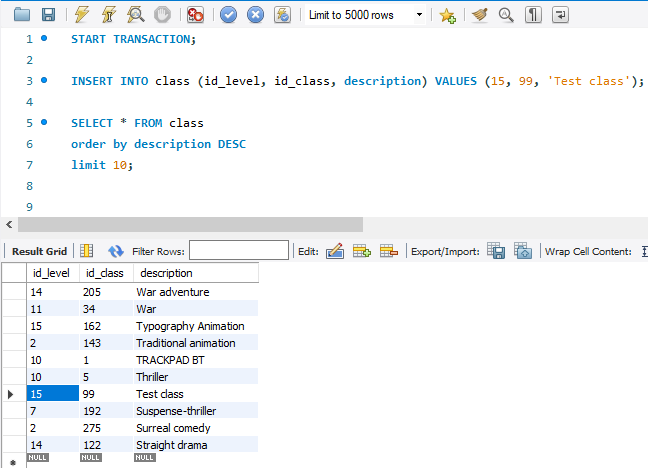
****

Vamos con una nueva operación del tipo DML.

Al igual que con los pasos anteriores, esta se hace efectiva apenas realizada.

Igualmente, para confirmar de manera definitiva, debemos ejecutar el comando commit:





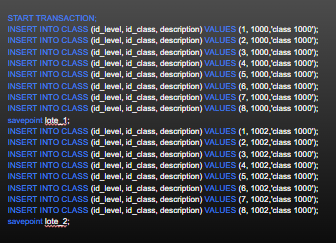
*SAVEPOINT*

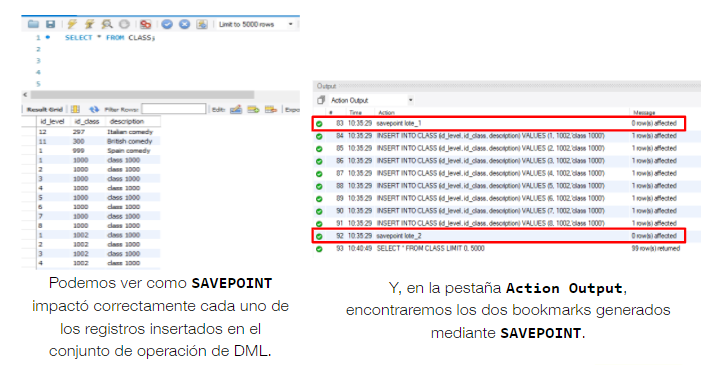
Veamos a continuación cómo **SAVEPOINT** nos ayuda a controlar una modificación masiva de registros, pudiendo **confirmar o deshacer por lotes**, según consideremos, acorde a la lógica operativa.

Ten presente que, **toda instrucción asociada a este comando**, es ejecutable solo en **bases de datos innoDB.**

El comando **SAVEPOINT** requiere establecer un identificador cada cierto punto para definir la posición del lote de registros insertados.

El criterio del nombre de cada savepoint lo definimos nosotros.





*ROLLBACK TO SAVEPOINT*

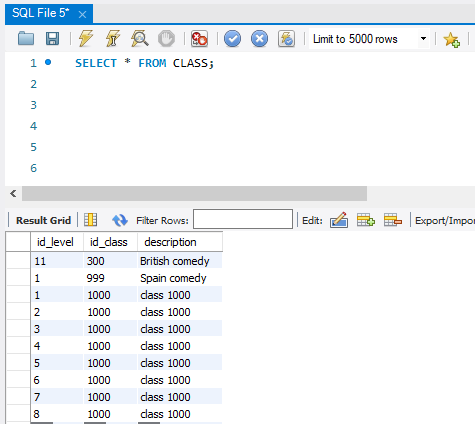
A través del comando **ROLLBACK TO SAVEPOINT**, podemos retroceder o “*deshacer*” el lote de comandos ejecutados hasta ese momento, de forma rápida y práctica

Su sentencia es:



Si ejecutamos **ROLLBACK TO lote\_1**  **desharemos los registros 11 al 20** insertados ,

El criterio del nombre de cada savepoint lo definimos nosotros.



*RELEASE SAVEPOINT*

También, en el caso que lo consideremos necesario, podemos eliminar un **SAVEPOINT** ejecutando la sentencia:



*IMPLEMENTACIÓN DE TRANSACCIONES*

La implementación de un control de transacciones, tanto para confirmar como para deshacer las mismas, es muy utilizado principalmente dentro   
de **Stored Procedures**

Combinando el mismo con el uso de variables, y la ejecución de varias sentencias DML que dependen unas de otras, el control de transacciones nos ayudará a mantener consistente cada una de las tablas de la base de datos.