



Clase 20. Curso SQL

SUBLENGUAJE TCL

***RECUERDA PONER A GRABAR LA
CLASE***





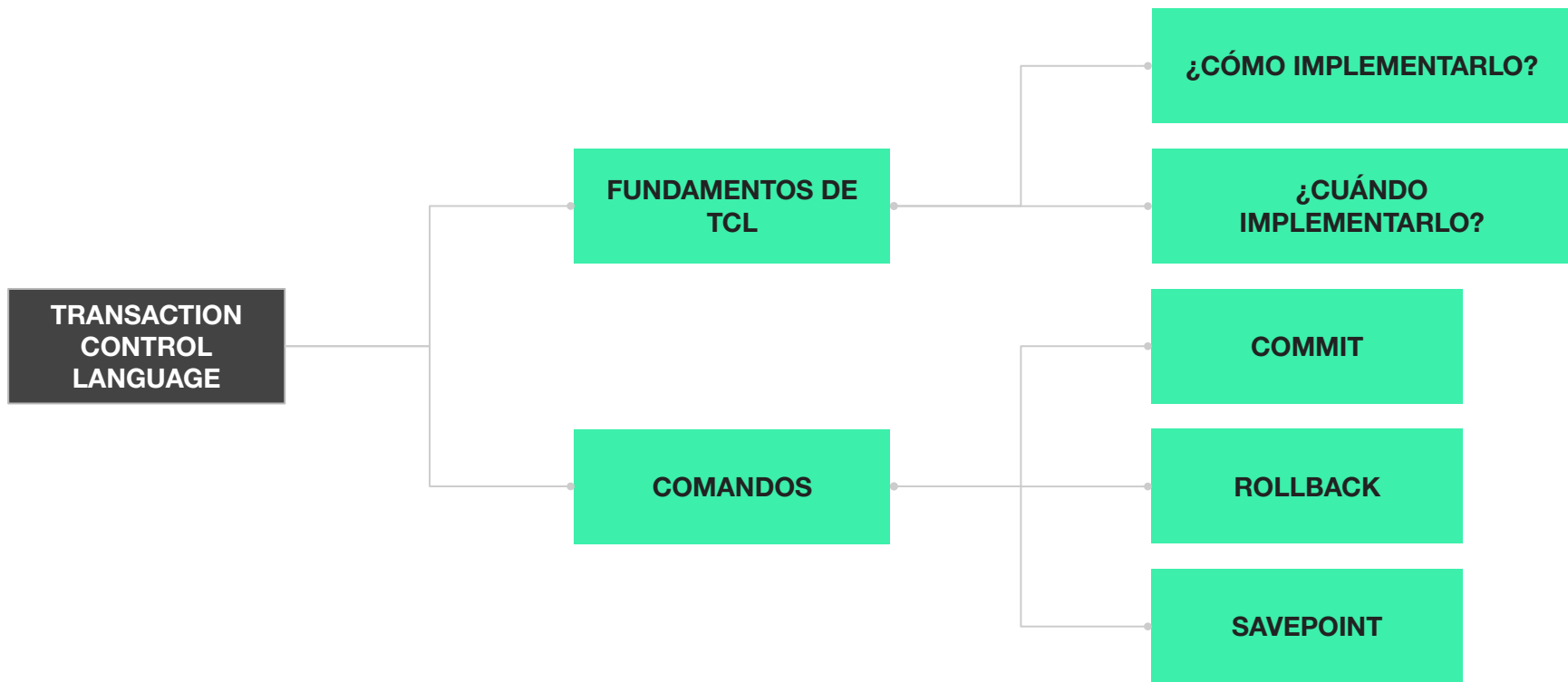
OBJETIVOS DE LA CLASE

- Reconocer e implementar las sentencias del sublenguaje TCL
- Identificar en qué situación usar cada sentencia.

MAPA DE CONCEPTOS

MAPA DE CONCEPTOS CLASE 20

¡Para
recordar!



SUBLENGUAJE TCL

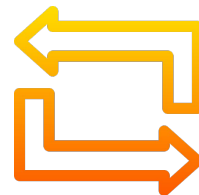
FUNDAMENTOS DE TCL

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**.

FUNDAMENTOS DE TCL

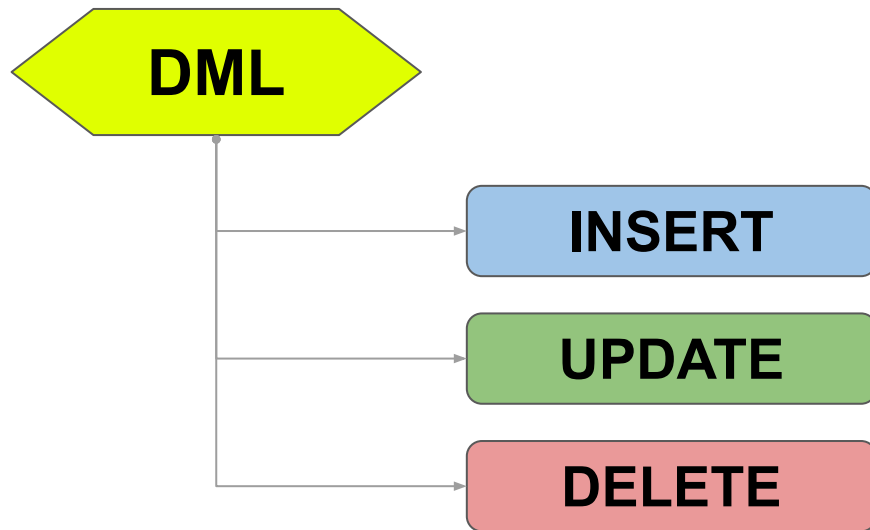


El papel de TCL es fundamental ya que, a través del mismo, controlamos las cláusulas u operaciones DML, agrupandolas de manera tal que se establezca una lógica transaccional cuando realizamos múltiples operaciones que afectan a una o más tablas.

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

FUNDAMENTOS DE TCL

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

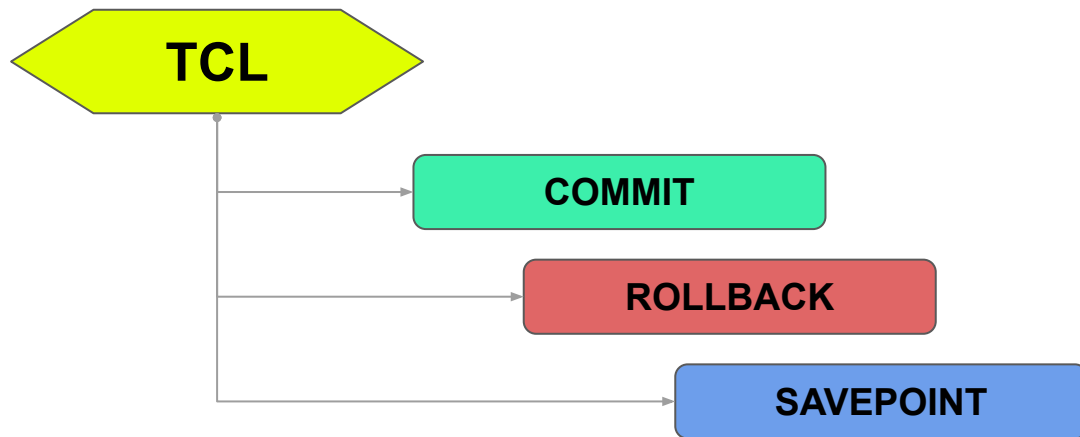


COMANDOS TCL PARA EL CONTROL DE TRANSACCIONES

COMANDOS TCL

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.

Estos comandos son:



COMANDO COMMIT

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.

COMANDO COMMIT

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

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

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.

COMANDO SAVEPOINT

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 representan el **ámbito de espacio temporal** para cualquier tipo de modificación que se realice sobre la DB cual puede involucrar numerosas sentencias de DML.

Para iniciar una transacción primeramente se debe escribir el comando **START TRANSACTION**, y luego se escriben aquellas sentencias de DML que se requiera modificar la DB.



LÍMITES DE UNA TRANSACCIÓN

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**.

PASO 1 INICIO DE UNA TRANSACCIÓN

Ejemplo
en vivo



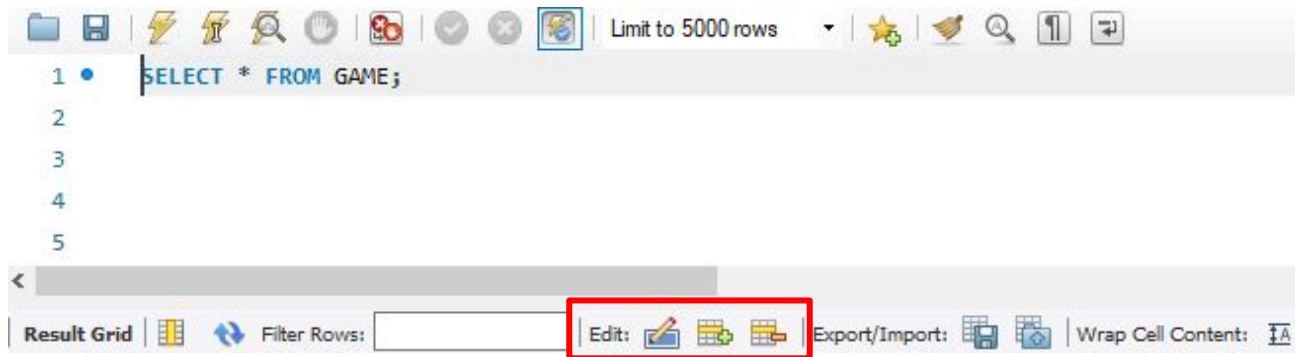
Antes de ingresar en las prácticas con código SQL, veamos cómo **Mysql Workbench** ayuda a entender las transacciones, a través de sus herramientas gráficas, para manipular operaciones DML.

The screenshot shows the MySQL Workbench interface for a local instance of MySQL 8.0. The 'Schemas' pane on the left shows the 'gammers_model' database selected, with a list of tables including 'class', 'comment', 'commentary', 'game', 'level_game', 'new_class', 'new_level_game', 'play', 'suggest', 'system_user', 'user_type', and 'vote'. The 'Query' pane in the center contains the SQL query: `SELECT * FROM GAME;`. The 'Result Grid' at the bottom displays the results of the query, showing a table with columns 'id_game', 'name', 'description', 'id_level', and 'id_class'. The results are as follows:

	id_game	name	description	id_level	id_class
1	1	Forza Horizon 5	odio donec	2	143
2	2	Call of Duty: Vanguard	morbi non	6	153
3	3	Shin Megami Tensei 5	turpis integer aliquet massa id	3	243
4	4	Marvels Guardianes de la Galaxia	lobortis sapien sapien non mi	4	245
5	5	Age of Empires IV		2	50
6	6	Football Manager 22	nulla suspendisse potenti	8	236
7	7	Football Manager 22	mauris lacinia sapien quis libero	11	173

PASO 2 *INICIO DE UNA TRANSACCIÓN*

Ejemplo
en vivo



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.

PASO 3 INICIO DE UNA TRANSACCIÓN

Ejemplo
en vivo



Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

	id_game	name	description	id_level	id_class
	93	Blast Brigade	in sapien iaculis congue vivamus	8	19
	94	Until You Fall	sollicitudin ut suscipit a feugiat	10	258
	95	The Dark Pictures: House Of As...	aenean lectus pellentesque e...	11	80
	96	Project Zero: Maiden of Black W...	ac nulla	15	162
	97	Actraiser Renaissance	viverra eget congue	3	231
	98	Call of Duty: Vanguard	odio donec vitae nisi	13	14
	99	Chernobylite	fermentum donec ut mauris eget	1	217
	100	Cyberpunk 2077		12	115
	150	Mortal Kombat	play station	2	143
	151	Mario Bros	Aventura	2	143
	NULL	NULL	NULL	NULL	NULL

GAME 1 x

Result Grid

Form Editor

Field Types

Apply Revert

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**.

CODER HOUSE

AUTOCOMMIT

CODER HOUSE

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:

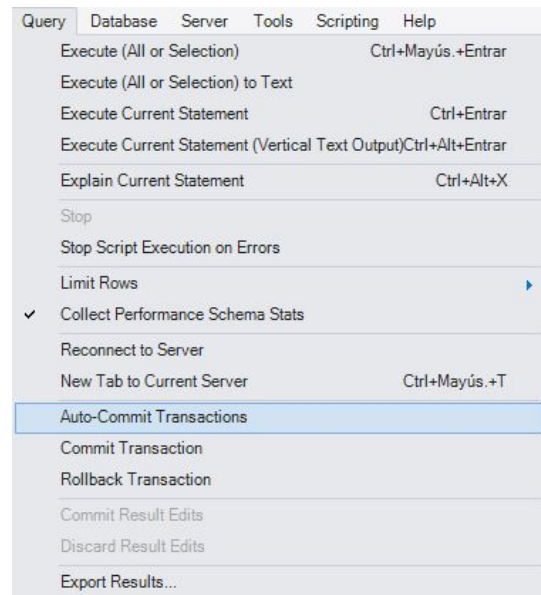
```
SELECT @@AUTOCOMMIT;
```

AUTOCOMMIT

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

```
SET AUTOCOMMIT = 0;
```

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.





BREAK

¡5/10 MINUTOS Y VOLVEMOS!

START TRANSACTION

START TRANSACTION

Llevemos a 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.

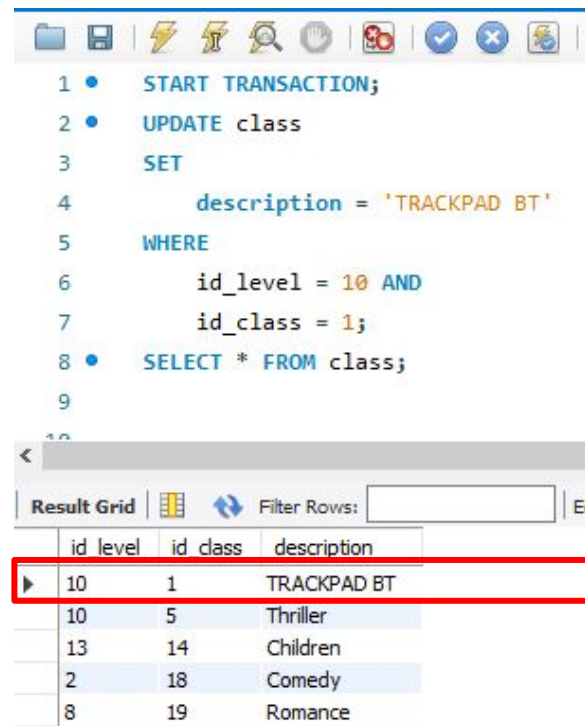
START TRANSACTION

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

```
START TRANSACTION;  
UPDATE class  
SET  
    description = 'TRACKPAD BT'  
WHERE  
    id_level = 10 AND  
    id_class = 1;  
SELECT * FROM class;
```

START TRANSACTION

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.



The screenshot shows the MySQL Workbench interface. The top toolbar contains icons for file operations, execution, and navigation. The SQL editor displays the following code:

```
1 • START TRANSACTION;
2 • UPDATE class
3   SET
4     description = 'TRACKPAD BT'
5   WHERE
6     id_level = 10 AND
7     id_class = 1;
8 • SELECT * FROM class;
9
```

Below the editor, the 'Result Grid' tab is active, showing a table with the following data:

	id_level	id_class	description
▶	10	1	TRACKPAD BT
	10	5	Thriller
	13	14	Children
	2	18	Comedy
	8	19	Romance

The first row of the result grid, containing the updated record, is highlighted with a red rectangular border.

ROLLBACK TRANSACTION

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.

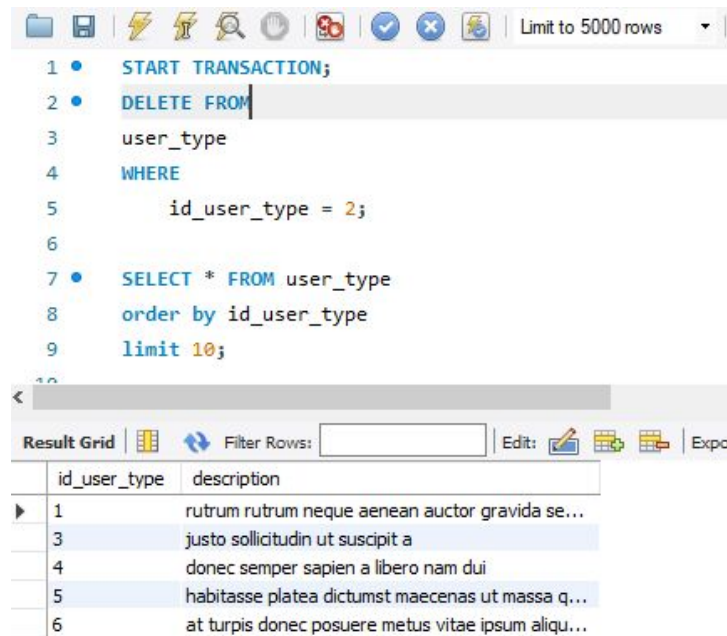
```
START TRANSACTION;  
DELETE FROM  
    user_type  
WHERE  
    id_user_type = 2;  
  
SELECT * FROM user_type  
order by id_user_type  
limit 10;
```

ROLLBACK TRANSACTION

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:

ROLLBACK;



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and search. The script editor contains the following SQL code:

```
1 • START TRANSACTION;
2 • DELETE FROM
3   user_type
4   WHERE
5     id_user_type = 2;
6
7 • SELECT * FROM user_type
8   order by id_user_type
9   limit 10;
```

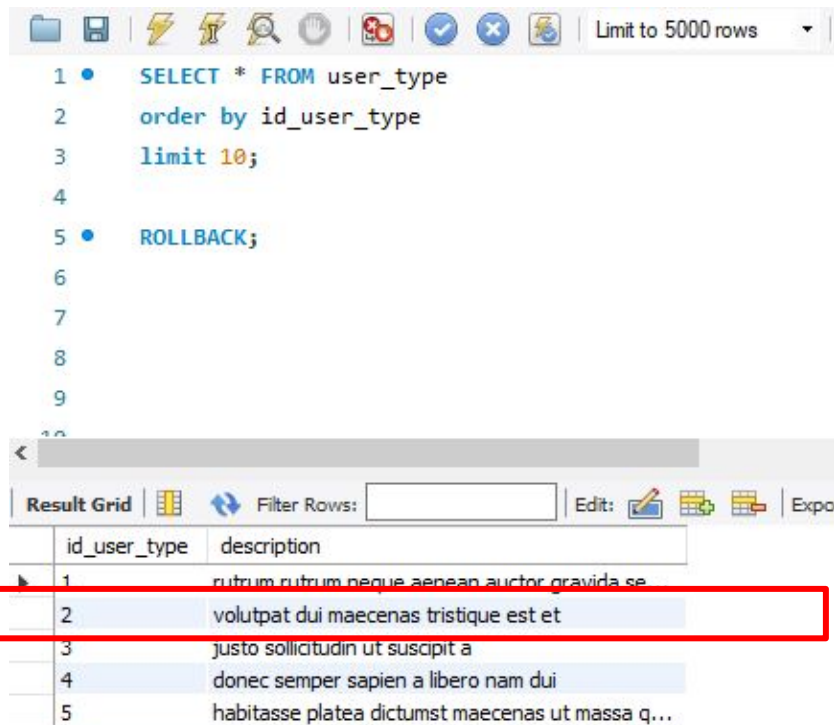
Below the script editor, the 'Result Grid' tab is active, displaying a table with two columns: 'id_user_type' and 'description'. The table contains six rows of data:

id_user_type	description
1	rutrum rutrum neque aenean auctor gravida se...
3	justo sollicitudin ut suscipit a
4	donec semper sapien a libero nam dui
5	habitasse platea dictumst maecenas ut massa q...
6	at turpis donec posuere metus vitae ipsum aliqu...

ROLLBACK TRANSACTION

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

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



The screenshot shows a database management tool interface. At the top, there is a toolbar with various icons and a dropdown menu set to 'Limit to 5000 rows'. Below the toolbar, a SQL query is entered in a text area:

```
1 • SELECT * FROM user_type
2   order by id_user_type
3   limit 10;
4
5 • ROLLBACK;
6
7
8
9
10
```

Below the query editor, there is a 'Result Grid' section. It contains a table with two columns: 'id_user_type' and 'description'. The table has five rows of data. The second row is highlighted with a red border:

id_user_type	description
1	rutrum rutrum neque aenean auctor gravida se
2	volutpat dui maecenas tristique est et
3	justo sollicitudin ut suscipit a
4	donec semper sapien a libero nam dui
5	habitasse platea dictumst maecenas ut massa q...

COMMIT TRANSACTION

START TRANSACTION

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

```
START TRANSACTION;  
UPDATE class  
SET  
    description= '2D Digital Animation'  
WHERE  
    id_level = 7 AND  
    id_class = 145;  
  
SELECT * FROM class  
order by description  
limit 10;
```

COMMIT TRANSACTION

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

```
START TRANSACTION;
```

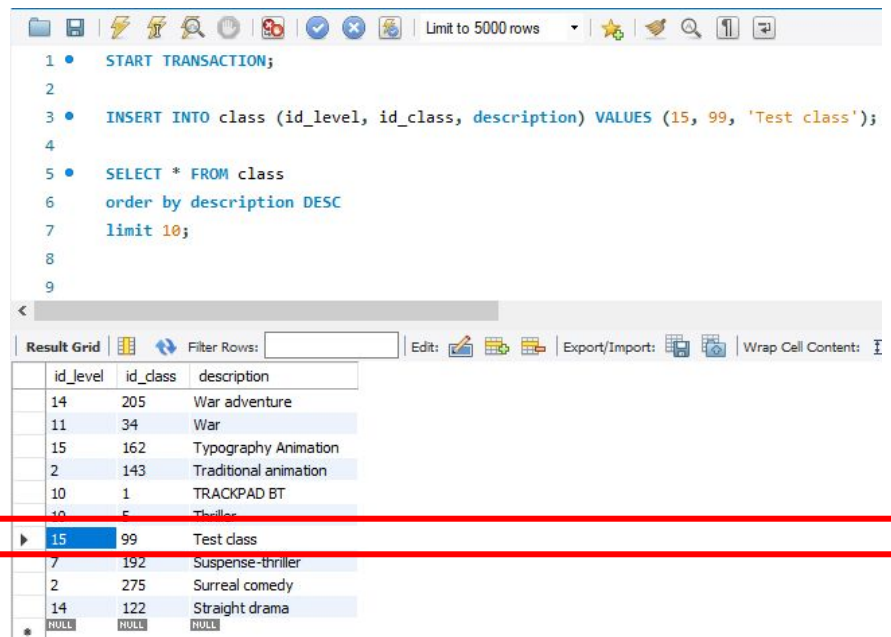
```
INSERT INTO class (id_level, id_class,  
description) VALUES (15, 99, 'Test class');
```

```
SELECT * FROM class  
order by description DESC  
limit 10;
```

COMMIT TRANSACTION

Vamos con una nueva operación del tipo DML.

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



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and settings. The SQL editor contains the following code:

```
1 • START TRANSACTION;  
2  
3 • INSERT INTO class (id_level, id_class, description) VALUES (15, 99, 'Test class');  
4  
5 • SELECT * FROM class  
6   order by description DESC  
7   limit 10;  
8  
9
```

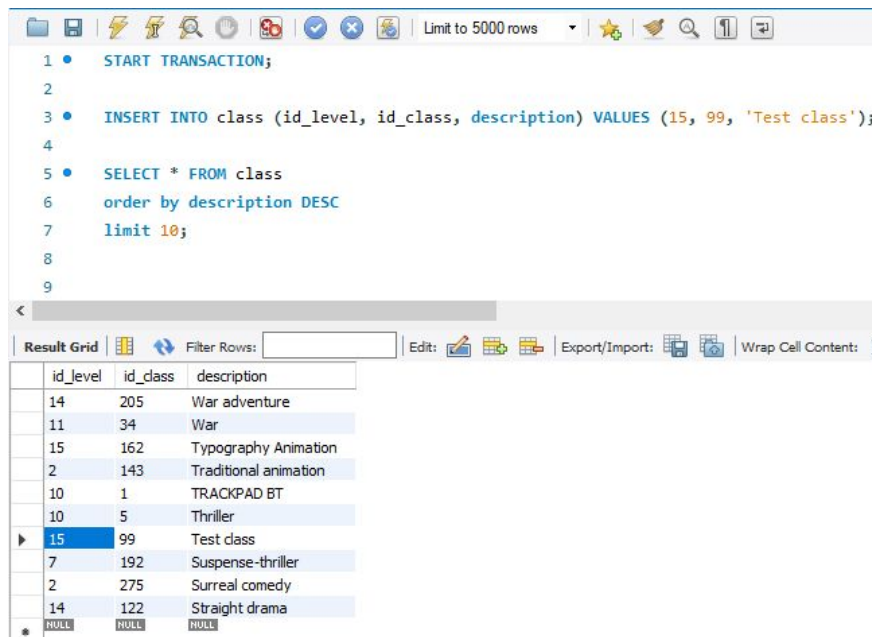
Below the editor, the 'Result Grid' is displayed, showing the results of the SELECT statement. The grid has columns for 'id_level', 'id_class', and 'description'. The row with 'id_level' 15 and 'id_class' 99 is highlighted in blue and enclosed in a red rectangle.

id_level	id_class	description
14	205	War adventure
11	34	War
15	162	Typography Animation
2	143	Traditional animation
10	1	TRACKPAD BT
10	5	Thriller
15	99	Test class
7	192	Suspense-thriller
2	275	Surreal comedy
14	122	Straight drama
NULL	NULL	NULL

COMMIT TRANSACTION

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

COMMIT;



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and settings. The SQL editor contains the following code:

```
1 • START TRANSACTION;
2
3 • INSERT INTO class (id_level, id_class, description) VALUES (15, 99, 'Test class');
4
5 • SELECT * FROM class
6   order by description DESC
7   limit 10;
8
9
```

Below the editor, the 'Result Grid' tab is active, displaying a table of results. The table has three columns: 'id_level', 'id_class', and 'description'. The row with 'id_level' 15 and 'id_class' 99 is highlighted in blue, indicating it is the current row. The table contains 10 rows of data, including the newly inserted 'Test class'.

id_level	id_class	description
14	205	War adventure
11	34	War
15	162	Typography Animation
2	143	Traditional animation
10	1	TRACKPAD BT
10	5	Thriller
15	99	Test class
7	192	Suspense-thriller
2	275	Surreal comedy
14	122	Straight drama
NULL	NULL	NULL

SAVEPOINT

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**.

SAVEPOINT

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.

```
START TRANSACTION;  
INSERT INTO CLASS (id_level, id_class, description) VALUES (1, 1000,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (2, 1000,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (3, 1000,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (4, 1000,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (5, 1000,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (6, 1000,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (7, 1000,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (8, 1000,'class 1000');  
savepoint lote_1;  
INSERT INTO CLASS (id_level, id_class, description) VALUES (1, 1002,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (2, 1002,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (3, 1002,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (4, 1002,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (5, 1002,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (6, 1002,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (7, 1002,'class 1000');  
INSERT INTO CLASS (id_level, id_class, description) VALUES (8, 1002,'class 1000');  
savepoint lote_2;
```


Limit to 5000 rows

```
1 SELECT * FROM CLASS;
```

Result Grid

	id_level	id_class	description
12	297	Italian comedy	
11	300	British comedy	
1	999	Spain comedy	
1	1000	class 1000	
2	1000	class 1000	
3	1000	class 1000	
4	1000	class 1000	
5	1000	class 1000	
6	1000	class 1000	
7	1000	class 1000	
8	1000	class 1000	
1	1002	class 1000	
2	1002	class 1000	
3	1002	class 1000	
4	1002	class 1000	

Podemos ver como **SAVEPOINT** impactó correctamente cada uno de los registros insertados en el conjunto de operación de DML.

Output

Action Output

#	Time	Action	Message
83	10:35:29	savepoint lote_1	0 row(s) affected
84	10:35:29	INSERT INTO CLASS (id_level, id_class, description) VALUES (1, 1002,'class 1000')	1 row(s) affected
85	10:35:29	INSERT INTO CLASS (id_level, id_class, description) VALUES (2, 1002,'class 1000')	1 row(s) affected
86	10:35:29	INSERT INTO CLASS (id_level, id_class, description) VALUES (3, 1002,'class 1000')	1 row(s) affected
87	10:35:29	INSERT INTO CLASS (id_level, id_class, description) VALUES (4, 1002,'class 1000')	1 row(s) affected
88	10:35:29	INSERT INTO CLASS (id_level, id_class, description) VALUES (5, 1002,'class 1000')	1 row(s) affected
89	10:35:29	INSERT INTO CLASS (id_level, id_class, description) VALUES (6, 1002,'class 1000')	1 row(s) affected
90	10:35:29	INSERT INTO CLASS (id_level, id_class, description) VALUES (7, 1002,'class 1000')	1 row(s) affected
91	10:35:29	INSERT INTO CLASS (id_level, id_class, description) VALUES (8, 1002,'class 1000')	1 row(s) affected
92	10:35:29	savepoint lote_2	0 row(s) affected
93	10:40:49	SELECT * FROM CLASS LIMIT 0, 5000	99 row(s) returned

Y, en la pestaña **Action Output**, encontraremos los dos bookmarks generados mediante **SAVEPOINT**.

ROLLBACK TO SAVEPOINT

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

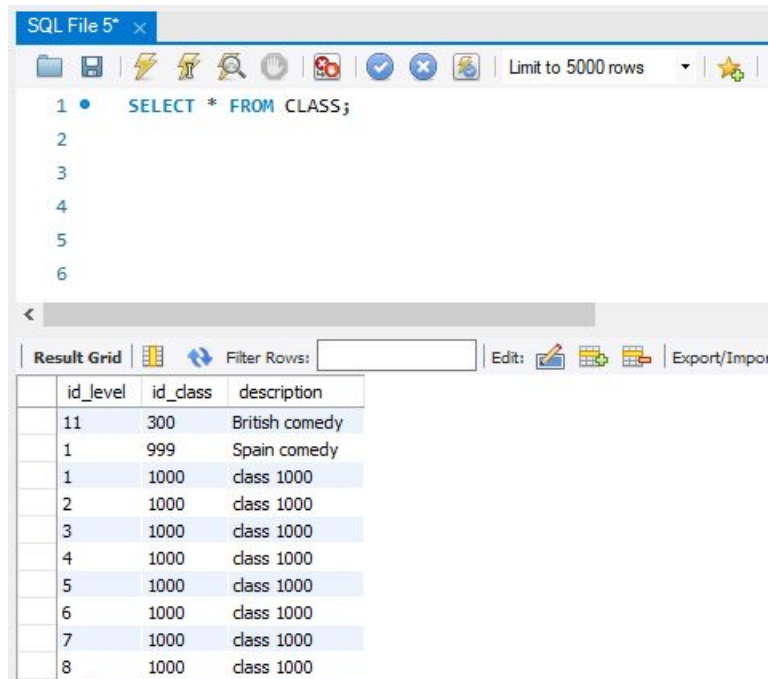
Su sentencia es:

```
ROLLBACK TO <savepoint>;
```

SAVEPOINT

Si ejecutamos **ROLLBACK TO lote_1**
desharemos los registros 11 al 20
insertados ,

El criterio del nombre de cada savepoint lo
definimos nosotros.



The screenshot shows a SQL IDE window titled "SQL File 5* x". The query editor contains the SQL statement: `SELECT * FROM CLASS;`. Below the editor, the "Result Grid" tab is active, displaying the results of the query. The results are shown in a table with three columns: `id_level`, `id_class`, and `description`. The table contains 11 rows of data. The first row has `id_level` 11, `id_class` 300, and `description` "British comedy". The second row has `id_level` 1, `id_class` 999, and `description` "Spain comedy". The remaining nine rows have `id_level` values from 1 to 8, `id_class` 1000, and `description` "class 1000".

	id_level	id_class	description
1	11	300	British comedy
2	1	999	Spain comedy
3	1	1000	class 1000
4	2	1000	class 1000
5	3	1000	class 1000
6	4	1000	class 1000
7	5	1000	class 1000
8	6	1000	class 1000
9	7	1000	class 1000
10	8	1000	class 1000
11			

RELEASE SAVEPOINT

CODER HOUSE

RELEASE SAVEPOINT

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

```
RELEASE <savepoint>;
```



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.



EJEMPLO EN VIVO

Veremos cómo implementar las sentencias start transaction, savepoint, commit y rollback.



MODIFICACIONES CONTROLADAS MEDIANTE TRANSACCIONES

Implementaremos el sublenguaje TCL.

Tiempo estimado: 15 minutos

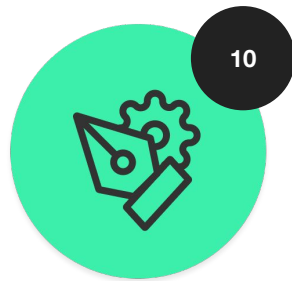
MODIFICACIONES CONTROLADAS MEDIANTE TRANSACCIONES

Desafío
generico



Trabajaremos con la tabla **PAY** de la DB GAMERS

1. Abrir una nueva pestaña script y elimina tres pagos de los que insertamos en la filmina 49, iniciando una transacción.
2. Validar en otra pestaña script, los registros eliminados.
3. A continuación, en la primera pestaña de script, revertir la eliminación de pagos.
4. En la tabla pagos, insertar un lote de 15 pagos, iniciando previamente la transacción y estableciendo un **savepoint** luego del registro **#5**, y otro luego del registro **#10**
5. Validar en otra pestaña script, los registros insertados
6. Eliminar el segundo savepoint
7. Confirmar la transacción y vuelve a validar en otra pestaña script los registros insertados



SENTENCIAS DEL SUBLENGUAJE TCL

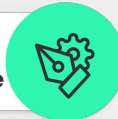
Presentaremos en formato .sql, el script de la implementación de dichas sentencias.

SENTENCIAS DEL SUBLENGUAJE TCL

Formato: El archivo a presentar debe tener como nombre "TCL+Apellido".

Sugerencia: Será, como siempre, un archivo del tipo .sql.

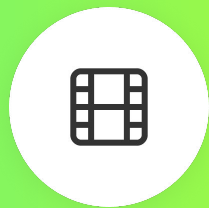
Desafío
entregable



>> Consigna: Elige dos tablas de las presentadas en tu proyecto. Realizarás una serie de modificaciones en los registros, controladas por transacciones.

>>Aspectos a incluir en el entregable:

1. En la **primera tabla**, si tiene registros, deberás eliminar algunos de ellos iniciando previamente una transacción. Si no tiene registros la tabla, reemplaza eliminación por inserción.
 2. Deja en una línea siguiente, comentado la sentencia Rollback, y en una línea posterior, la sentencia Commit.
 3. Si eliminas registros importantes, deja comenzado las sentencias para re-insertarlos.
-
1. En la segunda tabla, inserta ocho nuevos registros iniciando también una transacción.
 2. Agrega un savepoint a posteriori de la inserción del registro #4 y otro savepoint a posteriori del registro #8
 3. Agrega en una línea comentada la sentencia de eliminación del savepoint de los primeros 4 registros insertados



***¿QUIERES SABER MÁS? TE DEJAMOS
MATERIAL AMPLIADO DE LA CLASE***



- [Transacciones](#) | **Blog de José Juan Sánchez**
- [Savepoint y sentencias asociadas](#) | **Blog de José Juan Sánchez**

Disponible en nuestro repositorio.

¿PREGUNTAS?



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Transacciones.
- Start Transaction.
- Commit y Rollback.
- Savepoint.
- Release y Rollback to savepoint.



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE