



**Argentina
programa
4.0**



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

***primero
la gente***

Clase 30: Bases de datos Relacionales

Transaction Control Language

Agenda de hoy

- A. TRANSACTION CONTROL LANGUAGE
 - a. START TRANSACTION / BEGIN
 - b. COMMIT
 - c. ROLLBACK
 - d. Savepoint
- B. VISTAS SQL
 - a. Crear una Vista SQL
 - b. Modificar una Vista SQL
 - c. Eliminar una Vista SQL
- C. CONSULTAS SOBRE EL PROYECTO FINAL



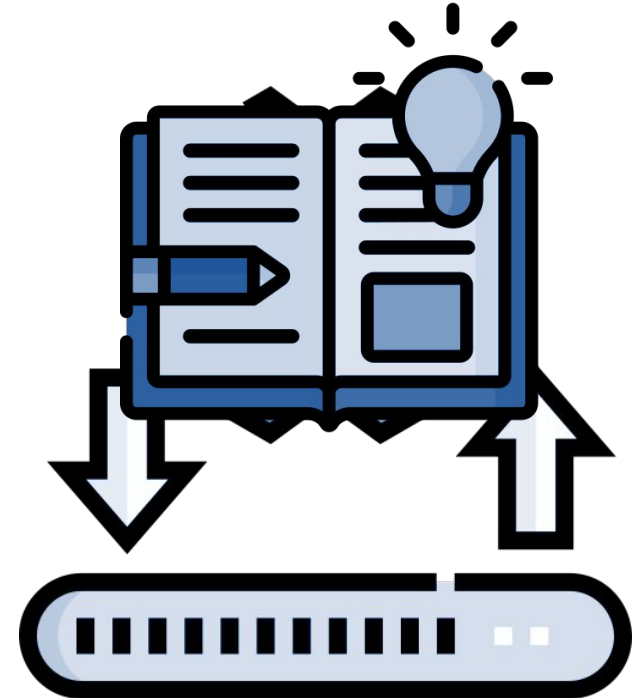
Transaction Control Language

Transaction Control Language

Hasta el momento, **todas las operaciones DML que realizamos sobre SQL manejan una estructura transaccional directa.**

Esto significa que, cuando ejecutamos cualquier operación de inserción, modificación o eliminación de datos, estas impactan directamente en la o las tablas vinculadas.

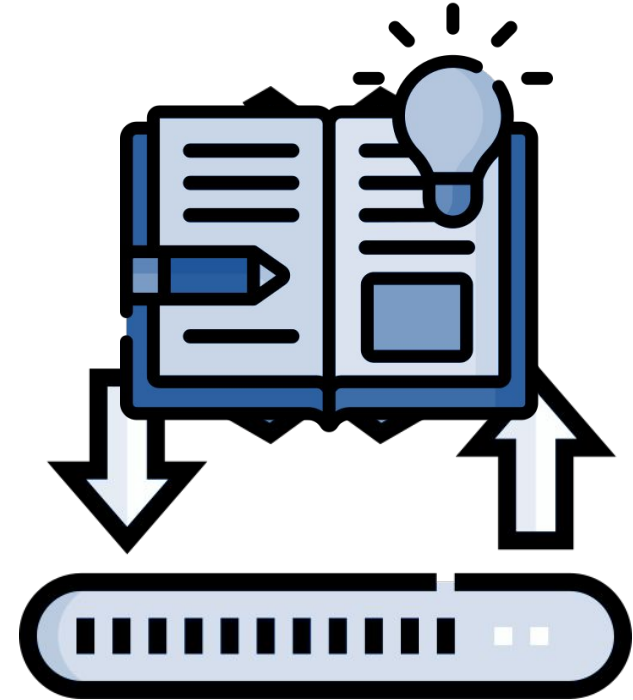
Pero si analizamos este proceso, **ante el más mínimo error que cometamos en la instrucción, esta afectará los datos de manera directa.**



Transaction Control Language

Pero, entre todas las opciones que maneja el lenguaje SQL, para los diferentes escenarios de operaciones que se pueden realizar, encontramos uno que nos ayudará a prevenir cualquier posible error: este es el sublenguaje **TCL**.

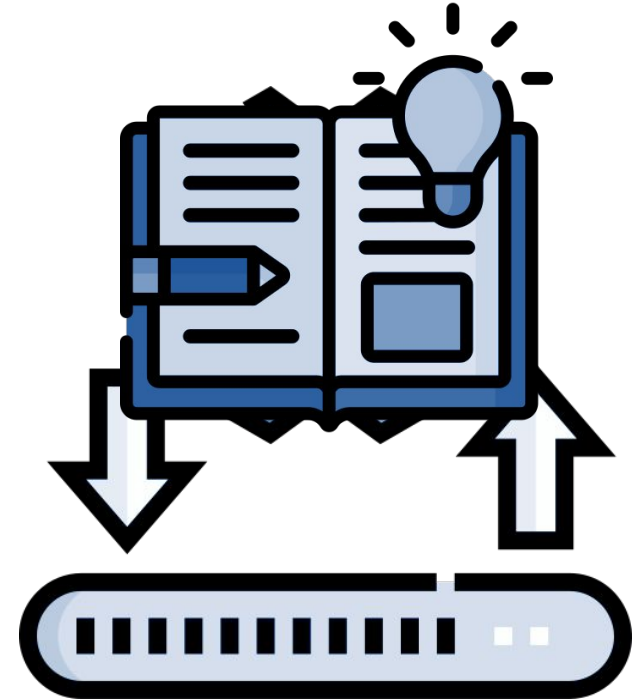
Su sigla proviene de **Transaction Control Language**, y es el sublenguaje encargado de administrar las transacciones en una base de datos. TCL es utilizado para administrar cada operación que realicemos mediante cláusulas DML.



Transaction Control Language

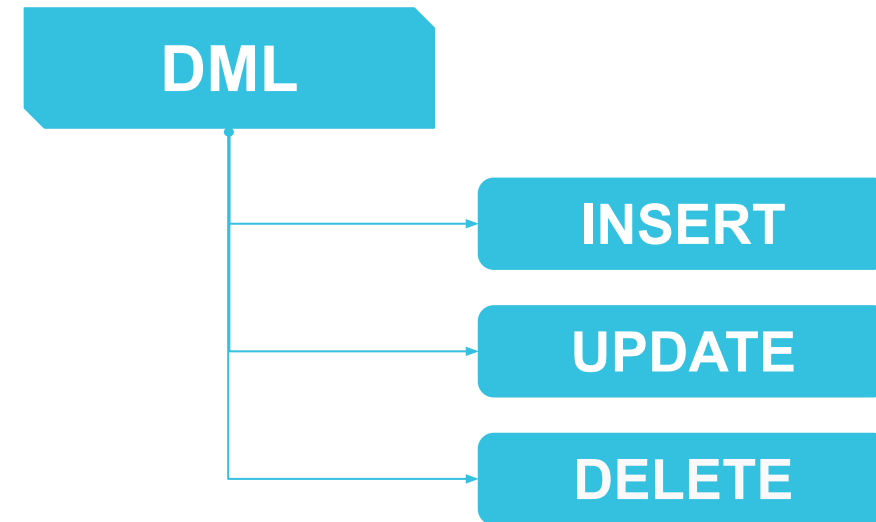
El rol de TCL es fundamental ya que, a través del mismo, obtenemos el control de las cláusulas operativas DML, agrupando las mismas de manera tal para que se establezca una lógica transaccional cuando realizamos múltiples operaciones afectando a los datos de una o más tablas.

TCL es la herramienta efectiva, la cual nos ayuda a mantener la integridad de los datos manipulados.



Transaction Control Language

Para refrescar conocimientos, las operaciones DML son aquellas que nos permiten trabajar con los tres tipos de operaciones más importantes sobre una bb.dd:

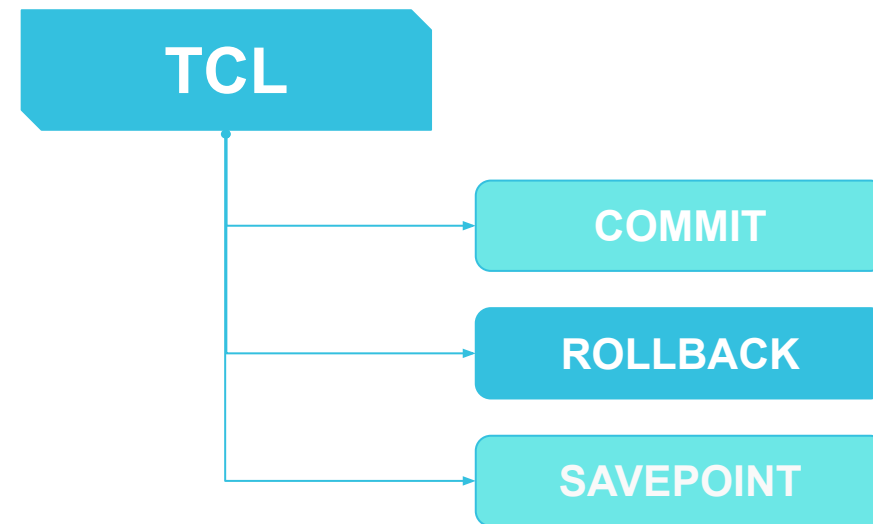


COMANDOS TCL PARA CONTROLAR TRANSACCIONES

Comandos TCL para controlar transacciones

MySQL incluye tres comandos integrados en el lenguaje SQL, los cuales se integran a las cláusulas homónimas, para controlar las operaciones DML durante el proceso de ejecución de las mismas.

Estos son:

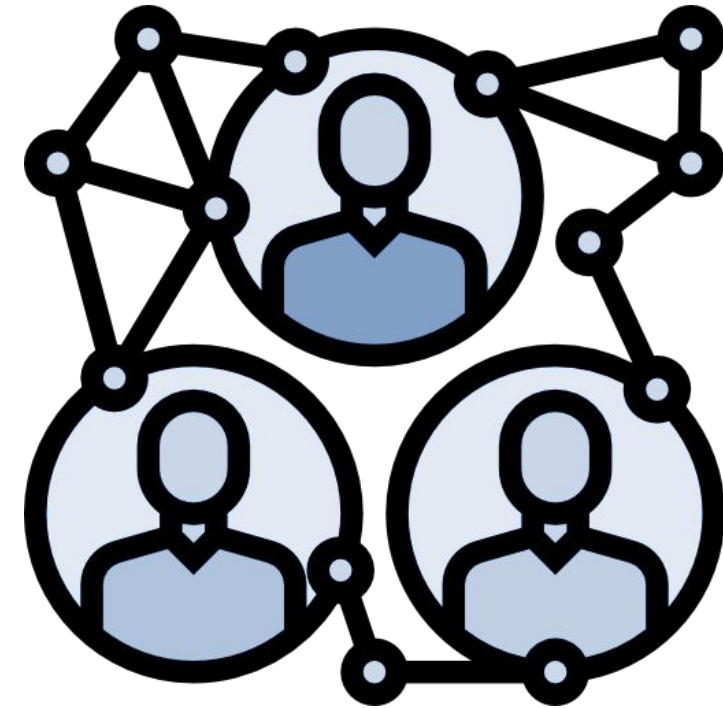


COMMIT

Comandos TCL para controlar transacciones

COMMIT es un comando el cual permite “*confirmar*” la transacción o transacciones realizadas sobre una o más tablas.

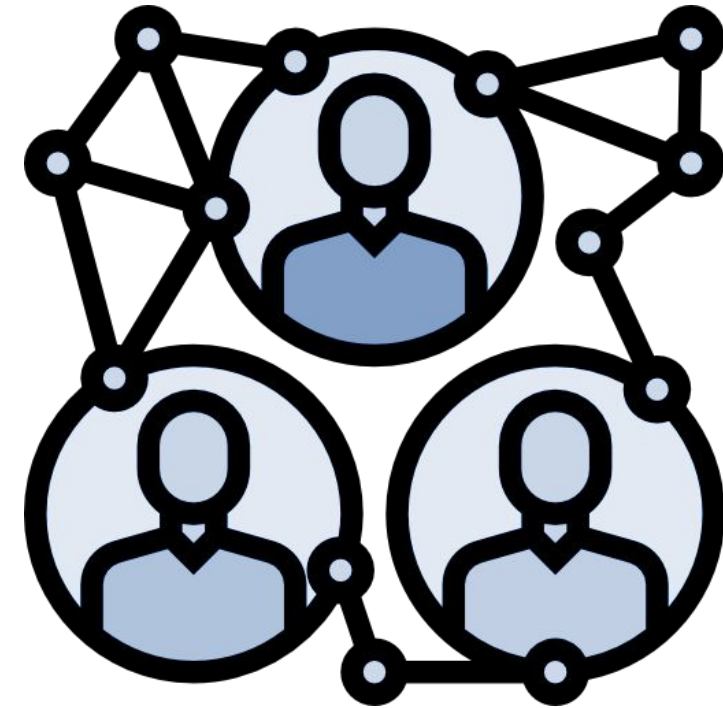
Cuando éste es ejecutado, se ocupa de guardar los cambios realizados en la o las tabla(s) a través del proceso de confirmación (commit), haciendo a estos permanentes.



Comandos TCL para controlar transacciones

Cuando ejecutamos alguna de las cláusulas DML, la(s) modificación(es) realizada(s) se guardan de forma “*temporal*” en la memoria de la computadora que las ejecuta.

Al invocar el comando COMMIT, dichas modificaciones terminan impactando de forma definitiva en el Servidor de base de datos, reflejando así los cambios en la tabla o las tablas.

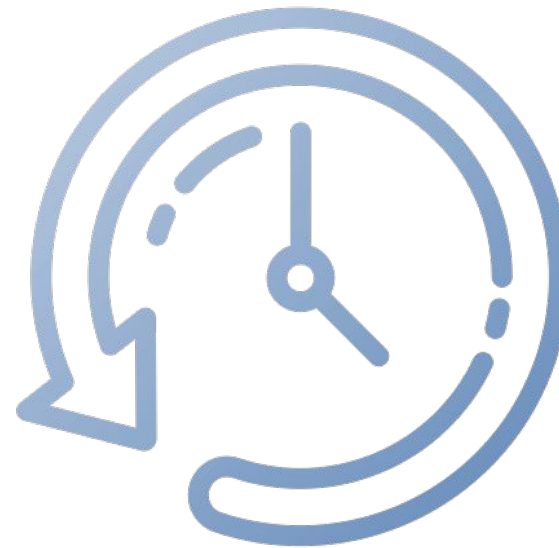


ROLLBACK

Comandos TCL para controlar transacciones

ROLLBACK “*deshace*” la operación DML realizada previamente. Su rol es, básicamente, volver al estado anterior todos los cambios aplicados sobre todas las tablas en cuestión.

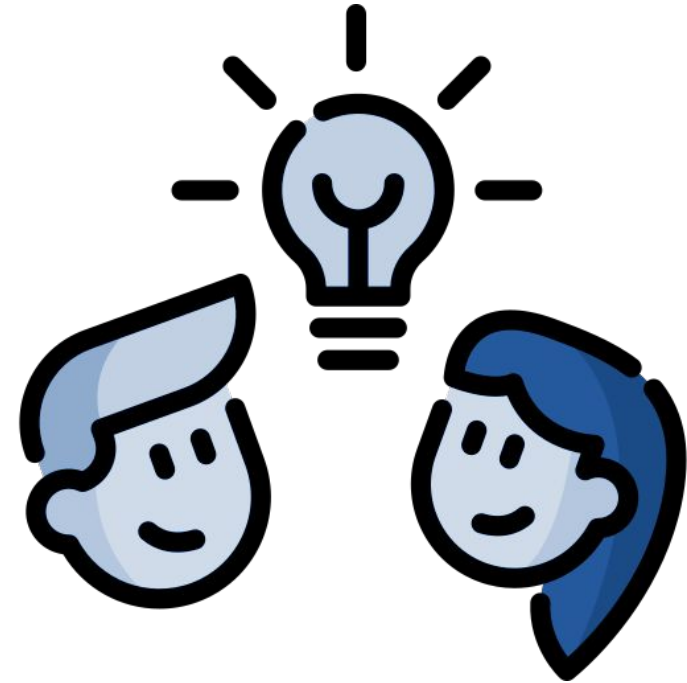
Su comportamiento es similar al uso del comando **Undo** (*deshacer*), o **Ctrl + Z**, que utilizamos de forma frecuente en cualquier aplicación de software en una computadora.



Comandos TCL para controlar transacciones

El comando ROLLBACK solo funciona para revertir modificaciones, en escenarios donde no se ha ejecutado previamente el comando COMMIT.

En el caso de realizar una operación DML y ejecutar el comando COMMIT inmediatamente, la operación habrá impactado en el motor de base de datos y ROLLBACK no surtirá efecto alguno.

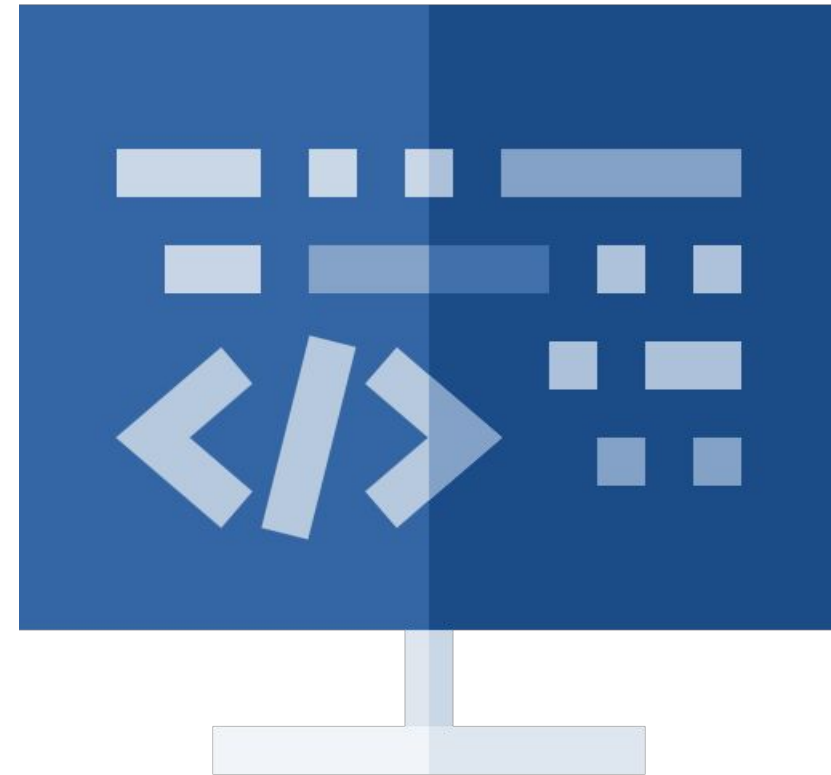


Savepoint

Comandos TCL para controlar transacciones

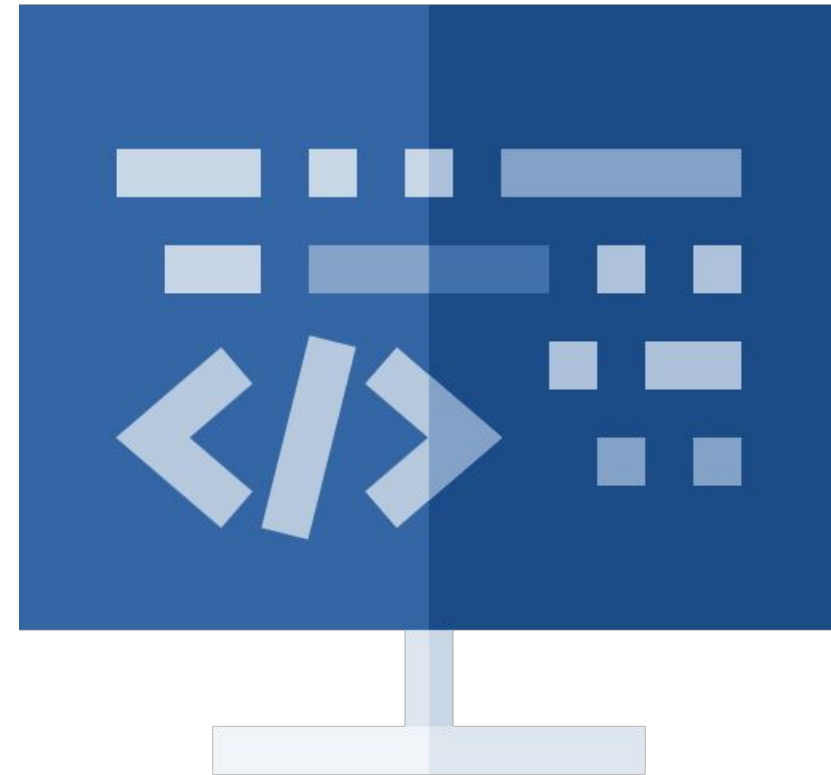
SAVEPOINT funciona en combinación con Rollback como una especie de Bookmark para establecer un punto de retroceso al momento de ejecutar el comando **ROLLBACK**.

Es ideal para implementarlo en modificaciones masivas de registros, estableciendo una marca específica cada cierto bloque de registros modificados.



Comandos TCL para controlar transacciones

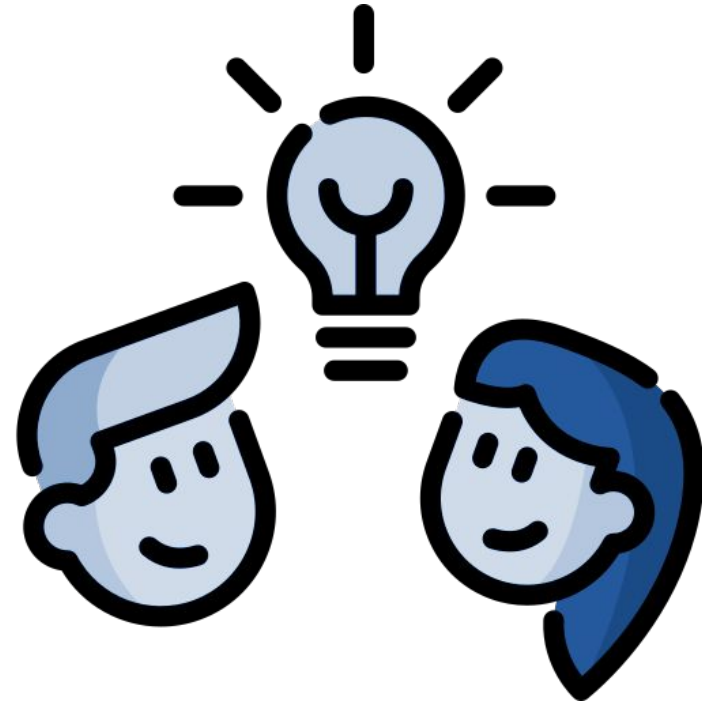
De esta forma, si en algún punto de la modificación masiva debemos ejecutar ROLLBACK, podemos hacerlo definiendo alguno de los savepoint definidos, para no tener que perder todo el bloque de registros modificados.



Comandos TCL para controlar transacciones

Al igual que lo visto anteriormente con el uso de Rollback, una vez ejecutado el comando COMMIT sobre alguna transacción en cuestión, todo tipo de SAVEPOINT que hayamos establecido previamente, se perderá.

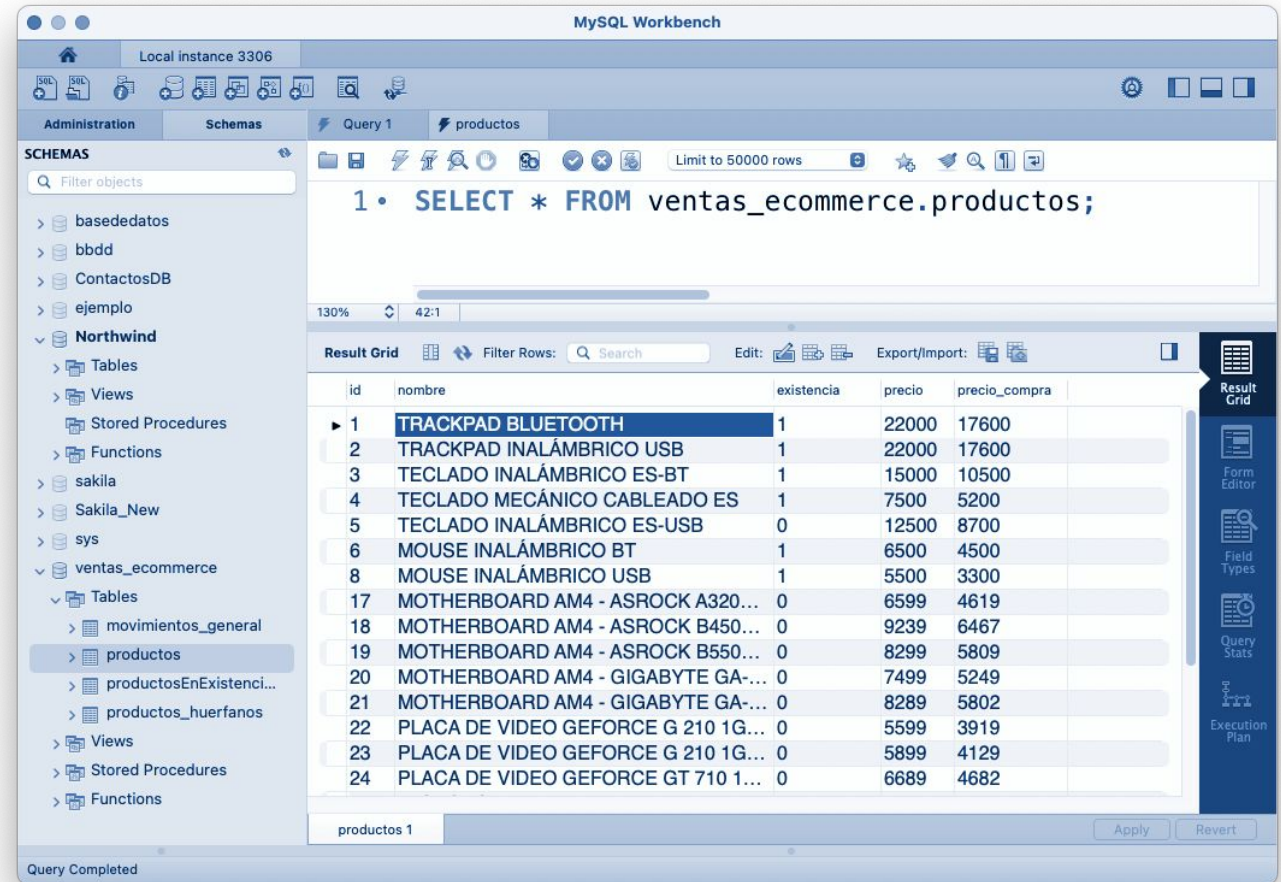
Esto sucede porque, SAVEPOINT, funciona en combinación con ROLLBACK.



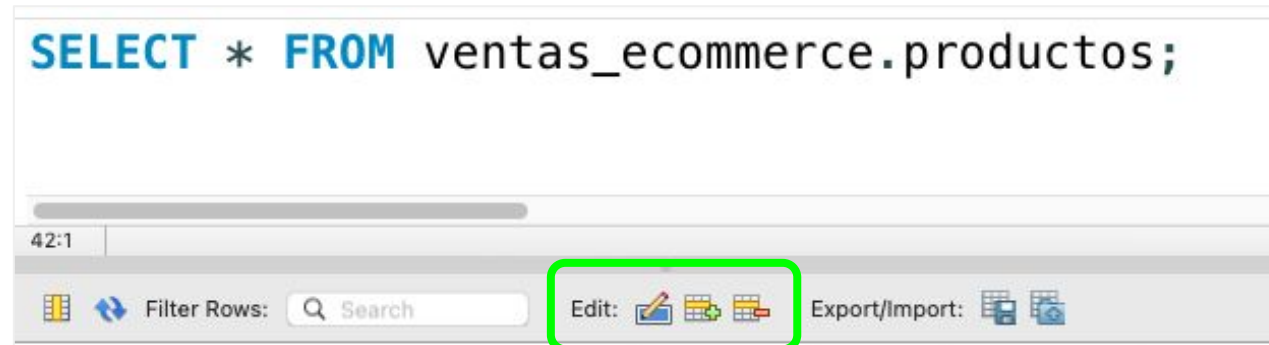
Escenario transaccional actual

Escenario transaccional actual

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.



Escenario transaccional actual



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.

Escenario transaccional actual

22	PLACA DE VIDEO GEFORCE G 210 1G...	0	5899	4129		Plan
23	PLACA DE VIDEO GEFORCE G 210 1G...	0	5899	4129		
24	PLACA DE VIDEO GEFORCE GT 710 1...	0	6689	4682		

productos 1

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, el botón **Apply** toma el rol de la cláusula **COMMIT**, y el botón **Revert** el rol de la cláusula **ROLLBACK**.

Begin Transaction

Si deseamos aplicar los comandos mencionados anteriormente, podemos hacerlo ante cualquier escenario u operación DML, escribiendo simplemente la cláusula **START TRANSACTION.**

Este será quien defina el ámbito de espacio temporal para cualquier tipo de modificación que realices, invocando una o más cláusulas DML.



Begin Transaction

Alternativamente, y a modo de poder ser compatible con otros motores SQL existentes en el mercado, MySQL también le da soporte a la cláusula **BEGIN**, propia de otros motores.

```
Transacciones  
  
START TRANSACTION  
  
-- o simplemente  
  
BEGIN
```

Autocommit

Begin Transaction

Mysql cuenta con una variable de entorno llamada **autocommit**, donde ajusta su valor a **1** para que cada operación DML impacte automáticamente en la tabla (*sin requerir confirmar una transacción*).

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

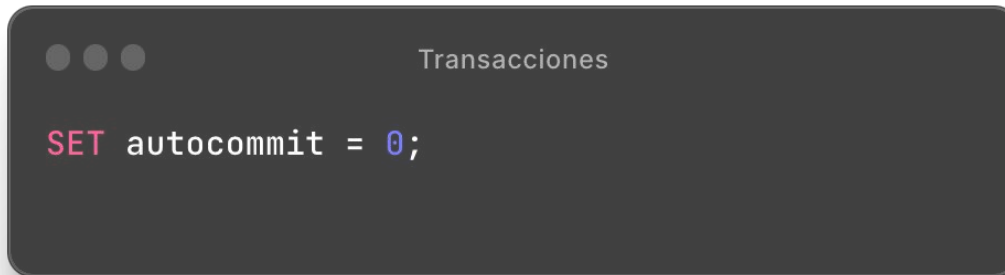
Escribamos en una ventana de script, lo siguiente:



```
Transacciones  
SELECT @@autocommit;
```

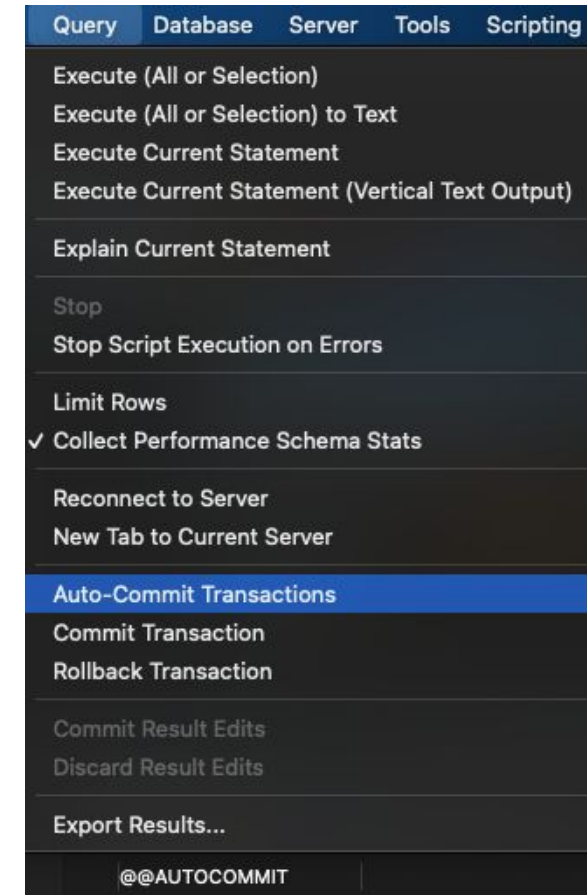
Begin Transaction

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.



Iniciar una Transacción

Iniciar una transacción

Llevemos estos ejemplos transaccionales a cada comando que vimos en la primera parte de esta clase. Comenzamos por el principal: **START TRANSACTION.**

Veamos cómo se comporta el mismo al momento de ejecutar operaciones DML en una tabla de datos.



Iniciar una transacción

Ejecutando una consulta DML del tipo **UPDATE**, iniciaremos previamente la sentencia **START TRANSACTION**.

Esto nos permitirá ver que, el registro afectado, se modificará sin problema alguno.

Elige para probar esta cláusula, cualquier tabla de cualquier bb.dd de tu motor MySQL.

```
Transacciones

START TRANSACTION;

UPDATE ventas_ecommerce.productos
SET
    nombre = 'TRACKPAD BT'
WHERE
    id = 1;
SELECT * FROM ventas_ecommerce.productos;
```


Iniciar una transacción

Luego de ejecutar esta cláusula, salimos de Mysql Workbench y volvemos a ingresar.

Volvemos a ejecutar una consulta de selección sobre esta tabla, y podremos ver que el cambio solicitado al registro de la tabla en cuestión, no se confirmó. Esto sucede porque no finalizamos la transacción, ejecutando **COMMIT** para aplicar los cambios.

```
9 • SELECT * FROM ventas_ecommerce.productos;
```

10

% 42:9

Result Grid Filter Rows: Search Edit: Export

id	nombre	existencia
1	TRACKPAD BLUETOOTH	1
2	TRACKPAD INALÁMBRICO USB	1
3	TECLADO INALÁMBRICO ES-BT	1
4	TECLADO MECÁNICO CABLEADO ES	1
5	TECLADO INALÁMBRICO ES-USB	0
6	MOUSE INALÁMBRICO BT	1
7	MOUSE CABLEADO	0
8	MOUSE INALÁMBRICO USB	1
16	UPS 10500 WATTS	1
17	MOTHERBOARD AM4 - ASROCK A320M-HDV	0
18	MOTHERBOARD AM4 - ASROCK B450M AC	0
19	MOTHERBOARD AM4 - ASROCK B550M HDV	0
20	MOTHERBOARD AM4 - GIGABYTE GA-A320M-H	0
21	MOTHERBOARD AM4 - GIGABYTE GA-A520M-H	0
22	PLACA DE VIDEO GEFORCE G 210 1GB MSI	0

Confirmar una Transacción

Confirmar una transacción

Repitamos la ejecución de la consulta de modificación usando la cláusula **UPDATE**, e iniciando previamente la sentencia **START TRANSACTION**.

Apliquemos el cambio reutilizando la cláusula anterior, si no borramos la sentencia en cuestión.

```
Transacciones

START TRANSACTION;

UPDATE ventas_ecommerce.productos
SET
    nombre = 'TRACKPAD BT'
WHERE
    id = 1;
SELECT * FROM ventas_ecommerce.productos;
```

Confirmar una transacción

Al finalizar la ejecución de la cláusula **UPDATE**, agreguemos el comando **COMMIT**, el cual nos permite validar la transacción previamente ejecutada.

Luego de ello, refrescamos la visualización de la tabla mediante una consulta **SELECT**.

```
Transacciones

START TRANSACTION;

UPDATE ventas_ecommerce.productos
SET
    nombre = 'TRACKPAD BT'
WHERE
    id = 1;
SELECT * FROM ventas_ecommerce.productos;

COMMIT;
```

Iniciar una transacción

Finalmente, veremos que el cambio ejecutado con la cláusula **UPDATE**, ha impactado de forma efectiva sobre el o los registro(s) indicados.

De esta manera, podremos tener el control necesarios sobre las operaciones riesgosas, pudiendo aplicar cláusulas DML y luego de validar las mismas, confirmar las operaciones.

```
SELECT * FROM ventas_ecommerce.productos;
```

id	nombre	existencia
1	TRACKPAD BT	1
2	TRACKPAD INALÁMBRICO USB	1
3	TECLADO INALÁMBRICO ES-BT	1
4	TECLADO MECÁNICO CABLEADO ES	1
5	TECLADO INALÁMBRICO ES-USB	0
6	MOUSE INALÁMBRICO BT	1
7	MOUSE CABLEADO	0
8	MOUSE INALÁMBRICO USB	1
16	UPS 10500 WATTS	1
17	MOTHERBOARD AM4 - ASROCK A320M-HDV	0
18	MOTHERBOARD AM4 - ASROCK B450M AC	0
19	MOTHERBOARD AM4 - ASROCK B550M HDV	0
20	MOTHERBOARD AM4 - GIGABYTE GA-A320M-H	0
21	MOTHERBOARD AM4 - GIGABYTE GA-A520M-H	0
22	PLACA DE VIDEO GEFORCE G 210 1GB MSI	0

Deshacer una Transacción

Confirmar una transacción

Ejecutemos una consulta de eliminación masiva de registros, sobre alguna tabla que permita aplicar una condición que afecte a múltiples filas de la tabla.

Iniciemos, como siempre, una transacción previo a ejecutar esta operación **DML** de eliminación.

```
Transacciones

START TRANSACTION;

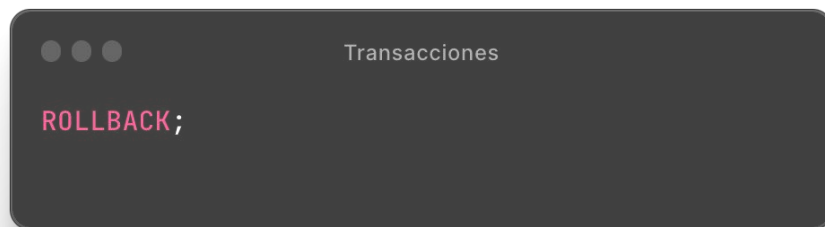
DELETE FROM
    ventas_ecommerce.productos
WHERE
    nombre = 'MOTHERBOARD%';

SELECT * FROM ventas_ecommerce.productos;
```

Confirmar una transacción

La cláusula SQL SELECT, nos mostrará que los registros en cuestión, han sido eliminados correctamente de la tabla.

Ahora, ejecutemos la sentencia rollback:



7 `SELECT * FROM ventas_ecommerce.productos;`

100% 10:8

Result Grid Filter Rows: Search Edit: Export/Import:

id	nombre	existencia	precio	precio_compra
1	TRACKPAD BLUETOOTH	1	22000	17600
2	TRACKPAD INALÁMBRICO USB	1	22000	17600
3	TECLADO INALÁMBRICO ES-BT	1	15000	10500
4	TECLADO MECÁNICO CABLEADO ES	1	7500	5200
5	TECLADO INALÁMBRICO ES-USB	0	12500	8700
6	MOUSE INALÁMBRICO BT	1	6500	4500
7	MOUSE CABLEADO	0	4100	2900
8	MOUSE INALÁMBRICO USB	1	5500	3300
16	UPS 10500 WATTS	1	15000	10500
22	PLACA DE VIDEO GEFORCE G 210 1GB MSI	0	5599	3919
23	PLACA DE VIDEO GEFORCE G 210 1GB EVGA	0	5899	4129
24	PLACA DE VIDEO GEFORCE GT 710 1GB MSI	0	6689	4682
25	MICROBOARD RPI 4TX	0	0	0
27	TRACKBALL VINTAGE GENIUS	0	950	1240
HULL	HULL	HULL	HULL	HULL

Confirmar una transacción

Con una nueva consulta de selección, posterior a **ROLLBACK**, veremos que el set de registros eliminados, volverá a su estado original.

```
7 • ROLLBACK;  
8 • SELECT * FROM ventas_ecommerce.productos;
```

Result Grid

	id	nombre	existencia	precio	precio_compra
▶	1	TRACKPAD BLUETOOTH	1	22000	17600
	2	TRACKPAD INALÁMBRICO USB	1	22000	17600
	3	TECLADO INALÁMBRICO ES-BT	1	15000	10500
	4	TECLADO MECÁNICO CABLEADO ES	1	7500	5200
	5	TECLADO INALÁMBRICO ES-USB	0	12500	8700
	6	MOUSE INALÁMBRICO BT	1	6500	4500
	7	MOUSE CABLEADO	0	4100	2900
	8	MOUSE INALÁMBRICO USB	1	5500	3300
	16	UPS 10500 WATTS	1	15000	10500
	17	MOTHERBOARD AM4 - ASROCK A320M-HDV	0	6599	4619
	18	MOTHERBOARD AM4 - ASROCK B450M AC	0	9239	6467
	19	MOTHERBOARD AM4 - ASROCK B550M HDV	0	8299	5809
	20	MOTHERBOARD AM4 - GIGABYTE GA-A320M-H	0	7499	5249
	21	MOTHERBOARD AM4 - GIGABYTE GA-A520M-H	0	8289	5802
	22	PLACA DE VIDEO GEFORCE G 210 1GB MSI	0	5599	3919

Savepoint

Savepoint

Por último, nos queda ver cómo **SAVEPOINT** nos ayudará a controlar una modificación masiva de registros, pudiendo confirmar o deshacer por lotes, según consideremos, acorde a la lógica operativa.

Debemos tener presente que, toda instrucción asociada a este comando, sólo será ejecutable en bases de datos **innoDB**.



Savepoint

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

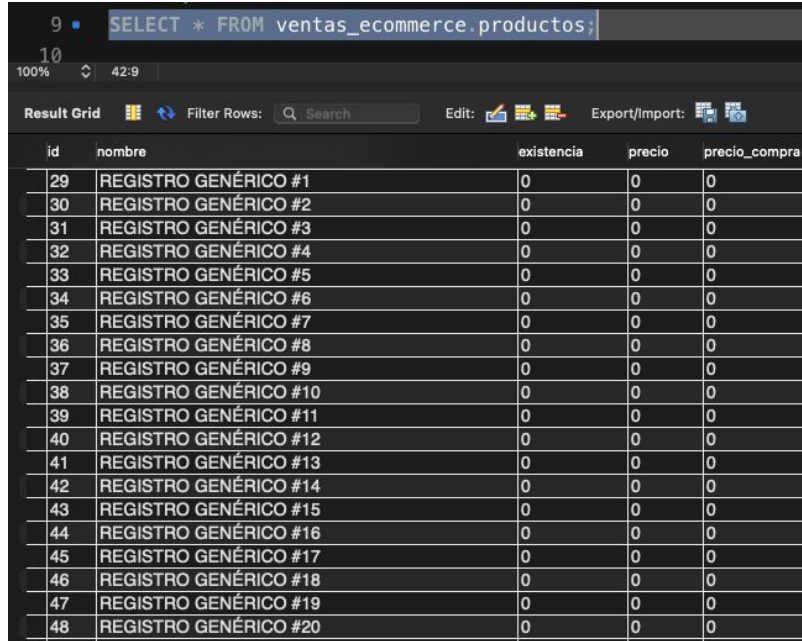
El criterio del nombre de cada es definido por nosotras, de acuerdo a algún parámetro válido o simplemente a discreción.

```
USE ventas_ecommerce;

START TRANSACTION;
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #1', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #2', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #3', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #4', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #5', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #6', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #7', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #8', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #9', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #10', 0, 0, 0);
SAVEPOINT lote_1_10;
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #11', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #12', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #13', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #14', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #15', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #16', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #17', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #18', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #19', 0, 0, 0);
INSERT INTO productos VALUES (NULL, 'REGISTRO GENÉRICO #20', 0, 0, 0);
SAVEPOINT lote_11_20;
...
```



Savepoint



9 • SELECT * FROM ventas_ecommerce.productos;

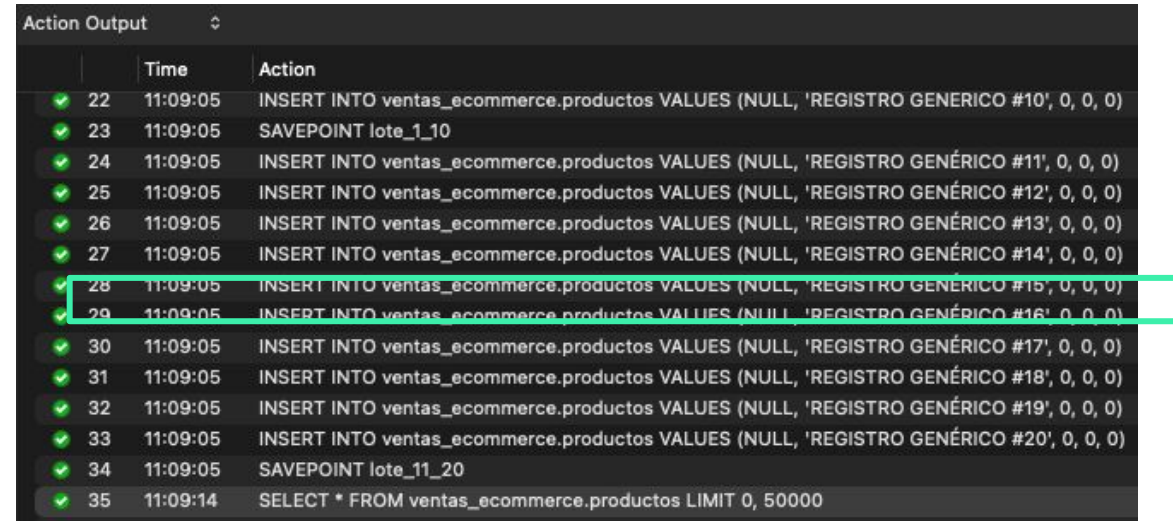
10

100% 42:9

Result Grid Filter Rows: Search Edit: Export/Import:

id	nombre	existencia	precio	precio_compra
29	REGISTRO GENÉRICO #1	0	0	0
30	REGISTRO GENÉRICO #2	0	0	0
31	REGISTRO GENÉRICO #3	0	0	0
32	REGISTRO GENÉRICO #4	0	0	0
33	REGISTRO GENÉRICO #5	0	0	0
34	REGISTRO GENÉRICO #6	0	0	0
35	REGISTRO GENÉRICO #7	0	0	0
36	REGISTRO GENÉRICO #8	0	0	0
37	REGISTRO GENÉRICO #9	0	0	0
38	REGISTRO GENÉRICO #10	0	0	0
39	REGISTRO GENÉRICO #11	0	0	0
40	REGISTRO GENÉRICO #12	0	0	0
41	REGISTRO GENÉRICO #13	0	0	0
42	REGISTRO GENÉRICO #14	0	0	0
43	REGISTRO GENÉRICO #15	0	0	0
44	REGISTRO GENÉRICO #16	0	0	0
45	REGISTRO GENÉRICO #17	0	0	0
46	REGISTRO GENÉRICO #18	0	0	0
47	REGISTRO GENÉRICO #19	0	0	0
48	REGISTRO GENÉRICO #20	0	0	0

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



	Time	Action
✓ 22	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #10', 0, 0, 0)
✓ 23	11:09:05	SAVEPOINT lote_1_10
✓ 24	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #11', 0, 0, 0)
✓ 25	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #12', 0, 0, 0)
✓ 26	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #13', 0, 0, 0)
✓ 27	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #14', 0, 0, 0)
✓ 28	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #15', 0, 0, 0)
✓ 29	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #16', 0, 0, 0)
✓ 30	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #17', 0, 0, 0)
✓ 31	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #18', 0, 0, 0)
✓ 32	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #19', 0, 0, 0)
✓ 33	11:09:05	INSERT INTO ventas_ecommerce.productos VALUES (NULL, 'REGISTRO GENÉRICO #20', 0, 0, 0)
✓ 34	11:09:05	SAVEPOINT lote_11_20
✓ 35	11:09:14	SELECT * FROM ventas_ecommerce.productos LIMIT 0, 50000

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

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:

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



Savepoint

Si ejecutamos la cláusula:

```
ROLLBACK TO lote_1_10;
```

desharemos los registros 11 al 20 insertados de forma masiva.

```
10 • ROLLBACK TO lote_1_10;
11 • SELECT * FROM ventas_ecommerce.productos;
```

Result Grid					
Filter Rows: Search Edit: Export/Import:					
id	nombre	existencia	precio	precio_compra	
7	MOUSE CABLEADO	0	4100	2900	
8	MOUSE INALÁMBRICO USB	1	5500	3300	
16	UPS 10500 WATTS	1	15000	10500	
17	MOTHERBOARD AM4 - ASROCK A320M-HDV	0	6599	4619	
18	MOTHERBOARD AM4 - ASROCK B450M AC	0	9239	6467	
19	MOTHERBOARD AM4 - ASROCK B550M HDV	0	8299	5809	
20	MOTHERBOARD AM4 - GIGABYTE GA-A320M-H	0	7499	5249	
21	MOTHERBOARD AM4 - GIGABYTE GA-A520M-H	0	8289	5802	
22	PLACA DE VIDEO GEFORCE G 210 1GB MSI	0	5599	3919	
23	PLACA DE VIDEO GEFORCE G 210 1GB EVGA	0	5899	4129	
24	PLACA DE VIDEO GEFORCE GT 710 1GB MSI	0	6689	4682	
25	MICROBOARD RPI 4TX	0	0	0	
27	TRACKBALL VINTAGE GENIUS	0	950	1240	
28	TRACKBALL VINTAGE MICROSOFT	0	0	0	
29	REGISTRO GENÉRICO #1	0	0	0	
30	REGISTRO GENÉRICO #2	0	0	0	
31	REGISTRO GENÉRICO #3	0	0	0	
32	REGISTRO GENÉRICO #4	0	0	0	
33	REGISTRO GENÉRICO #5	0	0	0	
34	REGISTRO GENÉRICO #6	0	0	0	
35	REGISTRO GENÉRICO #7	0	0	0	
36	REGISTRO GENÉRICO #8	0	0	0	
37	REGISTRO GENÉRICO #9	0	0	0	
38	REGISTRO GENÉRICO #10	0	0	0	
NULL	NULL	NULL	NULL	NULL	

Savepoint

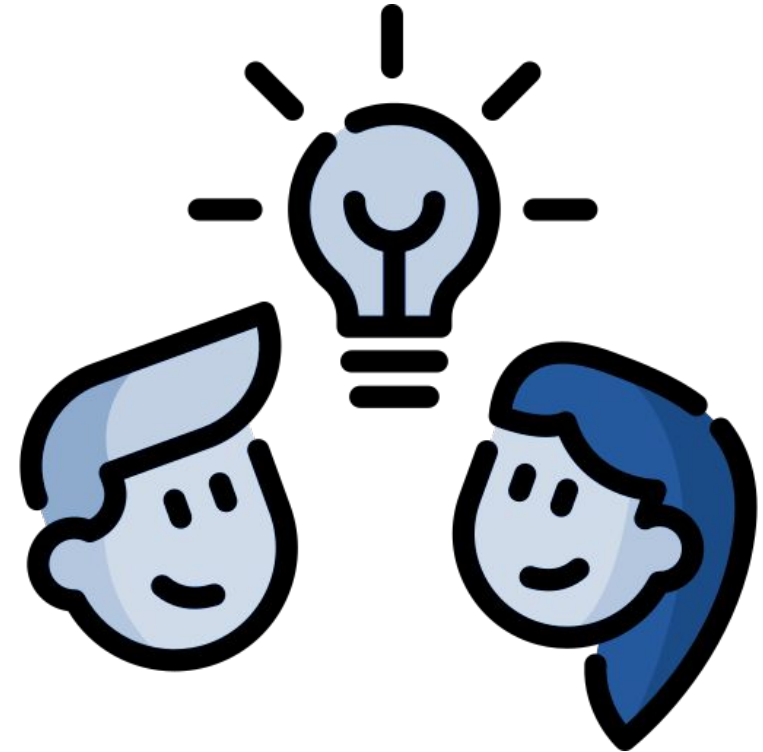
También, cuando el caso que lo amerite, podemos eliminar un **SAVEPOINT** ejecutando la cláusula:

```
Transacciones  
  
RELEASE <savepoint>;  
  
-- por ejemplo: RELEASE lote_1_10;
```


Savepoint

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

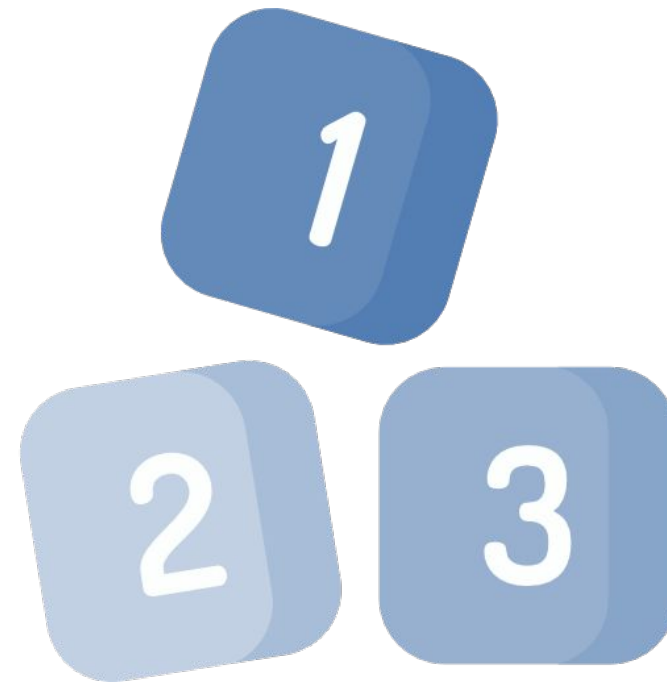
Combinando el mismo con el uso de variables, y la ejecución de cláusulas DML que dependan unas de otras, el control de transacciones ayudará a mantener consistente las tablas de la base de datos.



Vistas SQL

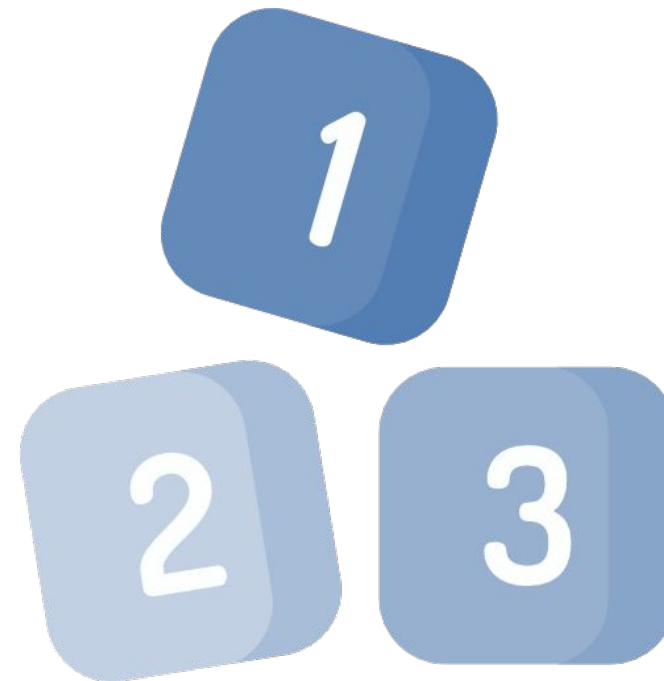
Vistas SQL

Desde el momento en el cual tengamos que repetir una consulta de selección de manera frecuente, la forma más fácil de simplificar esta tarea es creando una Vista SQL.



Vistas SQL

Las vistas almacenan la estructura de la consulta dentro del objeto **VIEWS** de nuestra base de datos, y desde allí podremos comenzar a ejecutarla de forma frecuente, sin tener que elaborar la consulta en cuestión cada vez que necesitemos acceder a la misma información.



Vistas SQL

Su estructura basada en **DDL** es muy simple. Cuando definimos el nombre de la Vista, es este mismo con el cual se guardará la misma en el apartado **VIEWS** de la base de datos en uso.

```
SQL VIEW

CREATE VIEW vista_nombre_de_consulta AS (
    -- CONSULTA DE SELECCIÓN DE DATOS
);
```

Vistas SQL

Veamos a continuación un ejemplo de implementación de Vista SQL, sobre la tabla **Orders** combinando en la misma la cláusula **WHEN THEN ELSE**, de acuerdo a diferentes estados de los registros aquí almacenados.

```
CREATE VIEW `new_view` AS
SELECT
  orders.OrderID AS OrderID,
  CAST(orders.OrderDate AS DATE) AS orderDate,
  CAST(orders.ShippedDate AS DATE) AS shippeDate,
  (CASE
    WHEN (orders.ShippedDate IS NULL) THEN 'Pendiente'
    WHEN (orders.ShippedDate > orders.OrderDate) THEN 'Entregado'
    ELSE 'En tránsito'
  END) AS OrderStatus
FROM
  orders;
```

Vistas SQL

La cláusula **CREATE VIEW** nos permite definir a continuación, el nombre que queremos darle a la vista. Luego, el comando **AS** nos permite referenciar la consulta de selección correspondiente para que ya quede almacenada como vista SQL.

```
Vistas SQL

CREATE VIEW `estadooderdenesdecompra` AS
SELECT
  orders.OrderID AS OrderID,
  CAST(orders.OrderDate AS DATE) AS orderDate,
  CAST(orders.ShippedDate AS DATE) AS shippeDate,
  (CASE
    WHEN (orders.ShippedDate IS NULL) THEN 'Pendiente'
    WHEN (orders.ShippedDate > orders.OrderDate) THEN 'Entregado'
    ELSE 'En tránsito'
  END) AS OrderStatus
FROM
  orders;
```

Vistas SQL

Una vez creada la vista, podremos acceder a la misma tal como si fuese una tabla más en la bb.dd, con la diferencia que su información será de sólo lectura.



The screenshot shows a SQL client window with a toolbar at the top. The query editor contains the following SQL statement:


```
1 • SELECT * FROM Northwind.estadoordenescompra;
```

Below the query editor, the results are displayed in a table. The table has five columns: OrderID, orderDate, shippeDate, OrderStatus, and an empty column. The data shows five rows of order information, all with a status of 'Entregado'.

	OrderID	orderDate	shippeDate	OrderStatus	
▶	10248	1996-07-04	1996-07-16	Entregado	
	10249	1996-07-05	1996-07-10	Entregado	
	10250	1996-07-08	1996-07-12	Entregado	
	10251	1996-07-08	1996-07-15	Entregado	
	10252	1996-07-09	1996-07-11	Entregado	

Vistas SQL

La misma vista, nos permitirá establecer filtros sobre los datos representados, ya que oficia de intermediaria sobre la o las tablas vinculadas dentro de ésta.



The screenshot shows a SQL query editor interface. At the top, there is a toolbar with various icons and a text box that says "Limit to 50000 rows". Below the toolbar, the SQL query is displayed in a text area:

```
2 FROM Northwind.estadoordenescompra
3 WHERE OrderID BETWEEN 10252 AND 10255
4 ORDER BY orderDate DESC;
```

Below the query, there is a horizontal scrollbar. Underneath the scrollbar, there is a status bar showing "130%" and "24:4". Below the status bar, there is a toolbar with "Result Grid", "Filter Rows:", "Search", and "Export:". Below the toolbar, there is a table with the following data:

	OrderID	orderDate	shippeDate	OrderStatus
▶	10255	1996-07-12	1996-07-15	Entregado
	10254	1996-07-11	1996-07-23	Entregado
	10253	1996-07-10	1996-07-16	Entregado
	10252	1996-07-09	1996-07-11	Entregado

Vistas SQL

A través del menú contextual del mouse, podremos acceder a la opción **ALTER VIEW**, para poder editar la consulta asignada a esta vista, y aplicarle cualquier cambio.



```
1 • CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = `root`@`localhost`
4     SQL SECURITY DEFINER
5     VIEW `estadoordenescompra` AS
6     SELECT
7         `orders`.`OrderID` AS `OrderID`,
8         CAST(`orders`.`OrderDate` AS DATE) AS `orderDate`,
9         CAST(`orders`.`ShippedDate` AS DATE) AS `shippeDate`,
10        (CASE
11            WHEN (`orders`.`ShippedDate` IS NULL) THEN 'Pendiente'
12            WHEN (`orders`.`ShippedDate` > `orders`.`OrderDate`) THEN 'Entregado'
13            ELSE 'En tránsito'
14        END) AS OrderStatus
15    FROM
16        orders
```

Espacio de Consultas

Abrimos este espacio de cara a realizar consultas, comentarios, o repasar algún tema específico, relacionado a finalizar de forma efectiva el último proyecto integrador.



Espacio de consultas

Recordemos qué debemos realizar en este:

- diseña un modelo relacional de bb.dd utilizando la información del archivo **trailerflix.json**
 - la bb.dd debe contar con un estimado de 6+ tablas relacionales, y los datos JSON migrados
- crea los endpoint necesarios para ver:
 - información de las películas y series
 - actrices/actores y sus trabajos fílmicos
 - filtrar por una película o serie específica
 - ver solo películas
 - ver solo series
 - y otros endpoint que veas viable incluir
- debes realizar la documentación acorde que explique cómo utilizar los endpoint existentes



Espacio de consultas

Finalmente, con el trabajo realizado, deberás crear un repositorio en Github, y publicar:

- la bb.dd trailerflix con sus tablas y datos cargados
- el modelo diseñado para su creación
- el código del proyecto Node
- la documentación en formato Markdown



Muchas gracias.



Ministerio de Economía
Argentina

Secretaría de
Economía del Conocimiento

*primero
la gente*