Politécnico Internacional de Occidente PIO

Curso Metodologías Ágiles para la Programación

Profesor Julio Cesar Castaño

Estudiante Laura Barona Saavedra

Grupo 6

Septiembre 2024

Comandos SQL

Comandos DDL (**Data Definition Language – Lenguaje Definición de Datos**) para creación, modificación y eliminación de la estructura y objetos de la base de datos.

CREATE – Utilizado para crear objetos (tablas, vistas, índices) en la base de datos

CREATE DATABASEcoleccion;

Al crear una tabla de base de datos debemos considerar

CREATE TABLE nombre\_tabla (

nombre\_columna tipo\_columna [ cláusula\_defecto ] [ vínculos\_de\_columna ]

[ , nombre\_columna tipo\_columna [ cláusula\_defecto ] [ vínculos\_de\_columna ] … ]

[ , [ vínculo\_de tabla] … ] )

**Nombre del campo** deben empezar por un carácter alfabético y ser lo más comprensible posible para entender que datos se almacenan en ese campo.

**Tipo del campo** (lista de los más importante):

Numérico: bigint, int, smallint, tinyint, numeric, bit, decimal, money, smallmoney.

Cadena de caracteres: char, varchar, text.

Fecha y hora: date, datetime, time, timestamp.

Cadenas binarias: binary, varbinary, image.

**Cláusula defecto** se le asigna valor por defecto al campo si no se le indica en el momento que se inserta una fila. Puede iniciarse con un valor o nulo.

DEFAULT { valor | NULL }

**Vínculos de integridad** que se aplica a cada campo

* NOT NULL no permite valor NULL
* PRIMARY KEY para indicar que es la clave primaria de la tabla. Puede formarse por más de un campo, directamente debe almacenar valor único y que no sea NULL
* FOREIGN KEY indica la clave foránea haciendo referencia a otra tabla, estableciendo la relación. Tiene las cláusulas ON DELETE y ON UPDATE indican que acción debe ejecutarse en el caso que la clave foránea (a quién hace referencia) es eliminada o borrada. Las acciones pueden ser:
  + CASCADE: elimina o modifica la tupa que tiene el campo referenciado
  + SET DEFAULT: asigna valor por defecto a la columna referenciada
  + SET NULL: asigna valor NULL a la columna referenciada
* Control de valor permite asignar o no un valor a la columna dependiendo del resultado de la condición

CHECK { expresión\_condicional }

**Vínculos de integridad que se pueden aplicar a más campos de la tabla**

* Clave primaria PRIMARY KEY ( columna1 [ , columna2 … ] )
* Clave foránea FOREIGN KEY ( columna1 [ , columna2 … ] )

CREATE TABLE consola (id\_consola bigint IDENTITY(1,1),

nombre varchar (50),

tipo varchar (15) DEFAULT ‘Sobremesa’,

marca varchar (50),

PRIMARY KEY (id\_consola)

)

CREATE TABLE juego(id\_juego bigint IDENTITY(1,1),

titulo varchar(50),

genero varchar(50),

PRIMARY KEY (id\_juego)

)

CREATE TABLE consola\_juego(id\_consola bigint,

id\_juego bigint,

PRIMARY KEY (id\_consola, id\_juego),

FOREIGN KEY (id\_consola) REFERENCES consola(id\_consola) ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY (id\_juego) REFERENCES juego(id\_juego) ON UPDATE CASCADE ON DELETE CASCADE

)

También podemos crear índices, se usan para encontrar un dato más rápidamente mejorando la eficacia de la consulta

CREATE INDEX index\_marca

ON consola (marca)

CREATE UNIQUE INDEX index\_nombre\_marca

ON consola (nombre, marca)

ALTER – Utilizado para modificar las tablas agregando o cambiando la definición de los campos

ALTER TABLE consola ALTER COLUMN marca char (SQL Server)

ALTER TABLE consola MODIFY COLUMN marca char (MySQL)

ALTER TABLE consola ADD fecha\_lanzamiento date

ALTER TABLE consola DROP COLUMN fecha\_lanzamiento

¿Qué pasaría si creamos el campo fecha con valores por defecto y después intentamos eliminar el propio campo?

ALTER TABLE consola ADD fecha\_lanzamiento DATE NOT NULL DEFAULT ‘2017-01-01’

ALTER TABLE consola DROP COLUMN fecha\_lanzamiento

Msg 5074, Level 16, State 1, Line 1

The object ‘DF\_\_consola\_\_fecha\_l\_\_276EDEB3’ is dependent on column ‘fecha\_lanzamiento’.

Msg 4922, Level 16, State 9, Line 1

ALTER TABLE DROP COLUMN fecha\_lanzamiento failed because one or more objects access this column.

DECLARE @ConstraintName nvarchar(MAX)

SELECT @ConstraintName = Name FROM SYS.DEFAULT\_CONSTRAINTS

WHERE PARENT\_OBJECT\_ID = OBJECT\_ID(‘consola’)

AND PARENT\_COLUMN\_ID = (SELECT column\_id FROM sys.columns

WHERE NAME = N’fecha\_lanzamiento’

AND object\_id = OBJECT\_ID(N’consola’))

IF @ConstraintName IS NOT NULL

EXEC(‘ALTER TABLE consola DROP CONSTRAINT ‘ + @ConstraintName)

ALTER TABLE consola DROP COLUMN fecha\_lanzamiento

DROP – Utilizado para eliminar objetos en la base de datos

Podemos eliminar un índice

DROP INDEX consola.index\_nombre\_marca (SQL Server)

ALTER TABLE consola DROP INDEX consola.index\_nombre\_marca (MySql)

O la tabla entera

DROP TABLE consola

**TRUNCATE** – Borra todo el contenido de una tabla

TRUNCATE TABLE consola

Msg 4712, Level 16, State 1, Line 1

Cannot truncate table ‘consola’ because it is being referenced by a FOREIGN KEY constraint.

En este caso no nos lo permitirá al haber una clave foránea que hace referencia a la tabla consola

**COMMENT** – Agregar comentarios al diccionario de datos sobre un objeto de la base de datos

COMMENT ON COLUMN consola.marca

IS ‘desarrolladora del producto’; (ORACLE)

COMMENT ON COLUMN consola.marca (ORACLE)

IS ‘ ‘;

**RENAME** – Renombrar la tabla de la base de datos

ALTER TABLE consola RENAME TO consola1 (ORACLE y MySQL)

sp\_rename ‘consola’,’consola1′; (SQL SERVER)

Caution: Changing any part of an object name could break scripts and stored procedures.

**Si modificamos el nombre, los script podrían dejar de funcionar. Hay que tener en cuenta también las relaciones que hay en la tabla, en Oracle por ejemplo si una tabla tiene una clave foránea, generará error.**

Comandos DML (**Data Manipulation Language – Lenguaje Manipulación de Datos**) para recuperar y trabajar con datos.

**SELECT** – Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado

SELECT [ ALL | DISTINCT ] lista\_elementos\_selección

FROM lista\_referencias\_tabla

[ WHERE expresión\_condicional ]

[ GROUP BY lista\_columnas ]

[ HAVING expresión\_condicional ]

[ ORDER BY lista\_columnas ]

Se puede buscar todos los valores de un campo o varios campos de una tabla, o con DISTINCT eliminar aquellas filas cuyo campo se repite. Si no se indica ALL | DISTINCT por defecto se realizará ALL. Nunca se debe usar DISTINCT en la clave primaria, ni en las candidatas ya que de por sí son únicas

WHERE se utiliza para generar una condición que debe cumplirse, aquellos campos que no la cumplan no se seleccionarán.

GROUP BY para agrupar en una sola línea el campo o campos especificados.

HAVING del resultado de GROUP BY elimina las que no cumplan su expresión condicional.

ORDER BY para ordenar el orden de las columnas de formas ascendente (ASC) o descendente (DESC). Por defecto el orden se hace de forma ascendente.

Además de la sintaxis, podemos encontrarnos con consultas que utilicen las siguientes funciones de agregado: COUNT, SUM, AVG, MAX, MIN.

Operadores lógicos: AND, OR, NOT.

Operadores de comparación: >, <, , =, =, !=, !, BETWEEN para intervalo de valores, LIKE para comparación entre cadenas de caracteres con los pattern ‘%’ y ‘\_’.

IN para especificar registros de una base de datos.

Para ver un ejemplo, vamos a ver las tablas de nuestra base de datos agregando campo STOCK a la tabla CONSOLA\_JUEGO, para saber cuántas unidades hay de cada juego y por consola.

Tabla

Descripción generada automáticamente

Consulta: Mostrar los tipos de consolas que hay registrados

SELECT DISTINCT tipo

FROM consola

Consulta: Nombre de los juegos que sólo están en 3DS

SELECT titulo

FROM juego

WHERE juego.id\_juego IN(

SELECT id\_juego

FROM consola\_juego

WHERE consola\_juego.id\_consola IN(

SELECT id\_consola

FROM consola

WHERE consola.nombre=’3DS’)

)

Consulta: Stock total de aquellos juegos cuyo género sea de Acción

SELECT SUM(consola\_juego.stock)

FROM consola\_juego

WHERE consola\_juego.id\_juego IN(

SELECT id\_juego

FROM juego

WHERE juego.genero=’accion’)

Consulta: Titulo de aquellos juegos con stock total que supere las 20 unidades

SELECT juego.titulo

FROM juego

WHERE juego.id\_juego IN(

SELECT id\_juego

FROM consola\_juego

GROUP BY id\_juego

HAVING SUM(consola\_juego.stock) > 20)

**INSERT** – Utilizado para cargar lotes de datos en la base de datos en una única operación

INSERT INTO table\_name

VALUES (value1,value2,value3,…);

INSERT INTO consola(nombre, marca) VALUES(‘PS4′,’Sony’);

INSERT INTO consola(nombre, marca) VALUES(‘XBOX ONE’,’Microsoft’);

INSERT INTO consola(nombre, tipo, marca) VALUES(‘3DS’,’Portatil’,’Nintendo’);

INSERT INTO juego(titulo, genero) VALUES(‘Dark Souls III’, ‘ROL’);

INSERT INTO juego (titulo, genero) VALUES(‘Project Cars’, ‘Conduccion’);

INSERT INTO juego(titulo, genero) VALUES(‘Metal Gear Solid V’, ‘Accion’);

INSERT INTO juego (titulo, genero) VALUES(‘Killer Instinct’, ‘Lucha’);

INSERT INTO juego(titulo, genero) VALUES(‘Metroid Prime’, ‘Accion’);

INSERT INTO juego(titulo, genero) VALUES(‘Dragon Quest VII’, ‘ROL’);

INSERT INTO consola\_juego (id\_consola, id\_juego, stock) VALUES(1,1, 45);

INSERT INTO consola\_juego (id\_consola, id\_juego, stock) VALUES(1,2, 30);

INSERT INTO consola\_juego (id\_consola, id\_juego, stock) VALUES(1,3, 15);

INSERT INTO consola\_juego (id\_consola, id\_juego, stock) VALUES(2,3, 15);

INSERT INTO consola\_juego (id\_consola, id\_juego, stock) VALUES(2,4, 19);

INSERT INTO consola\_juego (id\_consola, id\_juego, stock) VALUES(3,5, 34);

INSERT INTO consola\_juego (id\_consola, id\_juego, stock) VALUES(3,6, 23);

**UPDATE** – Utilizado para modificar los valores de los campos y registros especificados

UPDATE table\_name

SET column1=value1,column2=value2,…

WHERE some\_column=some\_value;

UPDATE juego

SET titulo=’The Legend of Zelda’

WHERE titulo=’Metroid Prime’

**DELETE** – Utilizado para eliminar registros de una tabla de una base de datos

DELETE FROM juego WHERE id\_juego=5

**MERGE** – Cuando se quiere actualizar/insertar datos de una tabla, actualiza cuando se cumple la condición e inserta cuando no se cumple

MERGE INTO [tabla\_destino]

USING ([tabla o vista o consulta])

ON ([condición de existencia de registro])

WHEN MATCHED THEN [sentencia de actualización]

WHEN NOT MATCHED THEN [sentencia de inserción];

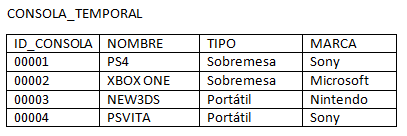
Para este ejemplo creamos la tabla consola\_temporal con los siguientes valores.

INSERT INTO consola\_temporal(nombre, marca) VALUES(‘PS4′,’Sony’);

INSERT INTO consola\_temporal(nombre, marca) VALUES(‘XBOX ONE’,’Microsoft’);

INSERT INTO consola\_temporal(nombre, tipo, marca) VALUES(‘NEW 3DS’,’Portatil’,’Nintendo’);

INSERT INTO consola\_temporal(nombre, tipo, marca) VALUES(‘PSVITA’,’Portatil’,’Sony’);



MERGE INTO consola C

USING (SELECT \* FROM consola\_temporal) CT

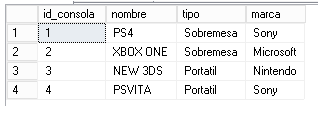
ON(C.id\_consola=CT.id\_consola)

WHEN MATCHED THEN UPDATE SET C.nombre=CT.Nombre

WHEN NOT MATCHED THEN insert (nombre, tipo, marca)

VALUES (CT.nombre, CT.tipo,CT.marca);

Si comprobamos los valores de la tabla consola, quedarán de la siguiente forma.



**CALL** – Para ejecutar un procedimiento en PL/SQL o Java

Para el desarrollo de procedimientos se realiza con la siguiente sintaxis.

En MySQL

CREATE PROCEDURE nombre ([parámetro1,parámetro2,…])

[Atributos de la rutina]

BEGIN instrucciones

END

DELIMITER //

CREATE PROCEDURE contador\_hasta\_n (IN n INT)

BEGIN

DECLARE contador INT DEFAULT 1;

WHILE contador<=n DO

SELECT contador;

SET contador = contador + 1 ;

END WHILE;

END//

DELIMITER //

CREATE PROCEDURE mostrar\_juegos()

BEGIN

SELECT \* FROM juego;

END//

DELIMITER ;

Para ejecutarlo usamos el comando CALL

CALL contador\_hasta\_n(5)

En SQL SERVER

USE coleccion

GO

CREATE PROCEDURE mostrar\_juegos

AS

SELECT \* FROM juego;

GO

exec mostrar\_juegos

**Podemos borrar el proceso para modificarlo si fuese necesario. Si se intenta crear el proceso con nombre ya existente, el RDBMS no lo permitirá.**

DROP PROCEDURE [IF EXISTS] nombre\_procedimiento

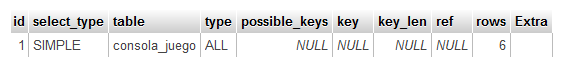
DROP PROCEDURE IF EXISTS contador\_hasta\_n

DROP PROCEDURE mostrar\_juegos

**EXPLAIN PLAN** – Nos permite analizar el plan de ejecución para poder optimizar la ejecución de las consultas

EXPLAIN SELECT \* FROM consola\_juego

Como resultado obtenemos lo siguiente



Básicamente nos indica que no existen índices para la consulta y como resultado tendremos 6 filas

Realizamos otra comprobación

EXPLAIN SELECT juego.titulo

FROM juego

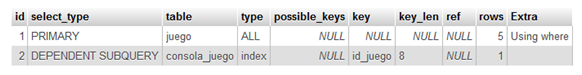
WHERE juego.id\_juego IN(

SELECT id\_juego

FROM consola\_juego

GROUP BY id\_juego

HAVING SUM(consola\_juego.stock) > 20)



Nos muestra que se realiza una consulta primaria y depende del resultado de la subconsulta, utilizando id\_juego como índex para filtrar los resultados.

**LOCK TABLE** – concurrencia de control, para controlar el flujo actual

LOCK TABLES

tbl\_name [AS alias] {READ [LOCAL] | [LOW\_PRIORITY] WRITE}

[, tbl\_name [AS alias] {READ [LOCAL] | [LOW\_PRIORITY] WRITE}] …

UNLOCK TABLES

LOCK TABLES consola READ;

INSERT INTO consola(nombre, tipo, marca) VALUES(‘NEW 3DS’,’Portatil’,’Nintendo’);

Nos permite bloquear la tabla consola y en modo lectura, si se intenta insertar sobre ella no lo permitirá indicando que la tabla está bloqueada y no se puede actualizar. De no usarse o no poder usar transacciones, es recomendable LOCK TABLE.

Para desbloquear ejecutamos.

UNLOCK TABLES;

LOCK TABLES consola READ;

SELECT \* FROM consola;

INSERT INTO consola(nombre, tipo, marca) VALUES(‘NEW 3DS’,’Portatil’,’Nintendo’);

UNLOCK TABLES; (MySQL En modo lectura no ponemos UNLOCK TABLES; porque sino entonces nos mostrar error pero nos mostrará mensaje de 0 columnas devueltas. Por defecto, la tabla después se libera)

LOCK TABLES consola WRITE;

SELECT \* FROM consola;

INSERT INTO consola(nombre, tipo, marca) VALUES(‘NEW 3DS’,’Portatil’,’Nintendo’);

UNLOCK TABLES;

En este caso, nos mostrará mensaje de 0 columnas devueltas y al mismo tiempo inserta los valores en la tabla.

Antes de continuar con los comandos, es necesario explicar las vistas en la base de datos. La diferencia entre una tabla y una vista, es que las tablas almacenan datos mientras que las vistas no. Si probamos ejecutar la siguiente consulta:

CREATE **VIEW** «NOMBRE\_VISTA» AS «Instrucción SQL»;

CREATE VIEW v\_ejemplo AS

INSERT INTO consola(nombre, tipo, marca) VALUES(‘NEW2 3DS’,’Portatil’,’Nintendo’);

Nos dará error de sintaxis. Eso es porque las vistas no están para consultas de inserción o modificación

Si probamos esta otra:

CREATE VIEW v\_ejemplo AS

SELECT \* FROM consola;

La dará por válida y se nos guardará la vista. En el momento que queramos consultarla, ejecutará la sentencia SQL.

Si ejecutamos:

INSERT INTO v\_ejemplo(nombre, tipo, marca) VALUES(‘NEW2 3DS’,’Portatil’,’Nintendo’);

Como la vista v\_ejemplo tiene relación que la tabla consola, se insertará en la tabla consola, de forma indirecta estamos aplicando cambios sobre la tabla

La gran utilidad que tienen las vistas es para guardar consultas que queramos ejecutar en cualquier momento, sin tener que estar escribiéndola nuevamente en la consola del **RDBMS**.

Comandos DCL (**Data Control Language -Lenguaje Control de Datos**) para controlar el acceso a los datos.

**GRANT** – Permite asignar permisos sobre el objeto de la base de datos. El usuario quien crea el objeto es el propietario y por defecto tiene todos los permisos, incluido el de dar permisos a otro usuario denominado permiso de concesión. El usuario propietario puede asignar los permisos que desee a otro usuario incluido el de concesión, si este usuario tiene el permiso de concesión puede asignar permisos a otro usuario.

GRANT lista\_privilegios ON objeto TO lista\_usuarios [ WITH GRANT OPTION ]

Permisos que se pueden asignar:

USAGE: para usar un objeto específico de la base de datos.

SELECT: para acceder a tablas o vistas.

INSERT [ (nombre\_columna) ]: Si se especifica el nombre\_columna, se otorga permiso para insertar en la columna especificada. Si se omite entonces se permite insertar valores en todas las columnas.

UPDATE [ (nombre\_columna) ]: Lo mismo que INSERT para modificar.

DELETE: para eliminar registros de una tabla o vista.

REFERENCES [ (nombre\_columna) ]: Si se especifica nombre\_columna permite referirse a la columna indicada como vínculo de integridad, si se omite se aplica a todas las comunas y también a las que se añadan después.

GRANT OPTION: permite dar permisos a otro usuario.

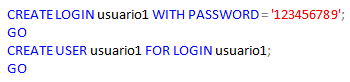
ALL: otorga todos los permisos menos GRANT OPTION.

El objeto suele ser la tabla o vista.

La lista de usuarios son usuarios o grupos que reconoce el sistema. Puede usarse PUBLIC para referirse a todos los usuarios y grupos del sistema. Para un grupo especifico usaremos GROUP

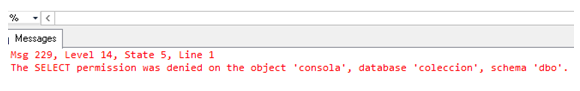
Necesitaremos primero un usuario para aplicar el ejemplo, lo creamos siguiendo la siguiente sintaxis.

**CREACIÓN DE USUARIOS. VINCULACIÓN USUARIO-LOGIN**



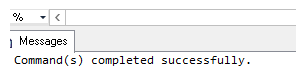
Si ahora intentamos hacer un **Select** a la tabla consola perteneciente a la base de datos colección, nos mostrará mensaje de permiso denegado.



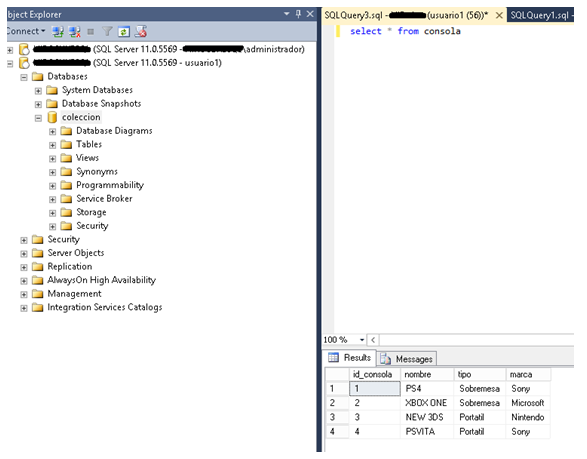


Los usuarios que crean una base datos al ser propietarios tienen asignados por defecto todos los permisos, para los demás usuario se les puede asignar o revocar permisos.





En este caso el usuario propietario le asignará el permiso SELECT mediante la sentencia GRANT. Esto significa que solo podrá ver datos de la tabla, pero no insertar o borrar.



Ejemplo: Asignamos permisos SELECT para mostrar registros de la tabla y también INSERT sólo en los campos nombre y marca de la tabla consola al usuario usergamer

GRANT SELECT, INSERT(nombre, marca) ON consola TO usergamer

**REVOKE** – Para quitar permisos.

REVOKE [ GRANT OPTION FOR ] lista\_privilegios ON objeto FROM lista\_usuarios { RESTRIC | CASCADE }

GRANT OPTION FOR: para quitar permiso de concesión.

RESTRIC se usa cuando existe en un solo objeto, esta podría fallar en el caso que el usuario al que se quiere quitar los permisos haya asignado permisos a otros usuarios. Para ello se usa CASCADE que permitirá los permisos al usuario y también a aquellos usuarios que les haya concedido permisos. También se destruirán los objetos que hayan sido creados por esos permisos.

Denegamos el permiso insertar en la tabla consola al usuario usuario1

REVOKE INSERT ON consola TO usuario1

Comandos TCL (**Transaction Control Language -Lenguaje Control de Transacciones**) para las transacciones de datos.

**COMMIT** – Finaliza la transacción y realiza los cambios hechos durante la transacción. Las transacción bloqueadas sobre la tabla quedan liberadas.

commit

**ROLLBACK** – Rechaza la transacción y no aplica cambios, volviendo al estado antes de iniciarse la transacción:

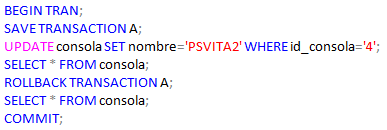
ROLLBACK [WORK] [TO SAVEPOINT]

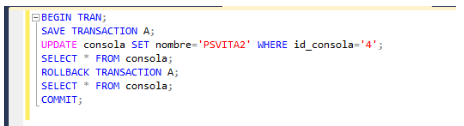
WORK: retorna al comienzo del último BEGIN TRANSACTION ejecutado

TO SAVEPOINT: al crearse un punto de guardado, se puede devolver los datos correspondientes al momento en que fue creado el punto creado.

SAVE POINT NAME: punto de rescate de una transacción que nos permite volver.

**SAVEPOINT**– Crea un punto en la transacción que se pueda volver mediante ROLLBACK.





Comenzamos una transacción salvando un punto que llamamos A, realizamos la actualización en la tabla consola y después un **ROLLBACK** para volver al momento exacto del punto de guardado. Cerramos la transacción con **COMMIT**.

**SET TRANSACTION** – Inicializa una transacción en la base de datos, indicando si quiere que sea de solo lectura o lectura/escritura.

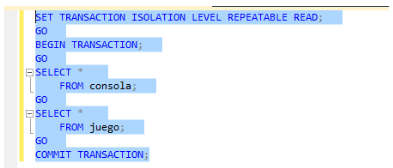
SET TRANSACTION [ READ WRITE | READ ONLY ];

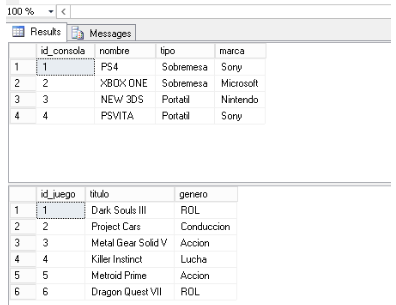
Argumentos de SET TRANSACTION en SQL SERVER:

* READ UNCOMMITTED: permite leer filas que han sido modificadas por otras transacciones sin ser confirmadas, es decir, no existe ningún bloqueo para que otras transacciones bloqueen la actual.
* READ COMMITED: no permite leer filas modificadas, pero no estar confirmadas por otras transacciones.
* REPEATABLE RED: no permite leer filas modificadas, pero no estar confirmadas por otras transacciones y ninguna otra transacción puede modificar ningún dato leído de la transacción actual hasta que finalice.
* SNAPSHOT: la transacción solo reconoce datos confirmadas antes del comienzo de la misma, las instrucciones que se ejecute en la transacción actual no se verán afectadas por otras transacciones después del inicio de la actual. Los datos estarían tal y como estaban al principio de la transacción.
* SERIALIZABLE: no permite leer datos que hayan sido modificados, pero no estar confirmados por otras transacciones; ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que finalice; y otras transacciones no pueden insertar filas nuevas con valores de clave que estén en el intervalo de claves leído por instrucciones de la transacción actual hasta que finalice.
* Se puede además aplicar SET TRANSACTION ISOLATION LEVEL, se aplica cuando al devolver el control de un objeto, se restablece en el nivel que fue invocado por el objeto.

Ejemplo con REPEATABLE RED

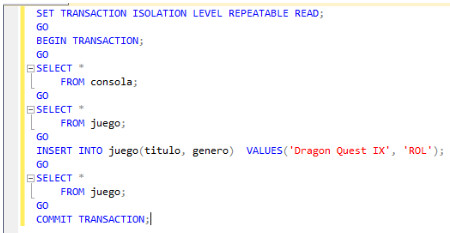
Comenzamos una transacción salvando un punto que llamamos A, realizamos la actualización en la tabla consola y después un ROLLBACK para volver al momento exacto del punto de guardado. Cerramos la transacción con COMMIT.

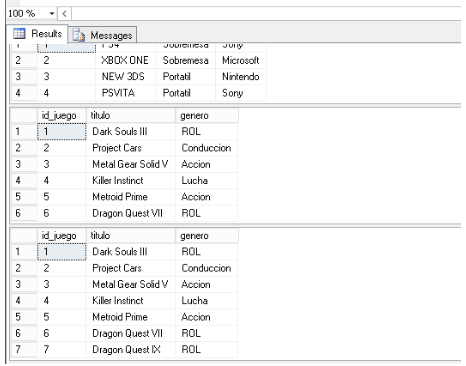




Obtenemos los registros de ambas tablas.

Probamos la misma consulta añadiendo un INSERT INTO.

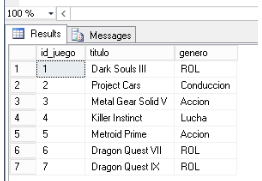




Y nos agrega la fila sin problemas.

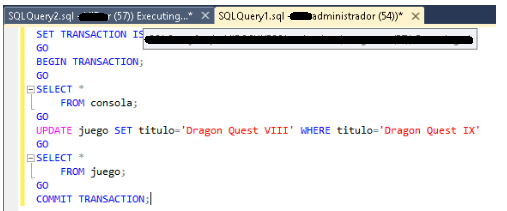
Si con otro usuario hacemos un SELECT en la tabla juego (teniendo los permisos para hacerlo)

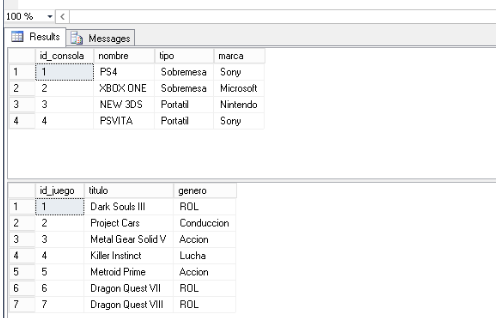




Vemos que podemos hacer la consulta sin problemas

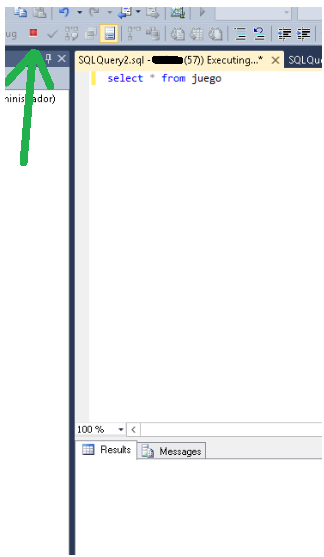
Ahora probamos con UPDATE





Y hemos modificado la fila que habíamos agregado antes.

Lo siguiente es ver si podemos consultar la tabla con el otro usuario.



Y no muestra nada, la consulta la intenta realizar pero no llega a terminar el proceso. Donde apunta la flecha verde es para cancelar la consulta, la causa de no poder ver el resultado de la consulta es porque al haber hecho un cambio y ser confirmada la transacción, se encuentra bloqueado el objeto de la base de datos, en este caso la tabla juego

Con esto terminamos los comandos SQL.