INGENIERÍA DE SISTEMAS

ASIGNATURA: BASE DE DATOS

CONTENIDISTA: Ing. Silvia Cobialca

UNIDAD 5:
TRANSACCIONES

INDICE – UNIDAD 5:

MAPA DE LA UNIDAD 5: Transacciones

PROPÓSITOS

En esta unidad nos proponemos explicarte qué es son las transacciones en el contexto de una base de datos. Veremos las definiciones y utilidad de las mismas y algunos ejemplos ilustrativos que servirán para aclarar los conceptos.

OBJETIVOS

- ✓ Entender la diferencia entre los diferentes tipos de transacciones.
- ✓ Conocer las sentencias sql que corresponden a transacciones: insert, delete, update.
- ✓ Conocer las sentencias que sirven para confirmar las transacciones: begin, commit, rollback.
- ✓ Conocer las programaciones que realiza el dbms para organizar las transacciones.

CONTENIDOS

Para que alcance los objetivos, los contenidos que abordará son los siguientes:

1) Transacciones.
2) Tipo de transacciones
3) Uso de sentencias DML para transacciones: select, insert, update, delete.
4) Utilización de transacciones.
5) Schedules y recuperación
6) Seriabilidad de schedules

PALABRAS CLAVES

Transacción, commit, rollback, select, insert, update, delete

BIBLIOGRAFÍA de consulta

Elmasri, Ramez/Navathe, Shamkant (2011). *Fundamentos de Sistemas de Base de Datos.* 6ta Ed, EEUU: Pearson / Addison Wesley.

EVALUACIÓN

A lo largo de cada unidad, encontrarás actividades de autoevaluación que te permitirán hacer un seguimiento de tu aprendizaje.

INTRODUCCIÓN

Para comenzar con esta unidad, y continuando desde la Unidad 3 donde vimos los conceptos básicos del lenguaje SQL. Aprenderemos a partir de los mismos ejemplos que trabajamos en la Unidad 3 de manera de unir todos los conocimientos adquiridos.

Comencemos…

1. TRANSACCIONES

Ya vimos todas las sentencias del lenguaje SQL que se utilizan para interactuar con la base de datos y obtener información a partir de sus datos.
Supongamos que nuestra base de datos se utiliza para gestionar las cuentas de un banco y que Juan Perez, cuyo código de CBU es el 23 tiene que hacer una transferencia de dinero a Pedro González cuyo código de CBU es 44. El monto que va a transferir es de $550.
La transferencia involucra que Juan va a retirar los $550 de su cuenta y luego va a ingresar a la cuenta de Pedro la misma suma de dinero. Si lo pensamos en operaciones DML como las que vimos en los apartados anteriores, lo podríamos realizar con dos updates, uno que incremente el saldo en 550 (en la cuenta de Pedro) y otro que lo disminuya en 550 (en la cuenta de Juan).
Pero tenés que pensar que estamos interactuando con la base de datos desde una aplicación, por ejemplo un sitio parecido a pagomiscuentas.como el sitio de uno de los bancos. Entonces Juan coloca los datos del CBU de la cuenta de Pedro y presiona el botón "Transferir".

Te preguntarás ahora: ¿Qué pasaría si se corta la conexión de internet de Juan justo cuando está realizando esta operación?

¡Este es un detalle muy importante! Debemos tener en cuenta a la hora de interactuar con los datos y las tablas el hecho de asegurarnos que las dos operaciones ocurran al mismo tiempo, y si por algún motivo una de ellas se cancela, tambien debe cancelarse la otra.

Una transacción es una unidad atómica de operación que debe ser realizada completamente o cancelada en caso de que suceda alguna falla.

En el caso del ejemplo de la transferencia bancaria es muy importante que se aplique este comportamiento ya que si falla el depósito en la cuenta de Pedro pero no la extraccción de la cuenta de Juan, ¿Dónde quedarían los $550 de Juan?

Entonces consideraremos esta operación como si fuera una transacción, es decir si falla una de ellas, se cancela todo y los saldos de ambos serían los mismos que antes de comenzar la transferencia.

Propiedades ACID de las transacciones

Las transacciones tienen que cumplir cuatro propiedades:

- **A**tomicidad: se realizan todas las operaciones involucradas con éxito o no se realiza ninguna.
- **C**onsistencia: la base de datos pasa de un estado consistente a otro.
- **A**islamiento: cada transacción es aislada, es decir que mientras está ocurriendo ninguna otra transacción puede acceder a los datos que estan siendo modificados por la misma.
- **D**urabilidad: los cambios ocurridos en la base de datos por causa de la transacción son persistentes, es decir que no se pueden perder por causa de una falla.

Por este motivo se dice que las transacciones tienen propiedades ACID (por sus siglas en inglés). En el lenguaje SQL, para marcar el comienzo de una transacción, se utiliza según el DBMS con el que estemos interactuando, una sentencia para marcarel inicio y otra para marcar el fin de la transacción. Por ejemplo para MySQL se utilizan `START TRANSACTION` y `COMMIT` y para SQL SERVER se utilizan `BEGIN TRAN` y `COMMIT` respectivamente. Si en lugar de cerrar la operación deseamos cancelar los cambios tenemos que ejecutar la sentencia `ROLLBACK`.

El DBMS implementa mecanismos que refuerzan el comportamiento de las transacciones y aseguran que ante una falla los estados de todas las tablas involucradas vuelvan a ser los mismos que antes de que comenzara la transacción. Estos son los llamados **mecanismos de recuperación**, ellos utilizan puntos de control o marcas de tiempo que les permiten poder volver a esos estados anteriores al comienzo de la transacción.

ACTIVIDAD 1.

Le pedimos que a continuación realice la actividad 6 en el campus

2. SENTENCIAS DML PARA MODIFICAR DATOS: INSERT, UPDATE, DELETE.

Como habrá observado, hasta ahora estuvimos siempre trabajando con tablas asumiendo que tenían datos.

Se preguntará entonces, ¿Qué pasa cuando necesitamos poner una T-upla o más aún, modificar un atributo de la misma?

Se habrá imaginado que existen sentencias DML especiales para realizar estas actividades dentro del lenguaje SQL. En efecto, ellas son:
Para agregar T-upla de datos: `INSERT`
Para modificar atributos de una o varias T-uplas: `UPDATE`
Para borrar T-uplas completas de una tabla: `DELETE`
Cada vez que usamos una de estas sentencias, lo que ocurre es una transacción.

INSERT

La sintaxis más simple del INSERT es así:
`INSERT INTO` tabla (<lista de atributos>)
`VALUES` (lista de valores)
La lista de atributos es opcional, pero si no se pone entonces el DBMS espera una lista de valores coherente con todos los atributos de la tabla. Veamos un ejemplo para entender bien la sintaxis.
Supongamos nuestra tabla de Productos del diagrama E-R que veníamos utilizando en los apartados anteriores:
Productos: {Código, Precio, Descripción}

Para insertar un nuevo producto utilizando la sintaxis indicada haríamos así:

Si ponemos todos los valores de cada uno de los atributos al momento de insertar no necesitamos poner la lista, es decir las tres sentencias siguientes son equivalentes:

```
INSERT INTO productos (codigo, precio, descripcion)
VALUES (26, 14,'Fecula de maíz')

INSERT INTO productos (precio, codigo, descripcion)
VALUES (14, 26, 'Fecula de maíz')
```

O por ejemplo si hubiéramos hecho:

```
INSERT INTO productos
VALUES (14, 26, 'Fecula de maíz')
```

Habría tomado 26 como el precio, ya que está en el lugar que corresponde a ese atributo en la definición de la tabla.

O si usábamos:

```
INSERT INTO productos (precio, codigo, descripcion)
VALUES (26, 14, 'Fecula de maíz')
```

Habría tomado 26 como el precio y no como el código, ya que está en el lugar del precio de acuerdo a la lista de atributos utilizada en la sentencia.

Es decir, si se coloca la lista de atributos, la lista de valores sigue esa secuencia, si no se coloca, el DBMS asume que la lista de valores está ordenada de acuerdo a la definición de la tabla.

UPDATE
Ahora veamos la sentencia `UPDATE` en su forma más simple:

```
UPDATE tabla
SET campo1=valor1,
Campo2= valor2,
….
campoN= valorN
WHERE  <lista de condiciones>
```

CUIDADO: ¡Si no hay cláusula `WHERE` lo que ocurrirá es que se actualizarán todas las T-uplas de la tabla!
Esto es un error muy común que cometen algunos usuarios avanzados, ya hablaremos sobre ese tema en la unidad 5.

Veamos algunos ejemplos de aplicación:

Si queremos modificar el precio del producto de código 14 que acabamos de insertar en el apartado anterior, quizás en lugar de 26 queremos que sea $28 podemos hacer lo siguiente:

```
UPDATE productos
SET precio= 28
WHERE  codigo=14;
```

O quizás queríamos incrementar su precio en un 15%, en cuyo caso haríamos así:

```
UPDATE productos
SET precio= precio * 1.15
WHERE  codigo=14;
```

Si lo queríamos disminuir en un 15%, hubiéramos hecho así:

```
UPDATE productos
SET precio= precio * 0.85
WHERE  codigo=14;
```

PERO, si lo que necesitábamos hacer era incrementar los precios de TODOS los productos en un 10%, la sentencia no utiliza la cláusula `WHERE` como habíamos indicado anteriormente.

```
UPDATE productos
SET precio= precio * 1.1;
```

DELETE
Por último, para eliminar registros de una tabla utilizamos ésta sentencia. Su forma más simple es:

```
DELETE FROM tabla
WHERE  <lista de condiciones>;
```

Acá hacemos la misma salvedad que en el caso del UPDATE, si no se usa la cláusula WHERE lo que ocurrirá es que se eliminarán TODAS las T-uplas de la tabla y la misma quedará vacía, lo que no es la intención habitual.
Usemos un ejemplo para ilustrar cómo borraríamos el producto que acabamos de ingresar cuando vimos la sentencia INSERT:

```
DELETE FROM productos
WHERE codigo=14;
```

Y si hacemos lo siguiente:

```
DELETE FROM productos;
```

¡Nos quedamos sin productos!

Todos los ejemplos que hemos visto son de transacciones IMPLICÍTAS.

ACTIVIDAD 2

Realizar la actividad 2 en el campus

3. UTILIZACIÓN DE TRANSACCIONES.

Como vimos en los apartados anteriores, tenemos sentencias especificas en el lenguaje SQL, para iniciar y finalizar una transacción. Si no las usamos, no quiere decir que no ocurran las transacciones, sino que las mismas serán implícitas.
Por ejemplo, cuando usamos el insert o el update en el apartado anterior, la transaccion comenzaba y terminaba con dicha sentencia, no existia una marca explícita de su inicio o fin.
Pero dichas marcas son útiles cuando queremos "encerrar" dentro de una transacción varias sentencias. Si una de las sentencias fallara, toda la transacción fallaría y no se ejecutaría ninguna de ellas.
Por ejemplo, volviendo al ejemplo de la transferencia de Juan a Pedro, se haría de esta manera: extraemos primero el monto de la cuenta de Juan y luego lo colocamos en la cuenta de Pedro. Esta operación consiste en realidad de dos updates. Imagina que pasaría si mientras está ocurriendo la primera parte de la misma hay un problema con la red eléctrica del banco. Eso provocaría que el dinero de Juan nunca llegaría a la cuenta de Pedro, si bien fue extraído ya de su cuenta.

Para resolver y prevenir ese tipo de inconvenientes, se colocan las dos operaciones juntas dentro de una misma transacción, como se muestra a continuación:

```
START TRANSACTION
UPDATE movimientos
SET monto=monto – 550
```

```
WHERE  CBU= (select CBU from cuentas where ID=23)

UPDATE movimientos
SET monto=monto +550
WHERE CBU= (SELECT CBU FROM cuentas WHERE ID=44)
COMMIT
```

Ahora si una de ellas falla o hay algun problema técnico se cancelará toda la transacción.

Sería buena idea que veas el video titulado "Transacciones" donde se muestran algunos ejemplos muy interesantes de cómo se trabaja con transacciones en el entorno MySQL Workbench.

ACTIVIDAD 3

Realizar la actividad 3 en el campus

4. SCHEDULES Y RECUPERACIÓN.

En el día a día de una base de datos, las transacciones se ejecutan intercaladas entre sí aunque no estén relacionadas. El orden de ejecución de las operaciones que tienen que ver con todas esas transacciones que se están ejecutando se denomina schedule o historial. Es importante estudiarlos ya que cuando algo falla durante la ejecución de un schedule necesitamos volver atrás y recuperar los estados anteriores de cada una de las instancias de la base de datos involucradas en las transacciones subyacentes.
Sea S un schedule o historial de n transacciones S(T1, T2, …, Tn), es un orden específico de la operación de dichas transacciones.

Como dijimos anteriormente, las transacciones pueden estar intercaladas en el schedule, pero, para una transacción determinada Ti, el orden de las operaciones que la conforman debe intercalarse en el mismo orden en que se ejecutarían si se tratara únicamente de la ejecución de Ti. El orden de las operaciones de cada transacción en S se dice que es TOTAL si todas las operaciones de cada transacción se ejecutan juntas antes de que comience la siguiente transacción.

De todas maneras, a los fines de lo que sucede o debe suceder en el schedule para que sea recuperable, solo nos interesa de cada transacción la operación de lectura *ri* y la de escritura *wi*. Así, cuando denotemos que una cierta transacción lee el item X, lo indecaremos con *ri(X),* y de forma similar para cuando se escriba X en esa misma transacción lo denotaremos con *wi(X).*

Entonces podremos escribir el siguiente schedule:
S: (r1(X); r2(X); w1(X); r1(Y); w2(X);w1(Y))

Operaciones conflictivas en un schedule

Se dice que dos operaciones de un schedule entran en conflicto si se satisfacen las siguientes condiciones:

1) Las operaciones pertenecen a diferentes transacciones
2) Las operaciones acceden al mismo item
3) Al menos una de las dos operaciones es una escritura

Por ejemplo, en el schedule S definido más arriba, las operaciones que estan en conflicto entre sí son:
R1(x) con w2(X), r2(X) con w1(X) (estos se llaman conflictos de lectura-escritura)
y w1(X) con w2(X) (estos se llaman conflictos de escritura-escritura).
Y las operaciones que no están en conflicto son r1(X) con r2(X) ya que ambas son de lectura, ni tampoco w2(X) con w1(Y) ya que si bien ambas son de escritura, ellas acceden a diferentes items. Las operaciones r1(X) con w1(X) no están en conflicto porque ambas pertenecen a la misma transacción.
Se dice que hay conflicto cuando al cambiar el orden de las operaciones, se obtienen diferentes resultados. Por ejemplo si el orden es r1(X);w2(X) y se cambia a w2(X);r1(X) el resultado cambiaría porque cuando la operación r1 lee el item X en el segundo caso su valor ha sido escrito por w2, cosa que no era así en el primer caso ya que se leía X ANTES del cambio.
Para los conflictos de escritura-escritura lo que sucedería sería que el valor final de X es diferente ya que es el valor del ULTIMO CAMBIO realizado sobre él.

ACTIVIDAD 4

Realizar la actividad 4 en el campus

Schedule recuperable

Para algunos schedules será bastante simple recuperarse de una falla, mientras que para otros será más difícil o incluso imposible realizar la recuperación. Entonces a continuación vamos a trabajar sobre la teoría a partir de la cual podremos definir la recuperabilidad o no del schedule.

Primero que nada debemos tener en cuenta que si la transacción ha sido commiteada, nunca será necesario hacer rollback de ella. Con esta restricción aseguramos la propiedad de durabilidad de las transacciones.
Entonces decimos que un schedule S es recuperable si:
No existe una transacción T que ejecute un commit antes que las demás transacciones Ti que han escrito un item X, que fuera leído por T ya han ejecutado su commit;

Una transacción T lee de una transacción T' en un schedule S si algún item X es primero escrito por T;
T' no podría haberse abortado antes de que T leyera el item X, y no debería haber ninguna transacción que escriben el item X despues que T' lo escribe y antes que T lo lea, a menos que dichas transacciones fueran abortadas antes que T lea X.
Veamos el siguiente schedule S:

S: r1(X); w1(X);r2(X); r1(Y);w2(X);c2;a1;

donde c2 es el commit de la transacción T2 y a1 es el abort de la transacción T1.
El schedule S es no recuperable porque la transacción T2 ha leído el item X que fue modificado por la transacción T1 que es abortada despues que T2 realiza el commit. Con estas condiciones, T2 debería tambien abortar despues que T1, pero como T2 ya ha realizado el commit (antes del aborto de T1), esto ocasiona la no recuperabilidad.
Para evitar este problema, se utiliza una restricción para los schedules según la cual ninguna transacción puede leer un item que haya escrito otra transacción que no ha sido commiteada.

Hay otro tipo de schedule que se denomina schedule estricto según el cual las transacciones no pueden ni leer ni escribir un item X hasta que la última transacción que escribió ese item haya efectuado el commit. Estos schedules simplifican la recuperabilidad ya que lo unico que se debe hacer en ese caso es poner el valor anterior del item X que se ha guardado en un log a tal efecto (imagen anterior de X).

3. SERIABILIDAD DE LOS SCHEDULES.

Se dice que un schedule es serial cuando cada las operaciones de cada transacción se ejecutan una detrás de otra sin ser intercaladas con las de otras transacciones, es decir se ejecutarán primero todas las operaciones de T1 y luego todas las de T2 (o viceversa).
Entonces claramente solo una transacción a la vez está activa.
El problema con el schedule serial es que no beneficia la concurrencia, si una transacción debe esperar por una operación de entrada-salida, no puede cambiar a la ejecución de otra transacción, como sucedería en la multiprogramación real. No es bueno para benefiiar la concurrencia de usuarios que lanzan transacciones una tras otra.
Por otro lado definimos un schedule serializable como un schedule S con n transacciones que es equivalente a algun schedule serial de las mismas n transacciones.
Cómo definimos un schedule serializable equivalente a uno serial? Porque el resultado es el mismo estado final de la base de datos!
Es decir, para el mismo estado inicial de un item X, el resultado de la operación del schedule serializable es el mismo que el resultado final del schedule serial con las transacciones individuales aplicadas en el mismo orden.

ACTIVIDAD 5

Realizar la actividad 5 en el campus

Si la Actividades fueron verificadas por su tutor usted habrá logrado comprender las principales características de las transacciones en las bases de datos.

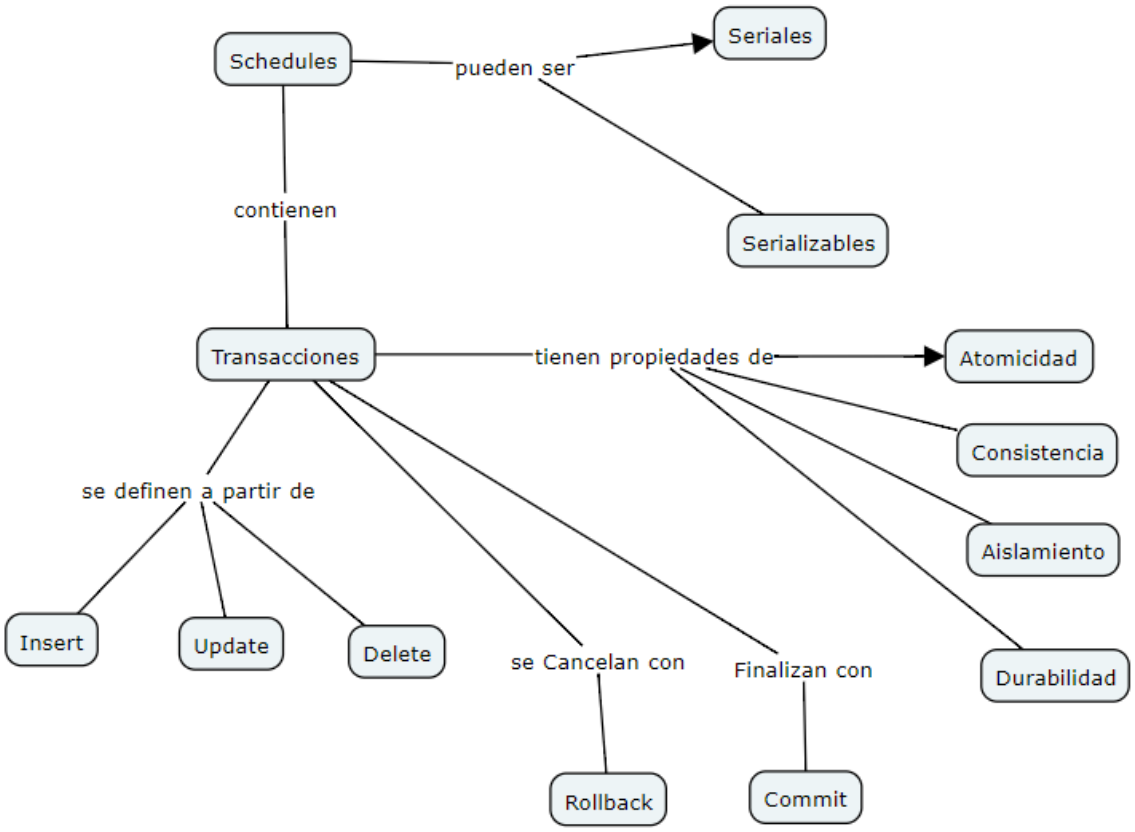
¡FELICITACIONES!

Ha logrado alcanzar los objetivos propuestos para esta Unidad.

SÍNTESIS DE LA UNIDAD

Hagamos un resumen de todo lo visto en esta Unidad.

En esta Unidad vimos los conceptos básicos que usted debe conocer para poder comenzar a trabajar con transacciones en lenguaje SQL para poder modificar los datos de las bases de datos.
Las TRANSACCIONES en una BASE DE DATOS se definen a partir de las operaciones INSERT, DELETE y UPDATE que pueden finalizarce con un COMMIT o cancelarse con un ROLLBACK. Dichas transacciones tienen que reunir las siguientes propiedades: ATOMICIDAD, CONSISTENCIA, AISLAMIENTO y DURABILIDAD. Además es importante tener en cuenta si las TRANSACCIONES son SERIALES o si son SERIALIZABLES para asegurar que los SCHEDULES sean RECUPERABLES.



AUTO-EVALUACIÓN

Aquí le ofrecemos la posibilidad de reflexionar sobre su aprendizaje y sobre la forma como está organizando y llevando a cabo su trabajo. Es una herramienta muy importante. Úsela y si tiene comentarios para compartir, póngalos en el Foro de Interacción con los Tutores o si no son consultas y sólo quiere compartir, en el Bar.

Aspecto a evaluar	MB	B	R	M
1. Mi distribución del tiempo de estudio y trabajo.				
2. Mi entrenamiento en técnicas de estudio en esta unidad. (¿Lo hice, aprendí, usé lo que aprendí?)				
3. Nuevos aprendizajes.				
4. Mi participación en los foros.				
5. Mi participación en el Glosario.				
6. Mi manejo del campus y medios técnicos.				
7. Mi contacto con los compañeros. (¿Intercambio ideas, socializo en el bar?)				
8. Mi contacto con los tutores (¿Pregunto, comento, pido aclaraciones?)				

Considere: ¿Qué debe mejorar?