Manipulación de datos y transaccionalidad en las operaciones

Transaccionalidad en las operaciones

{desafío} latam_

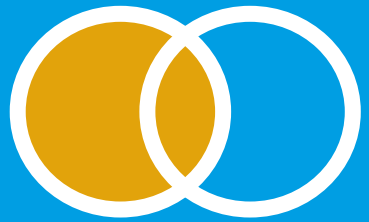
Utilizar lenguaje de manipulación de datos DML para la modificación de los datos existentes en una base de datos dando solución a un problema planteado

- Unidad 1:
  Bases de datos relacionales
- Unidad 2:
  Manipulación de datos y transaccionalidad en las operaciones
- Unidad 3:
  Definición de tablas
- Unidad 4:
  Modelos Entidad-Relación y Relacional

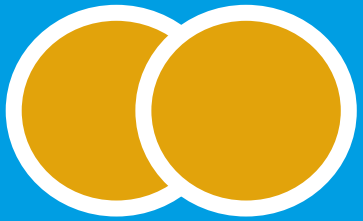
{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Reconoce el concepto de transaccionalidad y su importancia para mantener la consistencia de la información en una base de datos.*



¿Qué tipo de Join muestra la imagen?

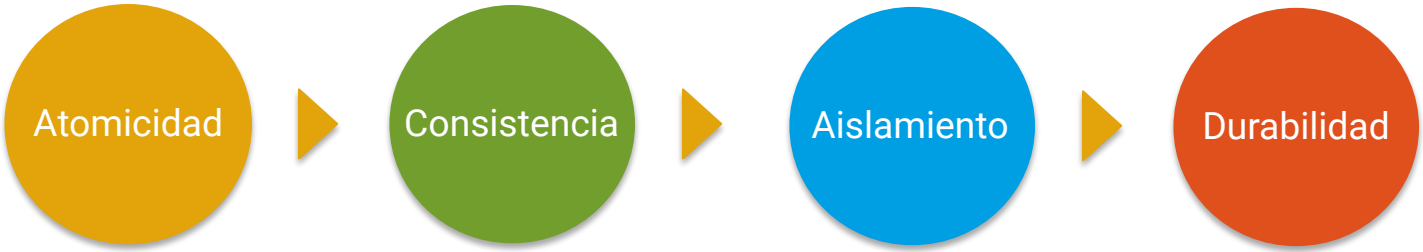


¿Qué tipo de Join muestra la imagen?

/* Qué es una transacción y por qué son importantes */

Transacciones

Las transacciones son secuencias de instrucciones ordenadas, las cuáles pueden ser indicadas de forma manual o pueden ser aplicadas automáticamente.
Las transacciones tienen las siguientes propiedades:



Comandos de transacciones SQL

Comando	Descripción
BEGIN	El sistema permite que se ejecuten todas las sentencias SQL que necesitemos.
COMMIT	Guarda los cambios de la transacción
ROLLBACK	Retrocede los cambios realizados
SAVEPOINT	Guarda el punto de partida al cual volver a la hora de aplicar ROLLBACK
SET TRANSACTION	Le asigna nombre a la transacción

Sintaxis de las transacciones

- Los comandos sólo pueden ser usados con las operaciones INSERT, UPDATE y DELETE.
- Lo que esté entre corchetes es de carácter opcional,

- Sintaxis para iniciar una transacción:

```
SET TRANSACTION [READ ONLY|WRITE][NAME
nombre_transaccion];
```

Sintaxis de las transacciones

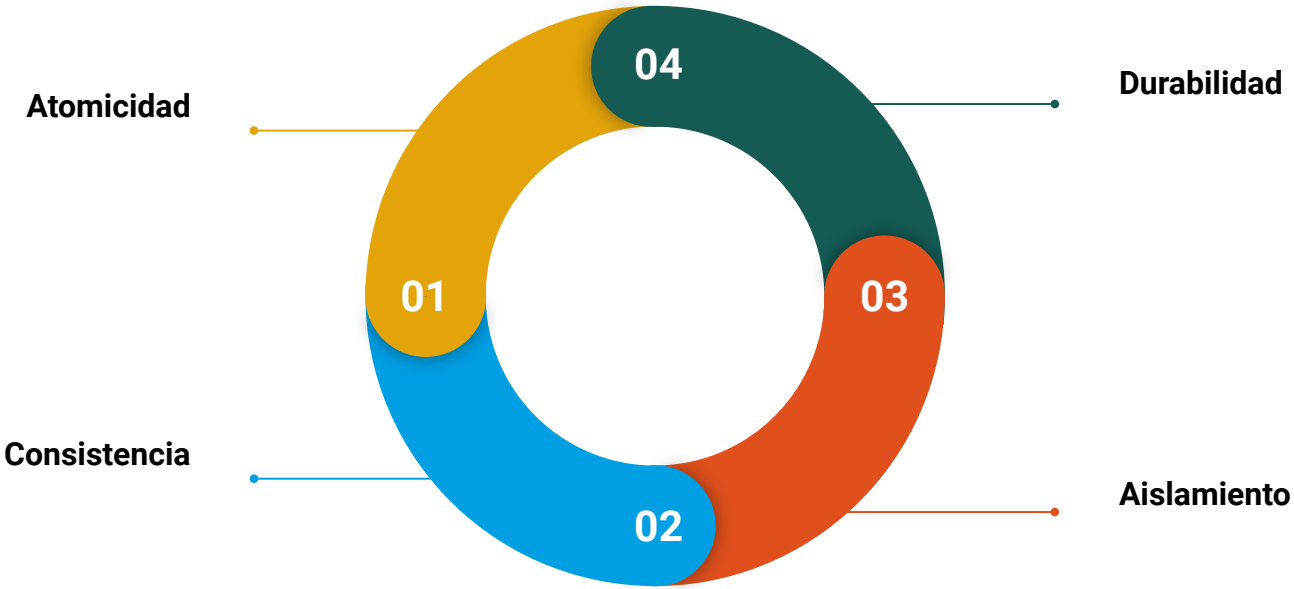
- READ ONLY para solamente leer la base de datos.
- READ WRITE para leer y escribir sobre ella, y poder nombrar la transacción con el comando NAME.

```
SET TRANSACTION [READ ONLY|WRITE][NAME nombre_transaccion];
```

/* Propiedades de las transacciones */

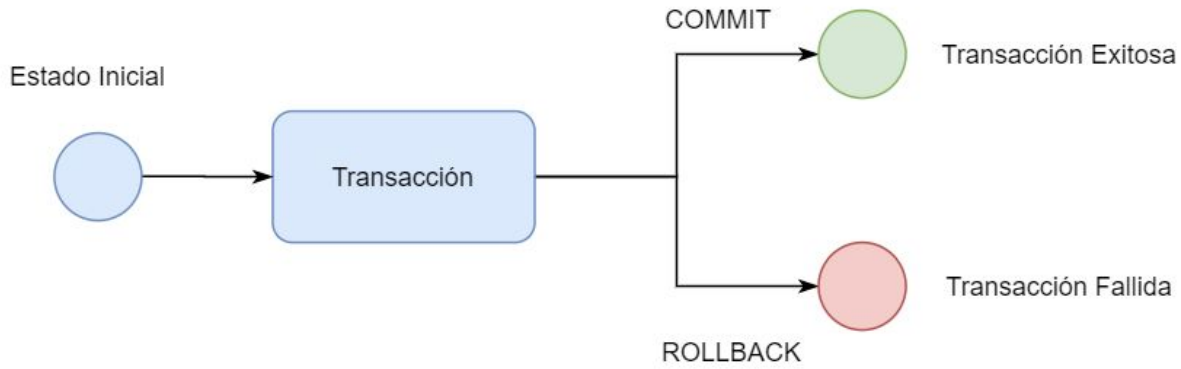
(atomicidad, consistencia, aislamiento, durabilidad)

Propiedades de las transacciones



Gráfica, flujo de una transacción SQL

Una transacción empaqueta varios pasos en una operación, de forma que se completen todos o ninguno, cuidando la integridad de la información, de la siguiente manera:



Importancia del modo transaccional

Imaginemos el siguiente caso: Tenemos una base de datos de 1000 clientes donde efectuamos un giro de dinero, realizamos el giro y nos damos cuenta de que cometimos un error en al menos 300 clientes.

- El modo transaction al momento de modificar una base de datos garantiza la integridad de los datos.
- Sin el modo transaction, los datos de una base de datos podrían quedar en un estado inconsistente si se produce un error durante la ejecución de una instrucción SQL.
- Con el modo transaction es posible volver atrás (Rollback).

/* Confirmación de una transacción */

Ejercicio guiado

"Commit de transacciones en una cuenta bancaria"

{desafío} latam_

Commit de transacciones en una cuenta bancaria

Un ejemplo muy sencillo de la utilidad de las transacciones, es pensar en el funcionamiento de las cuentas de los bancos.

Al hacer una transferencia, por ejemplo:

- ¿Cómo nos aseguramos que el dinero que se resta de una cuenta, se sume en la siguiente?
- ¿Cómo controlamos que efectivamente se mantenga la integridad de los datos en caso de fallas?

Commit de transacciones en una cuenta bancaria

- **Paso 1:** Creamos la base de datos transacciones y nos conectamos.

```
create database transacciones;
\c transacciones;
```

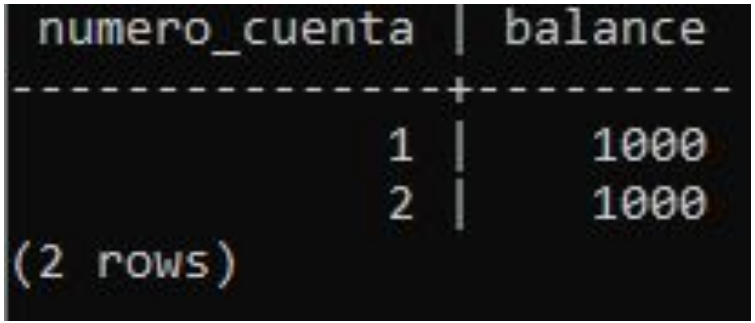
- **Paso 2:** Creamos una tabla llamada cuentas con los campos numero_cuenta y balance.

```
create table cuentas (numero_cuenta int not null unique primary key, balance float check(balance >= 0.00));
```

Commit de transacciones en una cuenta bancaria

- **Paso 3:** Insertamos dos registros a la tabla.

```
insert into cuentas (numero_cuenta, balance) values (1, 1000);
insert into cuentas (numero_cuenta, balance) values (2, 1000);
```



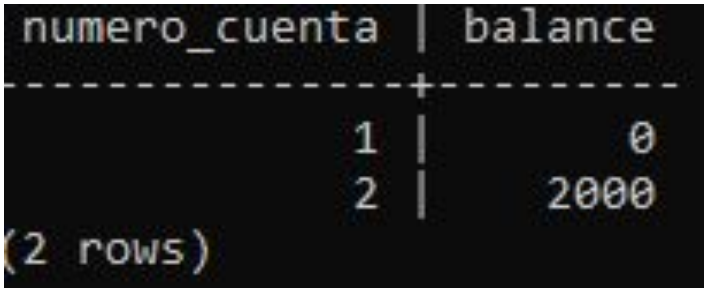
Commit de transacciones en una cuenta bancaria

- **Paso 4:** Si quisiéramos hacer una transferencia de $1000 desde nuestra cuenta 1 a la cuenta 2, una forma de asegurarnos que el monto de nuestro balance disminuya en $1000 y el de la segunda cuenta aumenta en la misma cifra, podría escribirse de la siguiente manera:

```
begin transaction;
UPDATE cuentas set balance = balance - 1000 where numero_cuenta = 1;
UPDATE cuentas set balance = balance + 1000 where numero_cuenta = 2;
```

Commit de transacciones en una cuenta bancaria

- **Paso 5:** Verificamos el estado de la tabla.



```
 numero_cuenta | balance
---------------+---------
             1 |       0
             2 |    2000
(2 rows)
```

Evidenciamos que se debitó el balance según lo esperado, pasamos 1000 de la cuenta 1 a la 2.

Commit de transacciones en una cuenta bancaria

- **Paso 6:** Confirmamos la transacción con commit.

```
commit;
```

Al iniciar nuestra transacción con "BEGIN TRANSACTION", podremos controlar todas aquellas transacciones que se desean realizar y con el “COMMIT” damos por finalizada la transacción para proceder con los cambios permanentes en la base de datos.

¿Cómo controlamos las transacciones SQL?

{desafío} latam_

/* Vuelta atrás de una transacción */

Rollback

- Con este comando podemos deshacer las transacciones que se hayan ejecutado.
- Revertimos los cambios realizados por una transacción hasta el último COMMIT o ROLLBACK ejecutado.
- Esto permite controlar los flujos de ejecución en nuestras transacciones, de manera que volvamos a un estado anterior, sin alterar los datos almacenados.

Rollback de transacción

- **Paso 7:** Continuemos con el ejercicio anterior y apliquemos el uso de ROLLBACK. Para ello haremos un nuevo insert a la base de datos.

```
insert into cuentas (numero_cuenta, balance) values (3, 1000);
```

- **Paso 8:** Iniciamos una transacción para transferir 1000 de la cuenta 3 a la 1.

```
begin transaction;
UPDATE cuentas set balance = balance - 1000 where numero_cuenta = 3;
UPDATE cuentas set balance = balance + 1000 where numero_cuenta = 1;
```

Rollback de transacción

Si consultamos nuevamente la tabla cuentas, veremos el siguiente resultado.

```
transacciones=*# select * from cuentas;
 numero_cuenta | balance
---------------+---------
             2 |    2000
             3 |       0
             1 |    1000
(3 rows)
```

Rollback de transacción

Para deshacer la transacción donde transferimos los 1000 de la cuenta 3 a la 1 ejecutamos el `ROLLBACK;` Esto nos dará como resultado lo siguiente.

```
transacciones=*# rollback;
ROLLBACK
transacciones=# select * from cuentas;
 numero_cuenta | balance
---------------+---------
             1 |       0
             2 |    2000
             3 |    1000
(3 rows)
```

Volvemos atrás y nuestra base de datos queda con la información del último commit.

Save Point

- Es posible tener un mayor control de las transacciones por medio de puntos de recuperación (SAVEPOINTS). Estos permiten seleccionar qué partes de la transacción serán descartadas bajo ciertas condiciones, mientras el resto de las operaciones sí se ejecutan.

- Después de definir un punto de recuperación seguido de su nombre representativo, se puede volver a él por medio de ROLLBACK TO. Todos los cambios realizados por la transacción, entre el punto de recuperación y el rollback se descartan.

En la guía de ejercicios aplicaremos Save Point y Rollback.

/* Modo autocommit */

¿Qué es el modo autocommit?

- En PostgreSQL, por defecto viene configurado el modo AUTOCOMMIT, es decir, que implícitamente una vez que hemos realizado una acción sobre la base de datos, ésta realiza un COMMIT.
- Podemos escribir en terminal `\echo :AUTOCOMMIT`.
- Esto retorna ON, es decir, que está activo.

¿Qué nos permite hacer Commit?

¿Qué habilita begin?

¿Cómo volvemos atrás una transacción SQL?

{desafío} latam_

¿Cuál es el modo por defecto de postgreSQL con los cambios que se realizan en una base de datos?

{desafío}
latam_

Próxima sesión...

- *Guía de ejercicios*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales