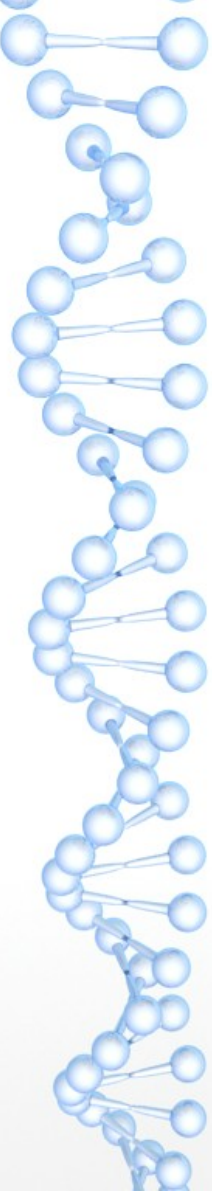


# Transacciones en Postgres

Jesús Reyes Carvajal



Una transacción empaqueta varios pasos en una operación, de forma que se ejecutan todos o ninguno. Los estados intermedios entre los pasos no son visible para otras transacciones ocurridas en el mismo momento.

En el caso de que ocurra algún fallo que impida que se complete la transacción, ninguno de los pasos se ejecutan y no afectan a los objetos de la base de datos.

### **Los pasos de una transacción:**

- BEGIN: Inicia una transacción
- COMMIT: Indica al sistema que han terminado correctamente todas las sentencias SQL
- ROLLBACK: Desecha todas las sentencias SQL que ha realizado.

```
BEGIN;  
INSERT INTO cuentas (n_cuenta, nombre, balance) VALUES (0679259, 'Pepe', 200);  
UPDATE cuentas SET balance = balance - 137.00 WHERE nombre = 'Pepe';  
UPDATE cuentas SET balance = balance + 137.00 WHERE nombre = 'Juan';  
SELECT nombre, balance FROM cuentas WHERE nombre = 'Pepe' AND nombre = 'Juan';  
COMMIT;
```

## EJEMPLO DE ROLLBACK

```
salas=# begin;  
BEGIN  
salas=# delete from items where id=19;  
DELETE 1  
salas=# select * from items;  
id | name | description  
---+-----+-----  
12 | ClaraLuz | milena  
14 | Carlos | Buendia  
16 | Carlos | Caceres  
17 | Julio | Jaramillo  
9 | Maria | Auxiliar  
22 | Sandra Milena | Celadora  
11 | Carlina | Sierra  
(7 rows)
```

```
salas=# rollback;  
ROLLBACK  
salas=# select * from items;  
id | name | description  
---+-----+-----  
12 | ClaraLuz | milena  
14 | Carlos | Buendia  
16 | Carlos | Caceres  
17 | Julio | Jaramillo  
19 | Lorenzo | Rojas  
9 | Maria | Auxiliar  
22 | Sandra Milena | Celadora  
11 | Carlina | Sierra  
(8 rows)
```

```
salas=# commit;
```

```
salas=# select * from items;  
id | name | description  
---+-----+-----  
12 | ClaraLuz | milena  
14 | Carlos | Buendia  
16 | Carlos | Caceres  
17 | Julio | Jaramillo  
19 | Lorenzo | Rojas  
9 | Maria | Auxiliar  
22 | Sandra Milena | Celadora  
11 | Carlina | Sierra
```



## EJEMPLO DE SAVEPOINT

SAVEPOINT: establece un nuevo punto de guardado dentro de la transacción actual.

```
BEGIN;  
  INSERT INTO categorias (cod_cat,nomb_cat) VALUES ('cat22','Refrescos');  
  SAVEPOINT savepoint1;  
  INSERT INTO categorias (cod_cat,nomb_cat) VALUES ('cat23','Endulzantes');  
  ROLLBACK TO SAVEPOINT savepoint1;  
  INSERT INTO categorias (cod_cat,nomb_cat) VALUES ('cat24','Colorantes');  
COMMIT;
```

No inserta la categoría Endulzantes



**Regla ACID: comprobar la propiedad “Atomicidad, lo que se ejecuta en una transacción se ejecuta todo o nada”, usando la tabla “clientes”.**

```
BEGIN;  
INSERT INTO clientes VALUES ('0007','JOSE','ALARCON','VALENCIA');  
INSERT INTO clientes VALUES ('0008','PEPE','MEDINA','VALENCIA');
```

```
-- deshacemos la transacción, en la tabla no se han introducido las filas  
ROLLBACK;
```

```
-- iniciamos una nueva transacción y esta vez la confirmamos:
```

```
BEGIN;  
  INSERT INTO clientes VALUES ('0009','JOSE','ALARCON','VALENCIA');  
  INSERT INTO clientes VALUES ('0010','PEPE','MEDINA','VALENCIA');  
COMMIT;
```

Ahora si están los datos.



## Regla ACID: comprobar la propiedad “**Consistencia - Integridad**”, usando la tabla “**clientes**”.

```
BEGIN;  
INSERT INTO clientes VALUES ('0010','JUAN','PALOMO','VALENCIA');
```

– responde que hay valores duplicados, si insertamos una operación correcta:

```
INSERT INTO clientes VALUES ('0012','JUAN12','PALOMO12','ALICANTE');
```

-- responde que esta transacción está abortada y que ignora todo hasta el fin de la transacción, tenemos que marcar fin transacción para poder seguir:

```
ROLLBACK;  
(si ejecutamos COMMIT, el hace ROLLBACK)
```

Comprobamos que no está ninguna de las filas anteriores, ahora iniciamos otra

```
BEGIN;  
INSERT INTO clientes VALUES ('0012','JUAN12','PALOMO12','ALICANTE');  
COMMIT;
```



**Regla ACID: comprobar la propiedad “Aislamiento, los cambios en una transacción no terminada no se ven en otra sesión”**

**Sesión 1:**

BEGIN;

INSERT INTO clientes VALUES ('0020','NOMBRE20','APELLIDO20','CIUDAD2');

INSERT INTO clientes VALUES ('0030','NOMBRE30','APELLIDO30','CIUDAD3');

**Sesión 2:**

SELECT \* FROM clientes WHERE dni IN ('0020', '0030');

– No aparecen datos

**Sesión 1:**

COMMIT;

**Sesión 2:**

SELECT \* FROM clientes WHERE dni IN ('0020', '0030');

– Ahora si ven datos.



**Regla ACID: comprobar la propiedad “Persistencia: se confirma una transacción, se para y enciende el servidor y se ve si están los datos.”**

```
BEGIN;  
INSERT INTO clientes VALUES ('0040','NOMBRE40','APELLIDO40','CIUDAD4');  
INSERT INTO clientes VALUES ('0050','NOMBRE50','APELLIDO50','CIUDAD5');
```

```
\q  – Sale sin dar COMMIT
```

```
$ pg_ctl -D $PGDATA stop                                --Para el servicio de  
Postgres
```

```
$ pg_ctl -D $PGDATA -l $PGLOG start  --Inicia el servicio de Postgres
```

```
$ psql -U postgres -d empresa  -- Ingresa a la base de datos
```

```
SELECT * FROM clientes WHERE dni IN ('0040', '0050')
```

Aparecen dos filas insertadas





## BLOQUEOS EN POSTGRES

Para ver los bloqueos de una base de datos, debemos consultar la tabla pg\_locks

```
SELECT count(granted),mode,datname as database  
FROM pg_locks l  
JOIN pg_database d ON (d.oid=l.database)  
GROUP BY mode,datname;
```

Un ejemplo de ejecución sería:

```
postgres=# SELECT count(granted),mode,datname as database FROM pg_locks l JOIN pg_database d ON (d.oid=l.database)  
GROUP BY mode,datname;
```

count	mode	database
1	AccessShareLock	postgres
66	AccessShareLock	ejemplo

(2 rows)

Los posible locks que pueden aparecer son los siguientes:

**AccessShareLock:** Bloqueo compartido para lectura: Varias consultas pueden leer de la tabla, pero no permite escribir en ella.

**RowShareLock:** Únicamente una fila tiene un bloqueo compartido de lectura

**RowExclusiveLock:** Fila bloqueada para escritura (INSERT, UPDATE, DELETE), no se puede ni leer ni escribir en ella

**ExclusiveLock:** Lo mismo que RowExclusiveLock pero a nivel de tabla