Titulación: **Grado en Ingeniería Informática e Ingeniería en Sistemas de Información**

Curso: **2020-2021. Convocatoria Ordinaria de Junio**

Asignatura: **Bases de Datos Avanzadas – Laboratorio**

Practica 3: Seguridad, Usuarios y Transacciones.

ALUMNO 1:

Nombre y Apellidos: Alejandro Hernández Martín

DNI: 09074363N

ALUMNO 2:

Nombre y Apellidos: Noelia Marca Retamozo

DNI: 09048809B

Fecha: 02/06/2021

Profesor Responsable: Manuel de Buenaga

Mediante la entrega de este fichero los alumnos aseguran que cumplen con la normativa de autoría de trabajos de la Universidad de Alcalá, y declaran éste como un trabajo original y propio.

En caso de ser detectada copia, se puntuará **TODA** la asignatura como Suspenso – Cero.

Plazos

Tarea en laboratorio:	Semana 4 de Mayo, Semana 11 de Mayo, Semana 18 de Mayo y semana 25 de Mayo.
Entrega de práctica:	Día 7 de Junio. Aula Virtual
Documento a entregar:	Este mismo fichero con las respuestas a las cuestiones planteadas, con el código SQL utilizado en cada uno de los apartados. Así mismo se debe de entregar el fichero de configuración postgresql.conf usado en la práctica, así como los ficheros de log generados en la misma. Si se entrega en

formato electrónico se entregará en un ZIP comprimido: **`DNI'sdelosAlumnos_PECL3.zip`**

AMBOS ALUMNOS DEBEN ENTREGAR EL FICHERO EN LA PLATAFORMA.

Introducción

El contenido de esta práctica versa sobre el manejo de las transacciones en sistemas de bases de datos, así como el control de la concurrencia y la recuperación de la base de datos frente a una caída del sistema. Las transacciones se definen como una unidad lógica de procesamiento compuesta por una serie de operaciones simples que se ejecutan como una sola operación. Entre las etiquetas BEGIN y COMMIT del lenguaje SQL se insertan las operaciones simples a realizar en una transacción. La sentencia ROLLBACK sirve para deshacer todos los cambios involucrados en una transacción y devolver a la base de datos al estado consistente en el que estaba antes de procesar la transacción. También se verá el registro diario o registro histórico del sistema de la base de datos (en PostgreSQL se denomina WAL: Write Ahead Loggin) donde se reflejan todas las operaciones sobre la base de datos y que sirve para recuperar ésta a un estado consistente si se produjera un error lógico o de hardware. La versión de postgres a utilizar deberá ser la versión 13.

Actividades y Cuestiones

En esta parte la base de datos **NETFLIX** deberá de ser nueva y no contener datos. Además, consta de 5 actividades:

- Conceptos generales.
- Manejo de transacciones.
- Concurrencia.
- Registro histórico.
- Backup y Recuperación

- pg_ctl -D ..\\data start

Cuestión 0: Configurar el fichero de Error Reporting and Logging de PostgreSQL para que aparezcan recogidas las sentencias SQL DDL (Lenguaje de Definición de Datos) + DML (Lenguaje de Manipulación de Datos) generadas en dicho fichero. No se pide activar todas las sentencias. No activar la duración de la consulta. También se debe de configurar el log para que en el comienzo de la línea de registro de la información del log (“line prefix”) aparezca el usuario y el nombre del host con su puerto.

log_statement = 'mod'

#log_duration = off

log_line_prefix = '%m [%p] %u %r'

Cuestión 1: Arrancar el servidor Postgres si no está y determinar si se encuentra activo el diario del sistema. Si no está activo, activarlo. Determinar cuál es el directorio y el archivo/s donde se guarda el diario. ¿Cuál es su tamaño? Al abrir el archivo con un editor de textos, ¿se puede deducir algo de lo que guarda el archivo?

Capítulo 19: Server Configuration.

Dentro de postgres.conf fsync está “on” por lo que se deduce que está activo.

Se guarda en el directorio C:\Program Files\PostgreSQL\13\data\pg_wal.

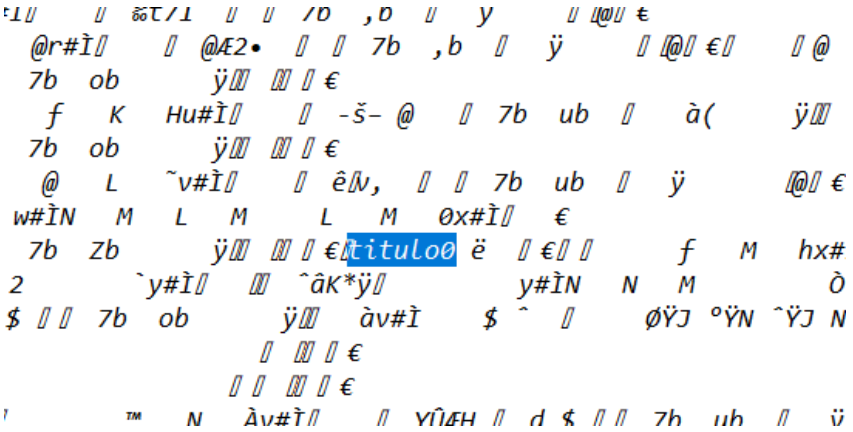
Los Archivos WAL (Write Ahead Log / REDO) son utilizados por postgreSQL para guardar toda la información sobre las transacciones y cambios realizados en la base de datos. Los archivos WAL se utilizan para garantizar la integridad de los datos grabados en la base de datos. También se utilizan para reparar automáticamente posibles inconsistencias en la base de de datos después de una caída súbita del servidor.

El tamaño del diario del sistema es de 1,00 GB (1.073.741.824 bytes). Dentro del diario del sistema Postgres lo fragmenta en archivos de 16 MB.

Estos archivos no son legibles por el ser humano y hay que hacer uso de pg_waldump. Dentro de la consola de sistema hemos ejecutado el siguiente comando:

Cuestión 2: Realizar una operación de inserción de un contenido que sea una serie sobre la base de datos **NETFLIX**. Abrir el archivo de diario ¿Se encuentra reflejada la operación en el archivo del sistema? ¿En caso afirmativo, por qué lo hará?

Para insertar una serie debemos insertar antes alguna tupla en contenidos ya que tiene FK procedentes de contenidos. Hemos insertado una tupla donde el atributo título es “título0”. Si abrimos el último archivo del wal que se ha modificado y hacemos control+B y buscamos “título0” nos sale asi que sabemos que se ha registrado en el diario del sistema.



Cuestión 3: Aplicar el comando pg_waldump.exe al último fichero de WAL que se haya generado. Obtener las estadísticas de ese fichero y comentar qué se está viendo.

```
C:\Program Files\PostgreSQL\13\bin>pg_waldump  -z -p "C:\Program Files\PostgreSQL\13\data\pg_wal" 000000010000000800000CC
Type                                           N      (%)          Record size      (%)             FPI size      (%)        Combined size      (%)
----                                           -      ---          -----------      ---             --------      ---        -------------      ---
XLOG                                         135 (  0,77)                9120 (  0,63)                7920 (  0,96)               17040 (  0,75)
Transaction                                   26 (  0,15)               75037 (  5,22)                   0 (  0,00)               75037 (  3,31)
Storage                                      180 (  1,02)                7560 (  0,53)                   0 (  0,00)                7560 (  0,33)
CLOG                                           0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Database                                       9 (  0,05)                 362 (  0,03)                   0 (  0,00)                 362 (  0,02)
Tablespace                                     0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
MultiXact                                      0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
RelMap                                         0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Standby                                      248 (  1,41)               11576 (  0,80)                   0 (  0,00)               11576 (  0,51)
Heap2                                       8714 ( 49,38)              514761 ( 35,79)               13028 (  1,57)              527789 ( 23,29)
Heap                                        2756 ( 15,62)              418515 ( 29,10)              253908 ( 30,65)              672423 ( 29,67)
Btree                                       5578 ( 31,61)              401228 ( 27,90)              553448 ( 66,82)              954676 ( 42,12)
Hash                                           0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Gin                                            0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Gist                                           0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Sequence                                       0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
SPGist                                         0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
BRIN                                           0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
CommitTs                                       0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
ReplicationOrigin                              0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Generic                                        0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
LogicalMessage                                 0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
                                         --------                     --------                     --------                     --------
Total                                       17646                     1438159 [63,45%]              828304 [36,55%]              2266463 [100%]
pg_waldump: fatal: error en registro de WAL en 8/CC237C80: invalid record length at 8/CC237CB8: wanted 24, got 0
```

1. Type:

En esta columna aparecen los tipos de clasificación de wal

2. N:

Representa el número de veces que ocurre un tipo wal, como se muestra en la imagen, el heap2 tiene 8714. La columna de porcentaje de después, indica la proporción de este tipo de wal, en este mismo caso 49,38.

3. Record size,FPI size y Combined size:

La longitud total ocupada por un determinado tipo de wal se muestra en la columna Tamaño combinado, donde la longitud de los datos de FPI está en la columna de tamaño de FPI y la longitud de los datos que no son de FPI está en la columna de Tamaño de registro.

Cuestión 4: Determinar el identificador de la transacción que realizó la operación anterior. Aplicar el comando anterior al último fichero de WAL que se ha generado y mostrar los registros que se han creado para esa transacción. ¿Qué se puede ver? Interpretar los resultados obtenidos.

Para determinar el identificador hemos realizado:

```
1  select pg_current_wal_insert_lsn();
```

Data Output Messages

	pg_current_wal_insert_lsn pg_lsn
1	8/CC237CB8

```
C:\Program Files\PostgreSQL\13\bin>pg_waldump  -z -p "C:\Program Files\PostgreSQL\13\data\pg_wal" -s 8/CC237EB8 -e 8/CC237F48
el primer registro está ubicado después de 8/CC237EB8, en 8/CC237F10, saltándose 88 bytes
Type                                           N      (%)          Record size      (%)             FPI size      (%)        Combined size      (%)
----                                           -      ---          -----------      ---             --------      ---        -------------      ---
XLOG                                           0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Transaction                                    0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Storage                                        0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
CLOG                                           0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Database                                       0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Tablespace                                     0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
MultiXact                                      0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
RelMap                                         0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Standby                                        1 (100,00)                  50 (100,00)                   0 (  0,00)                  50 (100,00)
Heap2                                          0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Heap                                           0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Btree                                          0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Hash                                           0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Gin                                            0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Gist                                           0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Sequence                                       0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
SPGist                                         0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
BRIN                                           0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
CommitTs                                       0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
ReplicationOrigin                              0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
Generic                                        0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
LogicalMessage                                 0 (  0,00)                   0 (  0,00)                   0 (  0,00)                   0 (  0,00)
                                         --------                     --------                     --------                     --------
Total                                          1                           50 [100,00%]                  0 [0,00%]                   50 [100%]
```

Cuestión 5: Se va a crear un backup del clúster de postgres. Este backup será utilizado más adelante para recuperar el sistema frente a una caída del sistema. Realizar solamente el backup mediante el procedimiento descrito en el apartado 25.3 del manual (versión 13 es *"Continous Archiving and point-in-time recovery (PITR)"*.

Antes de crear el backup debemos configurar el WAL en postgres.conf:

- Wal_level = replica
- Archive_mode = on
- archive_command ='copy "C:\Program Files\PostgreSQL\13\data\pg_wal"'
- Max_wall_senders = 10

Comando para hacer el backup:

```
C:\Program Files\PostgreSQL\13\bin>pg_basebackup -h localhost -U postgres -p 5433 -D D:\NOELIA\backup2021
Contraseña:
```

Este equipo > Disco local (D:) > NOELIA > backup2021

Nombre	Fecha de modificación	Tipo	Tamaño
base	31/05/2021 17:45	Carpeta de archivos	
global	31/05/2021 17:45	Carpeta de archivos	
log	31/05/2021 17:45	Carpeta de archivos	
pg_commit_ts	31/05/2021 17:45	Carpeta de archivos	
pg_dynshmem	31/05/2021 17:45	Carpeta de archivos	
pg_logical	31/05/2021 17:45	Carpeta de archivos	
pg_multixact	31/05/2021 17:45	Carpeta de archivos	
pg_notify	31/05/2021 17:45	Carpeta de archivos	
pg_replslot	31/05/2021 17:45	Carpeta de archivos	
pg_serial	31/05/2021 17:45	Carpeta de archivos	
pg_snapshots	31/05/2021 17:45	Carpeta de archivos	
pg_stat	31/05/2021 17:45	Carpeta de archivos	
pg_stat_tmp	31/05/2021 17:45	Carpeta de archivos	
pg_subtrans	31/05/2021 17:45	Carpeta de archivos	
pg_tblspc	31/05/2021 17:45	Carpeta de archivos	
pg_twophase	31/05/2021 17:45	Carpeta de archivos	
pg_wal	31/05/2021 17:45	Carpeta de archivos	
pg_xact	31/05/2021 17:45	Carpeta de archivos	
backup_label	31/05/2021 17:44	Archivo	1 KB
backup_manifest	31/05/2021 17:45	Archivo	280 KB
current_logfiles	31/05/2021 17:45	Archivo	1 KB
pg_hba	31/05/2021 17:45	Archivo CONF	5 KB
pg_ident	31/05/2021 17:45	Archivo CONF	2 KB
PG_VERSION	31/05/2021 17:45	Archivo	1 KB
postgresql.auto	31/05/2021 17:45	Archivo CONF	1 KB
postgresql	31/05/2021 17:45	Archivo CONF	29 KB

Cuestión 6: Qué herramientas disponibles tiene PotsgreSQL para controlar la actividad de la base de datos en cuanto a la concurrencia y transacciones? ¿Qué información es capaz de mostrar? ¿Dónde se guarda dicha información? ¿Cómo se puede mostrar?

Internamente, la consistencia de los datos se mantiene gracias al modelo multiversión de control de concurrencia, MVCC. Este modelo tiene la ventaja de que si está en lectura no bloquea la escritura y si está en escritura no bloquea la lectura.

Esto evita que las declaraciones vean datos inconsistentes producidos por transacciones concurrentes que realizan actualizaciones en las mismas filas de datos, proporcionando aislamiento de transacciones para cada sesión de base de datos. MVCC, evitando las metodologías de bloqueo de los sistemas de bases de datos tradicionales, minimiza la contención del bloqueo para permitir un rendimiento razonable en entornos multiusuario.

Se encarga de hacer cumplir la coherencia con las transacciones serializables y de hacer cumplir la consistencia con bloqueos de bloqueo explícitos.

A la hora de realizar la lectura, existen varios niveles de aislamiento de las transacciones:

1. Serializable: el cual es el más estricto. Los siguientes solo se producen en casos más aislados resultantes de intereacción entre transacciones concurrentes:
1. Lectura no repetible: Una transacción vuelve a leer los datos que ha leído previamente y encuentra que los datos han sido modificados por otra transacción
2. Lectura fantasma: Una transacción vuelve a ejecutar una consulta que devuelve un conjunto de filas que satisfacen una condición de búsqueda y descubre que el conjunto de filas que cumple la condición ha cambiado debido a otra transacción recientemente confirmada.
3. Lectura sucia: Una transacción lee datos escritos por una transacción no confirmada concurrente.

A la hora de los bloqueos, tenemos también varios niveles:

1. A nivel de tabla.
2. A nivel de fila.
3. A nivel de página.
4. Puntos muertos.
5. Bloqueos de asesoramiento.

Cuestión 7: Crear dos usuarios en la base de datos que puedan acceder a la base de datos **NETFLIX** identificados como usuario1 y usuario2 que tengan permisos de lectura/escritura a la base de datos tienda, pero que no puedan modificar su estructura. Describir el proceso seguido.

Primero se crea el usuario1 y posteriormente, se le asignan los privilegios de select, insert, update y delete que corresponden a lectura y escritura.

Query Editor Query History Notifications Explain

```
1 CREATE USER usuario1 WITH PASSWORD 'usuario1'
```

netflix1/postgres@PostgreSQL 13

Query Editor Query History Notifications Explain

```
1 grant select,insert,update,delete on
2 contenidos,cuenta_usuario,genero,genero_contenidos,pago,peliculas,series,suscripcion_contenidos to usuario1
```

Data Output Messages

```
Query returned successfully in 87 msec.
```

Después se crea el usuario2 y después se le asignan los privilegios de select, insert, update y delete que corresponden a lectura y escritura.

Query Editor Query History Notifications Explain

```
1 CREATE USER usuario2 WITH PASSWORD 'usuario2'
```

netflix1/postgres@PostgreSQL 13

Query Editor Query History Notifications Explain

```
1 grant select,insert,update,delete on
2 contenidos,cuenta_usuario,genero,genero_contenidos,pago,peliculas,series,suscripcion_contenidos to usuario2
```

Data Output Messages

```
Query returned successfully in 79 msec.
```

Cuestión 8: Abrir una transacción que inserte un nuevo contenido que sea película en la base de datos (NO cierre la transacción). Realizar una consulta SQL para mostrar los

contenidos de la base de datos dentro de esa transacción. Consultar la información sobre lo que se encuentra actualmente activo en el sistema. ¿Qué conclusiones se pueden extraer?

Se ha iniciado la transacción, se ha insertado la pelicula nueva y hemos hecho la consulta pertinente para que nos muestre las peliculas creadas. Este es el resultado:

```
1  begin;
2  INSERT INTO public.contenidos(
3      "producto_ID", titulo, fecha_lanzamiento, valoracion)
4      VALUES (2,'titulo_2', '15-04-2020', 6);
5  INSERT INTO public.peliculas(
6      duracion,"producto_ID_contenidos")
7      VALUES (40,2);
8
```

```
8
9  select * from peliculas
```

Data Output | Messages

	duracion numeric	producto_ID_contenidos [PK] numeric
1	40	2

Actualmente, se encuentran 2 sesiones en el sistema. Como se puede apreciar en el primer cuadro, hay una transacción activa y otra en 'idle', lo que quiere decir que la transacción está abierta (dentro de BEGIN) e inactiva.

Server activity

Sessions | Locks | Prepared Transactions | Search

PID	User	Application	Client	Backend start	State	Wait event
15796	postgres	pgAdmin 4 - CONN:6450102	::1	2021-05-31 18:42:17 CEST	idle	Client: Client
17080	postgres	pgAdmin 4 - DB:netflix1	::1	2021-05-31 18:41:34 CEST	active	
18920	postgres	pgAdmin 4 - CONN:2423089	::1	2021-05-31 18:42:56 CEST	idle in transaction	Client: Client
19196	postgres	pgAdmin 4 - CONN:6427988	::1	2021-05-31 18:42:18 CEST	idle	Client: Client
23544	postgres	pgAdmin 4 - CONN:6077372	::1	2021-05-31 18:42:27 CEST	idle	Client: Client

Cuestión 9: Cierre la transacción anterior. Abrir una transacción T1 en el usuario1 que realice las siguientes operaciones sobre la base de datos **NETFLIX**. NO termine la transacción. Simplemente:

Hemos inicado sesión con Usuario1 y con esta consulta se puede ver que nos hemos mbiado correctamente.

Query Editor | Query History | N

```
1 SELECT CURRENT_USER
2
3
```

Data Output | Messages

	current_user name
1	usuario1

- Inserte una nueva suscripción con suscripción_ID 1500.

```
1 begin;
2 INSERT INTO public.suscripcion(
3     tipo, precio, fecha, "suscripcion_ID")
4     VALUES (2, 2.56, '03-12-2020', 1500)
```

Data Output | Messages

Query returned successfully in 65 msec.

- Inserte un nuevo usuario asociado a la suscripción anterior.

```
insert into public.cuenta_usuario("ID",nombre,
                                  direccion,e_mail,telefono,
                                  "suscripcion_ID_suscripcion")
        VALUES (1,'nombre1','direccion1','email1@email.com',69493833,1500)
```

Data Output | Messages

```
Query returned successfully in 113 msec.
```

- Inserte un nuevo pago del usuario anterior debido a la suscripción que tiene asociada con pago_ID 98765.

```
insert into public.pago("pago_ID",metodo,fecha,cantidad,
                        "ID_cuenta_usuario","suscripcion_ID_suscripcion")
        VALUES (98765,'Tarjeta','20-02-2020',300,1,1500)
```

Data Output | Messages

```
Query returned successfully in 76 msec.
```

Cuestión 10: Realizar cualquier consulta SQL que muestre los datos anteriores insertados para ver que todo está correcto.

```
select * from suscripcion where "suscripcion_ID"=1500;
select * from cuenta_usuario where "suscripcion_ID_suscripcion"=1500;
select * from pago where "pago_ID"=98765;
```

Data Output | Messages

	tipo numeric	precio double precision	fecha date	suscripcion_ID [PK] numeric
1	2	2.56	2020-12-03	1500

```
select * from cuenta_usuario where "suscripcion_ID_suscripcion"=1500;
select * from pago where "pago_ID"=98765;
```

Data Output | Messages

	ID [PK] numeric	nombre text	direccion text	e_mail text	telefono numeric	suscripcion_ID_suscripcion numeric
1	1	nombre1	direccion1	email1...	69493833	1500

```
select * from pago where "pago_ID"=98765;
```

Data Output | Messages

	pago_ID [PK] numeric	metodo text	fecha date	cantidad double precision	ID_cuenta_usuario numeric	suscripcion_ID_suscripcion numeric
1	98765	Tarjeta	2020-0...	300	1	1500

Cuestión 11: Establecer una **nueva conexión** a la base de datos con el usuario2 (abrir otra sesión diferente a la abierta actualmente que pertenezca al usuario2) y realizar la misma consulta. ¿Se nota algún cambio? En caso afirmativo, ¿a qué puede ser debido el diferente funcionamiento en la base de datos para ambas consultas? ¿Qué información de actividad hay registrada en la base de datos en este momento?

Como se observa en las imágenes a continuación, hay cambios. Esto es debido a que la transacción T1 aún no ha hecho commit, y por lo tanto no se han aplicado los cambios a la base de datos. En cuanto a la actividad, tenemos una transacción en curso, 2 sesiones de usuario2 y la del superusuario postgres.

```
1 select * from suscripcion where "suscripcion_ID"=1500;
2 select * from cuenta_usuario where "suscripcion_ID_suscripcion"=1500;
3 select * from pago where "pago_ID"=98765;
```

Data Output | Messages

tipo numeric	precio double precision	fecha date	suscripcion_ID [PK] numeric

```
2 select * from cuenta_usuario where "suscripcion_ID_suscripcion"=1500;
3 select * from pago where "pago_ID"=98765;
```

Data Output | Messages

ID [PK] numeric	nombre text	direccion text	e_mail text	telefono numeric	suscripcion_ID_suscripcion numeric

```
3 select * from pago where "pago_ID"=98765;
```

Data Output | Messages

pago_ID [PK] numeric	metodo text	fecha date	cantidad double precision	ID_cuenta_usuario numeric	suscripcion_ID_suscripcion numeric

Cuestión 12: ¿Se encuentran los nuevos datos físicamente en las tablas de la base de datos? Entonces, ¿de dónde se obtienen los datos de la cuestión 10 y/o de la 11?

No, aún no se han guardado en memoria. Los datos de la T1 solo los conocerá usuario1 mientras no se haga ningún commit (los datos se obtienen de realizar una "dirty read" de una transacción aun no comprometida) y en la cuestión 11, usuario2 lee de la memoria global los datos.

Cuestión 13: Finalizar con éxito la transacción T1 y realizar la consulta de la cuestión 10 y 11 sobre ambos usuarios conectados. ¿Qué es lo que se obtiene ahora? ¿Por qué?

Una vez hecho el commit en T1, se puede ver que usuario2 y usuario1 poseen los mismos valores en la base de datos, esto se debe a que cuando se hace commit en cualquier transacción, a partir de ahí, cualquiera que lea tablas modificadas por transacciones leerá los valores registrados tras el commit.

netflix1/usuario2@PostgreSQL 13

Query Editor | Query History | Notifications | Explain

```
select * from suscripcion where "suscripcion_ID"=1500;
select * from cuenta_usuario where "suscripcion_ID_suscripcion"=1500;
select * from pago where "pago_ID"=98765;
```

Data Output | Messages

	tipo numeric	precio double precision	fecha date	suscripcion_ID [PK] numeric
1	2	2.56	2020-1...	1500

Cuestión 14: Sin ninguna transacción en curso, abrir una transacción en un usuario cualquiera y realizar las siguientes operaciones:

- Insertar una suscripción nueva con suscripción_ID a 3000.
- Insertar un usuario nuevo de la suscripción 3000.
- Insertar un pago del usuario anterior con pago_ID 45678.

netflix1/postgres@PostgreSQL 13

Query Editor | Query History | Notifications | Explain

```
BEGIN;
INSERT INTO public.suscripcion(
    tipo,precio,fecha,"suscripcion_ID")
    VALUES (5,8.9,'22-03-2020',3000);

INSERT INTO public.cuenta_usuario(
    "ID",nombre,direccion,e_mail,telefono,"suscripcion_ID_suscripcion")
    VALUES (5,'Nombre5','direccion5','email5@gmail.com',69998521,3000);

INSERT INTO public.pago (
    "pago_ID",metodo,fecha,cantidad,"ID_cuenta_usuario","suscripcion_ID_suscripcion")
    VALUES (45678,'Efectivo','23-04-2020',96.32,5,3000)
```

Data Output | Messages

Query returned successfully in 84 msec.

- Hacer una modificación de la suscripción anterior para cambiar su ID de 3000 a 1500.

Nos encontramos en el Usuario2 y como solo tiene permisos de lectura y escritura no puede realizar updates. Este es el motivo por el que se nos ha denegado el permiso.

Query Editor | Query History | Notifications | Explain

```
update suscripcion
set "suscripcion_ID"=1500
where "suscripcion_ID"=3000
```

Data Output | Messages

Query returned successfully in 65 msec.

- Cerrar la transacción.

```
commit
```

Data Output | Messages

```
Query returned successfully in 120 msec.
```

¿Cuál es el estado final de la base de datos? ¿Por qué?

El estado final de la base de datos, es que todas las operaciones se han realizado correctamente y por tanto, se ha actualizado. En este caso, solo va a haber una suscripcion con ID=1500, puesto que antes de cerrar la transacción, se ha hecho un update de las suscripción con ID 3000 que estaba en memoria pero no en disco, y se ha actualizado el valor de la memoria a 1500. Posteriormente, al hacer el commit, se ha actualizado en disco.

Query Editor | Query History | Notifications | Explain

```
select * from suscripcion where "suscripcion_ID"=3000
```

Data Output | Messages

tipo numeric	precio double precision	fecha date	suscripcion_ID [PK] numeric

Query Editor | Query History | Notifications | Explain

```
select * from suscripcion where "suscripcion_ID"=1500
```

Data Output | Messages

	tipo numeric	precio double precision	fecha date	suscripcion_ID [PK] numeric
1	5	8.9	2020-0...	1500

Los siguientes datos, pertenecen a los dos insert correspondientes de los cuales no se ha hecho ninguna modificación, por lo que se almacena en disco con los valores de memoria que se han pedido en la consulta:

```
select * from cuenta_usuario where "suscripcion_ID_suscripcion"=3000
select * from pago where "pago_ID"=45876
```

Data Output | Messages

	ID [PK] numeric	nombre text	direccion text	e_mail text	telefono numeric	suscripcion_ID_suscripcion numeric
1	5	nombre5	direccion5	email@...	67772111	3000

```
select * from pago where "pago_ID"=45678
```

Data Output | Messages

	pago_ID [PK] numeric	metodo text	fecha date	cantidad double precision	ID_cuenta_usuario numeric	suscripcion_ID_suscripcion numeric
1	45678	Efectivo	2020-0...	96.32	5	3000

Todos los usuarios pueden ver los últimos datos insertados porque la transacción se ha comprometido ya.

Cuestión 15: Repetir la cuestión 9 con otra suscripción, usuario y pago. Realizar la misma consulta de la cuestión 10, pero ahora terminar la transacción con un ROLLBACK y repetir la consulta con los mismos dos usuarios. ¿Cuál es el resultado? ¿Por qué?

```
BEGIN;
INSERT INTO public.suscripcion(
    tipo, precio, fecha, "suscripcion_ID")
    VALUES (7, 33.98, '01-01-2020', 789456);
INSERT INTO public.cuenta_usuario(
    "ID",nombre,direccion,e_mail,telefono,"suscripcion_ID_suscripcion"
) VALUES (10,'nombre10','direccion10','email10@email.com',666666669,789456);

INSERT INTO public.pago (
    "pago_ID",metodo,fecha,cantidad,"ID_cuenta_usuario","suscripcion_ID_suscripcion"
) VALUES (12,'Paypal','22-04-2020',99.97,10,789456);
```

Data Output | Messages

```
Query returned successfully in 77 msec.
```

Aplicamos rollback, y vemos que todos los datos que hemos insertado se han eliminado porque la función de rollback deshace todo lo que se ha hecho en esa transacción salvo que hubiéramos puesto un punto a partir del cual no se hiciera rollback, pero no es el caso.

```
14
15 ROLLBACK;
```

Data Output | Messages

```
Query returned successfully in 113 msec.
```

```
14 select * from suscripcion where "suscripcion_ID"=789456;
15 select * from cuenta_usuario where "suscripcion_ID_suscripcion"=789456;
16 select * from pago where "pago_ID"=12;
```

Data Output | Messages

tipo numeric	precio double precision	fecha date	suscripcion_ID [PK] numeric

```
15 select * from cuenta_usuario where "suscripcion_ID_suscripcion"=789456;
16 select * from pago where "pago_ID"=12;
```

Data Output | Messages

ID [PK] numeric	nombre text	direccion text	e_mail text	telefono numeric	suscripcion_ID_suscripcion numeric

```
16 select * from pago where "pago_ID"=12;
```

Data Output | Messages

pago_ID [PK] numeric	metodo text	fecha date	cantidad double precision	ID_cuenta_usuario numeric	suscripcion_ID_suscripcion numeric

Cuestión 16: ¿Qué es lo que ocurre en el sistema gestor de base de datos si dentro de una transacción que cambia la cantidad del pago con ID 98765 se abre otra transacción que borre dicho pago? ¿Por qué?

```
3 begin;
4 update pago set cantidad=100.99 where "pago_ID"=98765;
5
```

Data Output | Messages

```
Query returned successfully in 103 msec.
```

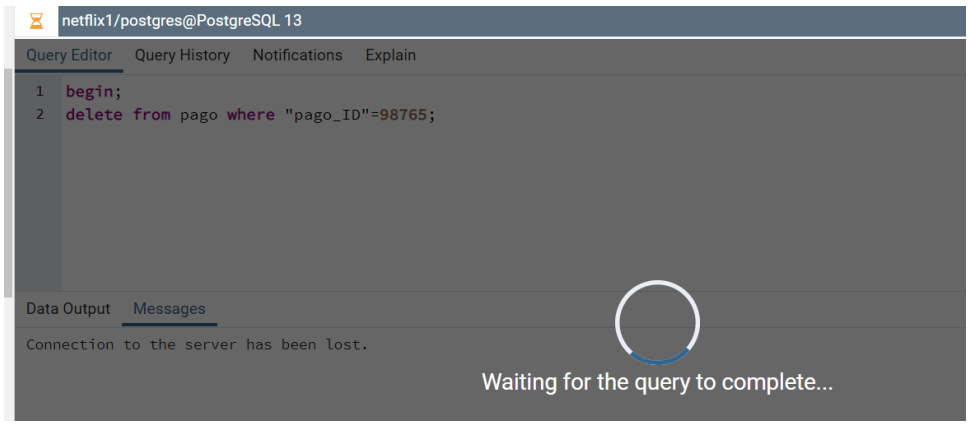
```
6  begin;
7  delete from pago where "pago_ID"=98765;
```

Data Output Messages

```
WARNING:  ya hay una transacción en curso
DELETE 1

Query returned successfully in 126 msec.
```

La transacción T2 se ha bloqueado y no se ejecuta ya que se pide modificar la misma tabla que se pide modificar en T1, la cual aún no se ha comprometido. T2 queda a la espera de que termine T1 para ejecutarse:



Si comprometemos la T1 se ejecuta la T2. Si hacemos una lectura de un pago cuyo ID sea 98765 no aparecerá nada.

Query Editor Query History Notifications Explain

```
1  begin;
2  delete from pago where "pago_ID"=98765;
```

Data Output Messages

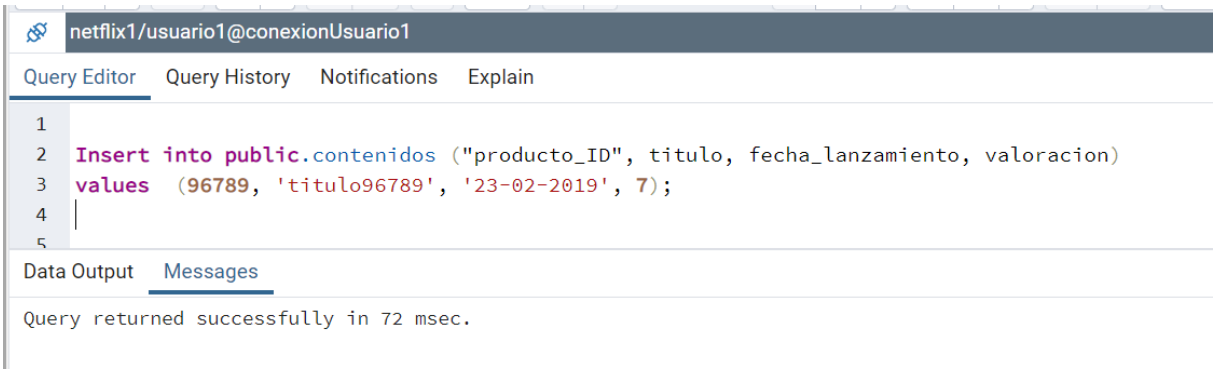
```
Query returned successfully in 5 min 56 secs.
```

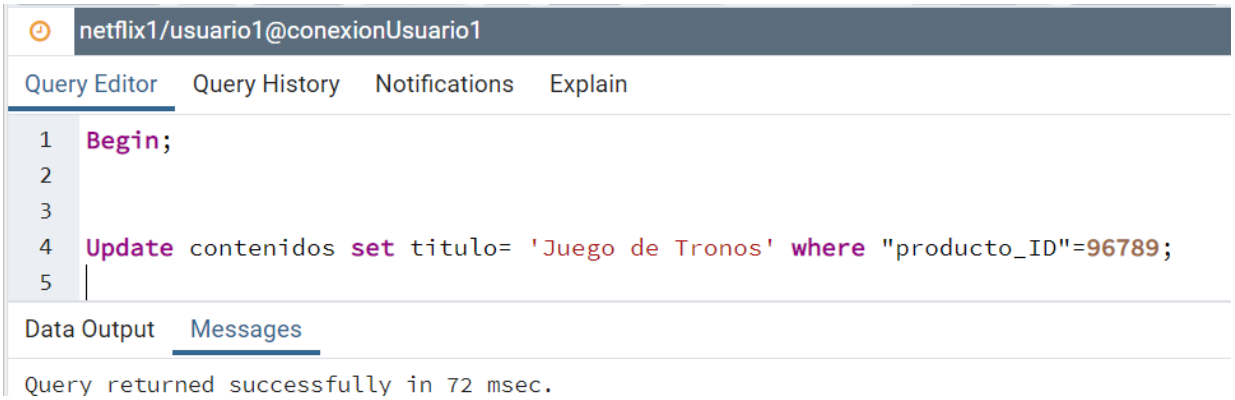
```
10 commit
11
12 select * from pago where "pago_ID"=98765
```

Data Output Messages

pago_ID [PK] numeric	metodo text	fecha date	cantidad double precision	ID_cuenta_usuario numeric	suscripcion_ID_suscripcion numeric

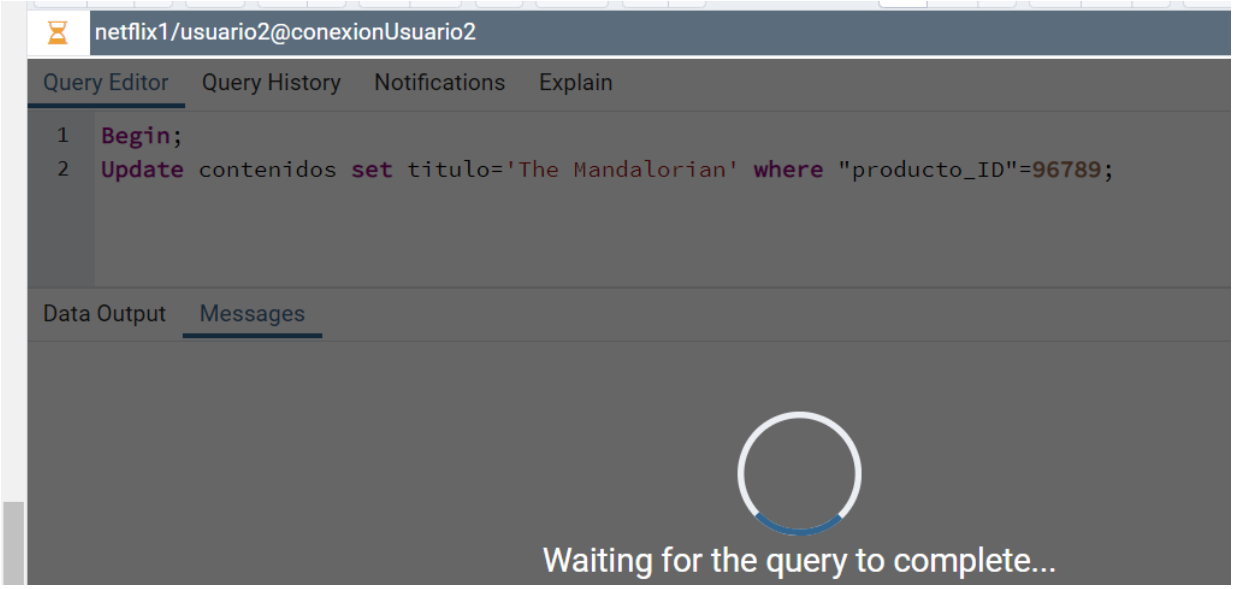
Cuestión 17: Cerrar todas las sesiones anteriores. Abrir una sesión con el usuario1 de la base de datos **NETFLIX**. Insertar en la tabla tienda un nuevo contenido con producto_ID a 96789. Abrir una transacción T1 en este usuario y realizar una modificación de la tienda con producto_ID 96789 para actualizar el título a "Juego de Tronos". No cierre la transacción.

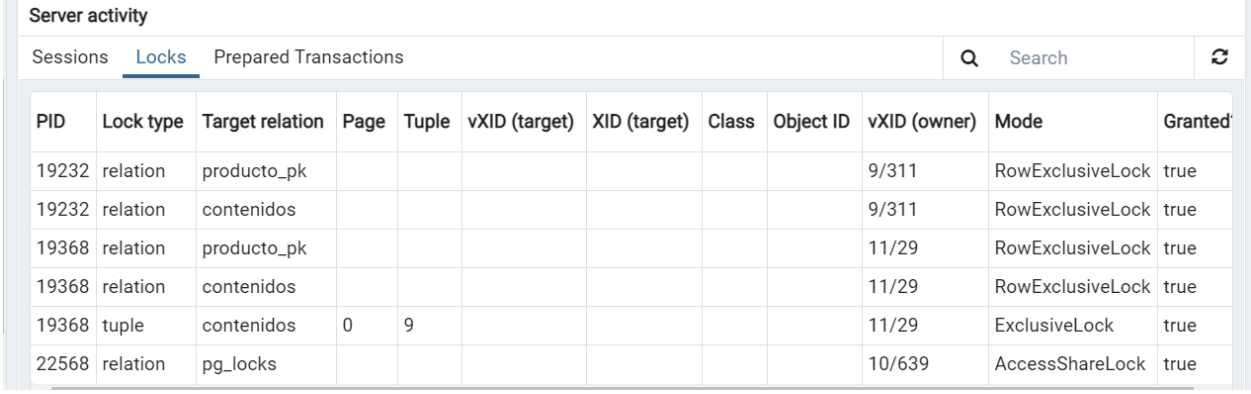




Cuestión 18: Abrir una sesión con el usuario2 de la base de datos **NETFLIX**. Abrir una transacción T2 en este usuario y realizar una modificación del contenido con producto_ID 96789 y cambiar el título a "The Mandalorian". No cierre la transacción. ¿Qué es lo que ocurre? ¿Por qué? ¿Qué información se puede obtener de la actividad?

La transacción t2 se encuentra bloqueada, el motivo es que al querer escribir (ya sea insert, delete o update) sobre la misma tupla, Postgres le da un XLOCK a la T2 y hasta que no se comprometa la T1, la T2 no podrá escribir sobre la tupla y tabla de contenidos. En la segunda captura se puede apreciar como en la dashboard aparece que la sesión con PID 19368 tiene *RowExclusiveLock* (xlock sobre la tupla) y *ExclusiveLock* (xlock sobre la tabla).

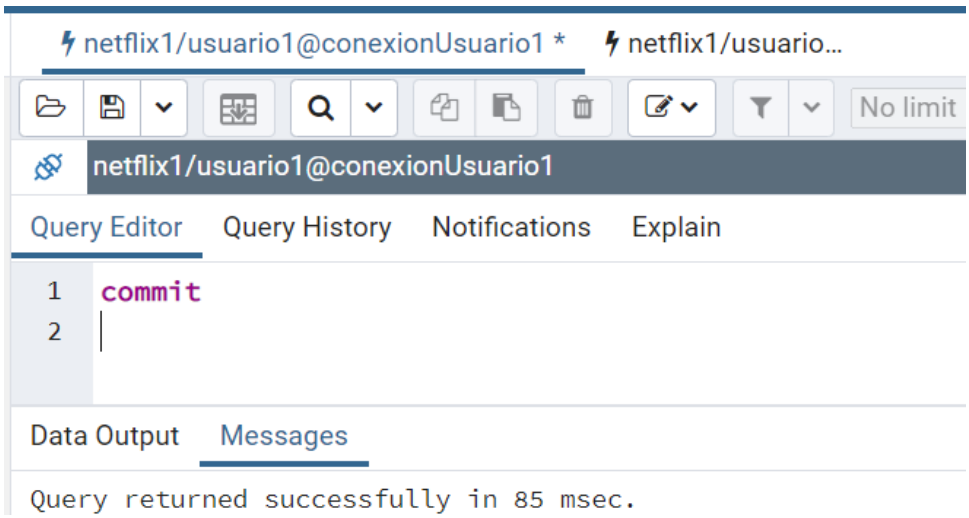




PID	Lock type	Target relation	Page	Tuple	vXID (target)	XID (target)	Class	Object ID	vXID (owner)	Mode	Granted
19232	relation	producto_pk							9/311	RowExclusiveLock	true
19232	relation	contenidos							9/311	RowExclusiveLock	true
19368	relation	producto_pk							11/29	RowExclusiveLock	true
19368	relation	contenidos							11/29	RowExclusiveLock	true
19368	tuple	contenidos	0	9					11/29	ExclusiveLock	true
22568	relation	pg_locks							10/639	AccessShareLock	true

Cuestión 19: Comprometa la transacción T1, ¿Qué es lo que ocurre? ¿Por qué? ¿Cuál es el estado final de la información del contenido con código 96789 para ambos usuarios? ¿Por qué?

Se compromete la t1;



Se puede apreciar que la t2, se ha ejecutado inmediatamente después de comprometer la t1:

netflix1/usuario2@conexionUsuario2

Query Editor | Query History | Notifications | Explain

```
1 Begin;
2 Update contenidos set titulo='The Mandalorian' where "producto_ID"=96789;
```

Data Output | Messages

```
Query returned successfully in 17 min 31 secs.
```

Realizamos un select para comprobar qué cambios se han efectuado. Podemos observar que en el usuario1 y el usuario de postgres no observamos el último cambio que se hizo desde usuario2 de actualizar el nombre a 'The Mandalorian'. Sigue apareciendo 'Juego de tronos'. Esto es así, porque se ha comprometido la t1 donde se guardaba este dato, pero no la t2 donde se cambiaba de nombre:

netflix1/postgres@PostgreSQL 13

Query Editor | Query History | Notifications | Explain

```
1 select * from contenidos where "producto_ID"=96789
```

Data Output | Messages

	producto_ID [PK] numeric	titulo text	fecha_lanzamiento date	valoracion numeric
1	96789	Juego de Tronos	2019-02-23	7

Vista postgres

netflix1/usuario1@conexionUsuario1

Query Editor | Query History | Notifications | Explain

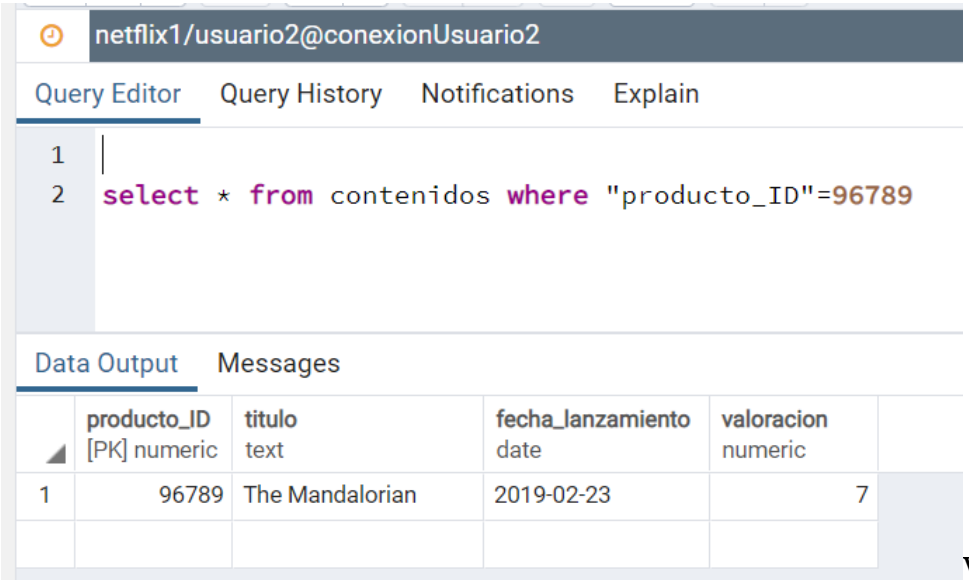
```
1 select * from contenidos where "producto_ID"=96789
```

Data Output | Messages

	producto_ID [PK] numeric	titulo text	fecha_lanzamiento date	valoracion numeric
1	96789	Juego de Tronos	2019-02-23	7

Vista usuario1

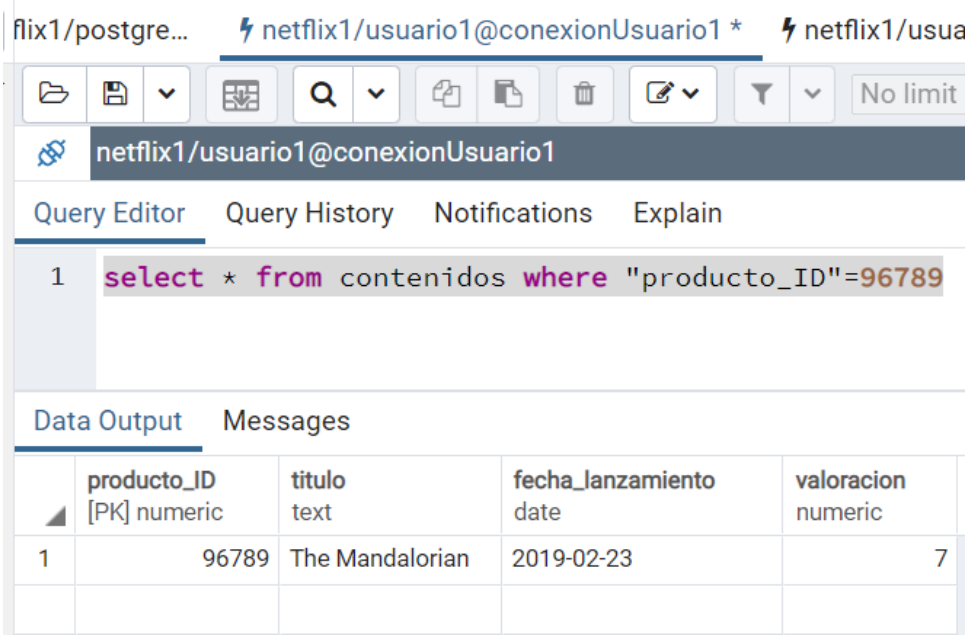
En cambio, viendo a través de la conexión de usuario2, podemos observar que sí aparece el cambio puesto que está en memoria de usuario2. Pero no se ha guardado a disco puesto que no se ha realizado ningún commit.



Vista usuario2

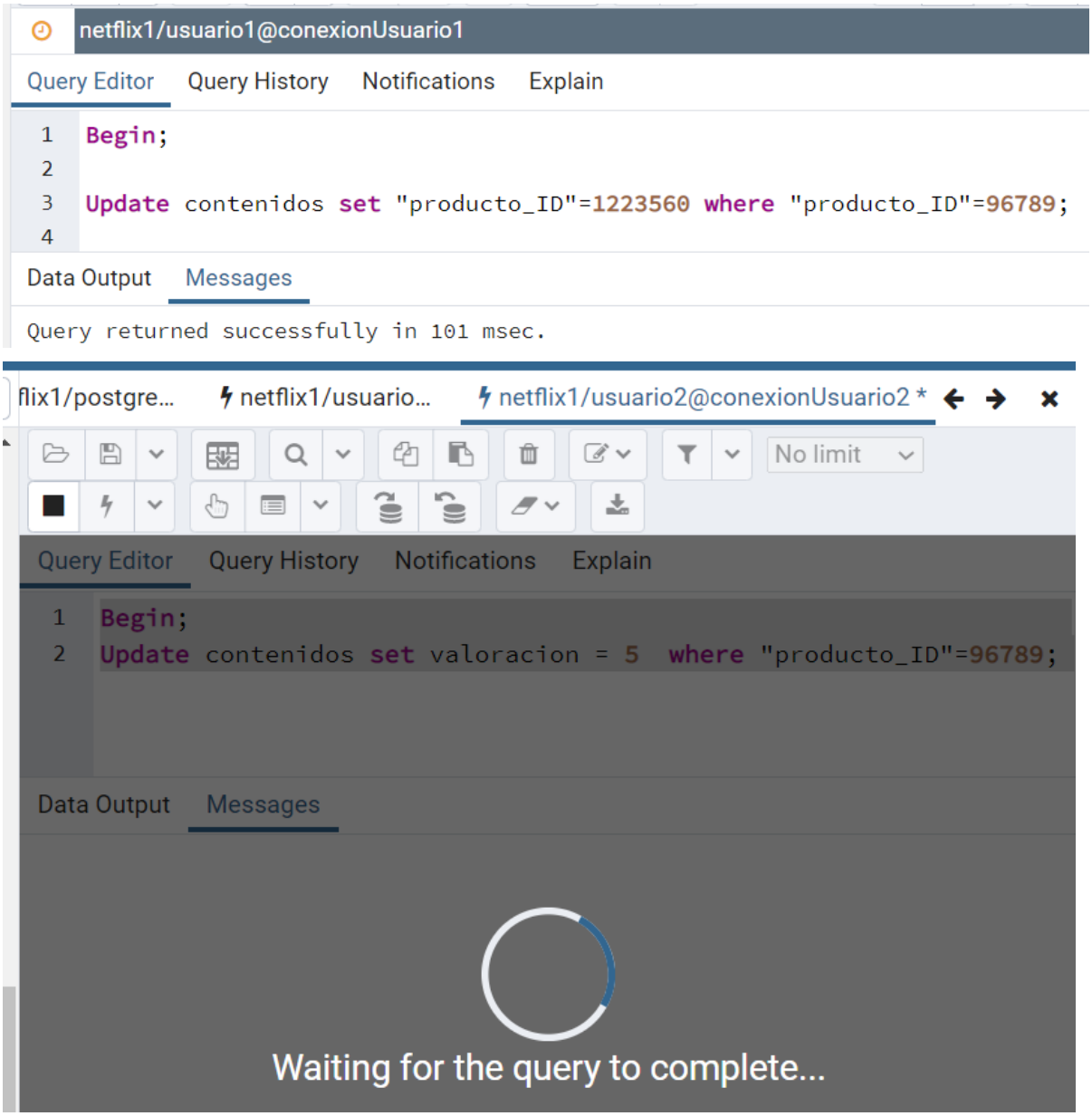
Cuestión 20: Comprometa la transacción T2, ¿Qué es lo que ocurre? ¿Por qué? ¿Cuál es el estado final de la información de la tienda con código 96789? ¿Por qué?

Se desbloquea la T2 y haríamos las mismas consultas que hemos hecho en usuario1 y una vez hecho commit en esta los datos serán apreciables para todos los usuarios.



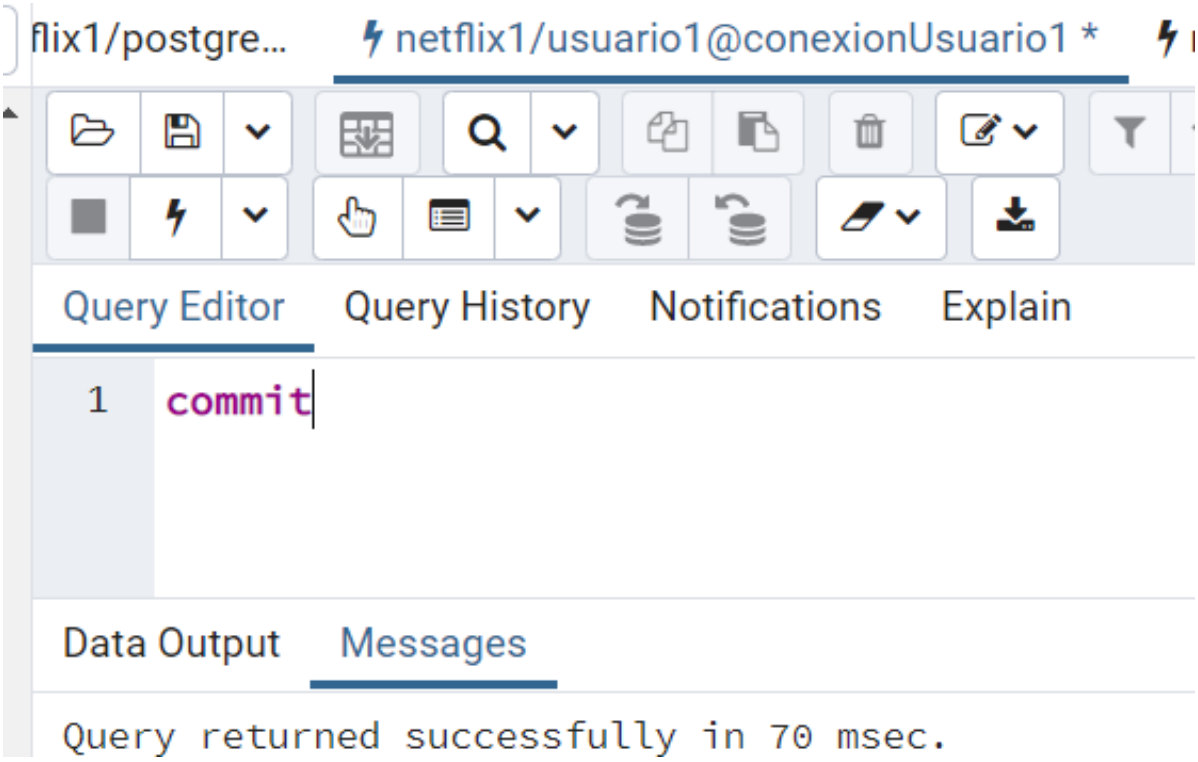
El estado final de la consulta se queda con el título 'The Mandalorian', ya que se ha realizado el commit, y ahora sí se ha guardado en disco.

Cuestión 21: Cerrar todas las sesiones anteriores. Abrir una sesión con el usuario1 de la base de datos **NETFLIX**. Abrir una transacción T1 en este usuario y realizar una modificación del contenido con producto_ID 96789 para cambiar su código a 1223560. Abra otro usuario diferente del anterior y realice una transacción T2 que cambie la valoración del contenido con producto_id 96789 a 8. No cierre la transacción.

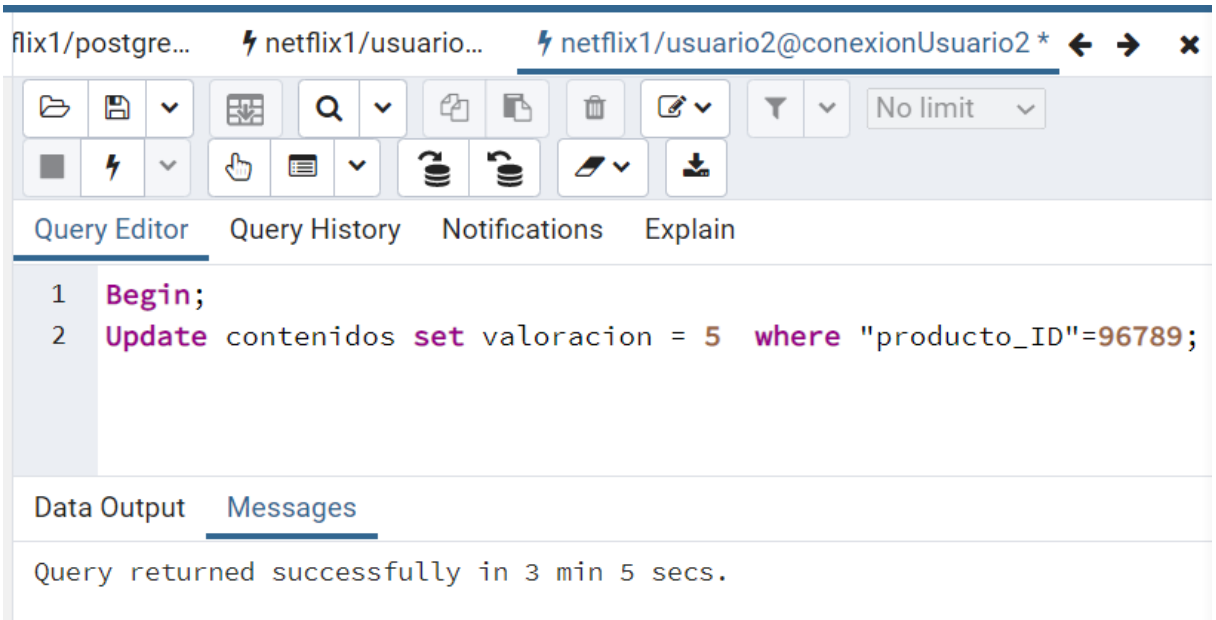


Cuestión 22: Comprometa la transacción T1, ¿Qué es lo que ocurre? ¿Por qué? ¿Cuál es el estado de la información del contenido con producto_ID 96789 para ambos usuarios? ¿Por qué?

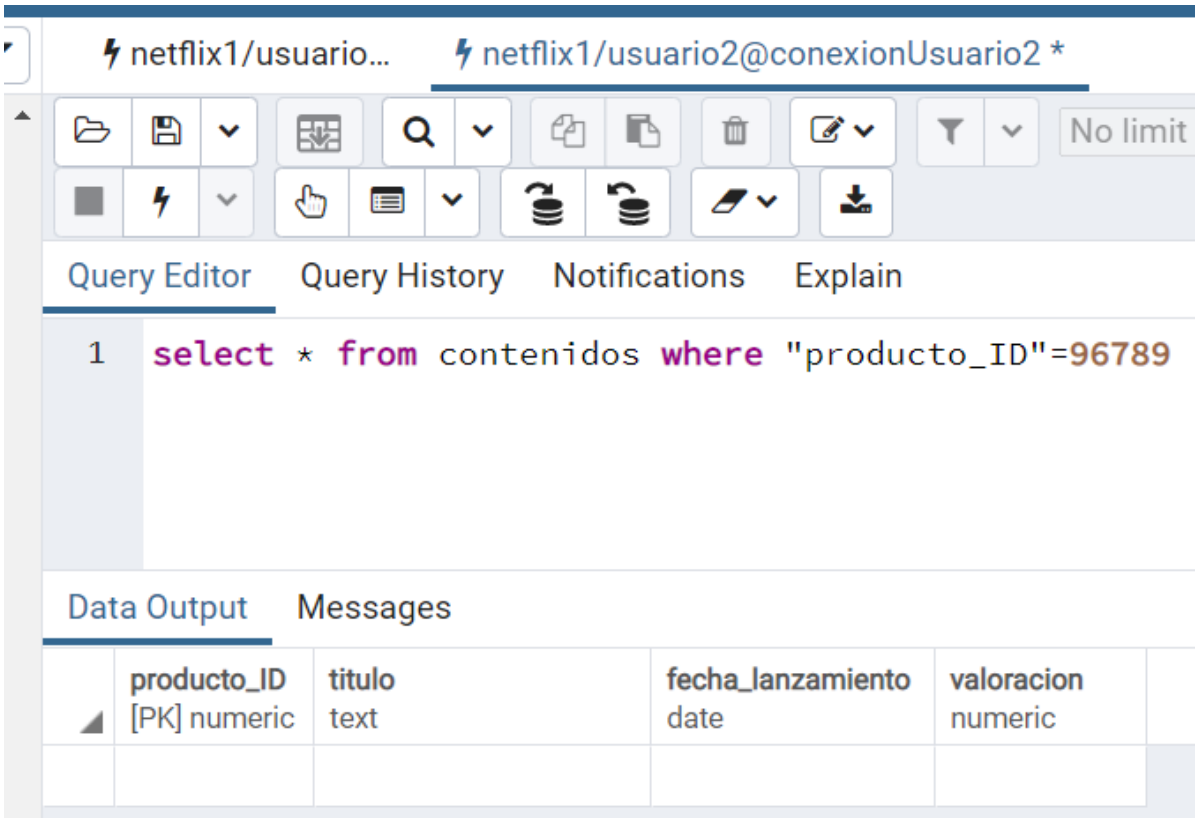
Realizamos un commit en la t1.



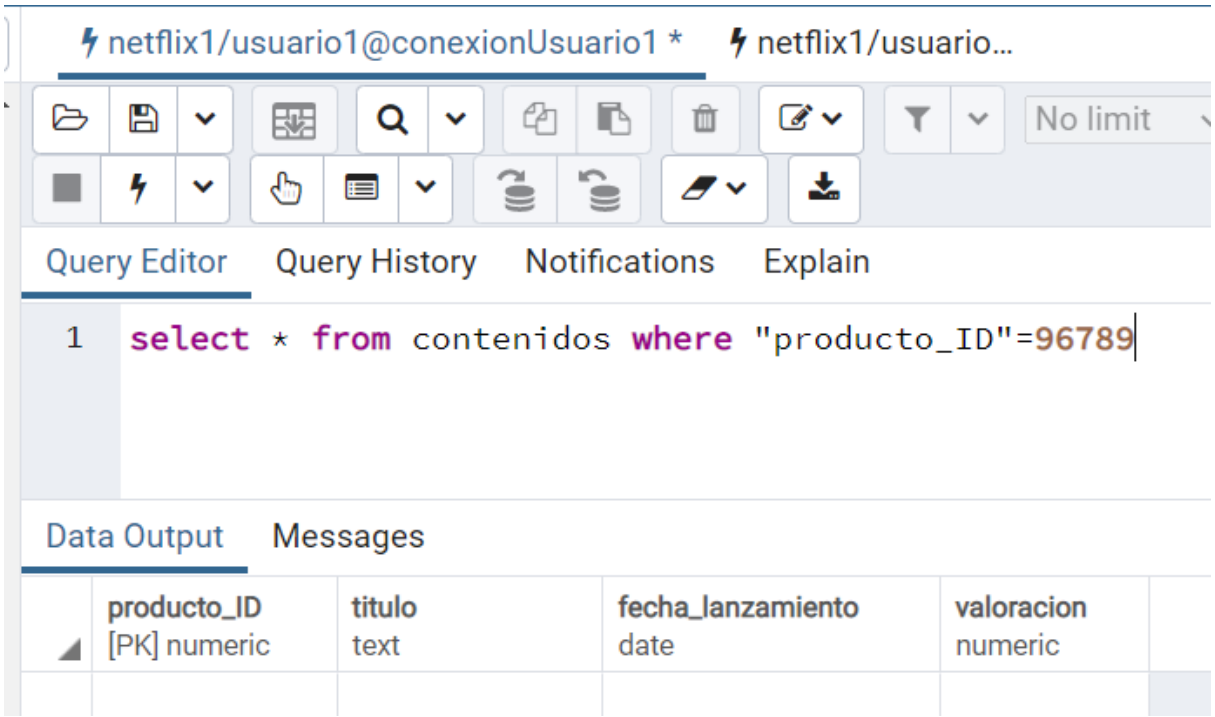
Se ejecuta la transacción t2 automáticamente.



Lo que ocurre es que, al comprometer la t1 se ha ejecutado la transacción y se ha actualizado el valor de la ID para ese contenido, con lo cual, ya no existe un contenido con un ID=96789. Si hacemos una vista desde usuario2, tampoco existe ese contenido con esa ID, ya que previamente se ha actualizado con los valores de t1, y después se ha actualizado con la transacción de t2 (aún no comprometida), pero en la t1 ya dejó de existir el ID 96789, por lo que, la t2 al ir después de esta, tampoco tiene ese contenido.



Cuestión 23: Comprometa la transacción T2, ¿Qué es lo que ocurre? ¿Por qué? ¿Cuál es el estado final de la información del contenido con producto_ID 96789 para ambos usuarios? ¿Por qué?



Haciendo un select con el ID nuevo realizado en la t1, podemos ver que tanto en usuario1 y usuario2 , aparece este resultado:

```
3  select * from contenidos where "producto_ID"=1223560
4
```

Data Output Messages

	producto_ID [PK] numeric	titulo text	fecha_lanzamiento date	valoracion numeric
1	1223560	The Mandalorian	2019-02-23	7

Esto es así puesto que la t2 no se ha llegado a realizar, debido a que previamente hubo un cambio de ID , entonces la cláusula del where de la t2 no se ha ejecutado y sigue estando valoración = 7, en vez de valoración = 5.

Cuestión 24: Cerrar todas las sesiones anteriores. Abrir una sesión con el usuario1 de la base de datos **NETFLIX**. Insertar la siguiente información en la base de datos:

- Insertar un contenido con producto_ID de 34341.
- Insertar una serie que pertenezca al contenido anterior.

netflix1/usuario1@conexionUsuario1

Query Editor Query History Notifications Explain

```
1  Insert into public.contenidos ("producto_ID", titulo, fecha_lanzamiento, valoracion) values
2  (34341, 'titulo34341', '26-06-2019', 8);
3
4
5
```

Data Output Messages

Query returned successfully in 136 msec.

netflix1/usuario1@conexionUsuario1

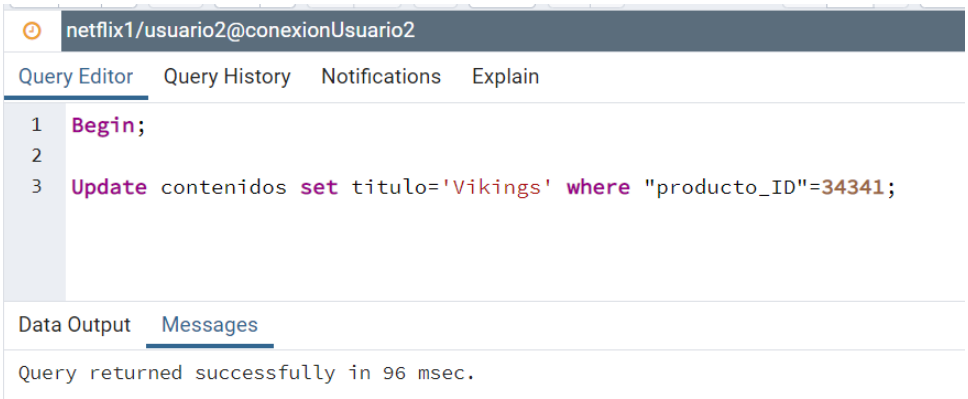
Query Editor Query History Notifications Explain

```
1  Insert into public.series(capitulos,  temporadas, "producto_ID_contenidos")
2  values (4, 2, 34341);
```

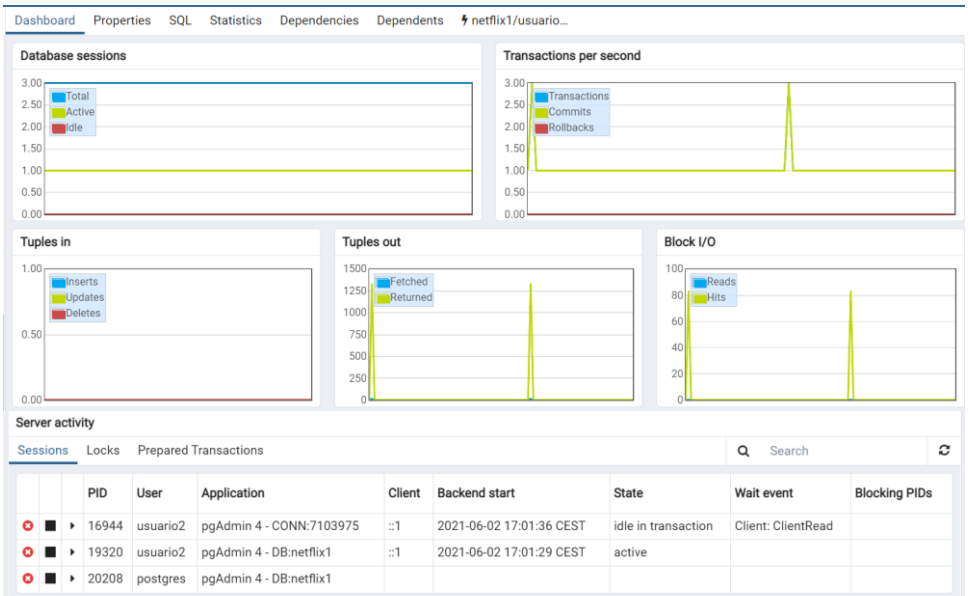
Data Output Messages

Query returned successfully in 88 msec.

Cuestión 25: Abrir una sesión con el usuario2 a la base de datos **NETFLIX**. Abrir una transacción T2 en este usuario2 y realizar una modificación del contenido con producto_ID de 34341 para cambiar el título a “Vikings”. ¿Qué actividad hay registrada en la base de datos? ¿Cuál es la información guardada en la base de datos? ¿Por qué?



Hay 3 conexiones en total, de las cuales 1 está activa.



Desde usuario2 se verán las modificaciones de la base de datos, aunque no se haya hecho commit pero desde las otras conexiones no se verá la última modificación.

Cuestión 26. Abra una transacción T1 en el usuario1. Haga una actualización de la serie con producto_ID 34341 para cambiar las temporadas a 10. ¿Qué actividad hay registrada en la base de datos? ¿Cuál es la información guardada en la base de datos? ¿Por qué?

netflix1/usuario1@conexionUsuario1

Query Editor | Query History | Notifications | Explain

```
1  begin;
2  Update series set temporadas=10 where "producto_ID_contenidos"=34341
3
```

Data Output | Messages

```
Query returned successfully in 94 msec.
```

Hay 5 sesiones en total (1 de postgres que es el superusuario, 2 de usuario1 y 2 de usuario2). Hay una sesión activa que es de usuario1.

Server activity

Sessions | Locks | Prepared Transactions | Search

	PID	User	Application	Client	Backend start	State	Wait event	Blocking PIDs
	16944	usuario2	pgAdmin 4 - CONN:7103975					
	17764	usuario1	pgAdmin 4 - DB:netflix1	::1	2021-06-02 17:07:12 CEST	active		
	19320	usuario2	pgAdmin 4 - DB:netflix1					
	20208	postgres	pgAdmin 4 - DB:netflix1					
	23384	usuario1	pgAdmin 4 - CONN:7115839	::1	2021-06-02 17:07:23 CEST	idle in transaction	Client: ClientRead	

Desde usuario1 se verán las modificaciones de la base de datos de la transacción del usuario1, aunque no se haya hecho commit pero desde las otras conexiones no se verá la última modificación.

Desde usuario2 se aprecia su último update en la tabla contenidos:

netflix1/usuario2@conexionUsuario2

Query Editor | Query History | Notifications | Explain

```
1  select * from contenidos where "producto_ID"=34341
```

Data Output | Messages

	producto_ID [PK] numeric	titulo text	fecha_lanzamiento date	valoracion numeric
1	34341	Vikings	2019-06-26	8

Desde usuario1 el título modificado en la transacción de usuario2 **no** sale actualizado todavía ya que no se ha hecho commit, por lo que no se podrá ver desde otros usuarios:

netflix1/usuario1@conexionUsuario1

Query Editor | Query History | Notifications | Explain

```
1  select * from contenidos where "producto_ID"=34341
2
3
```

Data Output | Messages

	producto_ID [PK] numeric	titulo text	fecha_lanzamiento date	valoracion numeric
1	34341	titulo3...	2019-06-26	8

Desde usuario1 se aprecia la modificación del número de temporadas en la tabla series:

netflix1/usuario1@conexionUsuario1

Query Editor | Query History | Notifications | Explain

```
1
2  select * from series where "producto_ID_contenidos"=34341
3
4
```

Data Output | Messages

	capitulos numeric	temporadas numeric	producto_ID_contenidos [PK] numeric
1	4	10	34341

Desde usuario2 **no** se aprecia la modificación del número de temporadas en la tabla series:

netflix1/usuario2@conexionUsuario2

Query Editor | Query History | Notifications | Explain

```
1 select * from series where "producto_ID_contenidos"=34341
2
3
```

Data Output | Messages

	capitulos numeric	temporadas numeric	producto_ID_contenidos [PK] numeric
1	4	2	34341

En ambos casos, se ha podido realizar la consulta correctamente ya que no se han hecho dos transacciones que utilicen la misma tabla o tupla, por lo tanto, no se producen bloqueos.

Cuestión 27: En la transacción T2, realice una modificación de la serie con producto_ID 34341 para poner capítulos a 40. ¿Qué actividad hay registrada en la base de datos? ¿Cuál es la información guardada en la base de datos? ¿Por qué?

netflix1/usuario2@conexionUsuario2

Query Editor | Query History | Notifications | Explain

```
5
6
7 Update series set capitulos=40 where "producto_ID_contenidos"=34341;
```

Data Output | Messages

Waiting for the query to complete...

La transacción se queda bloqueada, ya que se está realizando una modificación de la misma tabla que en la t1 y esta al tener un bloqueo de tipo "ExclusiveLock" (por la escritura de la nueva tupla) no permite que se modifique hasta realizar un commit. Podemos apreciar este bloqueo en el dashboard:

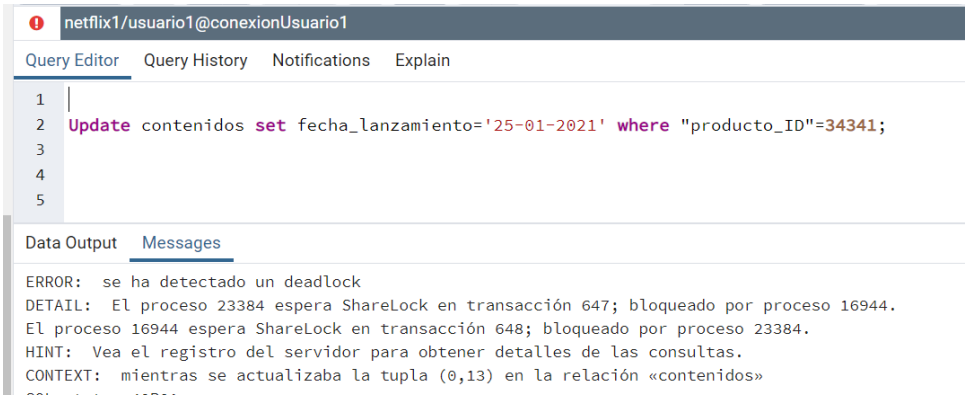
16944	usuario2	pgAdmin 4 - CONN:7103975									23384

16944	tuple	series	0	4					7/1364	ExclusiveLock	true

Se puede apreciar que el usuario2 tiene el bloqueo exclusivo.

Cuestión 28: En la transacción T1, realice una modificación del contenido con producto_ID 34341 para poner la fecha de lanzamiento a 25 de Enero de 2021. ¿Qué actividad hay registrada en la base de datos? ¿Cuál es la información guardada en la base de datos? ¿Por qué?

Update contenidos set fecha_lanzamiento='25-01-2021' where "producto_ID"=34341;

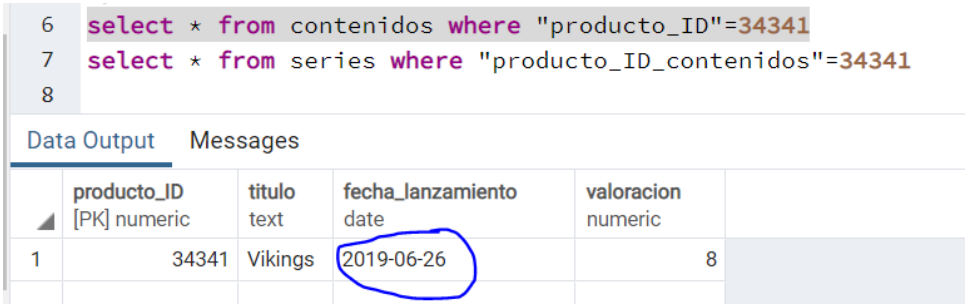


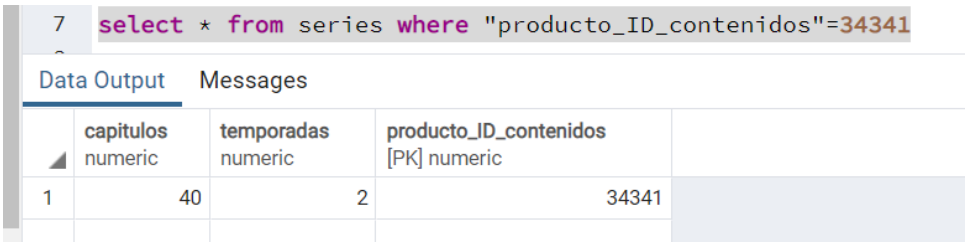
Cuestión 29: Comprometa ambas transacciones T1 y T2. ¿Cuál es el valor final de la información modificada en la base de datos **NETFLIX**? ¿Por qué?

Transacción 1 (usuario1)	Transacción 2 (usuario2)
	Contenidos(Título) = 'Vikings'
Series(temporadas)=10	
	Series(capitulos)=40
Contenidos(fecha_lanzamiento)='25-01-2021'	
Commit	Commit

Como se puede observar se produce un xlock al acceder a la misma tupla que está siendo modificada por T1 y no se ha comprometido. En T1 se hace un deadlock asi que una vez se comprometan ambas no se guardará en la base datos esa modificación.

Finalmente, la tupla de series y contenidos tendrían el siguiente aspecto:





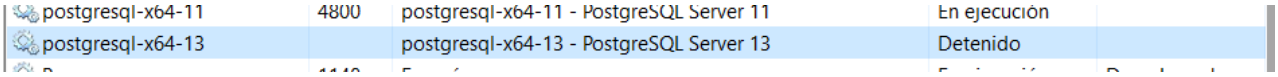
Cuestión 30: Suponer que se produce una pérdida del cluster de datos y se procede a restaurar la instancia de la base de datos del punto 6. Realizar solamente la restauración (recovery) mediante el procedimiento descrito en el apartado 25.3 del

manual (versión, 12) *"Continous Archiving and point-in-time recovery (PITR).* ¿Cuál es el estado final de la base de datos? ¿Por qué?

Lo primero que hemos hecho es apagar el servidor de Postgres. Seguido de esto hemos realizado un backup de la base de datos que teníamos actualmente para evitar de que en caso de que tuviéramos un error a la hora de poner en recuperación la base de datos, no perdiéramos lo que teníamos hecho hasta ahora. Después hemos copiado el backup que realizamos en el ejercicio 6 en el data y hemos creado dentro un archivo de texto vacío llamado "recovery.signal". Como último paso hemos procedido a iniciar de nuevo los servidores de postgres.

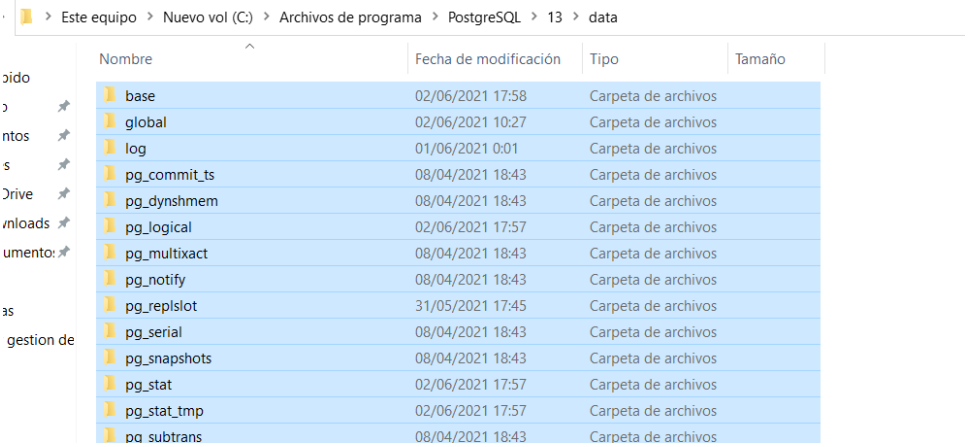
Según el capítulo 25.3 los pasos serían los siguientes:

1. Detener el servidor



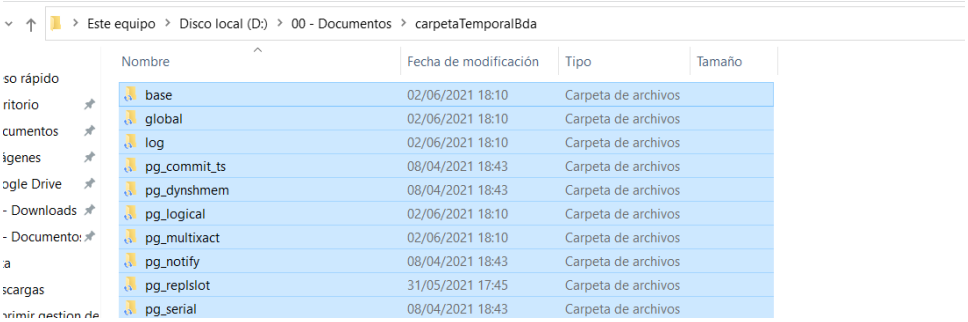
postgresql-x64-11	4800	postgresql-x64-11 - PostgreSQL Server 11	En ejecución
postgresql-x64-13		postgresql-x64-13 - PostgreSQL Server 13	Detenido

2. Si tiene espacio para hacerlo, copie todo el directorio de datos del clúster y cualquier espacio de tabla en una ubicación temporal en caso de que los necesite más adelante. Tenga en cuenta que esta precaución requerirá que tenga suficiente espacio libre en su sistema para contener dos copias de su base de datos existente. Si no tiene suficiente espacio, al menos debe guardar el contenido del pg_wal subdirectorio del clúster , ya que puede contener registros que no se archivaron antes de que el sistema se cayera.



Este equipo > Nuevo vol (C:) > Archivos de programa > PostgreSQL > 13 > data

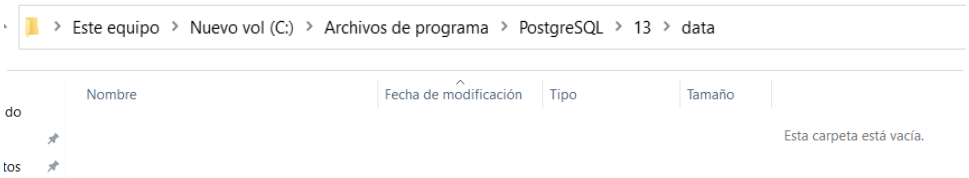
Nombre	Fecha de modificación	Tipo	Tamaño
base	02/06/2021 17:58	Carpeta de archivos	
global	02/06/2021 10:27	Carpeta de archivos	
log	01/06/2021 0:01	Carpeta de archivos	
pg_commit_ts	08/04/2021 18:43	Carpeta de archivos	
pg_dynshmem	08/04/2021 18:43	Carpeta de archivos	
pg_logical	02/06/2021 17:57	Carpeta de archivos	
pg_multixact	08/04/2021 18:43	Carpeta de archivos	
pg_notify	08/04/2021 18:43	Carpeta de archivos	
pg_replslot	31/05/2021 17:45	Carpeta de archivos	
pg_serial	08/04/2021 18:43	Carpeta de archivos	
pg_snapshots	08/04/2021 18:43	Carpeta de archivos	
pg_stat	02/06/2021 17:57	Carpeta de archivos	
pg_stat_tmp	02/06/2021 17:57	Carpeta de archivos	
pg_subtrans	08/04/2021 18:43	Carpeta de archivos	



Este equipo > Disco local (D:) > 00 - Documentos > carpetaTemporalBda

Nombre	Fecha de modificación	Tipo	Tamaño
base	02/06/2021 18:10	Carpeta de archivos	
global	02/06/2021 18:10	Carpeta de archivos	
log	02/06/2021 18:10	Carpeta de archivos	
pg_commit_ts	08/04/2021 18:43	Carpeta de archivos	
pg_dynshmem	08/04/2021 18:43	Carpeta de archivos	
pg_logical	02/06/2021 18:10	Carpeta de archivos	
pg_multixact	02/06/2021 18:10	Carpeta de archivos	
pg_notify	08/04/2021 18:43	Carpeta de archivos	
pg_replslot	31/05/2021 17:45	Carpeta de archivos	
pg_serial	08/04/2021 18:43	Carpeta de archivos	

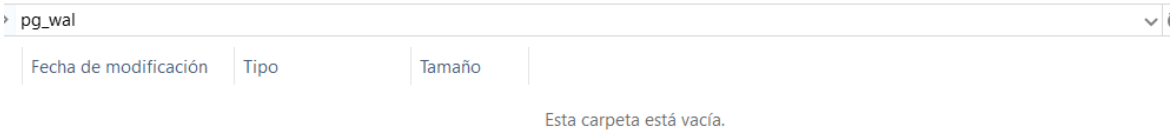
3. Elimine todos los archivos y subdirectorios existentes en el directorio de datos del clúster y en los directorios raíz de cualquier espacio de tabla que esté utilizando.



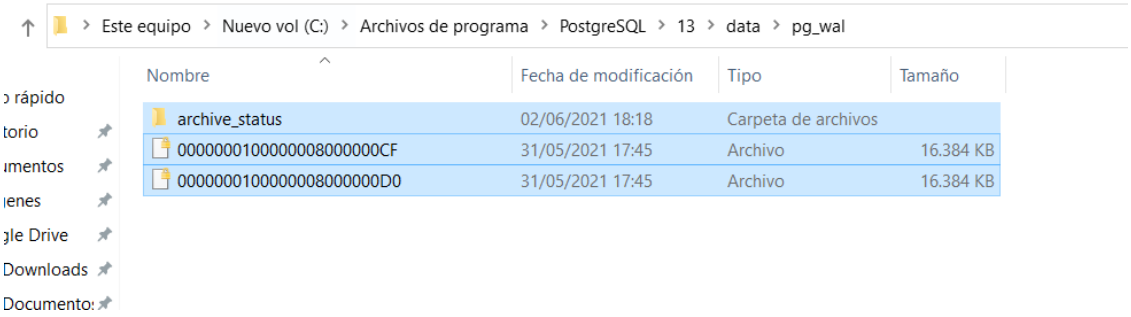
4. Restaure los archivos de la base de datos desde la copia de seguridad de su sistema de archivos. Asegúrese de que se restauran con la propiedad correcta (el usuario del sistema de base de datos, rooti no !) Y con los permisos correctos. Si está utilizando espacios de tabla, debe verificar que los enlaces simbólicos se pg_tblspc/ hayan restaurado correctamente.

Se ha realizado correctamente la copia de los archivos del backup .

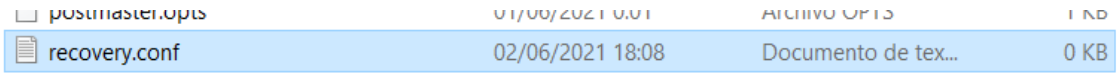
5. Eliminar cualquier archivo presente en pg_wal/; estos provienen de la copia de seguridad del sistema de archivos y, por lo tanto, probablemente sean obsoletos en lugar de actuales. Si no archivó pg_wal/en absoluto, luego vuelva a crearlo con los permisos adecuados, teniendo cuidado de asegurarse de restablecerlo como un enlace simbólico si lo configuró de esa manera antes.



6. Si tiene archivos de segmento WAL sin archivar que guardó en el paso 2, cópielos en pg_wal/. (Es mejor copiarlos, no moverlos, de modo que aún tenga los archivos sin modificar si ocurre un problema y tiene que comenzar de nuevo).



7. Cree un archivo de comando de recuperación recovery.conf en el directorio de datos del clúster (consulte el Capítulo 27). También es posible que desee modificar temporalmente pg_hba.confpara evitar que los usuarios comunes se conecten hasta que esté seguro de que la recuperación fue exitosa.



Iniciamos el servidor. Su estado ha sido recuperado y por tanto la base de datos está recuperada exitosamente.

8. El estado final de la base de datos es igual al que fue en su momento en el ejercicio 6.

Cuestión 31: A la vista de los resultados obtenidos en las cuestiones anteriores, ¿Qué tipo de sistema de recuperación tiene implementado postgreSQL? ¿Qué protocolo de gestión de la concurrencia tiene implementado? ¿Por qué? ¿Genera siempre planificaciones secuenciables? ¿Genera siempre planificaciones recuperables? ¿Tiene rollbacks en cascada? Justificar las respuestas.

El sistema de recuperación que posee es diferida puesto que no se aplican los cambios a la memoria global hasta que no haya un commit.

PostgreSQL utiliza MVCC, control de concurrencia multiversión, con la que evita bloqueos innecesarios y le permite manejar los registros sin que los usuarios tengan que esperar a que estén disponibles. Con MVCC los datos antiguos no se pierden aunque se sobrescriban porque se guardan las versiones de estos.

Genera planificaciones secuenciables porque para dos transacciones Ti Tj, que se refieran al mismo elemento de datos, hay que tener en cuenta el orden de aparación. En caso de que se refieran a distintos elementos, se pueden intercambiar entre ellas.

Las planificaciones también recuperables ya que si una Ti leyese datos de Tj, Tj se comprometería antes que Ti.

También hemos visto que se produce rollback en cascada ya que en cuanto cometemos un error en la transacción nos obliga a realizar rollback y comenzar de nuevo la transacción.

Bibliografía

- Capítulo 13: Concurrency Control.
- Capítulo 19: Server Configuration.
- Capítulo 25: Backup and Restore.
- Capítulo 27: Monitoring Database Activity.
- Capítulo 29: Reliability and the Write-Ahead log.