

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 4: Replicación e Implementación de una Base de Datos Distribuida.

ALUMNO 1:

Nombre y Apellidos: Alejandro Hernández Martín

DNI: 09074363n

ALUMNO 2:

Nombre y Apellidos: Noelia Marca Retamozo

DNI: 09048809b

Fecha: 05/06/2021

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 un pequeño tutorial de la implementación de la replicación y la base de datos distribuida, las pruebas realizadas de su funcionamiento; y los ficheros de configuración del maestro y del esclavo utilizados en replicación; y de la configuración de los servidores de la base de datos

distribuida. También los ficheros de log generados en la realización de la práctica de los postgres involucrados. Fichero a subir DNI's de los Alumnos_PECL4.zip

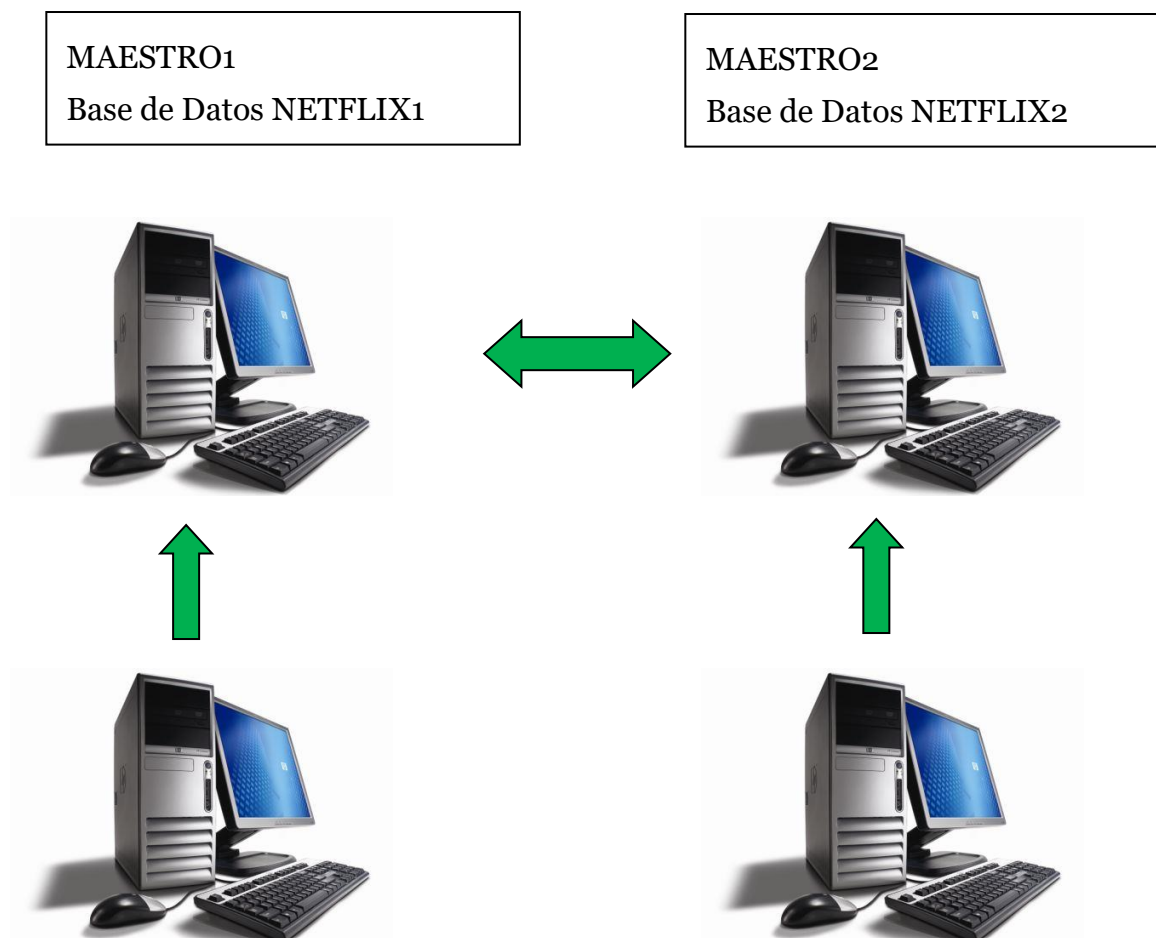
AMBOS ALUMNOS DEBEN ENTREGAR EL FICHERO EN LA PLATAFORMA.

Introducción

El contenido de esta práctica versa sobre la Replicación de Bases de Datos con PostgreSQL e introducción a las bases de datos distribuidas. Concretamente se va a utilizar los servicios de replicación de bases de datos que tiene PostgreSQL. Para ello se utilizará PostgreSQL 13.x con soporte para replicación. **Se prohíbe el uso de cualquier otro programa externo a PostgreSQL para realizar la replicación, como puede ser Slony.**

También se va a diseñar e implementar una pequeña base de datos distribuida. Una base de datos distribuida es una base de datos lógica compuesta por varios nodos (equipos) situados en un lugar determinado, cuyos datos almacenados son diferentes; pero que todos ellos forman una base de datos lógica. Generalmente, los datos se reparten entre los nodos dependiendo de donde se utilizan más frecuentemente.

El escenario que se pretende realizar se muestra en el siguiente esquema:



ESCLAVO1

Replicando NETFLIX1

ESCLAVO2

Replicando NETFLIX2

Se van a necesitar 4 máquinas: 2 maestros y 2 esclavos. Cada maestro puede ser un ordenador de cada miembro del grupo con una base de datos de NETFLIX en concreto (NETFLIX1 y NETFLIX2). Dentro de cada maestro se puede instalar una máquina virtual, que se corresponderá con el esclavo que se encarga de replicar la base de datos que tiene cada maestro, es decir, hace una copia o backup continuo de la base de datos NETFLIX1 o de la base de datos NETFLIX2. También es posible no usar otra máquina virtual para el esclavo y levantar otra estancia de postgres para guardar la réplica en un clúster diferente. Hay más de una solución.

Se debe de entregar una memoria descriptiva detallada que posea como mínimo los siguientes puntos:

En el ordenador de Alejandro se encuentra el Maestro1 y su esclavo (mediante una máquina virtual de VirtualBox en Windows 10).

En el ordenador de Noelia se encuentra el Maestro2 y su esclavo (mediante una máquina virtual de VirtualBox en Windows 10).

1. 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
```

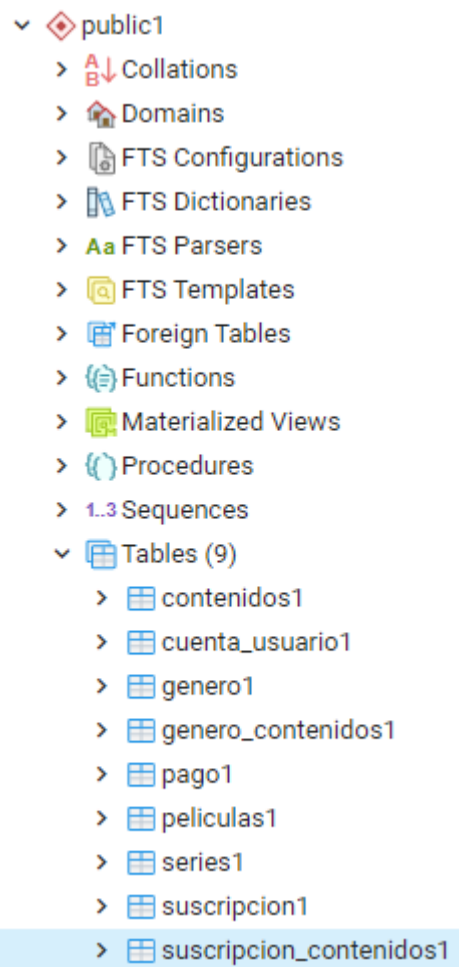
```
log_line_prefix = '%m [%p] %u %r'
```

2. Configuración de cada uno de los nodos maestros de la base de datos de NETFLIX1 y NETFLIX2 para que se puedan recibir y realizar consultas sobre la base de datos que no tienen implementadas localmente.

El Maestro1 se encuentra en el portátil de Alejandro y contendrá parte de la base datos localmente y que será compartida con el Maestro2.

MAESTRO 1:

Se ha cambiado el nombre de las tablas, del esquema y de la base de datos añadiendo un "1" al final para que quede más claro y se sepa que es del Maestro1.



En **pg_hba.conf** se han modificado los permisos para dar acceso a la base de datos de todas las conexiones locales.

#	TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only					
local	all		all		scram-sha-256
# IPv4 local connections:					
host	all		all	127.0.0.1/32	scram-sha-256
host	all		all	0.0.0.0/0	trust
# IPv6 local connections:					
host	all		all	:::1/128	scram-sha-256
host	all		all	0.0.0.0/0	trust

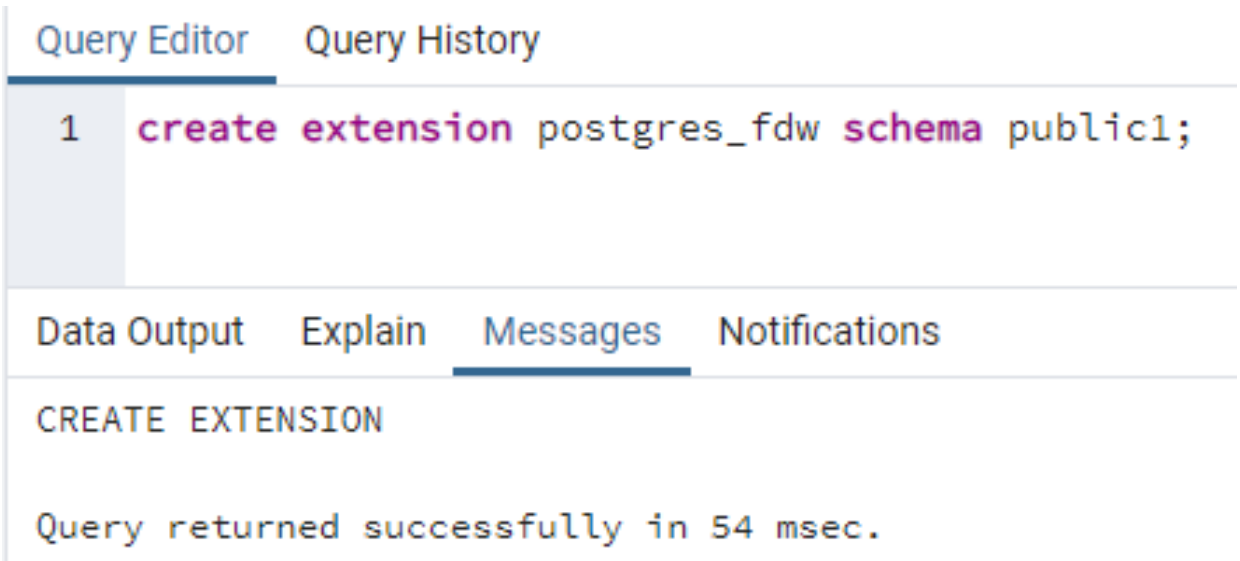
En **postgres.conf** se ha modificado para que escuche todas las IP's. Al estar en una red local no es posible correr dos servidores PostgreSQL diferentes en el mismo puerto, así que los puertos de cada servidor serán diferentes. En el caso del servidor del Maestro1 su puerto es el 5433. Finalmente nos vamos al administrador de tareas y reiniciamos Postgres.

- Connection Settings -

```
listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (change requires restart)
port = 5433                     # (change requires restart)
```

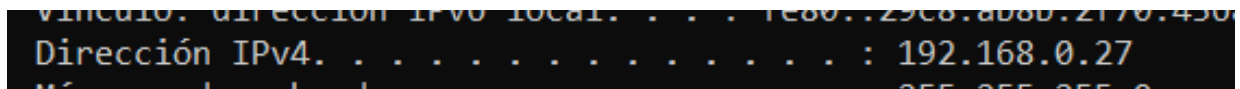
El siguiente paso es usar la extensión **postgres_fdw** que viene explicada en “Appendix F: Additional Supplied Modules. F.33.” en el manual.

create extension postgres_fdw schema public1



The screenshot shows a web-based PostgreSQL interface. At the top, there are tabs for 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, showing a single query: `1 create extension postgres_fdw schema public1;`. Below the query editor, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Messages' tab is active, displaying the message: `CREATE EXTENSION` and `Query returned successfully in 54 msec.`

Ahora crearemos el servidor del que nos copiaremos los datos de su base de datos, es decir, del nodo Maestro 2. La IP del Maestro2 (Noelia) es la siguiente:



The screenshot shows a network configuration interface. It lists several network interfaces. The 'Dirección IPv4' (IPv4 Address) for the selected interface is shown as `192.168.0.27`.

Ahora conectaremos el Maestro1 al Maestro2 gracias a un servidor extranjero que recibe la información del Maestro2 ubicada en 192.168.0.27, puerto 5435 y el nombre de la base de datos es 'netflix2'. Hay que asegurarse de que el firewall del Maestro2 no esté bloqueando los puertos de red ya que si está activado no podremos conectarnos correctamente al otro nodo.

```
1 Create server Maestro2 foreign data wrapper postgres_fdw options (host '192.168.0.27', port '5435', dbname 'netflix2')
```

Más tarde, se mapea un usuario local en la base de datos extranjera. Al haber configurado todas las IPs como trust, no hace falta contraseña.

Create user mapping for postgres server Maestro2 options (user 'postgres')

```
1 create user mapping for postgres server Maestro2 options (user 'postgres')
```

Importamos el schema del Maestro2 en nuestra bbdd:

```
1 import foreign schema public2 from server maestro2 into public1
```

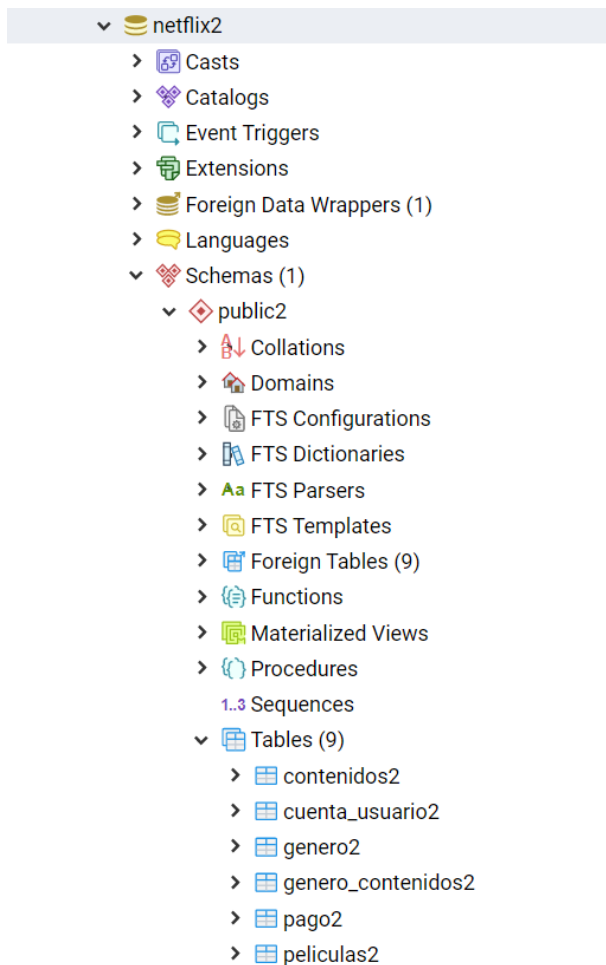
Para comprobar que se ha conectado correctamente el Maestro1 a la base de datos del nodo Maestro2 se ha hecho un select de la tabla “genero2” y se han mostrado satisfactoriamente los datos que se habían importado en Maestro2 previamente.

The screenshot shows a database management interface. On the left, a tree view displays the database schema. Under 'Schemas (1)', the 'public1' schema is expanded, showing various database objects. The 'Foreign Tables (9)' section is highlighted, and the 'genero2' table is selected. On the right, the 'Query Editor' shows a SQL query: `SELECT * FROM public1.genero2 LIMIT 100`. Below the query editor, the 'Data Output' tab displays the results of the query in a table format.

	genero_ID numeric	descripcion text
1	0	Acción
2	1	Aventuras
3	2	Comedia
4	3	Drama
5	4	Terror
6	5	Musical
7	6	Ciencia ficcion
8	7	Guerra
9	8	Acutal
10	9	Antigua
11	10	Estreno
12	11	Clasico
13	12	Muda

MAESTRO 2:

Se ha cambiado el nombre de las tablas, del esquema y de la base de datos añadiendo un “2” al final para que quede más claro y se sepa que es del Maestro2.



En **pg_hba.conf** del maestro2 se han modificado los permisos para dar acceso a la base de datos de todas las conexiones locales.

```
80
81 # TYPE  DATABASE        USER            ADDRESS                 METHOD
82
83 # "local" is for Unix domain socket connections only
84 local    all             all                                     scram-sha-256
85
86 # IPv4 local connections:
87 host     all             all             127.0.0.1/32            scram-sha-256
88 host     all             all             0.0.0.0/0                trust
89 # IPv6 local connections:
90 host     all             all             ::1/128                 scram-sha-256
91 host     all             all             0.0.0.0/0                trust
```

En **postgres.conf** se ha modificado para que escuche todas las IP's. Al estar en una red local no es posible correr dos servidores PostgreSQL diferentes en el mismo puerto, así que los puertos de cada servidor serán diferentes. En el caso del servidor del Maestro1 su puerto es el 5435. Finalmente nos vamos al administrador de tareas y reiniciamos Postgres.

```

7  # - Connection Settings -
8
9  listen_addresses = '*'      # what IP address(es) to listen on;
10                             # comma-separated list of addresses;
11                             # defaults to 'localhost'; use '*' for all
12                             # (change requires restart)
13  port = 5435 # (change requires restart)

```

Usaremos la extensión **postgres_fdw** :

create extension postgres_fdw schema public2

```

1  create extension postgres_fdw schema public2;

```

Data Output Messages

Query returned successfully in 74 msec.

Ahora crearemos el servidor del que nos copiaremos los datos de su base de datos, es decir, del nodo Maestro 1. La IP del Maestro2 (Alejandro) es la siguiente:

Dirección IPv4. : 192.168.0.16

Ahora conectaremos el Maestro2 al Maestro1 gracias a un servidor extranjero que recibe la información del Maestro1 ubicada en 192.168.0.16, puerto 5433 y el nombre de la base de datos es 'netflix1'. Hay que asegurarse de que el firewall del Maestro1 no esté bloqueando los puertos de red ya que si está activado no podremos conectarnos correctamente al otro nodo.

```

1  create server Maestro1 foreign data wrapper postgres_fdw options (host '192.168.0.16', port '5433', dbname 'postgres')

```

Data Output Messages

Query returned successfully in 74 msec.

Más tarde, se mapea un usuario local en la base de datos extranjera. Al haber configurado todas las IPs como trust, no hace falta contraseña.

Create user mapping for postgres server Maestro1 options (user 'postgres')

```

1  create user mapping for postgres server Maestro1 options (user 'postgres')

```

Data Output Messages

Query returned successfully in 102 msec.

Importamos el schema del Maestro1 en nuestra bbdd:

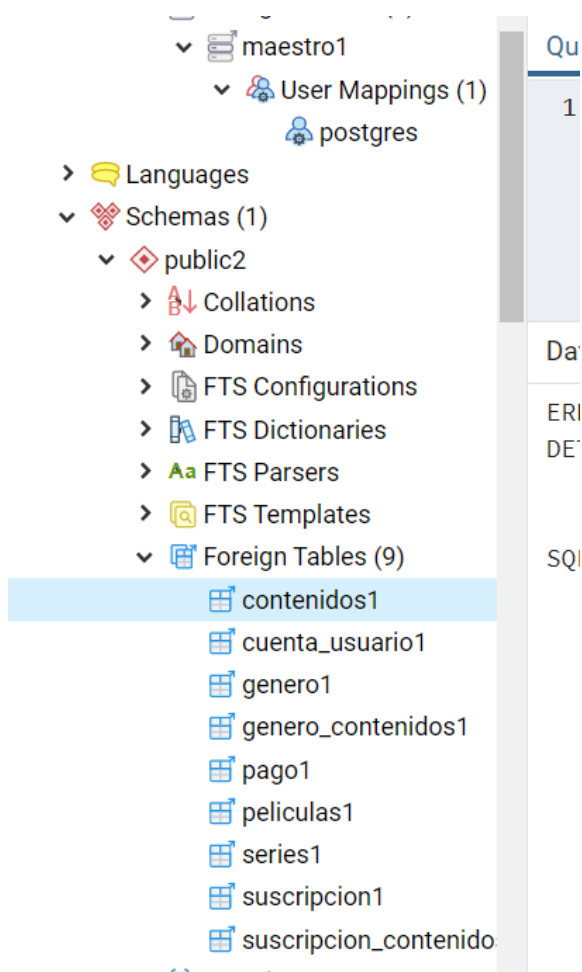

```
Query Editor  Query History  Notifications  Explain

1  import foreign schema public1 from server maestro1 into public2

Data Output  Messages

Query returned successfully in 1 min 2 secs.
```

Podemos observar que las bases de datos se han conectado correctamente. Las foreign tables son las del maestro1 y aparecen todas las tablas de su base de datos.



3. Configuración completa de los equipos para estar en modo de replicación.

Respondido junto a pregunta 4

4. Configuración del nodo maestro. Tipos de nodos maestros, diferencias en el modo de funcionamiento y tipo elegido. Tipos de nodos esclavos, diferencias en el modo de funcionamiento y tipo elegido, etc.

Entre todos los tipos de soluciones que nos ofrece Postgres para implantar la replicación entre servidores hemos elegido **WAL shipping**. Con esto el servidor Esclavo puede replicar continuamente el servidor Maestro. En caso de que sufra un

fallo el servidor principal, el servidor Esclavo contaría con la mayoría de sus datos y se los pasaría.

Este sistema consiste en realizar un backup del servidor Maestro al servidor Esclavo y enviarle los logs del Maestro al Esclavo. El resultado será que el Esclavo se encontrará continuamente sincronizado con el Maestro.

Puerto de cada nodo:

- Maestro1: 5433
- Esclavo1: 5434
- Maestro2: 5435
- Esclavo2: 5436

Configuración de Maestro1 y Maestro2:

En ambos nodos la configuración será igual, tanto en pg_hba como en postgres.conf (lo único que es distinto es el puerto).

Ponemos el wal_level a replica porque wal_level determina cuánta información se escribe en el WAL y replica, lo que hará será escribir suficientes datos para admitir el archivado y la replicación WAL, incluida la ejecución de consultas de solo lectura en un servidor en espera.

```
# - Settings -  
wal_level = replica  
#
```

PostgreSQL recomienda que el número de sincronizaciones concurrentes tenga asignado un numero alto:

```
max_wal_senders = 16
```

Especifica el número mínimo de segmentos de archivos de registro anteriores que se mantienen en el directorio pg_xlog, en caso de que un servidor en espera necesite recuperarlos para la replicación de transmisión.

```
wal_keep_size = 0 # in megabytes; 0 disables
```

Normalmente no hay necesidad de ajustar wal_keep_size, ya que no es una manera fiable de asegurar que todos los segmentos de WAL necesarios estén disponibles a recursos seguros.

En el pg_hba se ha realizado la siguiente modificación para dar permiso para la replicación de la base de datos a cualquier usuario desde cualquier parte de la red y sin necesidad de autenticarse:

```
# Allow replication connections from localhost, by a user with the  
# replication privilege.  
local    replication    all                                     scram-sha-256  
host     replication    all          127.0.0.1/32          scram-sha-256  
host     replication    all          ::1/128              scram-sha-256  
host     replication    all          0.0.0.0/0            trust
```

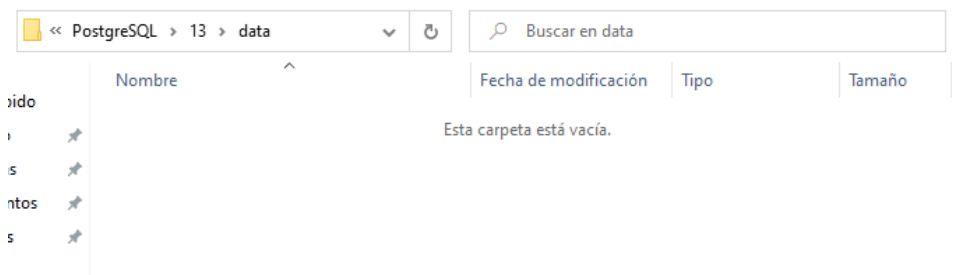
Configuración de Esclavo1 y Esclavo2:

Para conectar los esclavos a sus maestros hay varios pasos:

1. Pararemos el servidor del esclavo:

```
C:\Archivos de programa\PostgreSQL\13\bin>pg_ctl stop -D ../data
esperando que el servidor se detenga.... listo
servidor detenido
```

2. Borramos todo lo que haya dentro de la carpeta "data".



3. Realizamos el basebackup del nodo Maestro en su respectivo Esclavo.

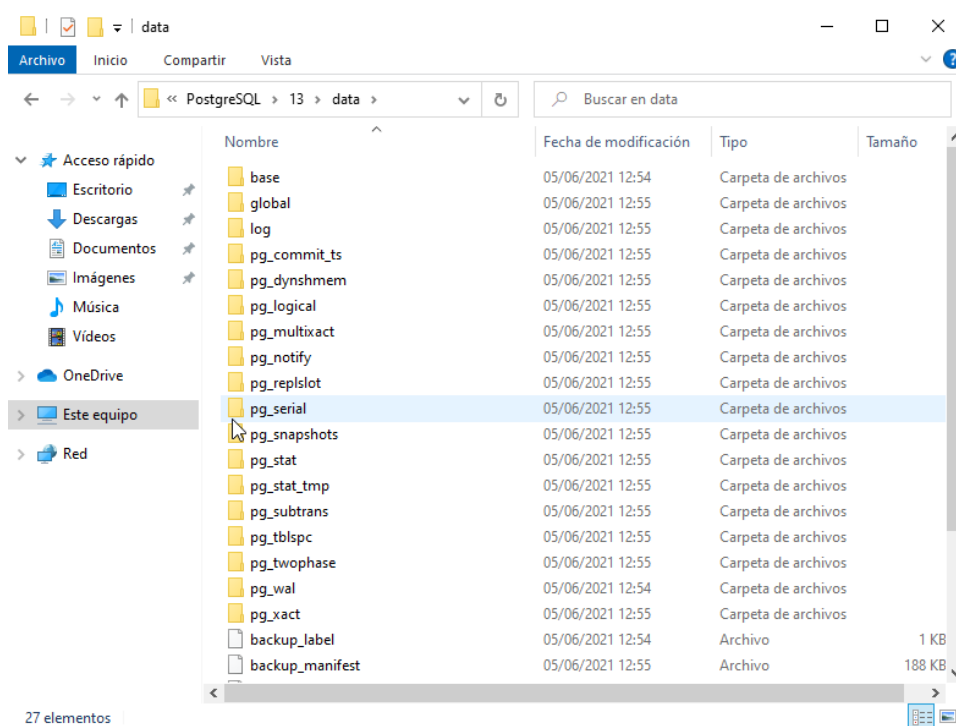
Esclavo1:

```
C:\Archivos de programa\PostgreSQL\13\bin>pg_basebackup -h 192.168.0.16 -U postgres -p 5433 -D ../data -Fp -Xs -P -R
1797367/1797367 kB (100%), 1/1 tablespace
```

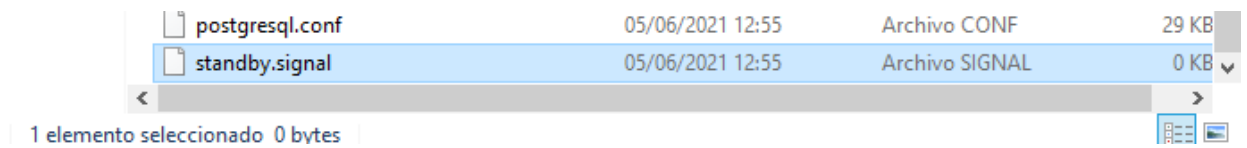
Esclavo2:

```
C:\Program Files\PostgreSQL\13\bin>pg_basebackup -h 192.168.0.27 -U postgres -p 5435 -D ../data -Fp -Xs -P -R
1667153/1667153 kB (100%), 1/1 tablespace
```

Como se puede apreciar en la imagen, se ha realizado correctamente el pg_basebackup y contiene los archivos del maestro.



Se crea automáticamente el archivo standby.signal que sirve para indicar al servidor que debe iniciarse en modo esclavo (standby).



4. Dentro de postgres.conf debemos indicar la información del servidor primario.

Esclavo1:

```
primary_conninfo = 'host=192.168.0.16 port=5433 dbname=netflix1'
```

Esclavo2:

```
primary_conninfo = 'host=192.168.0.27 port=5435 dbname=netflix2'
```

5. Para que el esclavo pueda realizar consultas desde el servidor Esclavo es necesario realizar la siguiente modificación en postgres.conf:

```
hot_standby = on
```

6. Ya podemos volver a iniciar el servidor:

```
C:\Archivos de programa\PostgreSQL\13\bin>pg_ctl start -D ../data
esperando que el servidor se inicie...2021-06-05 13:11:26.433 CEST [3512] LOG:  redirigiendo la salida del registro al
proceso recolector de registro
2021-06-05 13:11:26.433 CEST [3512] HINT:  La salida futura del registro aparecerá en el directorio «log».
.. listo
servidor iniciado
```

Si vamos al dashboard podemos observar que hay procesos relacionados con que el servidor está recibiendo el WAL del servidor maestro.

		PID	Database	User	Application	Client	Backend start	State	Wait event
✖	■ ▶	3984	postgres	postgres	pgAdmin 4 - DB:postgres	::1	2021-06-05 13:21:03 CEST	active	
✖	■ ▶	4364					2021-06-05 13:20:51 CEST		Activity: RecoveryWalStream
✖	■ ▶	4392					2021-06-05 13:20:53 CEST		Activity: WalReceiverMain

Se puede observar en la siguiente imagen que el servidor maestro está enviando desde mi la IP del Maestro2 (192.168.0.27) al servidor Esclavo2.

Sessions

Locks

Prepared Transactions

Configuration

Q

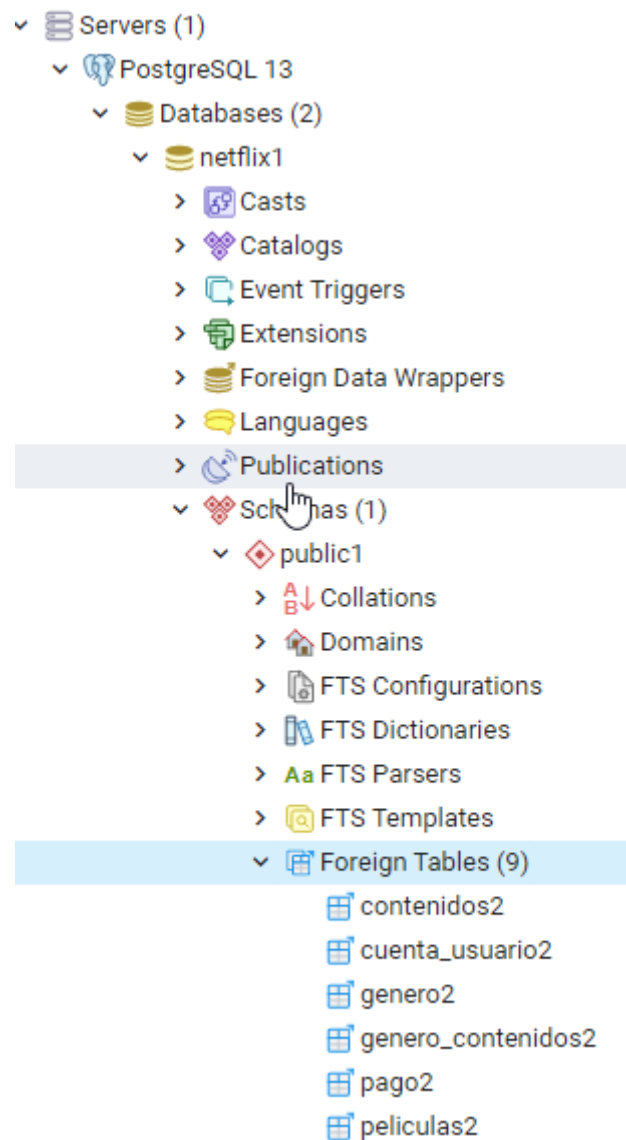
Search

		PID	Database	User	Application	Client	Backend start	State	Wait event
		3056		postgres	walreceiver	192.168.0.27	2021-06-05 13:12:29 CEST	active	Activity: WalSenderMain

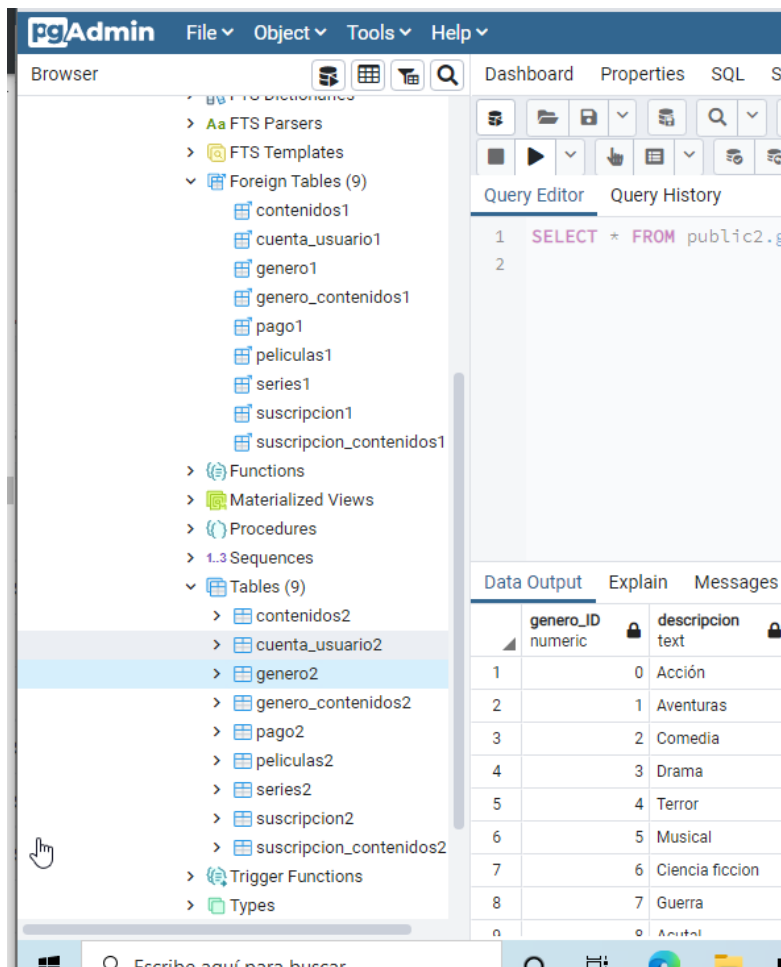
Y desde el servidor Esclavo2 podemos ver que se están recibiendo los wal del servidor Maestro2:

✖	■ ▶	7520					2021-06-05 13:20:09 CEST		Activity: WalReceiverMain
---	-----	------	--	--	--	--	--------------------------	--	---------------------------

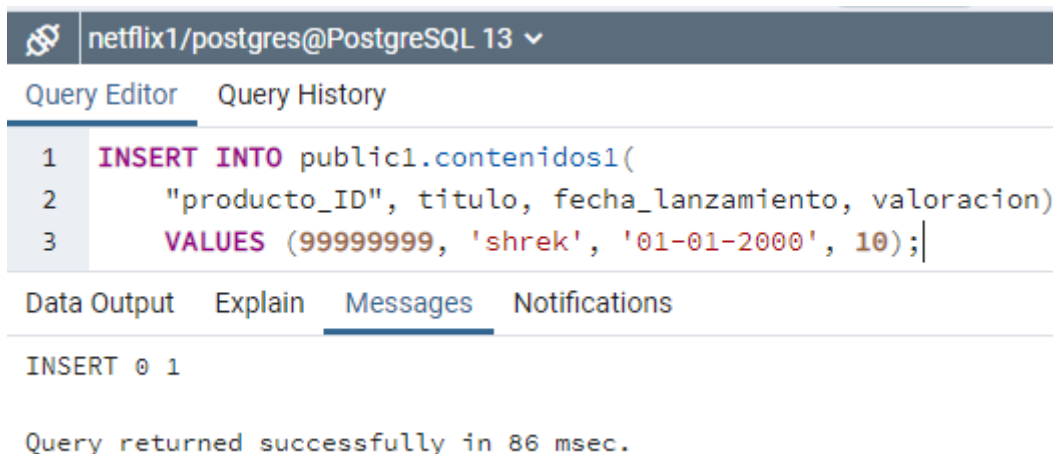
La estructura de del Maestro1 se ha replicado correctamente en el Esclavo1:



La estructura del Maestro2 se ha replicado correctamente:



Si se inserta un producto por ejemplo desde el servidor Maestro1, se puede ver reflejado en ambos servidores esclavos (ya que se han habilitado las consultas desde estos) y desde el Maestro2.



Haciendo la consulta **desde esclavo2**, podemos observar que aparece la nueva tupla insertada en el maestro1:

Query Editor Query History Scratch Pad

```
1 select * from public2.contenidos1 where titulo = 'shrek'
```

Data Output Explain Messages Notifications

	producto_ID numeric	titulo text	fecha_lanzamiento date	valoracion numeric
1	99999999	shrek	2000-01-01	10

Haciendo la consulta **desde esclavo1**, podemos observar que aparece la nueva tupla insertada en el maestro1:

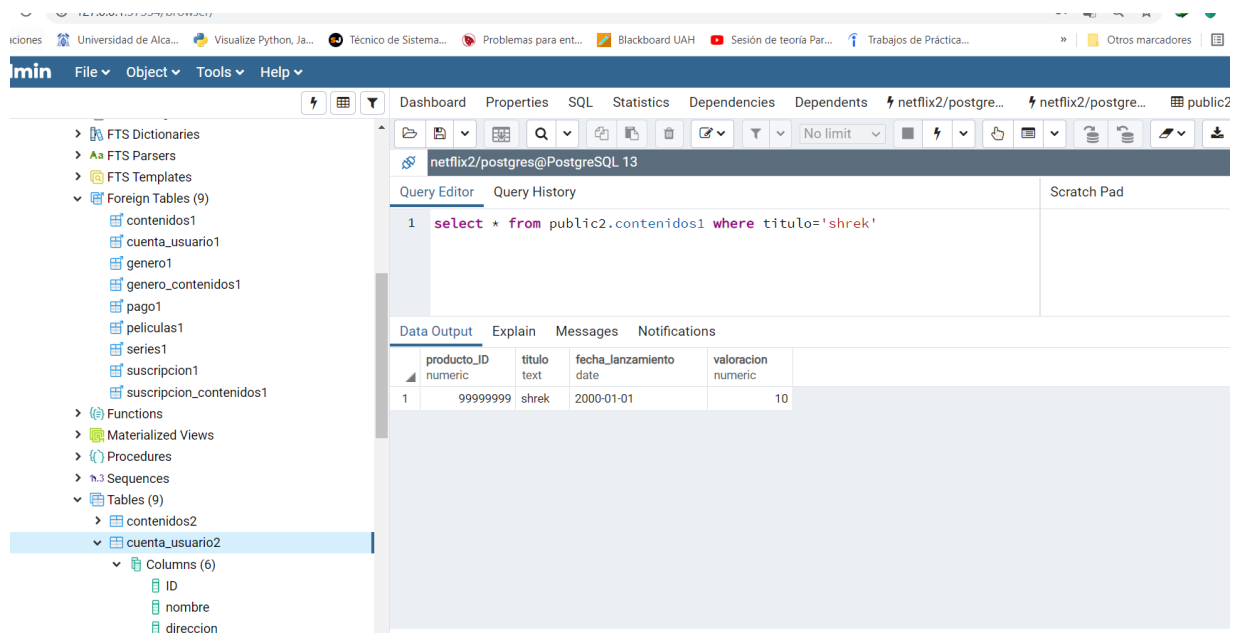
Query Editor Query History

```
1 select * from public1.contenidos1 where titulo = 'shrek'
```

Data Output Explain Messages Notifications

	producto_ID [PK] numeric	titulo text	fecha_lanzamiento date	valoracion numeric
1	99999999	shrek	2000-01-01	10

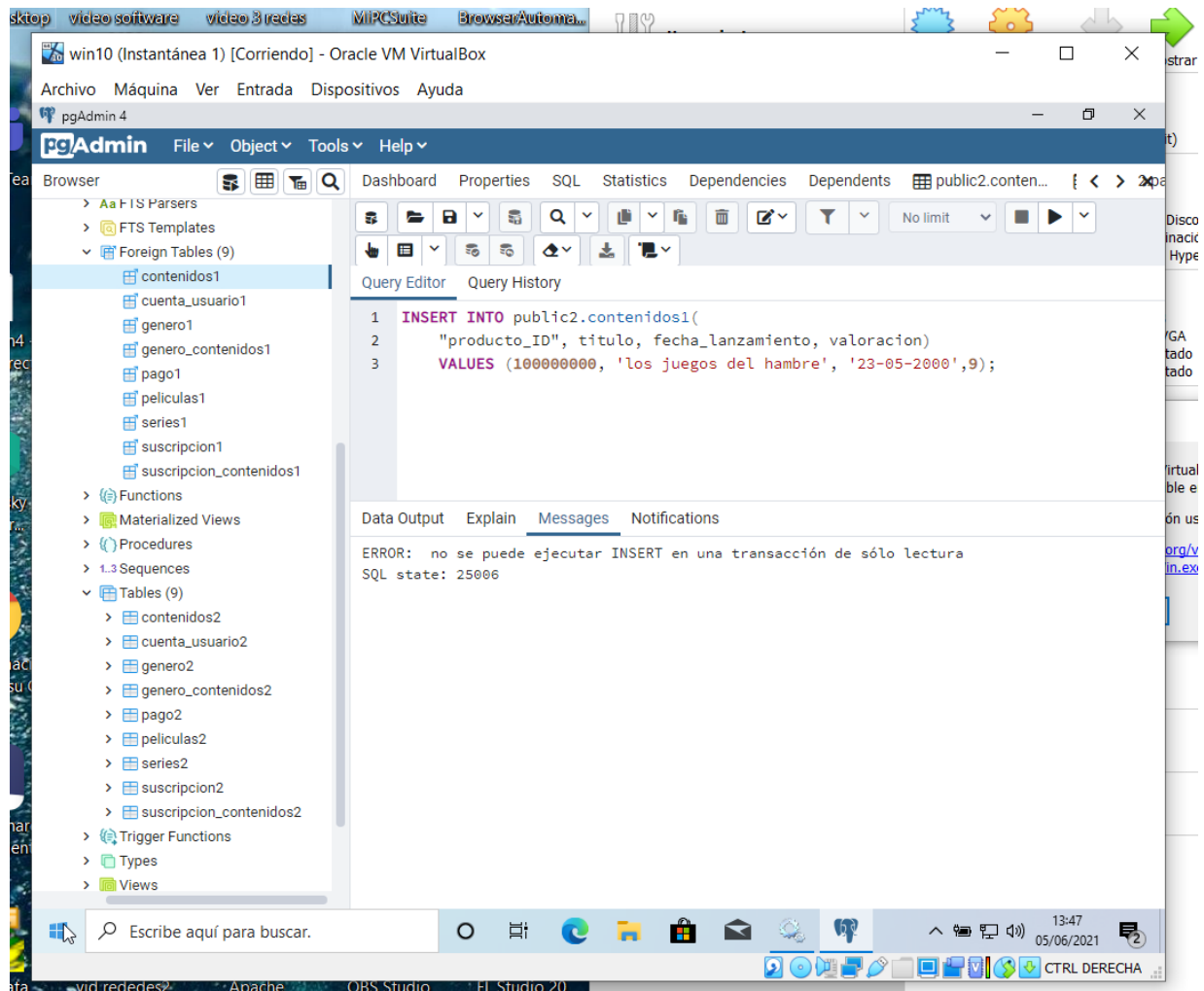
Haciendo la consulta **desde maestro2**, podemos observar que aparece la nueva tupla insertada en el maestro1:



5. Operaciones que se pueden realizar en cada tipo de equipo de red. Provocar situaciones de caída de los nodos y observar mensajes, acciones correctoras a realizar para volver el sistema a un estado consistente.

1. Operaciones que se pueden realizar en cada tipo de equipo de red

En los esclavos no podemos realizar operaciones de escritura, tan solo de lectura. Los servidores están funcionando correctamente replicando simplemente la información del servidor Maestro y no pueden hacer ningún tipo de alteración u operación de escritura en la base de datos. Solo pueden consultar y almacenar su información para no perderla en caso de que el servidor Maestro sufriese alguna caída o error. En la siguiente captura podemos ver el mensaje de error que nos salta al intentar insertar en el esclavo2:



2. Caída de los nodos y acciones correctoras para situaciones de caída de los nodos

A) Caída de un nodo esclavo

Si un nodo esclavo sufre una caída y en la base de datos se siguen produciendo operaciones de escritura, a la hora de iniciarse y recuperarse el servidor Esclavo, obtendría los logs de la base de datos del servidor Maestro y por tanto, se actualizaría y tendría los datos al día.

Para simular esta caída, primero detenemos el servidor Esclavo2:

```
C:\Program Files\PostgreSQL\13\bin>pg_ctl stop -D ../data
esperando que el servidor se detenga.... listo
servidor detenido

C:\Program Files\PostgreSQL\13\bin>
```

Mirando desde Maestro2, podemos observar que en el dashboard ya no aparece el WalSenderMain, ya que no está enviando los logs del Wal.

Server activity									
Sessions							Q	Search	↺
		PID	Database	User	Application	Client	Backend start	State	Wait event
✖	■ ▶	9944					2021-06-05 12:33:39 CEST		Activity: BgWriterHibernate
✖	■ ▶	11564					2021-06-05 12:33:39 CEST		Activity: WalWriterMain
✖	■ ▶	17432	netflix2	postgres	pgAdmin 4 - CONN:1865755	::1	2021-06-05 18:23:32 CEST	idle	Client: ClientRead
✖	■ ▶	20604	netflix2	postgres	pgAdmin 4 - DB:netflix2	::1	2021-06-05 12:52:45 CEST	idle	Client: ClientRead
✖	■ ▶	20640		postgres			2021-06-05 12:33:39 CEST		Activity: LogicalLauncherMain
✖	■ ▶	22696					2021-06-05 12:33:39 CEST		Activity: AutoVacuumMain
✖	■ ▶	23832					2021-06-05 12:33:39 CEST		Activity: CheckpointerMain
✖	■ ▶	25884	postgres	postgres	pgAdmin 4 - DB:postgres	::1	2021-06-05 13:14:53 CEST	active	

Insertamos desde Maestro1 otra tupla en contenidos:

Query Editor
Query History

```

1 INSERT INTO public1.contenidos1(
2     "producto_ID", titulo, fecha_lanzamiento, valoracion)
3     VALUES (99999998, '50 sombras', '01-01-2020', 10);

```

Data Output
Explain
Messages
Notifications

Insertamos desde Maestro2 otra tupla en contenidos:

Procedures
Sequences
Tables (9)

- contenidos2
- cuenta_usuario2
 - Columns (6)
 - ID
 - nombre
 - direccion
 - e_mail
 - telefono
 - suscripcion_
- Constraints
- Indexes
- Rules
- Triggers
- genero2
- genero_contenidos:
- pago2

netflix2/postgres@PostgreSQL 13

Query Editor
Query History

```

1 INSERT INTO public2.contenidos2(
2     "producto_ID", titulo, fecha_lanzamiento, valoracion)
3     VALUES (789456123789, 'crepusculo', '09-06-2001', 6);

```

Data Output
Explain
Messages
Notifications

Query returned successfully in 84 msec.

Se recupera el esclavo2 que estaba caído iniciándolo de nuevo:

```

C:\Program Files\PostgreSQL\13\bin>pg_ctl start -D ../data
esperando que el servidor se inicie....2021-06-05 14:03:58.247 CEST [2496] LOG: redirigiendo la salida del registro a
l proceso recolector de registro
2021-06-05 14:03:58.247 CEST [2496] HINT: La salida futura del registro aparecerá en el directorio «log».
listo
servidor iniciado

```

Y se puede observar la tupla insertada desde el Maestro2:

idos1	Query Editor	Query History
._usuario1	1	<code>select * from public2.contenidos2 where titulo = 'crepusculo'</code>
1		
._contenidos1		
as1		
icion1		
icion_contenidos1		
	Data Output	Explain Messages Notifications
id Views	producto_ID [PK] numeric	titulo text
s	fecha_lanzamiento date	valoracion numeric
;	1 789456123789	crepusculo 2001-06-09 6
idos2		
._usuario2		

Y también, la del Maestro1:

ollations	Query Editor	Query History
mainas	1	<code>select * from public2.contenidos1 where "producto_ID" = 99999998</code>
'S Configurations		
'S Dictionaries		
'S Parsers		
'S Templates		
oreign Tables (9)		
contenidos1		
cuenta_usuario1		
genero1		
genero_contenidos1		
pago1		
peliculas1		
series1		
suscripcion1		
	Data Output	Explain Messages Notifications
	producto_ID numeric	titulo text
	fecha_lanzamiento date	valoracion numeric
	1 99999998	50 sombras 2020-01-01 10

Lo que indica que el servidor ha actualizado su base de datos mediante los logs recibidos.

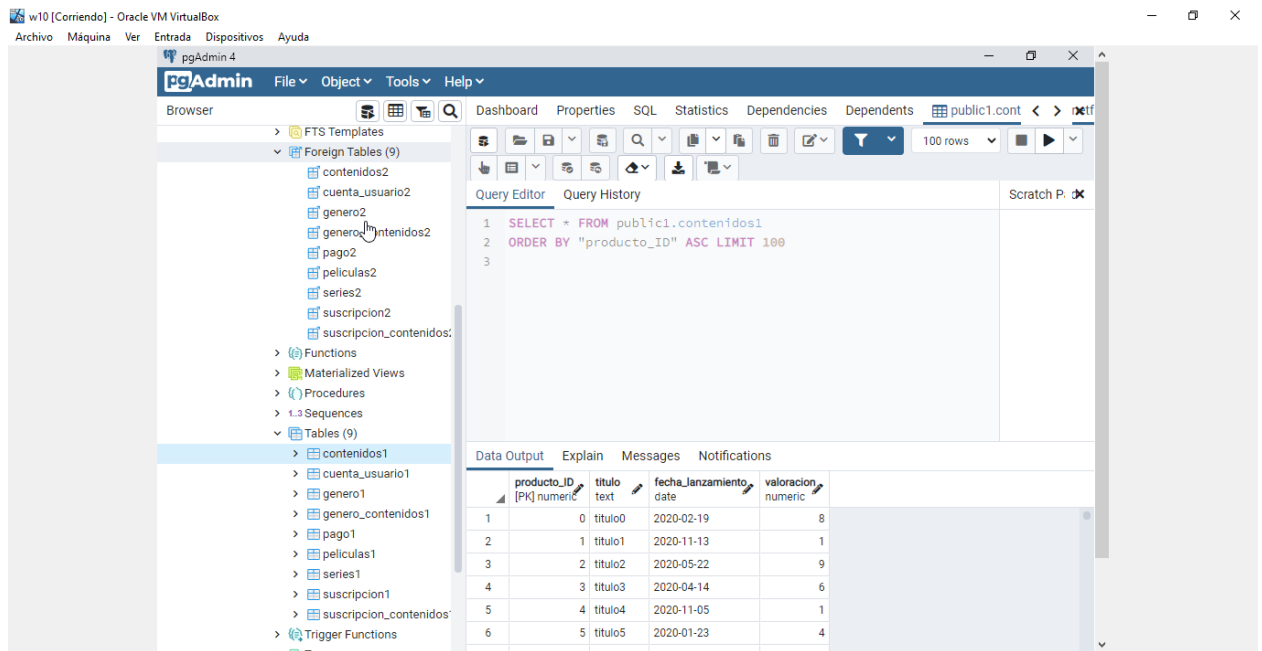
B) Caída de un nodo maestro

En caso de que el servidor del Maestro sufriese alguna caída, la parte de la base de datos situada en ese servidor quedaría inaccesible, por lo que el servidor de su Esclavo quedaría incomunicado, pero actualizado al último log del Maestro antes de la caída. Si se quisiese acceder a las tablas del Maestro caído desde otro Maestro u otro Esclavo ajeno a su servidor no tendrían acceso ya que el servidor se encuentra cerrado, en cambio, desde el Esclavo del mismo servidor se tiene acceso a la última actualización del log antes de la caída.

Simulamos la caída del Maestro1 deteniendo su servidor:

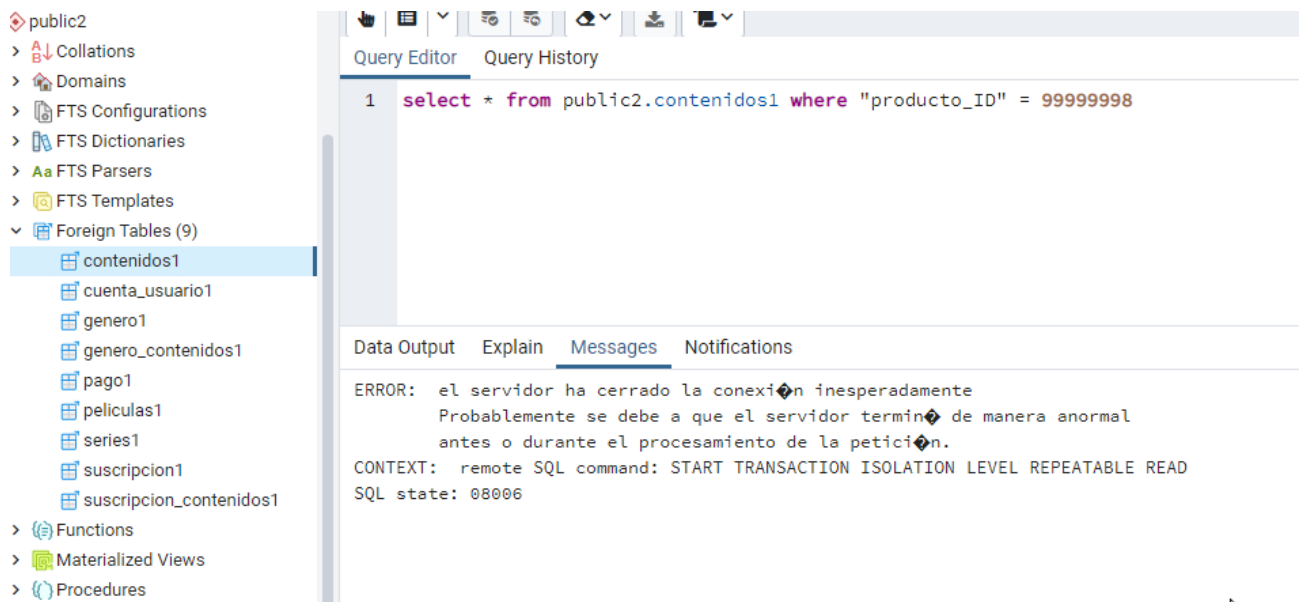
```
C:\Archivos de programa\PostgreSQL\13\bin>pg_ctl stop -D ../data
esperando que el servidor se detenga.... listo
servidor detenido
```

Si intentamos acceder desde esclavo1 nos sale muestra lo último actualizado antes de la caída:



Si intentamos acceder no será posible acceder a los datos ya que el servidor se ha cerrado. Haciendo una consulta desde Esclavo2 o Maestro2 nos sale el siguiente mensaje:

Desde Esclavo2:



Desde Maestro2:

The screenshot shows a PostgreSQL client window titled 'netflix2/postgres@PostgreSQL 13'. On the left, a tree view shows the database structure, including 'Foreign Tables (9)' and 'Tables (9)'. The 'Query Editor' tab is active, displaying a SQL query: `1 select * from public2.contenidos1 where "producto_ID"=99999998`. Below the query, the 'Messages' tab shows an error: `ERROR: could not connect to server "maestro1"`, with a detail explaining that the connection was refused because the server is not listening on the specified port (5433). The SQL state is `08001`.

Si insertamos desde Maestro2 con el servidor del Maestro1 parado:

The screenshot shows the same PostgreSQL client window. The 'Query Editor' tab now displays an insert query: `1 INSERT INTO public2.contenidos2(`
`2 "producto_ID", titulo, fecha_lanzamiento, valoracion)`
`3 VALUES (1593574862, 'ET', '03-03-2003', 8);`. The 'Messages' tab shows a success message: `Query returned successfully in 77 msec.`

Y volvemos a iniciar el servidor del Maestro1 se ven los datos insertados por Maestro2, el Maestro1 obtiene los nuevos logs y, por ello reconoce el nuevo contenido.

netflix1/postgres@PostgreSQL 13

Query Editor

Query History

1 select * from public1.contenidos2 where titulo = 'ET'

Data Output

Explain

Messages

Notifications

	producto_ID numeric	titulo text	fecha_lanzamiento date	valoracion numeric
1	1593574862	ET	2003-03-03	8


C) Caída de dos nodos o más:

Si fueran nodos Standby los que se caen, al iniciarse se actualizaría con los logs de su Maestro y por tanto no se perderían los datos y es solucionable.

Si fuesen nodos Maestros, no se podría realizar ninguna acción sobre sus datos, pero al recuperarse obtendría la información de los demás nodos maestros y sus nodos Standby la replicarían.

6. Insertar datos en cada una de las bases de datos del MAESTRO1 y del MAESTRO2. Realizar una consulta sobre el MAESTRO1 que permita obtener el número de contenidos que puede ver cada usuario en toda la base de datos distribuida (MAESTRO1 + MAESTRO2). Explicar cómo se resuelve la consulta y su plan de ejecución.

Desde Maestro1 se ha insertado contenidos con id = 99999999999999 y cuenta de usuario con id = 2000000 asignado a ese contenido.

 netflix1/postgres@PostgreSQL 13 ▾

Query Editor Query History

```
1 INSERT INTO public1.suscripcion1(
2     tipo, precio, fecha, "suscripcion_ID")
3     VALUES (2, 23.5, '23-02-2020', 2000000);
4 INSERT INTO public1.cuenta_usuario1(
5     "ID", nombre, direccion, e_mail, telefono, "suscripcion_ID_suscripcion")
6     VALUES (2000000, 'Alejandro', 'Alcala', 'alejandro@gmail.com', 722181777, 2000000);
7 INSERT INTO public1.suscripcion_contenidos1(
8     fecha_producto_suscripcion, "suscripcion_ID_suscripcion", "producto_ID_contenidos")
9     VALUES ('03-04-2020', 2000000, 999999999999999);
10 INSERT INTO public1.contenidos1(
11     "producto_ID", titulo, fecha_lanzamiento, valoracion)
12     VALUES (999999999999999, 'sssssshrek', '24-04-2020', 10);
13
```

Desde Maestro2 se han insertado los siguientes datos:

stgre... netflix2/postgres@PostgreSQL 13 * public2.suscrip...

Query Editor Query History

```

1 Insert into public2.suscripcion2(
2     tipo,precio,fecha,"suscripcion_ID"
3 )VALUES (10023022,8.60,'02-06-2021',2000000);
4
5 Insert into public2.cuenta_usuario2(
6     "ID",nombre,direccion,e_mail,telefono,"suscripcion_ID_suscripcion"
7 )VALUES (7878945694568,'Alejandrol','direccion789456','alejandro789456@gmail.com',
8         66666647,2000000);
9
10 INSERT INTO public2.contenidos2(
11     "producto_ID", titulo, fecha_lanzamiento, valoracion)
12     VALUES (159357486288000007, 'Barbie y las 12 princesas bailarinas2', '03-03-2001', 10
13
14 insert into public2.suscripcion_contenidos2(
15     fecha_producto_suscripcion,"suscripcion_ID_suscripcion","producto_ID_contenidos"
16 )VALUES ('02-06-2021',2000000,159357486288000000);
17

```

Data Output Explain Messages Notifications

Query returned successfully in 72 msec.

Hemos ejecutado la siguiente consulta

netflix2/postgres@PostgreSQL 13

Query Editor Query History

```

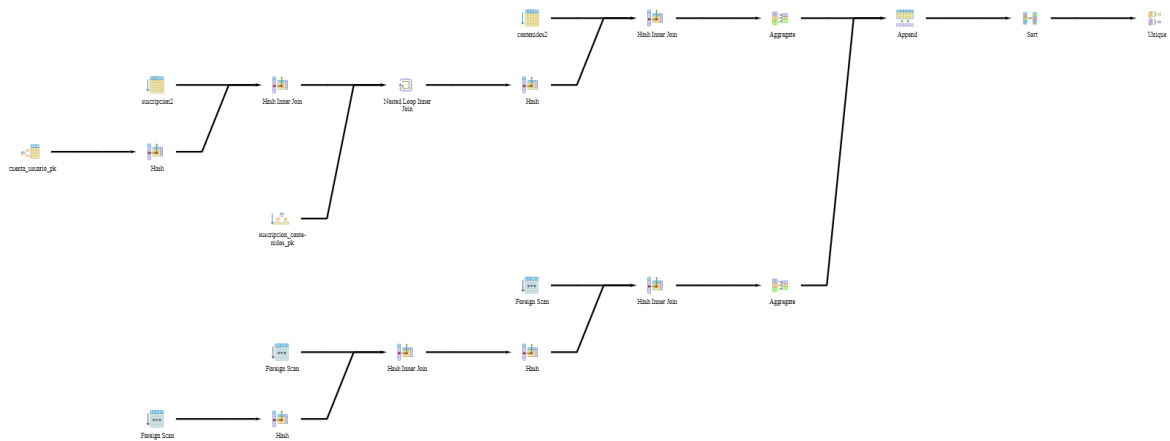
1 explain ((select count ("producto_ID") from (((public2.contenidos2 inner join public2.suscripcion_contenidos2
2 on "producto_ID"= "producto_ID_contenidos" ) inner join public2.suscripcion2 on "suscripcion_ID"=
3     "suscripcion_ID_suscripcion")inner join public2.cuenta_usuario2 on
4     "suscripcion_ID"=public2.cuenta_usuario2."suscripcion_ID_suscripcion")
5     where "ID"=2000000)
6
7 union
8
9 (select count ("producto_ID") from (((public2.contenidos1 inner join public2.suscripcion_contenidos1
10 on "producto_ID"= "producto_ID_contenidos" ) inner join public2.suscripcion1 on "suscripcion_ID"=
11     "suscripcion_ID_suscripcion")inner join public2.cuenta_usuario1 on
12     "suscripcion_ID"=public2.cuenta_usuario1."suscripcion_ID_suscripcion")
13     where "ID"=2000000))
14

```

Data Output Explain Messages Notifications

	QUERY PLAN
	text
1	Unique (cost=602.96..602.97 rows=2 width=8)
2	-> Sort (cost=602.96..602.97 rows=2 width=8)
3	Sort Key: (count(contenidos2."producto_ID"))
4	-> Append (cost=30.20..602.95 rows=2 width=8)
5	-> Aggregate (cost=30.20..30.21 rows=1 width=8)
6	-> Hash Join (cost=11.44..30.18 rows=8 width=32)
7	Hash Cond: (contenidos2."producto_ID" = suscripcion_contenidos2."producto_ID_contenidos")
8	-> Seq Scan on contenidos2 (cost=0.00..16.30 rows=630 width=32)
9	-> Hash (cost=11.34..11.34 rows=8 width=32)
10	-> Nested Loop (cost=8.47..11.34 rows=8 width=32)
11	-> Hash Join (cost=8.32..10.60 rows=1 width=10)
12	Hash Cond: (suscripcion2."suscripcion_ID" = cuenta_usuario2."suscripcion_ID_suscripcion")
13	-> Seq Scan on suscripcion2 (cost=0.00..2.00 rows=100 width=5)
14	-> Hash (cost=8.31..8.31 rows=1 width=5)

	QUERY PLAN
	text
15	-> Index Scan using cuenta_usuario_pk on cuenta_usuario2 (cost=0.29..8.31 rows=1 width=5)
16	Index Cond: ("ID" = '2000000':numeric)
17	-> Index Only Scan using suscripcion_contenidos_pk on suscripcion_contenidos2 (cost=0.15..0.70 rows=4 ...)
18	Index Cond: ("suscripcion_ID_suscripcion" = suscripcion2."suscripcion_ID")
19	-> Aggregate (cost=572.70..572.71 rows=1 width=8)
20	-> Hash Join (cost=449.58..568.36 rows=1739 width=32)
21	Hash Cond: (suscripcion1."suscripcion_ID" = suscripcion_contenidos1."suscripcion_ID_suscripcion")
22	-> Foreign Scan on suscripcion1 (cost=100.00..153.86 rows=1462 width=32)
23	-> Hash (cost=346.61..346.61 rows=238 width=96)
24	-> Hash Join (cost=281.20..346.61 rows=238 width=96)
25	Hash Cond: (contenidos1."producto_ID" = suscripcion_contenidos1."producto_ID_contenidos")
26	-> Foreign Scan on contenidos1 (cost=100.00..153.86 rows=1462 width=32)
27	-> Hash (cost=180.79..180.79 rows=33 width=96)
28	-> Foreign Scan (cost=100.00..180.79 rows=33 width=96)



Para las tablas realiza lecturas secuenciales. A medida que va subiendo de niveles en el árbol realiza hash join, bucles anidados y aggregate. Por último, realiza la unión, los ordena y elimina duplicados.

- Si el nodo MAESTRO1 se quedase inservible, ¿Qué acciones habría que realizar para poder usar completamente la base de datos en su modo de funcionamiento normal? ¿Cuál sería la nueva configuración de los nodos que quedan?

MAESTRO1
Base de Datos NETFLIX1

MAESTRO2
Base de Datos NETFLIX2



ESCLAVO2
Replicando NETFLIX2

Si el nodo Maestro1 fallara, se podría tener una solución para que el sistema no se quede inconsistente. El Esclavo1 contiene todos los datos del Maestro1 como ya se ha explicado anteriormente, por lo que tendría todos los datos del Maestro1 antes de que este diera fallo. Gracias a esto, podemos convertir el Esclavo1 en Maestro1 configurándolo de la manera que se ha explicado al principio del documento. De esta forma, seguirían todos los datos iguales, y el sistema estaría compuesto de Maestro1, Maestro2 y Esclavo2.

8. Según el método propuesto por PostgreSQL, ¿podría haber inconsistencias en los datos entre la base de datos del nodo maestro y la base de datos del nodo esclavo? ¿Por qué?

Existen algunas inconsistencias como la "limpieza temprana". La razón más común de conflicto entre las consultas en espera y la reproducción de WAL es la "limpieza temprana". Normalmente, PostgreSQL permite la limpieza de versiones de filas antiguas cuando no hay transacciones que necesiten verlas para garantizar la visibilidad correcta de los datos de acuerdo con las reglas de MVCC. Sin embargo, esta regla solo se puede aplicar a las transacciones que se ejecutan en el maestro. Por lo tanto, es posible que la limpieza en el maestro elimine las versiones de fila que aún son visibles para una transacción en el modo de espera.

Podría haber una inconsistencia también a la hora de realizar un DROP TABLE en una tabla que se está consultando actualmente en el servidor en espera. Claramente, la consulta en espera no puede continuar si se aplica DROP TABLE en el modo de espera. Si esta situación ocurriera en el primario, DROP TABLE esperaría hasta que la otra consulta hubiera terminado. Pero cuando DROP TABLE se ejecuta en el primario, el primario no tiene información acerca de las consultas que se están ejecutando en el modo de espera, por lo que no esperará ninguna de esas consultas en espera. Los registros de cambios de WAL pasan al modo en espera mientras la consulta en espera aún se está ejecutando, lo que provoca un conflicto. El servidor en espera debe retrasar la aplicación de los registros WAL (y todo lo que sigue a ellos también) o cancelar la consulta en conflicto para que se pueda aplicar DROP TABLE.

9. Conclusiones.

Gracias a los sistemas distribuidos la consistencia de la base de datos se mantiene mejor ya que, aunque algún nodo se caiga o falle al tener otros activos, si se reinicia de nuevo, gracias a los logs del WAL se recuperará la información de la base de datos. También se puede repartir la carga de datos entre los servidores.

A su vez, es útil para evitar la pérdida de datos, ya que los nodos esclavos al estar en continua actualización de los nodos maestros siempre van a ser un apoyo para evitar que esto ocurra, así como poder convertir un nodo esclavo en maestro cuando haya fallas del sistema y por tanto mantener la consistencia del sistema lo mejor posible.

Es útil también para manejar los mismos datos en diferentes sistemas donde no se tengan que estar haciendo copias continuadas de los datos para actualizarlos. Estos se actualizan solos.

En cuanto a los problemas que hemos tenido principalmente ha sido que a la hora de conectar ambos Maestros entre sí el firewall de Alejandro estaba activado y bloqueaba los puertos. Gracias a la ayuda de Iván descubrimos que ese era el fallo por el que no nos dejaba mapear el Maestro1 en el Maestro2. También había veces que al configurar el archivo postgres.conf, al reiniciar el servidor no se iniciaba y teníamos que revisar que los puertos, IP y otros parámetros que configuramos estuviesen correctos. Sin lugar a duda el principal problema ha sido el poder conectar ambos nodos Maestros

desde diferentes casas, así que la solución que encontramos fue realizar la práctica en la misma casa.

Bibliografía

- Capítulo 19: Server Configuration.
- Capítulo: 20.1. The pg_hba.conf File
- Capítulo 25: Backup and Restore.
- Capítulo 26: High Availability, Load Balancing, and Replication.
- Appendix F: Additional Supplied Modules. F.33. Postgres_fdw