

Titulación: Grado en Ingeniería Informática y Sistemas de Información

Curso: 2019-2020. 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:** **Adina** **Murg**

DNI:

ALUMNO 2:

Nombre **y** **Apellidos:** **Victoria** **Lorena** **Ordenes** **Orbegozo**

DNI:

Fecha: 08/06/2020__

Profesor Responsable: Iván González _____

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 online: Semana 27 de Abril y 4 de Mayo.

Entrega de práctica: Día 18 de Mayo de 2020 (**provisional**). Aula Virtual

Documento a entregar: **Este mismo fichero con los pasos 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. Obligatorio. Se debe de entregar en un ZIP comprimido: DNI 'sdelosAlumnos_PECL4.zip

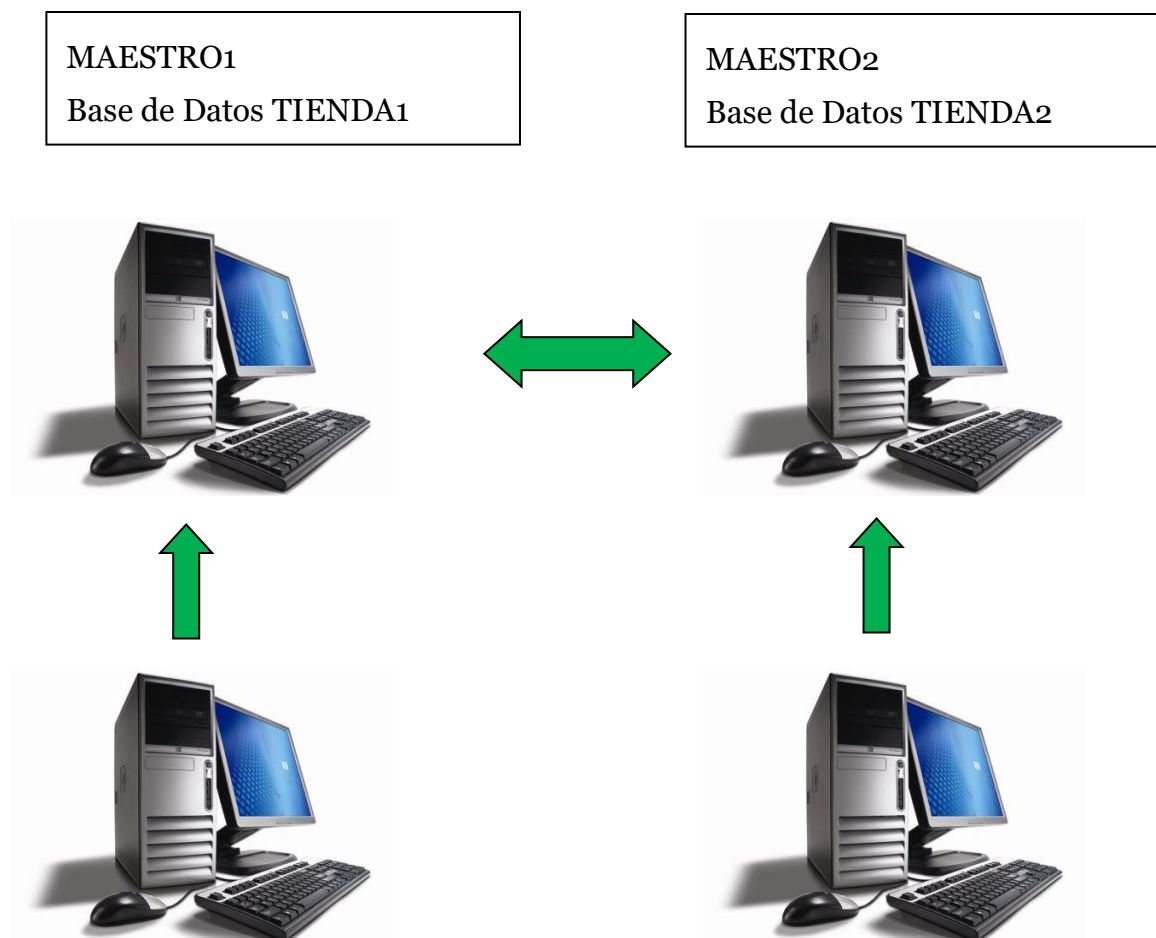
AMBOS ALUMNOS DEBEN ENTREGAR EL FICHERO EN LA PLATAFORMA.

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 12.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 TIENDA1

ESCLAVO2

Replicando TIENDA2

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 unas tiendas en concreto (TIENDA1 y TIENDA2). 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 TIENDA1 o de la base de datos TIENDA2.

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

1. Configuración de cada uno de los nodos maestros de la base de datos de TIENDA1 y TIENDA2 para que se puedan recibir y realizar consultas sobre la base de datos que no tienen implementadas localmente.
2. Configuración completa de los equipos para estar en modo de replicación. 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.
3. 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.
4. 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 nombre de todos los trabajadores junto con su tienda en la que trabajan que hayan realizado por lo menos una venta de algún producto en toda la base de datos distribuida (MAESTRO1 + MAESTRO2). Explicar cómo se resuelve la consulta y su plan de ejecución.
5. 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?
6. 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?
7. Conclusiones.

La memoria debe ser especialmente detallada y exhaustiva sobre los pasos que el alumno ha realizado y mostrar evidencias de que ha funcionado el sistema.

1. Configuración de cada uno de los nodos maestros de la base de datos de TIENDA1 y TIENDA2 para que se puedan recibir y realizar consultas sobre la base de datos que no tienen implementadas localmente.

1. Configuraciones en conf y hba

Primero tenemos que configurar y adaptar los archivos postgres.conf y pg_hba.conf para que los servidores y clientes puedan recibir conexiones entre ellas.

En el archivo postgres.conf en wal_level cambiaremos replica por hot_standby, que nos permite conectarnos a un nodo standby (servidor remoto) y hacer consultas de lectura además de permitirnos ejecutar consultas mientras el nodo se esté actualizando mediante replicación.

```
187 #-----
188 # WRITE-AHEAD LOG
189 #-----
190
191 # - Settings -
192
193 wal_level = hot_standby          # minimal, replica, or logical
194                                # (change requires restart)
```

Del manual ([26.5 Hot Standby](#)),

La siguiente parte no será necesaria modificarla, ya que por defecto viene de forma correcta, con el puerto asignado y el listen_addresses = '*' -> que nos indica las direcciones TCP/IP a las que se deberá escuchar. El '*' equivale a 0.0.0.0 que nos permitirá escuchar a cualquier dirección IPv4.

Del manual ([19.3 Connections and Authentication](#))

```
53 #-----
54 # CONNECTIONS AND AUTHENTICATION
55 #-----
56
57 # - Connection Settings -
58
59 listen_addresses = '*'          # what IP address(es) to listen on;
60                                # comma-separated list of addresses;
61                                # defaults to 'localhost'; use '*' for all
62                                # (change requires restart)
63 port = 5433                     # (change requires restart)
64 max_connections = 100           # (change requires restart)
65 #superuser_reserved_connections = 3 # (change requires restart)
```

A continuación pasamos a modificar el archivo pg_hba.conf ([del manual: 20.1 The pg_hba.conf file](#)). Este archivo nos permite controlar las autenticaciones.

Seguimos el esquema: **host database user IP-address IP-mask auth-method** del manual.

También desde CMD y con la instrucción ipconfig /all en el **maestro 2**(el segundo ordenador) obtenemos la ip que necesitamos en **maestro 1**:

Maestro 2

```
Adaptador de Ethernet Ethernet 2:

Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . :
Descripción . . . . . : Fortinet Virtual Ethernet Adapter (NDIS 6.30)
Dirección física. . . . . : 00-09-0F-FE-00-01
DHCP habilitado . . . . . : sí
Configuración automática habilitada . . . : sí

Adaptador de LAN inalámbrica Wi-Fi:

Sufijo DNS específico para la conexión. . . :
Descripción . . . . . : Intel(R) Dual Band Wireless-AC 3160
Dirección física. . . . . : 34-E6-AD-C0-48-74
DHCP habilitado . . . . . : sí
Configuración automática habilitada . . . : sí
Vínculo: dirección IPv6 local. . . : fe80::891:7106:c653:b31a%14(Preferido)
Dirección IPv4. . . . . : 192.168.1.107(Preferido)
Máscara de subred . . . . . : 255.255.255.0
Concesión obtenida. . . . . : martes, 2 de junio de 2020 12:55:18
La concesión expira . . . . . : viernes, 9 de julio de 2156 19:45:49
Puerta de enlace predeterminada . . . . : 192.168.1.1
Servidor DHCP . . . . . : 192.168.1.1
IAID DHCPv6 . . . . . : 104130221
DUID de cliente DHCPv6. . . . . : 00-01-00-01-23-85-BF-6F-68-F7-28-DE-A8-62
Servidores DNS. . . . . : 192.168.1.1
NetBIOS sobre TCP/IP. . . . . : habilitado

C:\Users\Vicky>
```

Maestro 1

```
Adaptador de LAN inalámbrica Wi-Fi:

Sufijo DNS específico para la conexión. . . :
Descripción . . . . . : Qualcomm Atheros QCA61x4A Wireless Network Adapter
Dirección física. . . . . : 98-22-EF-99-4B-D7
DHCP habilitado . . . . . : sí
Configuración automática habilitada . . . : sí
Vínculo: dirección IPv6 local. . . : fe80::5d14:af60:740b:9129%11(Preferido)
Dirección IPv4. . . . . : 192.168.1.106(Preferido)
Máscara de subred . . . . . : 255.255.255.0
Concesión obtenida. . . . . : martes, 2 de junio de 2020 12:40:12
La concesión expira . . . . . : viernes, 9 de julio de 2156 20:27:12
Puerta de enlace predeterminada . . . . : 192.168.1.1
Servidor DHCP . . . . . : 192.168.1.1
IAID DHCPv6 . . . . . : 110633711
DUID de cliente DHCPv6. . . . . : 00-01-00-01-25-71-C3-29-98-22-EF-99-4B-D7
Servidores DNS. . . . . : 192.168.1.1
NetBIOS sobre TCP/IP. . . . . : habilitado
```

#	TYPE	DATABASE	USER	ADDRESS	METHOD
# IPv4 local connections:					
host	all		all	127.0.0.1/32	md5
host	all		all	192.168.1.107/24	trust
# IPv6 local connections:					
host	all		all	::1/128	md5
# Allow replication connections from localhost, by a user with the					
# replication privilege.					
host	replication		all	127.0.0.1/32	md5
host	replication		all	::1/128	md5
host	replication		postgres	192.168.1.107/24	trust

En **database** y **user** marcaremos all (para aceptar conexiones a todas las bases de datos y de cualquier usuario, nada en específico). En **IP-address** e **IP-mask** especificaremos la IP de la máquina con la que realizaremos la conexión.

Dado que tenemos una IP de clase B, su máscara de subred es 255.255.255.0 que equivale al '/24', por lo que la dirección es 192.168.1.107/24.

En **method** (método de autenticación) seleccionamos trust, que nos permitirá conectarnos sin necesidad de introducir contraseñas.

Desactivamos el Firewall de ambos equipos (tanto para red pública como para privada) dado que tenemos problemas por tenerlo activado:

❌ Firewall de Windows Defender usa una configuración que puede hacer que el dispositivo no es seguro.

Restaurar configuración

🏠 Red de dominio

El firewall está desactivado.

Activar

2. Conexión y pruebas

Desde el manual ([F.33 Postgres fdw](#))

Seguimos las instrucciones que se nos indica:

1. Instalación de postgres_fdw usando CREATE EXTENSION

```
1 create extension postgres_fdw;
```

Explain Data Output Notifications Query History Messages

CREATE EXTENSION

Query returned successfully in 156 msec.

2. Creación de un servidor externo usando CREATE SERVER (sin especificar usuarios ni contraseñas)

```
Query Editor
1 create server tiendas
2   foreign data wrapper postgres_fdw
3   options (host '192.168.1.107', port '5432', dbname 'tienda2');
```

Explain Data Output Notifications Query History Messages

CREATE SERVER

Query returned successfully in 99 msec.

3. Creación de un usuario usando CREATE USER MAPPING (aquí sí hay que especificar usuarios y sus contraseñas)

```
1 CREATE USER MAPPING FOR postgres
2   server tiendas
3   OPTIONS(user 'adina', password '123');
```

Explain Data Output Notifications Query History Messages

CREATE USER MAPPING

Query returned successfully in 82 msec.

Con esta sentencia le asignamos al servidor existente previamente creado, un usuario que ya existe en la base de datos de maestro2.

Nota: Este usuario debe contar con los privilegios necesarios.

4. Uso de IMPORT FOREIGN SCHEMA para crear una tabla externa

```
1 create schema tienda2;
2 import foreign SCHEMA public
3   from server tiendas into tienda2;
```

Explain Data Output Notifications Query History Messages

IMPORT FOREIGN SCHEMA

Query returned successfully in 824 msec.

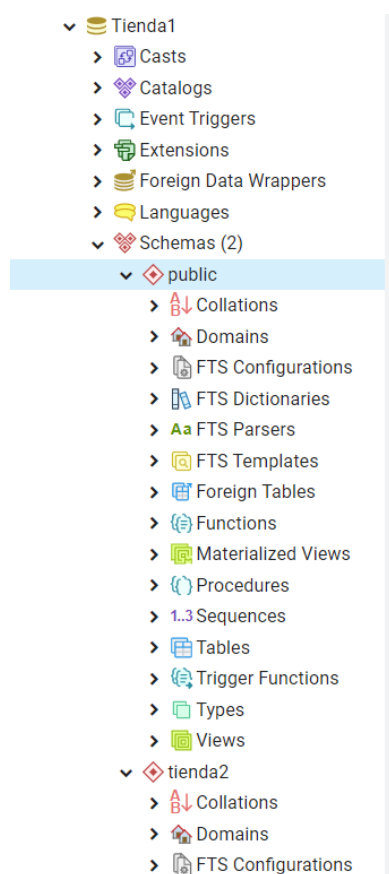
```
create server tiendas
foreign data wrapper postgres_fdw
options (host '192.168.1.107', port '5432', dbname 'tienda2');

CREATE USER MAPPING FOR postgres
server tiendas
OPTIONS(user 'adina', password '123');

create schema tienda2;
import foreign SCHEMA public
from server tiendas into tienda2;
```

Nosotros emplearemos import foreign schema (es más eficiente) dado que la base de datos la teníamos creada y de esta manera la estamos copiando sin necesidad de volver a escribir el código de la creación de la base de datos respetando el número de columnas y sus nombres (como pasaría si se empleara create foreign table ya que es manual).

También comprobamos que el servidor ha conectado correctamente:



Ordenador1-Maestro1

Dashboard Properties SQL Statistics Dependencies Dependents tienda1/postgres@PostgreSQL 12 *

Query Editor

```
1 select * from tienda2.tienda
```

Explain Data Output Notifications Query History Messages

	idtienda	nombre	ciudad	barrio	provincia
1	integer	text	text	text	text
1	1234	N1	C1	B1	P1

Ordenador-Maestro2

Dashboard Properties SQL Statistics Dependencies Dependents tienda2/postgres@PostgreSQL 12 *

Query Editor Query History

```
1 select * from tienda
```

Data Output Explain Messages Notifications

	idtienda	nombre	ciudad	barrio	provincia	
1	[PK] integer	text	text	text	text	
1		1234	N1	C1	B1	P1

Ahora repetiremos este mismo proceso en el ordenador2.

Dashboard Properties SQL Statistics Dependencies Dependents tienda2/postgres@PostgreSQL 12 *

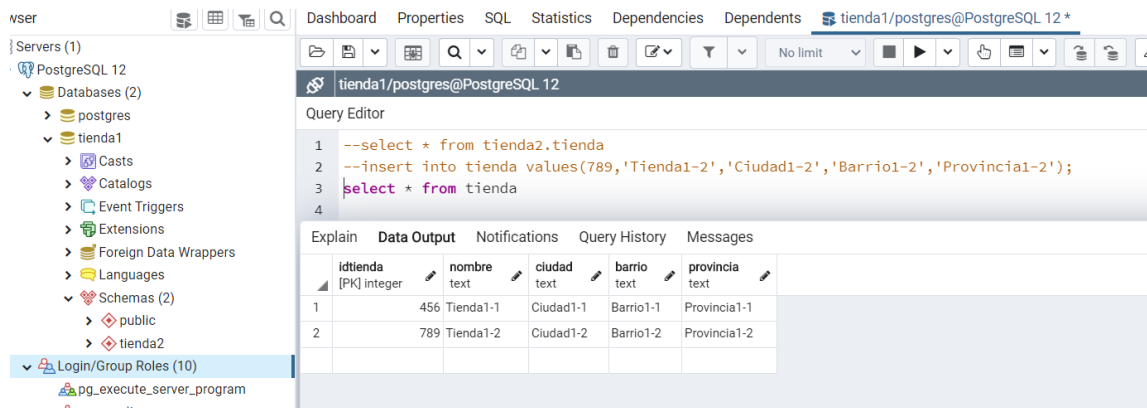
Query Editor Query History

```
4 -- options (host '192.168.1.106', port '5432', dbname 'tienda1');
5 --CREATE USER MAPPING FOR postgres
6 -- server tiendas
7 --OPTIONS(user 'victoria', password '123');
8
9 --create schema tienda1;
10 --import foreign SCHEMA public
11 --from server tiendas into tienda1;
12
13 select * from tienda1.tienda
```

Data Output Explain Messages Notifications

	idtienda	nombre	ciudad	barrio	provincia
1	integer	text	text	text	text
1	456	Tienda1-1	Ciudad1-1	Barrio1-1	Provincia1-1
2	789	Tienda1-2	Ciudad1-2	Barrio1-2	Provincia1-2

Hemos hecho un par de insert en tienda1-ordenador1 para verificar que si tienda1 sufre un cambio también lo puede ver tienda2.



Query Editor

```

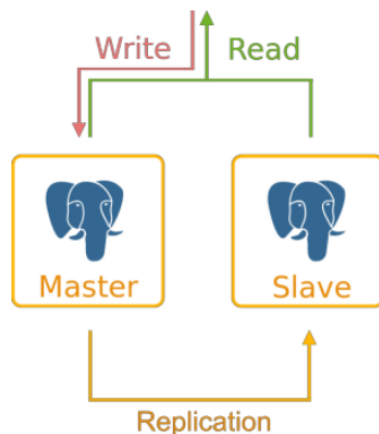
1 --select * from tienda2.tienda
2 --insert into tienda values(789,'Tienda1-2','Ciudad1-2','Barrio1-2','Provincia1-2');
3 select * from tienda
4

```

	idtienda	nombre	ciudad	barrio	provincia
	[PK] integer	text	text	text	text
1	456	Tienda1-1	Ciudad1-1	Barrio1-1	Provincia1-1
2	789	Tienda1-2	Ciudad1-2	Barrio1-2	Provincia1-2

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

Para evitar pérdidas de datos se hace uso de la tecnología de replicación, basada en la replicación de un servidor maestro en un esclavo o también llamado standby server. El Maestro podrá realizar consultas tanto de lectura como de escritura, mientras que el esclavo sólo podrá leer.



A continuación, se explicará cómo crear y configurar una base de datos con un Maestro-Esclavo haciendo uso de hot standby o replicación streaming, en la cual el esclavo se conecta al maestro el cual le transmitirá los archivos WAL a medida que se vayan generando, y el esclavo actuará como una especie de backup continuo.

Elegimos el método de streaming frente al de archivado continuo (WAL shipping) dado que este es más eficiente.

Desde el manual, [apartado 26: High Availability, Load Balancing and Replication](#)

Antes de empezar hay que tener en cuenta:

- Maestro y Esclavo deben tener la misma versión de Postgresql. En nuestro caso se empleará la versión 12.
- Ambas máquinas deben tener una arquitectura de x64 bits ya que Postgres versión 12 sólo está disponible para ésta.

Nosotros emplearemos como MAESTRO una máquina local y como ESCLAVO una máquina virtual de VirtualBox.

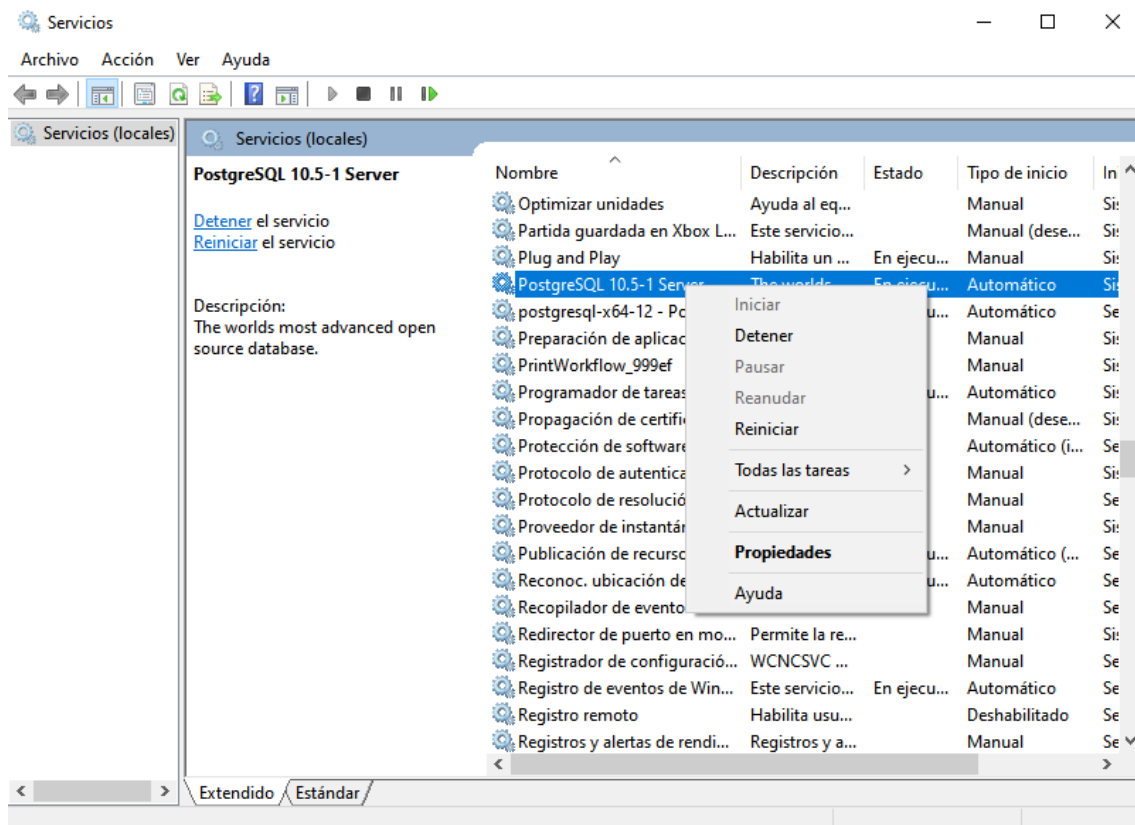
Trabajaremos con la siguiente información:

Nodo	IP	Máscara Subred	Especificaciones	Usuario
Maestro	192.168.1.106	255.255.255.0 (CIDR 24)	Ordenador con Windows 10 x64	Postgres (todos los permisos)
Esclavo	192.168.1.104	255.255.255.0 (CIDR 24)	MV con Windows 7 x64	Postgres (todos los permisos)
Maestro 2	192.168.1.107	255.255.255.0 (CIDR 24)	Ordenador con Windows 10 x64	Postgres (todos los permisos)
Esclavo 2	192.168.1.108	255.255.255.0 (CIDR 24)	MV con Windows 7 x64	Postgres (todos los permisos)

El usuario que debemos emplear es cualquiera que tenga permisos de replicación, o bien creamos un nuevo usuario con **CREATE ROLE nombreusuario WITH REPLICATION PASSWORD 'contraseña' LOGIN** o empleamos postgres, como es nuestro caso, que tiene todos los permisos sobre la base de datos (superusuario).

También desactivaremos los firewall tanto de maestro como de esclavo para evitar problemas.

Nota: algunas de las modificaciones requerirán de un reinicio de postgres. Para ello abriremos servicios, seleccionaremos postgres y pincharemos en reiniciar.



Para comenzar y comprobar la conexión entre máquina virtual (esclavo) y maestro realizamos una prueba de ping.

```
C:\Users\vicky>ping 192.168.1.104

Haciendo ping a 192.168.1.104 con 32 bytes de datos:
Respuesta desde 192.168.1.104: bytes=32 tiempo<1m TTL=128
Respuesta desde 192.168.1.104: bytes=32 tiempo<1m TTL=128
Respuesta desde 192.168.1.104: bytes=32 tiempo<1m TTL=128
Respuesta desde 192.168.1.104: bytes=32 tiempo<1m TTL=128

Estadísticas de ping para 192.168.1.104:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
        (0% perdidos),
    Tiempos aproximados de ida y vuelta en milisegundos:
        Mínimo = 0ms, Máximo = 0ms, Media = 0ms
```

Empezaremos configurando el **MAESTRO**:

1. Pasaremos a modificar el archivo postgresql.conf mirando las secciones WRITE-AHEAD LOG y REPLICATION de la siguiente manera:

```
.87 #-----
.88 # WRITE-AHEAD LOG
.89 #-----
.90
.91 # - Settings -
.92
.93 wal_level = hot_standby          # minimal, replica, or logical
```

Wal_level: nos indica cuánta información se escribirá en los WAL. Por defecto el valor es replica, esto significa que en los WAL se escribirán los datos suficientes para permitir la replicación, sin embargo, nosotros lo cambiaremos a hot_standby, que será importante para el esclavo, (no para el maestro) ya que le permitirá al esclavo el poder hacer lecturas. Sin hot_standby, no se podrán realizar consultas (select) en el esclavo.

```
#-----
# CONNECTIONS AND AUTHENTICATION
#-----
# - 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 = 5432                      # (change requires restart)
max_connections = 100            # (change requires restart)
```

En **listen_addresses** lo dejaremos por defecto con el valor '*' que significará que nuestro Maestro podrá escuchar a cualquier dirección IP sin necesidad de especificarlas en concreto.

```
15
16 # - Standby Servers -
17
18 # These settings are ignored on a master server.
19
20 #primary_conninfo = ''           # connection string to sending server
21 #                               # (change requires restart)
22 #primary_slot_name = ''         # replication slot on sending server
23 #                               # (change requires restart)
24 #promote_trigger_file = ''      # file name whose presence ends recovery
25 #hot_standby = off             # "off" disallows queries during recovery
26 #                               # (change requires restart)
27 #max_standby_archive_delay = 30s # max delay before canceling queries
28 #                               # when reading WAL from archive;
29 #                               # -1 allows indefinite delay
30 #max_standby_streaming_delay = 30s # max delay before canceling queries
31 #                               # when reading streaming WAL;
```

Por último, no marcamos on en **hot_standby**, que deberá estar obligatoriamente en off.

Guardamos el archivo.

Reiniciamos Postgres.

2. A continuación, modificaremos el archivo pg_hba.conf

```
|
# TYPE DATABASE USER ADDRESS METHOD
# IPv4 local connections:
host all all 127.0.0.1/32 md5
host all all 192.168.1.107/24 trust
# IPv6 local connections:
host all all ::1/128 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
host replication all 127.0.0.1/32 md5
host replication all ::1/128 md5
host replication postgres 192.168.1.107/24 trust
host replication postgres 192.168.1.104/24 trust
```

Ahora dentro de pg_hba.conf añadimos una nueva línea, un nuevo host con permisos de replicación con la IP que anteriormente conseguimos de nuestro esclavo (192.168.1.104/24) y al que le daremos como autenticación 'trust' para evitar pedirle contraseñas.

Guardamos el archivo.

Reiniciamos Postgres.

3. Ahora realizaremos un backup con `pg_basebackup`, que nos generará un data que será el que debemos sustituir por el data del esclavo.

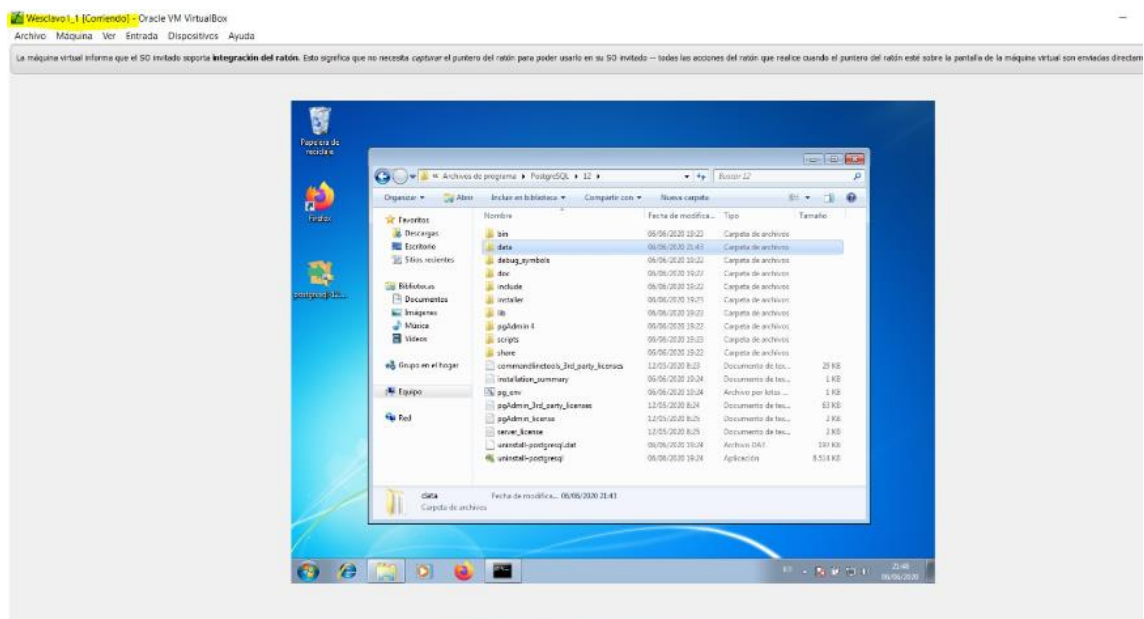
```
C:\Program Files\PostgreSQL\12\bin>pg_basebackup -h localhost -U postgres -D C:\Users\vicky\Desktop\CopiaSeguridad -R
Contraseña:
C:\Program Files\PostgreSQL\12\bin>
```

Y tras esta instrucción obtenemos el data:

Nombre	Fecha de modificación	Tipo	Tamaño
base	06/06/2020 21:40	Carpetas de archivos	
global	06/06/2020 21:40	Carpetas de archivos	
log	06/06/2020 21:40	Carpetas de archivos	
pg_commit_ts	06/06/2020 21:40	Carpetas de archivos	
pg_dynshmem	06/06/2020 21:40	Carpetas de archivos	
pg_logical	06/06/2020 21:40	Carpetas de archivos	
pg_multixact	06/06/2020 21:40	Carpetas de archivos	
pg_notify	06/06/2020 21:40	Carpetas de archivos	
pg_replicat	06/06/2020 21:40	Carpetas de archivos	
pg_serial	06/06/2020 21:40	Carpetas de archivos	
pg_snapshots	06/06/2020 21:40	Carpetas de archivos	
pg_stat	06/06/2020 21:40	Carpetas de archivos	
pg_stat_tmp	06/06/2020 21:40	Carpetas de archivos	
pg_subtrans	06/06/2020 21:40	Carpetas de archivos	
pg_tblspc	06/06/2020 21:40	Carpetas de archivos	
pg_twophase	06/06/2020 21:40	Carpetas de archivos	
pg_xact	06/06/2020 21:40	Carpetas de archivos	
backup_label	06/06/2020 21:40	Archivo	1 KB
current_logfiles	06/06/2020 21:40	Archivo	1 KB
pg_ctl	06/06/2020 21:40	Archivo	0 KB
pg_hba	06/06/2020 21:40	Archivo: CONF	5 KB
pg_ident	06/06/2020 21:40	Archivo: CONF	2 KB
PG_VERSION	06/06/2020 21:40	Archivo	1 KB
postgresql.auto	06/06/2020 21:40	Archivo: CONF	1 KB
postgresql	06/06/2020 21:40	Archivo: CONF	27 KB
standby.signal	06/06/2020 21:40	Archivo: SIGNAL	0 KB

Al terminar, deberá de aparecer junto con todo el data del backup el archivo que buscábamos, standby.signal. No importa que esté vacío, lo importante es su existencia

A continuación, pasaremos a modificar el **ESCLAVO**.



Nota: el esclavo debe estar PARADO (verificamos en servicios que lo esté).

4. Procedemos a renombrar la carpeta data del esclavo, pasando de llamarse 'data' a 'antiguo1'. Esta carpeta la mantendremos por seguridad, pero la que realmente queremos usar y emplearemos es la carpeta data del backup que realizamos anteriormente. Esta carpeta la dejaremos con el nombre data. Dentro de esta data realizaremos los siguientes puntos (la modificación del postgres.conf y pg_hba.conf).
5. Abrimos pg_hba.conf

```
# listen on a non-local interface via the listen_addresses
# configuration parameter, or via the -i or -h command line switches.

# TYPE      DATABASE      USER      ADDRESS      METHOD
# IPv4 local connections:
host        all         all        127.0.0.1/32  md5
host        all         all        192.168.1.106/24  trust
# IPv6 local connections:
host        all         all        ::1/128      md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
host        replication  all        127.0.0.1/32  md5
host        replication  all        ::1/128      md5
host        replication  postgres   192.168.1.106/24  trust
```

Modificamos la IP por la del Maestro (192.168.1.106), que será a lo que queremos que se conecte y lo que replique.

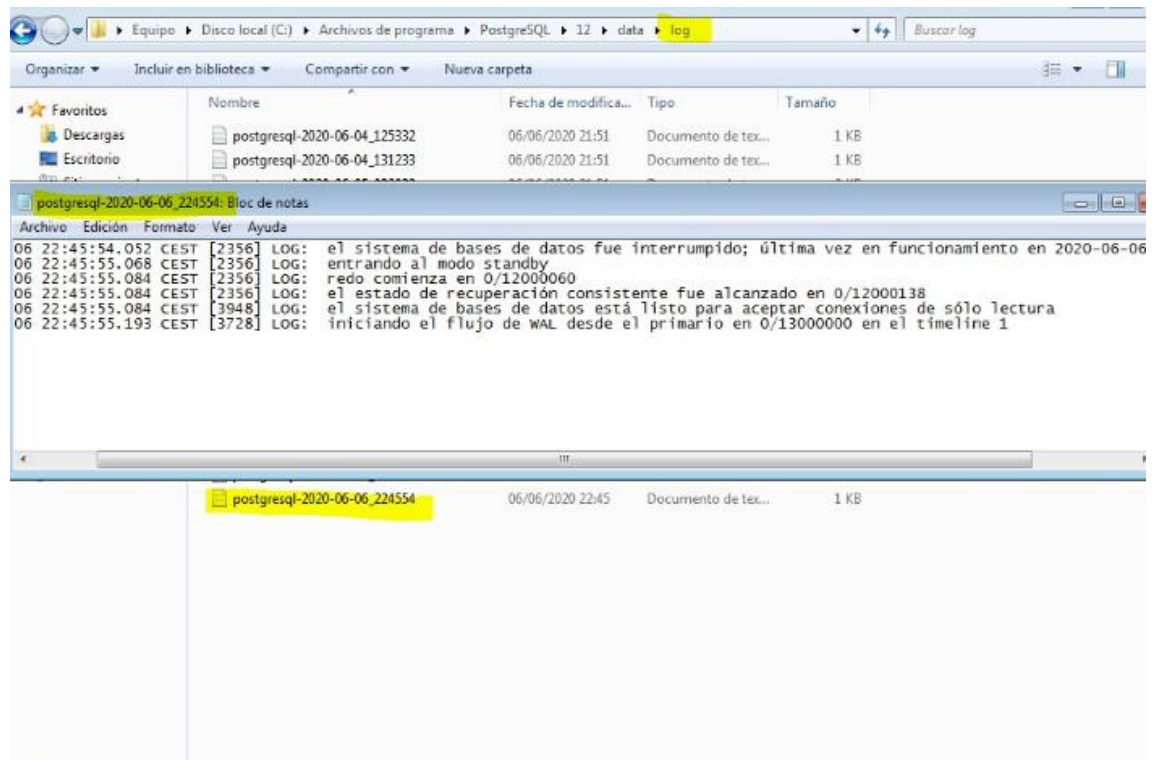
6. Volvemos a abrir postgresql.conf desde el esclavo

```
# and comma-separated list of application_name
# from standby(s); '*' = all
# number of xacts by which cleanup is delayed
#vacuum_defer_cleanup_age = 0
# - Standby Servers -
# These settings are ignored on a master server.
primary_conninfo = 'host= 192.168.1.106 port=5432 user=postgres password=root ' # connection string to sending server
# (change requires restart)
# replication slot on sending server
# (change requires restart)
# file name whose presence ends recovery
# "off" disallows queries during recovery
# (change requires restart)
# max delay before canceling queries
# when reading WAL from archive;
# -1 allows indefinite delay
# max delay before canceling queries
# when reading streaming WAL;
# -1 allows indefinite delay
# send replies at least this often
#wal_receiver_status_interval = 10s
# (change requires restart)
```

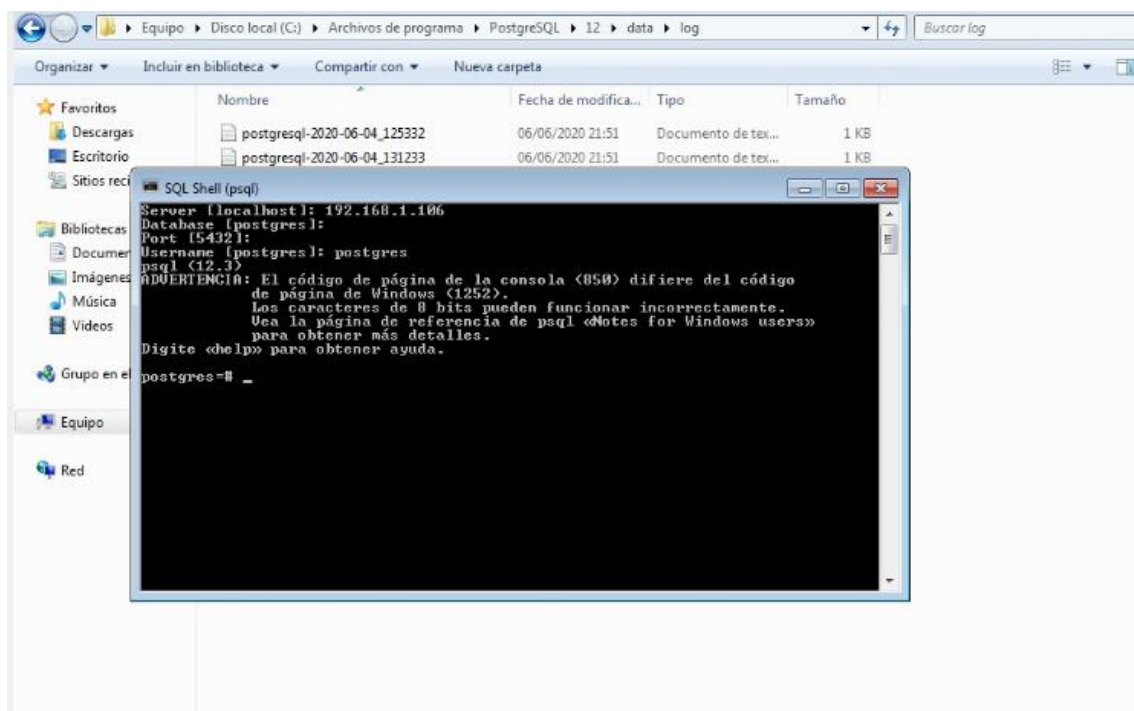
Marcamos **hot_standby** como ON (en el maestro se quedaba por defecto en OFF y comentado)

Modificaremos **primary_conninfo** para especificar la conexión que el esclavo usará para conectarse al maestro. En él hay que indicar el host, además del puerto y el usuario con privilegios de replicación (postgres) y su contraseña.

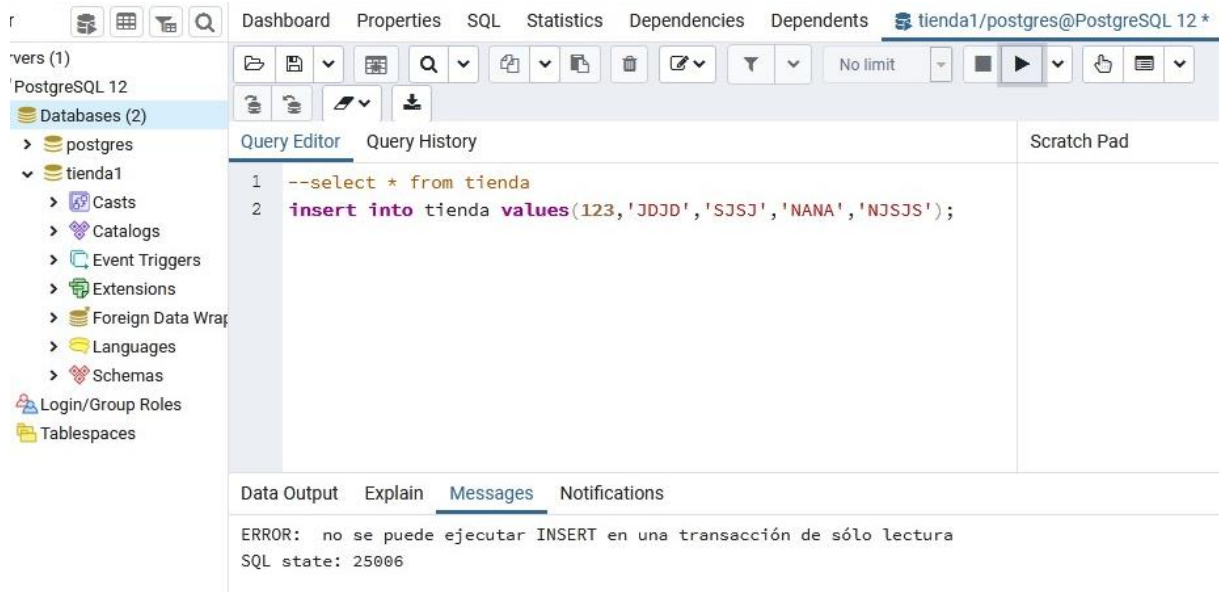
7. El esclavo estaba detenido inicialmente. Primero deberemos ir al maestro y lo inicializamos (porque es el primero que debe arrancar) y ahora inicializamos el esclavo. Comprobamos en los logs (donde aparecerían los errores) que se ha realizado correctamente.



8. También comprobamos por seguridad desde psql que la conexión es correcta:



Como ejemplo probamos a hacer un insert, y Postgres no nos lo deberá permitir. Como observamos, esto es correcto.



Repetiremos los puntos para el Maestro2-Esclavo2

3. 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.

En el equipo de red tenemos un maestro y un esclavo. El maestro tendrá todas las opciones posibles de escritura (insert, update, delete) y obviamente lectura. Por su parte, el esclavo, sólo tendrá permisos de lectura. Esto se da entre el Maestro y su propio esclavo. En ejercicios superiores (Ejercicio 2) mostramos cómo no se nos permite realizar un insert (escritura) desde un nodo esclavo.

Por otro lado, entre dos maestros, los permisos son tanto de lectura como de escritura (obviamente éstos son los máximos, se pueden aplicar restricciones a diferentes usuarios). En ejercicios superiores (Ejercicio 1) se muestran ejemplos.

Para provocar caídas, desde el manual: [26.3 Failover](#).

Información relevante adicional [aquí](#). (EDBPostgres – How does PostgreSQL master-slave failover work?)

Tenemos dos situaciones:

- Caída del nodo maestro. En este caso el nodo esclavo debe iniciar una acción correctora.
- Caída del nodo esclavo. En este caso el nodo maestro no necesita iniciar medidas correctoras. También podemos o bien esperar y recuperar el esclavo o bien crear uno nuevo.

La situación b) es la más sencilla de resolver, aunque se puede complicar si se da también la caída del maestro. En ese caso habría que recuperar lo realizado mirando los logs de WAL, donde aparecería las transacciones realizadas y comprometidas, que se procederían a rehacer de nuevo.

El suceso a) también se puede complicar si el nodo esclavo se ascendiera a maestro y después de este suceso el maestro real se reiniciará, ya que se producirían fallos. Para resolver este problema habría que instaurar un mecanismo para informar al maestro real y original de que su posición de maestro ya está obsoleta.

Ante estas situaciones se recomienda tener algún mecanismo que se asegure y verifique la conexión entre los nodos, ya que Postgres no proporciona soporte para identificar fallos en el nodo maestro e informar al esclavo secundario.

Procedemos a simular las situaciones:

1. Caída del nodo esclavo

Si se produce el caso de una caída del esclavo, el maestro no se enteraría, seguiría funcionando como si nada. Aparecerían mensajes de error en los logs, de que no se ha podido establecer conexión con el destino. La solución sería la de tratar de reiniciar el esclavo o bien buscar uno nuevo que replique en su lugar. En cualquier caso, no afectaría al maestro, el funcionaría sin problemas. Por seguridad, en los `pg_hba.conf` sería conveniente eliminar la dirección de dicho esclavo caído.

2. Caída del nodo maestro

Si el maestro cae, el esclavo queda 'inservible'. Aún mantendrá los WAL, pero ya no tendrá función de ser. Gracias a sus WAL, actúa como un backup y se podría recuperar el maestro a partir de dichos WAL. Se podría realizar, al igual pero a la inversa que en el ejercicio anterior, hacer un `pg_basebackup` e implantarlo (la carpeta data) sustituyendo el data original del maestro. Habría que realizar modificaciones sobre dicho data, como en los `pg_hba.conf` y adaptarlos al igual que `postgres.conf` de acuerdo que cumpla lo estipulado en el ejercicio 2. Luego se reiniciaría y observaríamos si todo vuelve a estar correcto.

Funcionaría como recuperar un backup normal.

3. Caída del nodo maestro, relevo del esclavo como maestro y reinicio del maestro anterior

Para elaborar este paso primero tendremos que detener el maestro con la instrucción `pg_ctl stop -D`. A continuación, ejecutaremos en el esclavo `pg_promote`, ya que a este esclavo lo tendremos que promocionar a maestro. Lo acompañaremos de `-D` en donde indicaremos la ruta del directorio data.

Una vez se haya promocionado, el archivo `standby.signal` desaparece (ya que éste indicaba que era un esclavo). Tras ejecutar el `pg_ctl promote`, el nuevo maestro mira en los registros del log de `pg_wal` y rehace las transacciones comprometidas del maestro.

Una vez el maestro original/antiguo vuelva a estar disponible, hay que evitar que vuelva a tomar el rol de maestro (de darse este caso podría producirse pérdida en la consistencia de datos). Para solucionar este problema, bastaría con desconectarla de

la red y sustituir su IP de los pg_hba.conf de los otros esclavos (si los hubiera) por el del nuevo maestro.

Una alternativa a todo este proceso sería el de, ante la carencia de Postgres de ofrecernos una solución ante caídas la de creación de un script o un *trigger file* en el esclavo al detectar que el maestro ha caído y de esta manera promocionarse él mismo automáticamente.

Siguiendo lo explicado: **(Ejercicio 5)**

1. Comprobamos con pg_isready que la conexión del maestro con el servidor Postgres está activa (desde powershell de Windows y desde el directorio bin).

```
PS C:\Program Files\PostgreSQL\12\bin> .\pg_isready -h 192.168.1.106 -p 5432
192.168.1.106:5432 - aceptando conexiones
PS C:\Program Files\PostgreSQL\12\bin>
```

2. A continuación, detenemos el maestro con pg_ctl stop

```
PS C:\Program Files\PostgreSQL\12\bin> .\pg_ctl stop -D 'C:\Program Files\PostgreSQL\12\data'
esperando que el servidor se detenga.... listo
servidor detenido
PS C:\Program Files\PostgreSQL\12\bin>
```

Y observamos que se ha detenido correctamente.

```
PS C:\Program Files\PostgreSQL\12\bin> .\pg_isready -h 192.168.1.106
192.168.1.106:5432 - sin respuesta
PS C:\Program Files\PostgreSQL\12\bin>
```

3. Ahora iremos al esclavo y lo promocionaremos a maestro con pg_ctl promote -D (directorio data)

```
C:\Program Files\PostgreSQL\12\bin> .\pg_ctl promote -D "C:\Program Files\PostgreSQL\12\data"
esperando que el servidor se promueva.... listo
servidor promovido
```

4. Observaciones: ha desaparecido el standby.signal correctamente, lo que significa que el nodo es maestro y deja de ser esclavo. También se realizó un insert cualquiera como ejemplo con lo que vemos que ahora sí se nos permite escribir, por que ya no es un esclavo.

5. Reiniciamos el antiguo maestro con pg_ctl start.

```
PS C:\Program Files\PostgreSQL\12\bin> .\pg_ctl start -D 'C:\Program Files\PostgreSQL\12\data'
esperando que el servidor se inicie...2020-06-07 23:52:08.493 CEST [5668] LOG:  iniciando PostgreSQL 12.3, compiled by Visual C++ build 1914, 64-bit
2020-06-07 23:52:08.499 CEST [5668] LOG:  escuchando en la dirección IPv6 «::», port 5432
2020-06-07 23:52:08.499 CEST [5668] LOG:  escuchando en la dirección IPv4 «0.0.0.0», port 5432
2020-06-07 23:52:08.561 CEST [5668] LOG:  redirigiendo la salida del registro al proceso recolector de registro
2020-06-07 23:52:08.561 CEST [5668] HINT:  La salida futura del registro aparecerá en el directorio «log».
listo
servidor iniciado
PS C:\Program Files\PostgreSQL\12\bin>
```

6. Paramos el esclavo/nuevo maestro. Ahora podemos tomar varias decisiones.
 - a) Desechamos el maestro revivido. Lo eliminamos y nos olvidamos de él.
 - b) Lo dejamos y mantenemos en maestro, pero conectándolo como en el ejercicio 1, por lo que ya no tendríamos esclavo, pero le podremos dar esta nueva función.
 - c) Dado que tenemos un nuevo maestro, podríamos considerar rebajar ese antiguo/revivido maestro a esclavo, estaríamos invirtiendo el orden inicial, pero lo que sí sería desaconsejable es revertirlo al original, ya que se perderían muchos datos además de recursos. La pérdida de datos se debe a la ligera demora que se produce entre maestro-esclavo. A pesar de que estemos en streaming, hay un leve retardo y esto podría generar problemas ya que los WAL se van reciclando y reutilizando (aunque serían casos muy especiales ya que no debería dar tiempo a perder ningún WAL).

4. 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 nombre de todos los trabajadores junto con su tienda en la que trabajan que hayan realizado por lo menos una venta de algún producto en toda la base de datos distribuida (MAESTRO1 + MAESTRO2). Explicar cómo se resuelve la consulta y su plan de ejecución.

Realizamos la consulta:

```
select trabajador.nombre, tienda.idtienda, tienda.nombre from
tienda
    inner join trabajador on tienda.idtienda=trabajador.idtienda
    inner join ticket on
trabajador.codigotrabajador=ticket.codigotrabajador
UNION
Select tienda2.trabajador.nombre,
tienda2.tienda.idtienda,tienda2.tienda.nombre from tienda2.tienda
    inner join tienda2.trabajador on
tienda2.tienda.idtienda=tienda2.trabajador.idtienda
    inner join tienda2.ticket on
tienda2.trabajador.codigotrabajador=tienda2.ticket.codigotrabajador
;
```

tienda1/postgres@PostgreSQL 12 *

tienda1/postgres@PostgreSQL 12

Query Editor

```

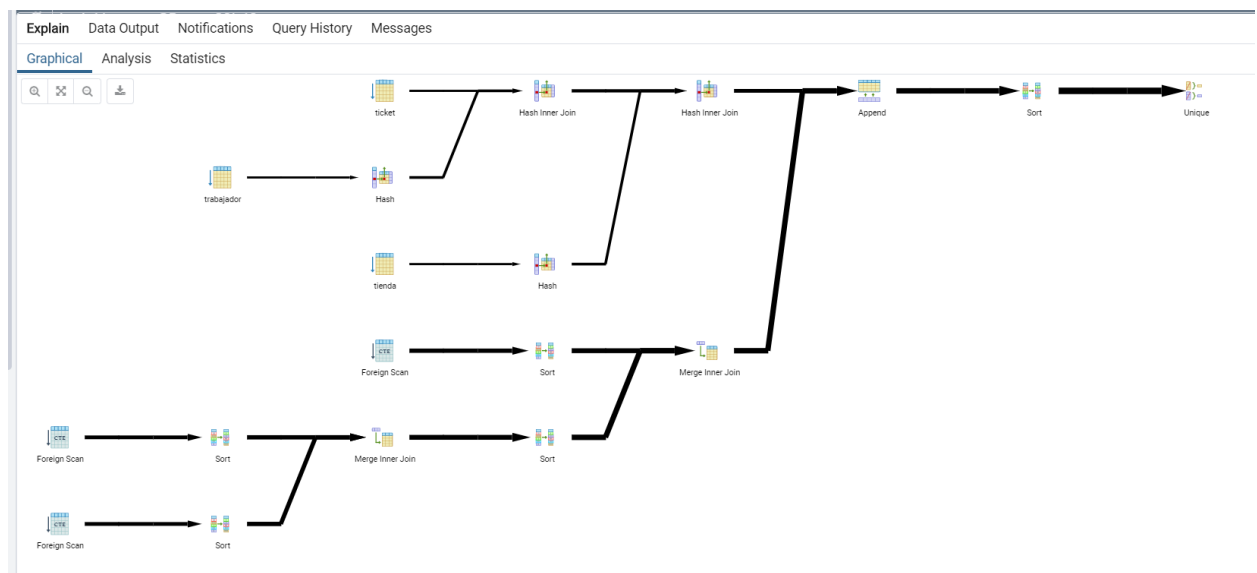
1 select trabajador.nombre, tienda.idtienda, tienda.nombre from tienda
2 inner join trabajador on tienda.idtienda=trabajador.idtienda
3 inner join ticket on trabajador.codigotrabajador=ticket.codigotrabajador
4 UNION
5 Select tienda2.trabajador.nombre, tienda2.tienda.idtienda,tienda2.tienda.nombre from tienda2.tienda
6 inner join tienda2.trabajador on tienda2.tienda.idtienda=tienda2.trabajador.idtienda
7 inner join tienda2.ticket on tienda2.trabajador.codigotrabajador=tienda2.ticket.codigotrabajador;
8
9

```

Explain Data Output Notifications Query History Messages

	nombre character varying	idtienda integer	nombre text
1	D5		2 Tienda2
2	N5		1 Tienda1

Plan de ejecución:



Cómo se resuelve la consulta:

Cuando utilizamos postgres_fdw, las tablas que se encuentran en Maestro 2 aparecen reflejadas en **Foreign Tables** por lo tanto son datos que están en otro servidor y cuando ejecutamos explain (format json)+consulta tendremos varios tipos de nodos, en caso el más interesante es Foreign Scan referente a las 3 consultas que escanean una tabla ajena a nuestro servidor. El resto ya los hemos ido viendo a lo largo de las prácticas anteriores.

5. 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?

En el ejercicio 3 se detalló en profundidad esta situación.

Si Maestro1 cae, el esclavo, para que la base recupere su modo de funcionamiento normal (se pueda volver a escribir y leer como antes) debe convertirse en maestro, por lo que se le promociona. De esta manera todo vuelve a lo habitual. Este esclavo/nuevo maestro rehará las transacciones comprometidas de los WAL desde el último punto que recibió. Se podría sufrir alguna pérdida debido al ligero retraso que se produce entre maestro-esclavo y perder los que todavía no llegó al esclavo.

La configuración de los nodos sería:

Maestro (exesclavo): pierde el standby.signal. Se elimina cualquier rastro de la IP del exmaestro del pg_hba.conf ya que no se necesitará más. También se podría eliminar del listen_addresses de postgres.conf en caso de haberse especificado ahí. Dentro de postgres.conf no es necesario modificar nada. Aunque algunos valores no se correspondan con lo que ‘deberían’, por ejemplo, hot_standby en el maestro original se mantenía en ‘off’, y en el esclavo en ‘on’. Cuando abramos el postgres.conf veremos que sigue marcado en ‘on’, no es necesario modificarlo ya que con la pérdida del standby.signal es suficiente.

Maestro caído: está inservible. En caso de recuperarlo, se explicó en el ejercicio 3 las opciones que tendríamos (abandonarlo, convertirlo en el esclavo del nuevo maestro o dejarlo en maestro y que se comunique con el nuevo maestro como en el ejercicio 1).

6. 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?

Seguimos el método propuesto en el manual y de las opciones disponibles. Nosotros hemos seguido la replicación por streaming, que consiste en enviar desde el servidor maestro los WAL al servidor esclavo de forma continua y constante, es casi una copia en ipso facto del maestro (como un backup continuo) a diferencia de la otra técnica disponible que es replicación por archivos que es un poco más lenta. Distinguimos dos métodos: asíncrona, en la que el commit en el maestro no se espera a que los esclavos reciban los WAL se limita a enviarlos y síncrona, en la que el commit del maestro se producirá cuando todos los esclavos reciban los WAL.

La replicación por transmisión que realizamos es asíncrona, por lo tanto sí podemos tener problemas de inconsistencia entre los datos del nodo maestro y el nodo esclavo, porque si hubiera algún cambio en el maestro o ya sea porque las configuraciones no se hicieron correctamente en ambos nodos o hubiera algún problema en la estabilidad de la red ocasionando un retardo que provocara que el maestro estuviera con un WAL más actualizado y el esclavo con una versión desactualizada. No hay que olvidar que el maestro sigue enviando WAL ocasionando la pérdida definitiva de esos datos.

7. Conclusiones.

Para empezar, encontramos bastante dificultad al realizar los esclavos porque inicialmente elegimos una máquina virtual en Google Cloud y tuvimos problemas de visualización de la red con nuestro ordenador. Después elegimos una Virtual Box y para prevenir ese problema nos aseguramos de que en su configuración estuviera habilitada el adaptador de red, en puente para que permitir la correcta comunicación entre Maestro1 y Esclavo1.

Al tener que implementar una “simulación” de base de datos distribuida hemos utilizado la extensión `postgres_fdw` para permitirnos acceder a datos de otros servidores (Maestro 2) de PostgreSQL. En cuanto a la replicación, descubrimos los dos tipos de los que ya hemos mencionado anteriormente en los ejercicios anteriores. También descubrimos las posibles desventajas que implica la técnica que hemos utilizado, si se produjera un retraso en la red, que podría ocasionar una pérdida de datos a la par que no hay que olvidar las ventajas y la rapidez con la que se transmiten los datos con lo que concluimos que es una copia casi instantánea y esto se logra modificando un par de parámetros de los archivos de los respectivos `pg_hba` y `postgresql.conf` de forma muy sencilla.

También los distintos tipos de nodos que podemos tener y los permisos de cada uno, como vemos los esclavos solo tienen de lectura ya que en principio lo hemos utilizado como “almacén”. Hemos emulado una situación de caída de un maestro que en la vida real nos puede pasar y bastante. La facilidad con la que podemos controlar mediante el comando `pg_ctl` a través del `cmd` con el que podemos inicializar, parar reiniciar PostgreSQL y en situaciones de caída con la que podemos promover un esclavo para que se convierta en un Maestro.

También es interesante considerar las múltiples opciones que tenemos para realizar diferentes posibilidades para ejecutar instrucciones, ya sea mediante powershell, `psql` e incluso la propia query tool.

Consideramos que esta práctica puede ser bastante útil de cara al futuro ya que nos ha ayudado a entender situaciones que parecen bastante cotidianas.

Bibliografía

- 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`