

**Titulación:** Grado en Ingeniería Informática y Sistemas de Información  
**Curso:** 2019-2020. Convocatoria Extraordinaria de Septiembre  
**Asignatura:** Bases de Datos Avanzadas – Laboratorio  
**Practica 4:** **Replicación e Implementación de una Base de Datos Distribuida.**

**ALUMNO 1:**

**Nombre y Apellidos:** Patricia Cuesta Ruiz

**DNI:** 03211093V

**ALUMNO 2:**

**Nombre y Apellidos:** Álvaro Golbano Durán

**DNI:** 03202759D

**Fecha:** 15/09/2020

**Profesor Responsable:** Óscar Gutiérrez Blanco

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 Suspense – Cero.

**Plazos**

**Entrega de práctica:** Día a convenir por el Aula Virtual a primeros de Septiembre. 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**

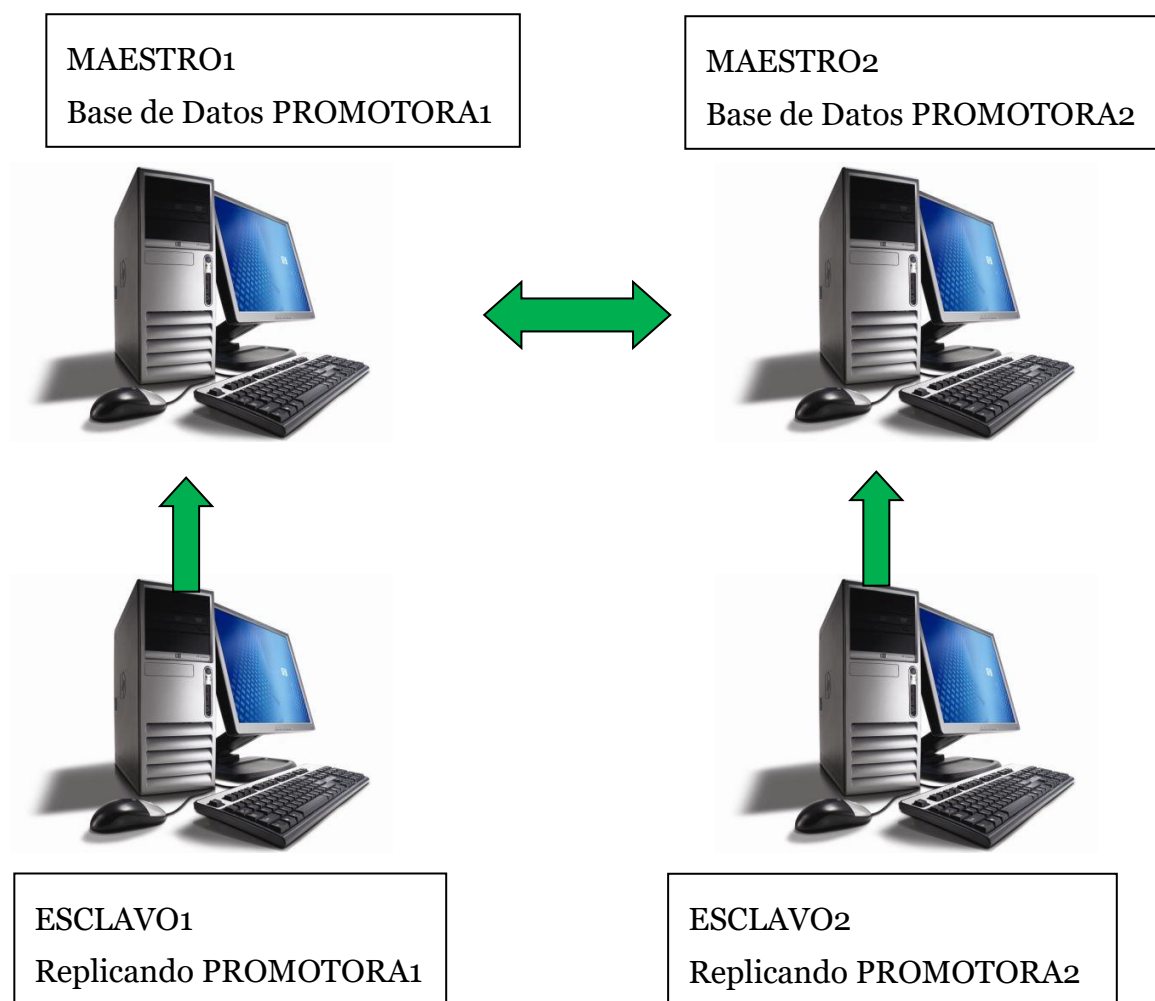
**AMBOS ALUMNOS DEBEN ENTREGAR EL FICHERO EN LA PLATAFORMA.**

## 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:



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 unos grupos musicales en concreto (PROMOTORA1 y PROMOTORA2). 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 PROMOTORA1 o de la base de datos PROMOTORA2.

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 PROMOTORA1 y PROMOTORA2 para que se puedan recibir y realizar consultas sobre la base de datos que no tienen implementadas localmente (módulo postgres\_fdw).

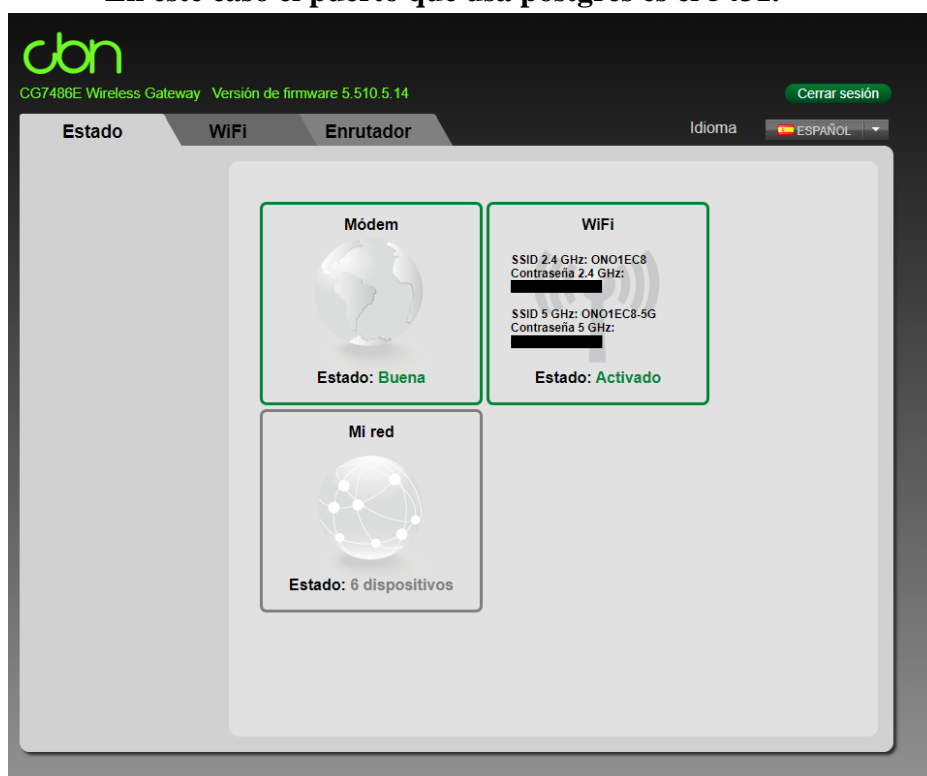
Para poder abrir los puertos del router y de que esta manera ambos podamos conectarnos usando la ip pública miramos en cmd nuestra ip local.

```
Adaptador de Ethernet Ethernet:

Sufijo DNS específico para la conexión. . . : home
Vínculo: dirección IPv6 local. . . : fe80::740c:daf3:25ad:7406%5
Dirección IPv4. . . . . : 192.168.1.9
Máscara de subred . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . . : 192.168.1.1
```

Tras conocer nuestra ip, en la zona de búsqueda por URL del navegador ponemos 192.168.1.1, lo cual nos conducirá a la configuración del router. Para iniciar sesión, por defecto, el usuario es *admin* y la contraseña es *admin* también. Después buscamos un apartado donde podamos modificar los puertos:


- Caso de Patricia (CBN): habría que acceder al apartado Enrutador, Avanzado y en Reenvío de puerto podremos modificar o eliminar los puertos. En este caso el puerto que usa postgres es el 5431.



Local			Externo					
Dirección IP	Puerto inicial	Puerto final	Puerto inicial	Puerto final	Protocolo	Descripción	Activado	
192.168.1.9	5431	5431	5431	5431	Ambos		sí	<div>Borrar todo</div> <div>Modificar</div> <div>Eliminar</div>

- Caso de Álvaro (Movistar): habría que acceder al apartado Puertos donde se pueden modificar o eliminar los puertos. En este caso el puerto que usa postgres es el 5432.

## Configuración del router



**Mi Router**  
**Mitrastar HGW-2501GN-R2**

- Línea teléfono: [REDACTED]
- Firmware: ES\_113WJ10b29
- Conectado: Sí

[Mi configuración actual](#)

Firmware actualizado.

Wifi / Contraseña

Puertos

Otras operaciones

Mis configuraciones

Red local

Nombre PostgreSQL	Protocolo TCP	Puerto/Rango Externo 5432:5432	Puerto/Rango Interno 5432:5432	Dirección IP 192.168.1.40	Activar <input checked="" type="checkbox"/>
PostgreSQL	UDP	5432:5432	5432:5432	192.168.1.40	<input checked="" type="checkbox"/>

Para la configuración de los dispositivos que se pueden conectar a la base de datos, tenemos que modificar los archivos postgres.conf y pg\_hba.conf.

En el pg\_hba.conf fijamos las ip y puertos que se pueden conectar a cualquiera cambiando los valores de este campo por todo 0.

```
# TYPE DATABASE USER ADDRESS METHOD
# IPv4 local connections:
host all all 127.0.0.1/32 md5
host all all 88.17.168.104/32 md5
# 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
```

En el archivo postgres.conf tenemos que ir al apartado CONNECTIONS AND AUTHENTICATION y comprobar lo siguiente:

- listen\_addresses = ‘\*’

Aquí se especifican las direcciones IP que queremos permitir que accedan a nuestro servidor. Si se pone un ‘\*’ es que permitimos que cualquier dirección pueda acceder.

- port = 5431 (en el caso de Patricia) o port = 5450 (en el caso de Álvaro).  
Para ver y configurar el puerto que usa postgres.

```
#-----
# 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 = 5431                     # (change requires restart)
```

#### En el apartado WRITE-AHEAD:

- **wal\_level = replica**

Define cuanta información se grabará en los archivos WAL generados. Se mantiene en replica.

```
#-----
# WRITE-AHEAD LOG
#-----

# - Settings -

wal_level = replica            # minimal, replica, or logical
                                # (change requires restart)
```

#### Dentro de WRITE-AHEAD nos vamos al apartado -Archiving-:

- **archive\_mode = on**

Para que se active la generación de archivos WAL en el servidor. Nos aseguramos de que esté en on.

```
# - Archiving -

archive_mode = on|             # enables archiving; off, on, or always
                                # (change requires restart)
```

#### Por último, en el apartado REPLICATION nos fijamos en:

- **max\_wal\_senders**

Indica las conexiones concurrentes o simultaneas desde los servidores esclavos.

- **Wal\_keep\_segments**

Especifica el número máximo de archivos WAL que serán retenidos en el directorio pg\_xlog en caso de retrasarse el proceso “Streaming replication”.

```
#-----
# REPLICATION
#-----

# - Sending Servers -

# Set these on the master and on any standby that will send replication data.

max_wal_senders = 10          # max number of walsender processes
                                # (change requires restart)
wal_keep_segments = 30        # in logfile segments; 0 disables
```

El módulo `postgres_fdw` proporciona el contenedor de datos foráneos `postgres_fdw`, que puede usarse para acceder a los datos almacenados en servidores externos PostgreSQL.

Creamos la extensión para crear la tabla foránea.

```
create extension postgres_fdw
```

Se crea un servidor externo conectándonos a la ip pública del compañero, con su ip y el nombre de la tabla a que se quiere acceder.

– Caso Patricia:

```
create server server2
foreign data wrapper postgres_fdw
options (host '83.46.104.201', port '5432', dbname 'PROMOTORA2');
```

– Caso Álvaro:

```
create server server1
foreign data wrapper postgres_fdw
options (host '81.203.35.55', port '5431', dbname 'PROMOTORA1');
```

Asignamos el usuario que se va a conectar a la base de datos introduciendo el nombre y la contraseña de este:

– Caso Patricia:

```
create user mapping for postgres
server server2
options(user 'postgres', password 'postgres');
```

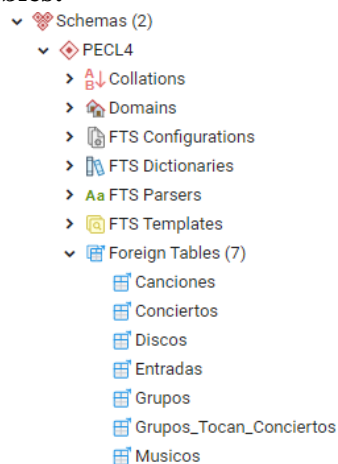
– Caso Álvaro:

```
create user mapping for postgres
server server1
options(user 'postgres', password 'abcd1234.');
```

Por último, hay que importar el foreign schema del compañero. Este paso debemos realizarlo para actualizar la información en caso de que el compañero meta registros nuevos a la base de datos. Para ello, hemos creado un nuevo Schema llamado PECL4, para evitar posibles errores.

```
import foreign schema public from server server2 into "PECL4";
import foreign schema public from server server1 into "PECL4";
```

Y ahora podemos observar que en el Schema PECL4 se han importado las foreign tables:



Es importante que ambos ordenadores tengan el servidor ejecutado ya que en caso opuesto nos saldrá un error en el output que nos indica que no se ha podido realizar la importación de foreign schema debido a que ‘no existe’ el servidor.

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. Especificar el tipo de replicación elegida.

## CONFIGURACIÓN DEL SERVIDOR MAESTRO

Desde `postgresql.conf`, debemos poner en `archive_command` del maestro el directorio en el que se encuentra el directorio `pg_wal`, donde están todos los archivos WAL, y definir `primary_conninfo` del esclavo más tarde. El servidor en espera puede leer WAL desde un archivo WAL o directamente desde el maestro a través de una conexión TCP.

# - Archiving -

```
archive_mode = on          # enables archiving; off, on, or always
                           # (change requires restart)
archive_command = 'C:\Program Files\PostgreSQL\12\data\pg_wal'
                           # command to use to archive a logfile segment
```

También tenemos que cambiar `max_wal_senders` y asegurarnos de que tiene un valor lo suficientemente elevado para asegurarnos de que los segmentos WAL no se reciclan demasiado pronto.

```
#-----
# REPLICATION
#-----
```

# - Sending Servers -

# Set these on the master and on any standby that will send replication data.

```
max_wal_senders = 10      # max number of walsender processes
```

Ahora en el archivo `pg_hba.conf` añadimos una nueva línea en la que ponemos la dirección IP privada de nuestro esclavo.

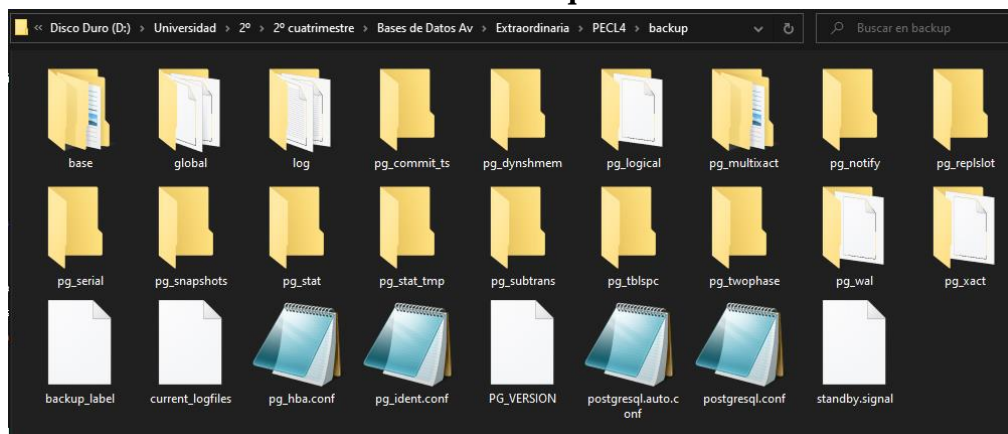
```
# Allow replication connections from localhost, by a user with the
# replication privilege.
host      replication      all             127.0.0.1/32             md5
host      replication      all             ::1/128                  md5
host      replication      all             192.168.1.6/32          md5
```

Una vez configurado el maestro habrá que realizar un backup de nuestra base de datos en el master para que luego esta copia se pueda pasar al esclavo.

Para hacer el backup se usa el siguiente comando:

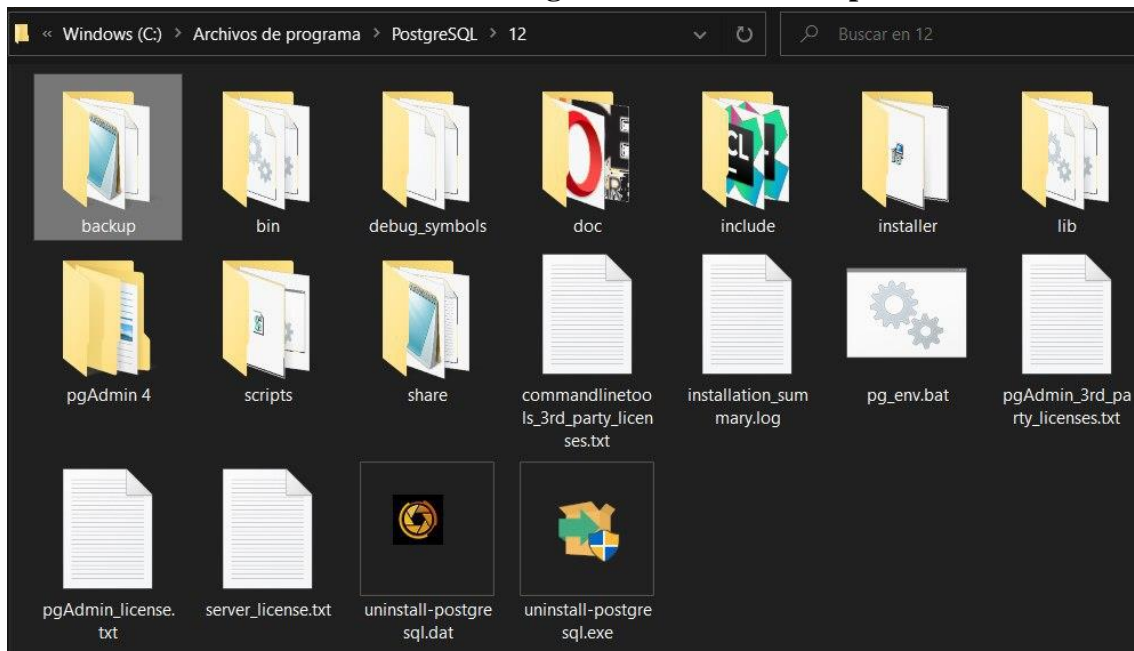
```
C:\Program Files\PostgreSQL\12\bin>pg_basebackup -h localhost -p 5431 -U postgres -R -D "D:\Universidad\2º\2º cuatrimestre\Bases de Datos Av\Extraordinaria\PECL4\backup"
Contraseña:
```

Y si vamos al directorio indicado vemos que ahí está todo el contenido:





Ahora en el ordenador en el que realizaremos la replicación habrá que eliminar la carpeta data y sustituirla con la base de datos obtenida del backup siempre y cuando el servidor esté detenido cambiando luego el nombre de la carpeta a data.

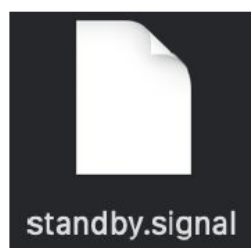


Para facilitar la realización de la replicación vamos a actualizar el archivo pg\_hba en el que hemos añadido antes la IP privada del ordenador esclavo y pondremos 0.0.0.0/0 para que cualquier dirección pueda hacer una replicación de nuestros datos.

```
# 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    all             0.0.0.0/0         md5
```

## CONFIGURACIÓN DEL SERVIDOR ESCLAVO

Se crea el archivo standby.signal al hacer el backup.



Primero debemos añadir en el esclavo en el apartado Standby Servers en primary\_conninfo la IP privada del maestro, el puerto en el que está y el usuario y la contraseña.

# - Standby Servers -

# These settings are ignored on a master server.

```
primary_conninfo = 'host=192.168.1.9 port=5431 user=postgres password=abcd1234'
```



Una vez guardado iniciamos el servidor desde el maestro.

```
C:\Program Files\PostgreSQL\12\bin>pg_ctl start -D "C:\Program Files\PostgreSQL\12\data"
esperando que el servidor se inicie...2020-09-14 15:28:35.631 CEST [13176] LOG:  iniciando PostgreSQL 12.4, compiled by Visual C++ build 1914, 64-bit
2020-09-14 15:28:35.632 CEST [13176] LOG:  escuchando en la dirección IPv6 «:::», port 5431
2020-09-14 15:28:35.633 CEST [13176] LOG:  escuchando en la dirección IPv4 «0.0.0.0», port 5431
2020-09-14 15:28:35.662 CEST [13176] LOG:  redirigiendo la salida del registro al proceso recolector de registro
2020-09-14 15:28:35.662 CEST [13176] HINT:  La salida futura del registro aparecerá en el directorio «log».
. listo
servidor iniciado
```

Podemos comprobar que ha funcionado insertando un registro desde el master y que nos aparezca desde el esclavo.

```
insert into public."Grupos"(
    codigo_grupo, nombre, genero_musical, pais, sitio_web)
values (1, 'ACDC', 'folk', 'EEUU', 'sitio');
```

Y desde el esclavo hacemos la siguiente consulta:

```
select *
from "Grupos"
where codigo_grupo = 1
```

Obteniendo lo siguiente:

	codigo_grupo [PK] integer	nombre text	genero_musical text	pais text	sitio_web text
1	1	ACDC	folk	EEUU	sitio

## TIPOS DE NODO MAESTRO

La documentación proporciona dos métodos para realizar consultar a servidores externos:

- **Dblink:** es sencillo de implementar al requerir poca configuración para poder conectarse a otras bases de datos. Dblink no soporta el modo de solo lectura ni sigue el estándar SQL. Las conexiones solo perduran durante la misma sesión.
- **postgres\_fdw:** no es compatible con versiones anteriores a la 9.3. Soporta el modo lectura y es más eficiente que Dblink, pero sus desventajas son la retrocompatibilidad y las conexiones existentes. Hemos usado este método ya que creemos que es la opción más eficiente.

## TIPOS DE NODO ESCLAVO

Según la documentación los tipos de nodo esclavo que podemos usar son:

- **Write-Ahead Log Shipping:** pone a nuestra disposición hot standby, que permite operaciones de lectura mientras se está recuperando el servidor. Se puede elegir entre síncrono o asíncrono. Si falla, el esclavo rápidamente puede convertirse en maestro con casi toda la totalidad de los datos. Hemos elegido esta opción ya que es de las pocas que emplean-maestro esclavo y nos ofrece lectura mientras se replica, podemos elegir entre síncrono o asíncrono y es rápido ante caídas.
- **Asynchronous Multimaster Replication:** destinado a servidores que no se conectan muy a menudo, permitiendo un uso independiente con algunas comunicaciones para los conflictos, los cuales se resolverán por los usuarios o con reglas.
- **Synchronous Multimaster Replication:** todos los tipos de servidores son buenos candidatos para utilizar este método. Se envían los datos modificados en la consulta antes de que se haga COMMIT. Si hay muchas escrituras esto provocará un rendimiento muy bajo, pero si es para lectura no hay problema.
- **Statement-Based Replication Middleware:** en este caso hay un programa que se encarga de recibir la consulta y enviársela a todos los servidores, que trabajan de forma independiente.

- **Logical Replication:** este crea un canal de datos entre los servidores con los cambios que ocurren en las tablas a partir del WAL. Además, el canal es bidireccional por lo que no hace falta definir maestro ni esclavo.
- **Trigger-Based Máster-Standby Replication:** es un método maestro-esclavo, donde el esclavo solo puede realizar consultas de lectura, el resto las hace el maestro. Envía datos de forma asíncrona al esclavo.
- **Shared Disk Failover:** ofrece un sistema de recuperación rápido ante un fallo sin pérdida de datos al tener un servidor en espera. Sin embargo, solo permite un único array de discos, por lo que, si falla o se corrompe, el sistema entero lo hace.
- **File System (Block Device) Replication:** es una modificación del anterior que permite ir copiando los cambios que ocurren en los archivos en otro dispositivo, asegurando que sean correctos. Así se solucionan los problemas de corrupción y fallo.

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

## OPERACIONES DESDE EL MASTER

Vamos a probar a insertar datos desde el Master, como hemos visto en la consulta anterior, podemos insertar datos desde el maestro sin ningún tipo de problema, replicándose en el esclavo.

```
insert into public."Grupos"(
    codigo_grupo, nombre, genero_musical, pais, sitio_web)
values (1515, 'Prueba', 'desde', 'master', 'Patri');
```

	codigo_grupo [PK] integer	nombre text	genero_musical text	pais text	sitio_web text
1	1515	Prueba	desde	master	Patri

Ahora vamos a probar a hacer un update desde el nodo maestro. Ejecutaríamos la siguiente consulta, ejecutándose correctamente y replicándose en el esclavo.

```
update public."Grupo"
set sitio_web = 'actualizado'
where codigo_grupo = 1515
```

Nos falta hacer una lectura de los datos de la tabla, ejecutamos esta consulta ejecutándose sin ningún problema.

```
select *
from "Grupos"
```

	codigo_grupo [PK] integer	nombre text	genero_musical text	pais text	sitio_web text
1	1515	Prueba	desde	master	actualizado

Por último, comprobaremos si es posible eliminar registros de la BD, ejecutando la siguiente consulta y viendo cómo se actualiza en el esclavo.

```
delete from "Grupos"
where codigo_grupo = 1515
```

Tras estas pruebas podemos comprobar que el nodo maestro funciona correctamente y puede realizar todos los tipos de consultas.

## OPERACIONES DESDE EL ESCLAVO

Probamos como en las operaciones del maestro una lectura desde el esclavo. Para ello ejecutamos la siguiente consulta:

```
select *  
from "Grupos"
```

	codigo_grupo [PK] integer	nombre text	genero_musical text	pais text	sitio_web text

Como podemos ver la replicación de los datos en el nodo esclavo funciona correctamente como cabía esperar ya que el registro con ID 1515 ya no está.

La siguiente prueba realizaremos una inserción de un nuevo registro en la base de datos:

```
insert into public."Grupos"(  
    codigo_grupo, nombre, genero_musical, pais, sitio_web)  
values (5151, 'Prueba', 'desde', 'esclavo', 'Patri');
```

A continuación, una eliminación de cualquier registro existente en nuestra base de datos:

```
delete from "Grupos"  
where codigo_grupo = 5151
```

Y por último trataremos de actualizar los datos de un registro ya existente en nuestra base de datos.

```
update public."Grupos"  
set sitio_web = 'actualizado'  
where codigo_grupo = 5151
```

Estas tres operaciones terminan con el mismo resultado por el output, un mensaje que indica que la única operación que puede realizar un servidor esclavo es la lectura.

```
ERROR: no se puede ejecutar INSERT en una transacción de sólo lectura  
SQL state: 25006
```

## OPERACIONES DE SIMULACIÓN DE CAÍDAS

### Caída del nodo maestro:

Como tenemos dos nodos maestros a la vez conectados, para simular una caída tendremos que detener el servidor mediante un comando en el CMD de Windows:

```
C:\Program Files\PostgreSQL\12\bin>pg_ctl stop -D "C:\Program Files\PostgreSQL\12\data"
```

Insertando este comando en la Shell conseguiremos mantener el esclavo operativo. Como se puede observar el esclavo funciona correctamente y se podría conseguir comenzar un proceso de recuperación de caídas si se detectan fallos en las conexiones.

La parada de alguno de los maestros supone que el otro maestro deja de tener acceso al otro nodo, si se intentase acceder a una de las tablas foráneas del otro nodo, se nos mostraría este mensaje de error:

```

53 select *
54 from "PECL4"."Grupos"
55
Data Output Explain Messages Notifications
ERROR: could not connect to server "server2"
DETAIL: no se pudo conectar con el servidor: Connection refused (0x0000274D/10061)
        Est el servidor en ejecución en el servidor 83.46.104.201 y aceptando
        conexiones TCP/IP en el puerto 5432?
SQL state: 08001

```

Deberíamos empezar un proceso de recuperación. Para ello tenemos que convertir el nodo esclavo del nodo maestro caído en el nuevo nodo maestro, para ello podemos usar `pg_ctl promote` desde el CMD. Con esto conseguiríamos que deje de tener un modo standby y el nodo tenga capacidad para realizar todas las operaciones y no solo la de lectura.

Para el caso de que el antiguo maestro se recuperase, habría que volver a invertir los papeles y hacer que el antiguo maestro volviera a su posición de maestro y el esclavo que actuaba de maestro volver a su estado anterior de esclavo. Para conseguirlo hay que volver a configurar las respectivas direcciones IP de los `postgres_fdw` y de los `pg_hba`.

### Caída del nodo esclavo:

Para simular una caída del nodo esclavo, basta con simplemente con apagar el ordenador o la máquina virtual en la cual se estuviera realizando la replica de ese nodo maestro. Esta caída no alteraría el funcionamiento el sistema ya que como se han podido ver en las pruebas que se acaban de realizar, la única función de una base de datos esclavo es leer datos, no modificarlos.

Para demostrar que ahora el esclavo no esta replicando aplicamos de nuevo este comando:

```

select pid, username, application_name, client_addr, backend_start, client_port, sync_state, state
from pg_stat_replication

```

Resultando esta ven en una respuesta en blanco ya que el nodo se ha caído.

pid	username	application_name	client_addr	backend_start	client_port	sync_state	state
integer	name	text	inet	timestamp with time zone	integer	text	text

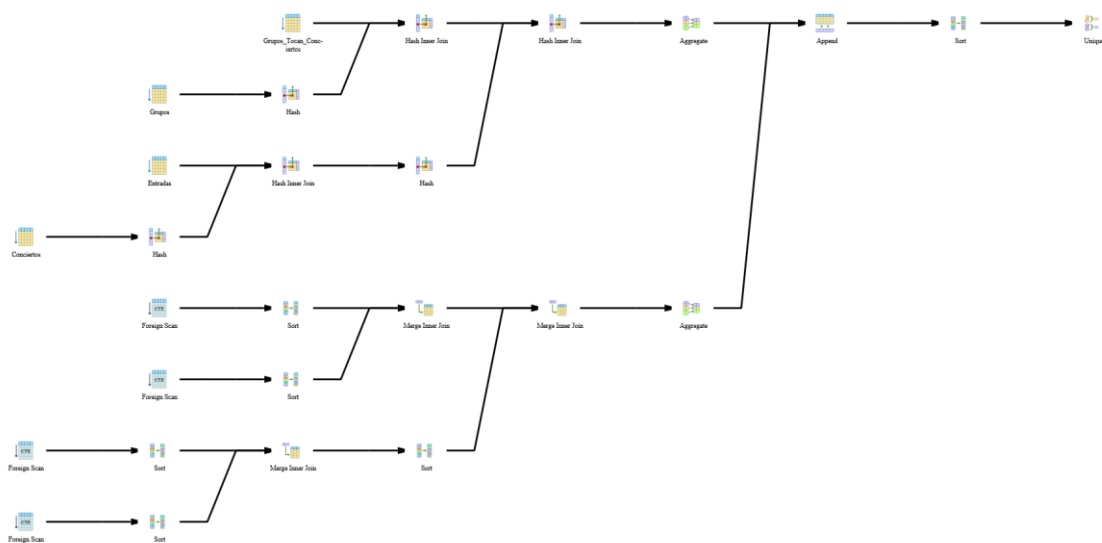
Hay un escenario peor, que sería en el que los dos nodos, es maestro y el esclavo se cayeran a la vez, esto provocaría una pérdida de datos que en ocasiones puede llegar a ser irrecuperable. Aquí es donde entraría el comando `restore_command` que usaremos para actualizar los datos mediante un recovery.

- 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 los grupos musicales junto con el número de entradas y precio recaudado en todos los conciertos que haya participado el grupo usando para ello toda la base de datos distribuida (MAESTRO1 + MAESTRO2). Explicar cómo se resuelve la consulta y su plan de ejecución.

**La consulta es la siguiente:**

```
select "Grupos".nombre, count(codigo_entrada), sum(precio)
from public."Grupos" inner join public."Grupos_Tocan_Conciertos" on "Grupos".codigo_grupo = "Grupos_Tocan_Conciertos"."codigo_grupo_Grupos"
inner join public."Conciertos" on "Grupos_Tocan_Conciertos"."codigo_concierto_Conciertos" = "Conciertos"."codigo_concierto"
inner join public."Entradas" on "Conciertos"."codigo_concierto" = "Entradas"."codigo_concierto_Conciertos"
group by "Grupos".nombre
union
select "Grupos".nombre, count(codigo_entrada), sum(precio)
from "PECL4"."Grupos" inner join "PECL4"."Grupos_Tocan_Conciertos" on "Grupos".codigo_grupo = "Grupos_Tocan_Conciertos"."codigo_grupo_Grupos"
inner join "PECL4"."Conciertos" on "Grupos_Tocan_Conciertos"."codigo_concierto_Conciertos" = "Conciertos"."codigo_concierto"
inner join "PECL4"."Entradas" on "Conciertos"."codigo_concierto" = "Entradas"."codigo_concierto_Conciertos"
group by "Grupos".nombre
```

**Y el árbol de ejecución generado con el comando explain es:**



La única diferencia circunstancial que podemos encontrar que distingue de cualquier consulta, es que al principio de esta en el árbol de ejecución se puede observar que aparecen varios `foreign scan`, que se refieren al escaneo de las tablas foráneas del otro nodo maestro ya que son necesarias para realizar la consulta.

La consulta de las tablas foráneas se puede ver que se encuentra en las ramas que parten de un escaneo foráneo y las consultas de las tablas locales no parten del mismo, una vez preprocesadas las consultas se unen con el comando `sql unión` que podemos ver en la consulta de arriba

- 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 caso de que el nodo maestro quedara inservible, se podría crear un nuevo maestro desde el esclavo que se tenía del nodo MAESTRO1. Si es servidor primario falla, el que se encuentra en espera se convierte en el nuevo primario y el antiguo primario procedería a reiniciarse, existe un mecanismo en el gestor de la base de datos llamado `STONITH` que informa al antiguo nodo primario de que este ya no es primario.

Para desencadenar este error en la conmutación por un error del maestro, hay que ejecutar `pg_ctl promote` o crear un archivo desencadenante con el nombre de la ruta y el archivo especificados por la configuración del `trigger_file` en el archivo `recovery.conf`. Si se intenta hacer por el primer método no es necesario crear ningún archivo de `trigger_file`. Otra opción que hay es la de el caso de la configuración de servidores de informes, que solo se utilizan para descargar consultas del primario en modo lectura y no para fines de alta disponibilidad de los servidores. Si es así, no es necesaria la promoción del servidor esclavo del nodo MAESTRO1.

Se recomienda, por tanto, en caso de error promocionar el esclavo del nodo caído a maestro, por su simplicidad frente a la creación de un `trigger_file`. Acabando así con dos maestros y un esclavo del nodo que aún sigue en pie.

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?

Si, puede haber inconsistencias, ya que dependiendo del tipo de replicación utilizado habrá distintos tipos de estas. Hay dos tipos de inconsistencias, de lectura y escritura.

Si el sistema de replicación utilizado es asíncrono, los casos de inconsistencia son más frecuentes que en el caso de los síncronos. Esto se debe a que el nodo maestro envía los datos del WAL al esclavo de forma asíncrona, sin esperar de ninguna manera a la confirmación por parte del nodo esclavo de que los ha recibido. Por tanto, en la replicación asíncrona se pueden ver los dos tipos de inconsistencias: la de lectura porque puede que los datos que se han leído del nodo esclavo son distintos que los del maestro, al haber la posibilidad de que aun los datos no se hayan actualizado, o también la inconsistencia de escritura ya que no se puede garantizar que el WAL escrito previamente por el maestro se haya escrito correctamente en el esclavo, estando así descompensados. Además, si por algún casual se cae el servidor maestro sin haber podido enviar los datos del WAL actualizados al nodo esclavo podemos encontrarnos con una pérdida de datos irrecuperable.

Si el sistema de replicación por otra mano usa un método síncrono, también pueden aparecer inconsistencias. Ya que en este caso siempre se escriben datos en el nodo maestro, espera a la confirmación del esclavo de que le han llegado los datos actualizados y posteriormente los escribe. Esto provoca una capacidad de evitar inconsistencias mayores que la del método de replicación asíncrona.

Este método nos proporciona una mayor durabilidad y confianza del usuario, ya que, la espera de confirmación por parte del esclavo garantiza un ameno pérdida de datos.

Para establecer una conexión síncrona hay dos opciones que activan el `synchronous_commit`:

- `Remote_apply`: en este caso además de la espera por parte del nodo maestro de confirmación del nodo esclavo, también espera a la aplicación de estos nodos antes de enviar cualquier dato al cliente. Evitando así la pérdida de datos salvo que se caigan ambos nodos a la vez.
- `Remote_write`: el nodo maestro para esta configuración síncrona espera la confirmación del nodo esclavo de que lo ha escrito en su propio SO, pero no espera a que los datos se descarguen en el disco del nodo esclavo. Por lo tanto, se puede producir una inconsistencia de lectura, ya que puede darse el caso de que un cliente lea un dato antes de que se aplique la escritura completa en su disco y los datos leídos no coincidan con los últimos actualizados.

## 7. Conclusiones.

**En conclusión, el método que hemos llevado a cabo, del streaming replication es el que más cómodo ha resultado de implementar y el que más ventajas consideramos que nos puede proporcionar, ya que, tiene varias ventajas sobre el método del envío del WAL (log-shipping). El corto tiempo de demora entre la confirmación de una transacción en el nodo maestro y los cambios que se hacen visibles en el nodo esclavo, que es prácticamente inmediato como también la posibilidad de pérdida de datos es muy pequeña, demuestran que es la opción más válida de replicación. Además, el método de implementación aplicado ha sido síncrono por las ventajas que ofrece como se explicó previamente y la poca complejidad que añade su uso.**

**Consideramos que estos métodos de replicación son muy útiles a la hora de tratar con bases de datos muy grandes, ya que, así podemos garantizar una continuidad de esta en caso de la caída de la base de datos principal (nodo maestro) y poder recuperarla a partir del nodo esclavo sin la pérdida de datos.**

**No nos ha sido excesivamente complicado llevar a cabo la práctica, ya que, en el manual hemos encontrado suficiente información como para realizarlo de una forma correcta, quizá lo que más nos ha penalizado a la hora de la realización ha sido el desconocimiento previo sobre los procesos que se nos pedía realizar y su funcionamiento.**

**El único problema serio que hemos encontrado en el desarrollo de esta práctica es la realización de los dos nodos maestros, los cuales nos ha sido imposible realizar los dos ya que por parte del ordenador de Álvaro ha sido imposible realizar la replicación, por lo que solo hemos replicado el nodo maestro del ordenador de Patricia.**

**Como conclusión sacamos que con esta práctica hemos aprendido a establecer conexiones entre distintos servidores creando bases de datos distribuidas y ejecutando consultas remotamente con postgres\_fdw. A su vez, hemos sabido diferenciar nodos hot\_standby comprendiendo su funcionamiento y las ventajas que aportan.**

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.

## **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