

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:** Laura Mambrilla Moreno

**DNI:** \_\_\_\_\_

### **ALUMNO 2:**

**Nombre y Apellidos:** Isabel Blanco Martínez

**DNI:**

**Fecha:** 30 de mayo de 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 Suspenseo – Cero.**

## **Plazos**

Tarea online: Semana 27 de Abril y 4 de Mayo.

Entrega de práctica: Día 8 de junio. 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****

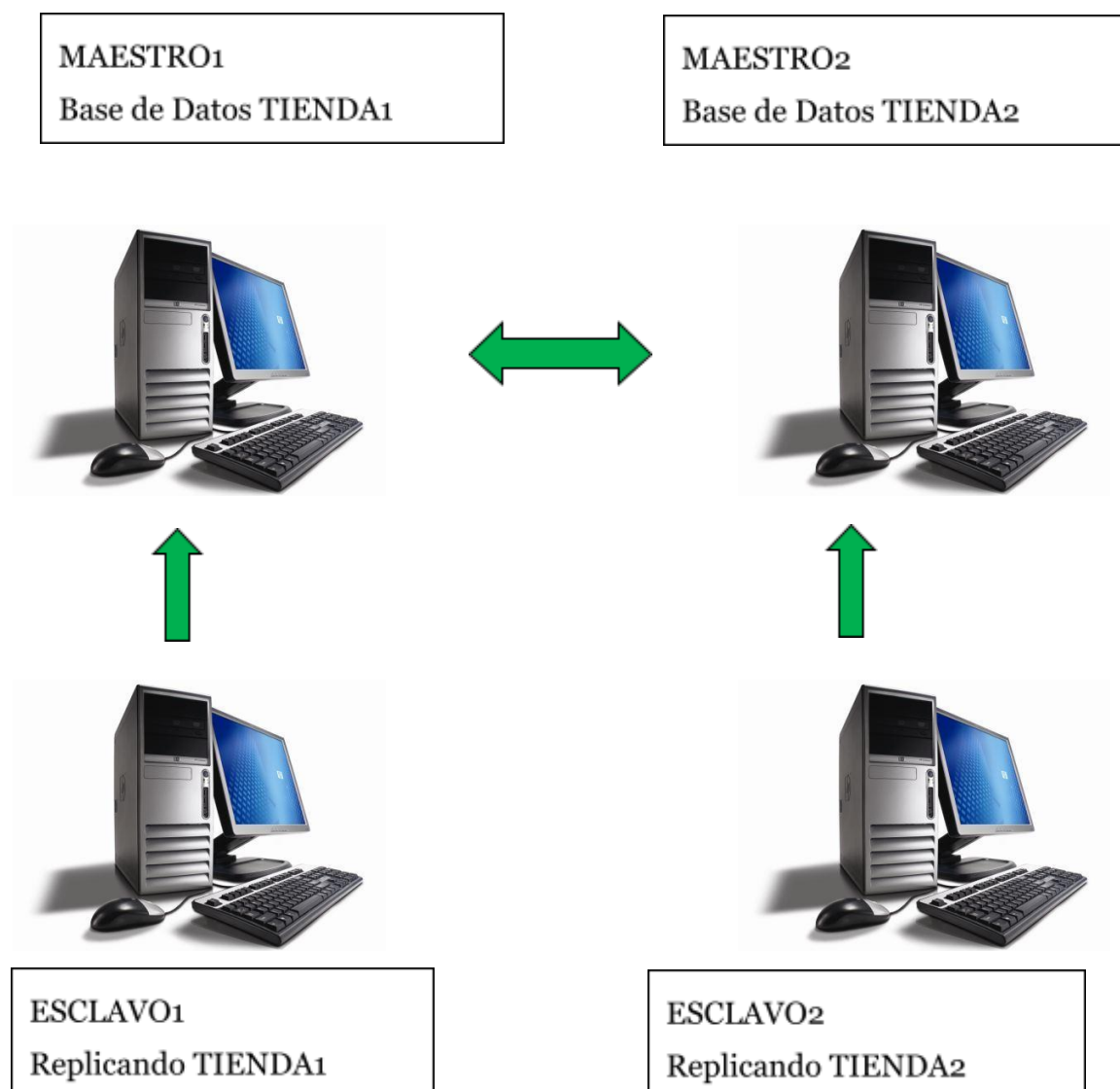
AMBOS ALUMNOS DEBEN ENTREGAR EL FICHERO EN LA PLATAFORMA.

## 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 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.

# Índice

1. Introducción
2. Establecemos conexión
3. postgres\_fdw
4. Replicación
  1. Tipos de replicación
  2. Replicación streaming
    1. Configuración del servidor maestro
    2. Configuración del servidor esclavo o standby
  3. Pruebas de replicación
  4. Tipo de nodo maestro
  5. Tipo de nodo esclavo
5. Operaciones en cada tipo de equipo red
  1. Operaciones desde master
  2. Operaciones desde esclavo
6. Simulación de caídas
  1. Caída de nodo esclavo
  2. Caída de nodo maestro
7. Resolución de consultas
  1. Inserción de datos en las tablas
  2. Consulta
8. Escenario maestro1 inservible
  1. Acciones para volver al funcionamiento normal
  2. Configuración de nodos
  3. Pruebas del correcto funcionamiento
9. Inconsistencias base de datos de maestro y esclavo
10. Conclusión
11. Bibliografía

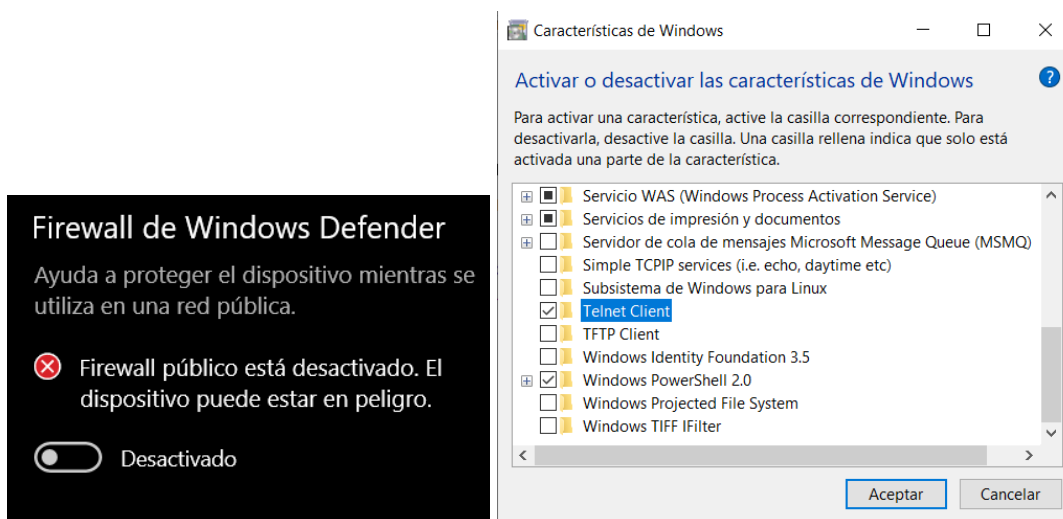
## 1. Introducción

En esta memoria analizaremos las distintas opciones a la hora de replicar una base de datos en PostgreSQL, además de conectar nuestro servidor maestro con otro maestro para compartir nuestra base de datos mediante el uso de la extensión `postgres_fdw`.

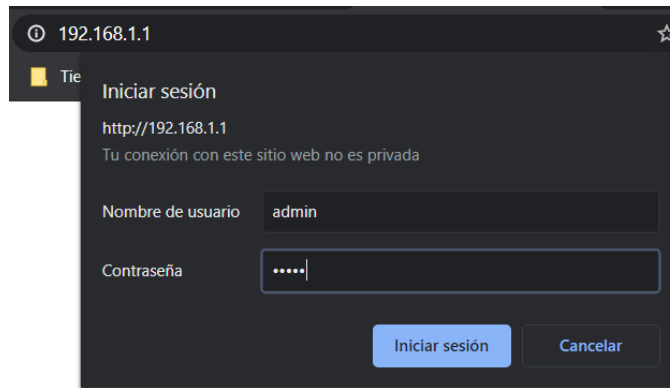
PostgreSQL nos ofrece un amplio abanico de métodos para llevar a cabo la replicación y para asegurar la disponibilidad de las bases de datos. A lo largo de esta memoria los iremos estudiando y viendo las características que les diferencian.

## 2. Establecemos conexión

Para poder acceder a la base de datos de nuestra compañera a través de nuestro router, tendremos que abrir el puerto del router (el que utiliza PostgreSQL es el 5432), pero antes tenemos que desactivar el firewall, ya que puede estar bloqueando los puertos. También activaremos el Telnet Client para comprobar que tenemos accesibilidad al puerto requerido.



Seguidamente, en la zona de búsqueda por URL del navegador ponemos 192.168.1.1, lo cual nos conducirá a la configuración del router. La dirección 192.168.1.1 es una IP privada que no se utiliza en Internet, y que se reserva para el uso exclusivo de los dispositivos en una red local. Primero debemos iniciar sesión, ya que por defecto el usuario es *admin* y la contraseña también es *admin*.



Según el operador que tengas, aparecerá una interfaz distinta, pero el contenido de todas es el mismo o muy similar, por lo que ambas buscamos un apartado donde podemos introducir la dirección IP local, el número del puerto inicial y el número de puerto final. En nuestros casos se encuentran en los siguientes apartados:

- **Caso de Isabel (Movistar):** Si accedemos a la configuración del router podemos ver un apartado (Puertos) en el que podemos modificar o eliminar los puertos. En este caso escogemos configuración avanzada para poder definir la dirección IP, en la que ponemos la ruta privada de mi ordenador.

## Configuración del router



**Mi Router**  
**Mitrastar GPT-2541GNAC**

- Línea teléfono: 925824082
- Firmware: ES\_g3.5\_100VNJob54\_7
- Conectado: Sí

[Mi configuración actual](#) Firmware actualizado.

Wifi / Contraseña

Wifi Plus / Contraseña

Puertos

Otras operaciones

Mis configuraciones

Red local

### MIS PUERTOS

Modifica o elimina tus puertos.

<input type="checkbox"/>	Nombre	Dirección IP	Protocolo	Interno Inicio	Interno Fin	Externo Inicio	Externo Fin
<input type="checkbox"/>	Puerto 1	192.168.1.41	TCP+UDP	5432	5432	5450	5450

Movistar te obliga a poner un puerto Interno y Externo distinto para asegurar el acceso al puerto. Elegimos como protocolo TCP + UDP.

- **Caso de Laura (Jazztel): Configuración Avanzada → NAT → Servidor Virtual**

Seleccione Idioma: ▼

**Router ADSL**



Estado	Configuración Básica	Configuración Avanzada	Gestión Acceso	Mantenimiento	Estado	Ayuda
	Información del Dispositivo	Log del Sistema	Configuración e-mail	Estadísticas		

Configuración Básica	Configuración Avanzada	Gestión Acceso	Mantenimiento	Estado
Firewall	Enrutamiento	NAT	ADSL	QoS
PortBinding	Port Mirror			

Servidor Virtual para : Única Cuenta/PVC IP0  
 Número de Puerto Inicial :   
 Número de Puerto Final :   
 Dirección IP Local :

Regla	Puerto Inicial	Puerto Final	Dirección IP Local	Editar	Eliminar
0	5432	5432	192.168.1.132		

Se puede saber si el puerto está o no abierto poniendo la IP pública y el puerto que quieres comprobar en esta página: <https://www.testdevelocidad.es/test-de-puertos/>.

**Prueba los puertos de tu ip**

**Dirección IP**

**Aplicaciones**

**Puertos**

Puerto	Resultado
5432	✓ Abierto

Otra manera de saberlo es ejecutar en la terminal el comando `telnet <IP privada> <puerto>` con el que establecemos conexión con el puerto que queremos.

```
Símbolo del sistema

C:\Users\laura>telnet 192.168.1.132 5432
```

```
Telnet 192.168.1.132
```

En el archivo **pg\_hba.conf** debemos añadir la línea señalada, poniendo la IP pública del que quieres establecer conexión. Le añadimos de método md5 que es un algoritmo de encriptación.

Una vez hecho esto, reiniciamos el servidor.

```
# TYPE DATABASE USER ADDRESS METHOD
# IPv4 local connections:
host all all 127.0.0.1/32 md5
host all all 81.35.76.6/32 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
```

```
# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all md5
# IPv4 local connections:
host all all 127.0.0.1/32 md5
host all all 195.133.117.211/32 md5
# IPv6 local connections:
host all all ::1/128 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local replication all md5
host replication all 127.0.0.1/32 md5
host replication all ::1/128 md5
```

En el archivo **postgresql.conf** tenemos que irnos al apartado de CONNECTIONS AND AUTHENTICATION y prestar especial atención a los siguientes puntos:

```
#-----
# 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)
```

- **port = 5432**

Podemos ver y configurar el puerto en el cual iniciamos PostgreSQL desde pgAdmin.



- **listen\_addresses = '\*'**

Aquí especificamos las direcciones IP que queremos permitir que accedan a nuestro servidor. Si ponemos '\*' estamos refiriéndonos a que permitimos que cualquier dirección pueda acceder.

Desde el apartado WRITE-AHEAD nos fijamos en wal\_level.

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

# - Settings -

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

- **wal\_level**

Define cuánta información se grabará en los archivos WAL generados. Lo mantenemos en replica.

Un poco más abajo desde el mismo WRITE-AHEAD nos vamos a -Archiving- y nos aseguramos de que esté en on.

```
# - Archiving -

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

- **archive\_mode = on**

Se activa la generación de archivos WAL en el servidor

Desde el apartado REPLICATION nos fijamos en max\_wal\_senders y wal\_keep\_segments.

```
#-----
# 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
```

- **max\_wal\_senders**

Indica las conexiones concurrentes o simultáneas 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”.

**En resumen, ejecutamos los siguientes cambios en la configuración de los máster:**

```
listen_addresses = '*'  
port = 5432  
wal_level = replica  
archive_mode = on  
max_wal_senders = 10  
wal_keep_segments = 30
```

### 3. postgres\_fdw

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.

Para utilizar postgres\_fdw, primero debemos:

1. Crear la extensión postgres\_fdw
2. Crear un servidor para cada base de datos. (host = IP pública del compañero)
3. Hacer el mapeo para poder leer la base de datos del compañero.
4. Importación del 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 completar este paso hemos creado un Schema nuevo denominado PECL4, para evitar posibles errores.

### Resultado final en el maestro de Isabel:

```
--creamos extensión postgres_fdw
CREATE EXTENSION postgres_fdw;

--creamos server2
CREATE SERVER server2 FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host '195.133.117.211',
dbname 'TIENDA2', port '5432');

--creamos el mapeado para conectar con el ordenador de Laura
CREATE USER MAPPING FOR postgres SERVER server2 OPTIONS(user 'postgres', password '299792458');

-- Importamos foreign schema
IMPORT FOREIGN SCHEMA public FROM SERVER server2 INTO "PECL4";
```

### Resultado final en el maestro de Laura:

```
1  --creamos extensión postgres_fdw
2  CREATE EXTENSION postgres_fdw;
3
4  --server1 porque hago referencia a TIENDA1
5  CREATE SERVER server1 FOREIGN DATA WRAPPER postgres_fdw
6  OPTIONS (host '81.35.76.6', dbname 'TIENDA1', port '5450');
7
8  --creamos el usuario para conectar con el ordenador de Isa
9  CREATE USER MAPPING FOR postgres SERVER server1
10 OPTIONS(user 'postgres', password '900799');
11
12 -- (Mapeamos) Importamos foreign schema (previamente hemos creado el SCHEMA PECL4
13 IMPORT FOREIGN SCHEMA public FROM SERVER server1 INTO "PECL4";
```

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.

(Desde la base de datos de Laura TIENDA2) Probamos a modificar las BBDD desde Tienda2, añadiendo una tienda a la base de datos de Isabel desde la de Laura y después otra en la de Laura desde ella misma...



```
1  -- estamos en la base de datos TIENDA2 (Laura)
2
3  -- insertamos tienda en TIENDA1 (Isabel)
4  INSERT INTO "PECL4"."Tienda"(
5      "Id_tienda", "Nombre", "Ciudad", "Barrio", "Provincia")
6      VALUES (2, 'FUNCIONA2', 'sdfg', 'sdfg', 'knjbhvg');
7
8  -- insertamos tienda en TIENDA2 (Laura)
9  INSERT INTO public."Tienda"(
10     "Id_tienda", "Nombre", "Ciudad", "Barrio", "Provincia")
11     VALUES (2, 'FUNCIONA2', 'sdfg', 'sdfg', 'knjbhvg');
```

...y comprobamos que en el schema PECL4 (la base de datos de Isabel TIENDA1)  
→ Foreign Tables → Tienda aparece la tienda que hemos introducido desde TIENDA2...

Query Editor

```
1 SELECT * FROM "PECL4"."Tienda"
```

Data Output

	Id_tienda integer	Nombre text	Ciudad text	Barrio text	Provincia text
1	1	FUNCIONA	sdfg	sdfg	knjbhvg
2	2	FUNCIONA2	sdfg	sdfg	knjbhvg

...y que en el SCHEMA public (la base de datos de Laura TIENDA2) → Tables → Tienda aparece la tienda que habíamos insertado:

Query Editor

```
1 SELECT * FROM public.\"Tienda\"
```

Data Output

	Id_tienda [PK] integer	Nombre text	Ciudad text	Barrio text	Provincia text
1	2	FUNCIONA2	sdfg	sdfg	knjbhvg

## 4. Replicación

### 4.1 Tipos de replicación

En una base de datos replicada, todos los cambios realizados a los datos deben ser aplicados en las múltiples copias, todas las copias tienen que ser idénticas. Encontramos dos tipos

- Replicación síncrona, todos los cambios son enviados a todas las copias nada más realizarse.
- Replicación asíncrona, maneja la propagación de los datos con un breve retraso desde la realización del cambio, esta es la que viene por defecto en PostgreSQL.

### 4.2 Replicación streaming

#### 4.2.1 Configuración del servidor maestro

Desde **postgresql.conf**, debemos poner una ruta en *archive\_command* en el maestro. El servidor en espera puede leer WAL desde un archivo WAL (*restore\_command*) o directamente desde el maestro a través de una conexión TCP (streaming).

Para ello en *archive\_command* debemos poner el directorio en el que se encuentra el directorio *pg\_wal*, donde están todos los archivos WAL, y definir *primary\_conninfo* en el esclavo más adelante.

```
archive_command = ????  
  
# - 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
```

Que no se nos olvide comprobar que *listen\_addresses* está activado y al menos con \* para aceptar todas las conexiones ajenas a la nuestra.

```
#-----
# 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
```

Después, vamos al archivo `pg_hba.conf` y debemos añadir un rol nuevo para permitir la replicación. El modo del campo de la base de datos debe estar en `replication`. Añadimos la dirección IP privada de nuestro esclavo.

```
# Allow replication connections from localhost, by a user with the
# replication privilege.
local    replication    all                                     md5
host     replication    all         127.0.0.1/32              md5
host     replication    all         ::1/128                  md5
host     replication    all         192.168.1.56/32            md5
```

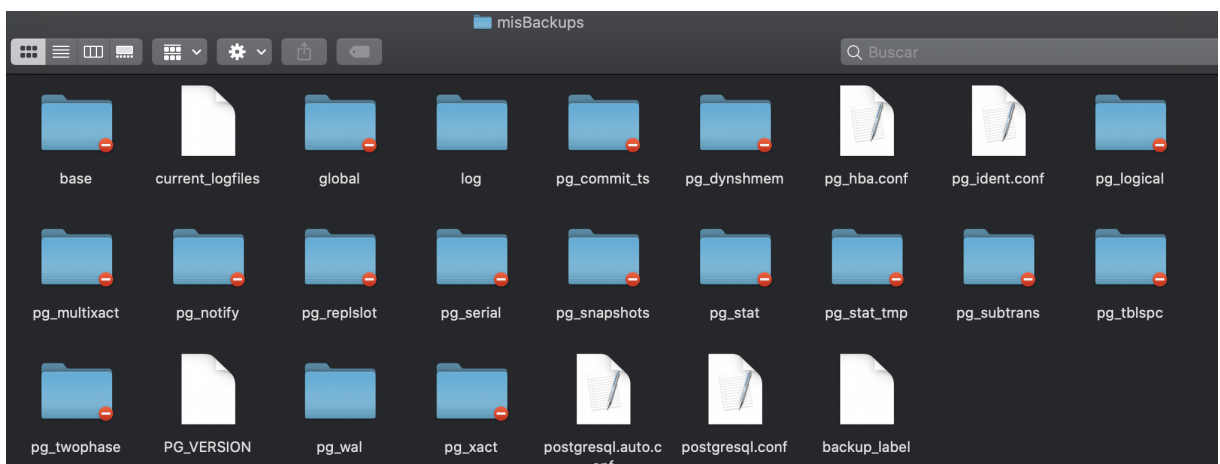
Lo primero que haremos tras configurar el maestro será realizar un backup de nuestro actual directorio de data (desde el Máster) para después, desde el esclavo, sustituir todo el directorio con el backup del máster.

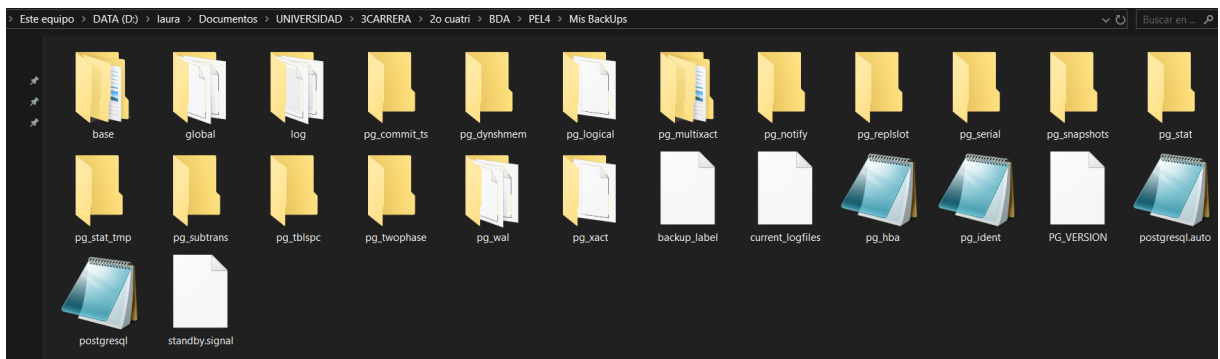
Para hacer el backup usaremos el siguiente comando:

```
Isabels-MacBook-Pro:bin postgres$ ./pg_basebackup -h localhost -p 5432 -U postgres -R -D /Library/PostgreSQL/12/data/pg_wal/misBackups
Password:
Isabels-MacBook-Pro:bin postgres$
```

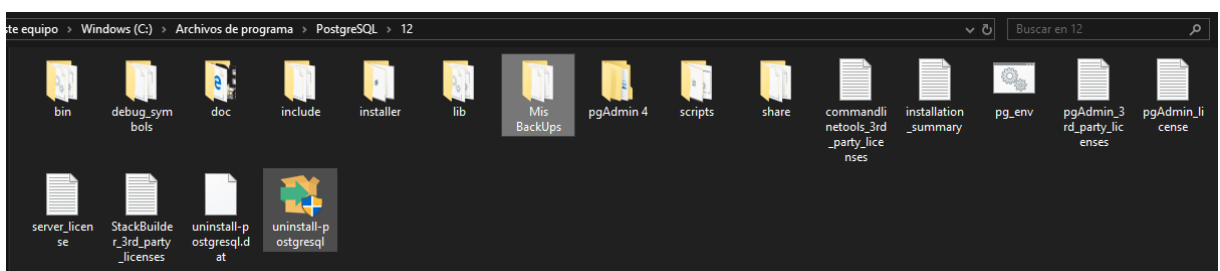
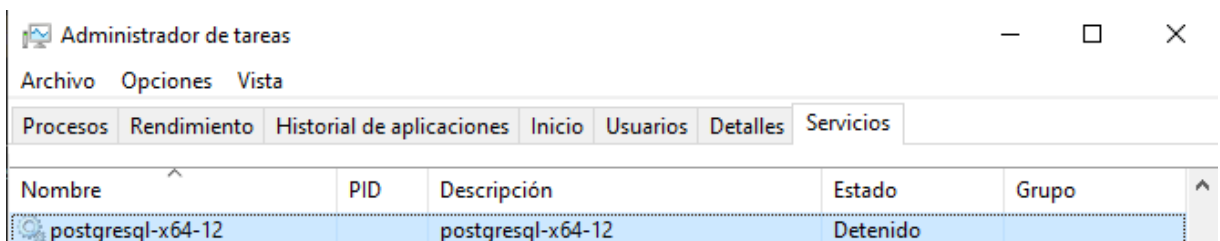
```
C:\Program Files\PostgreSQL\12\bin>pg_basebackup -h localhost -p 5432 -U postgres -R -D "D:\laura\Documents\UNIVERSIDAD\3CARRERA\2o cuatri\BDA\PEL4\Mis BackUps"
Contraseña:
```

Si nos vamos a dicho directorio podemos ver que su contenido se ha volcado de forma correcta

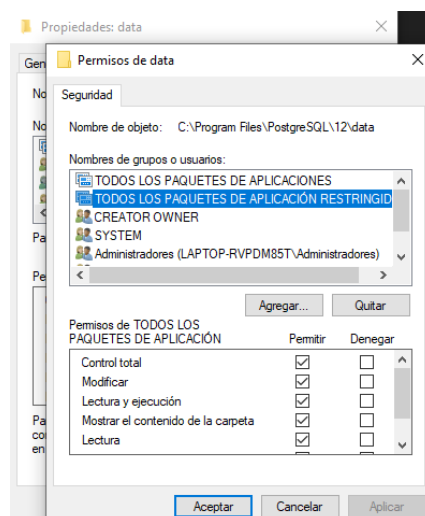




Ahora, nos vamos a nuestro ordenador o máquina virtual desde el que vamos a hacer la replicación y el primer paso que debemos realizar es eliminar la carpeta de data y sustituirla con la base de datos recién obtenida del backup del master, siempre y cuando el servidor esté detenido (seguimos respetando el nombre de "data").



Para asegurarnos, debemos conceder todos los permisos a los archivos de la carpeta "12":

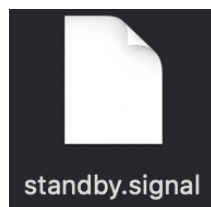


Para facilitar la realización, vamos a actualizar el archivo de `pg_hba` anteriormente modificado para especificar la dirección IP privada de nuestro esclavo como `0.0.0.0/0` para que cualquier dirección pueda hacer replica de nuestros datos, ya que las direcciones IP de nuestros ordenadores son dinámicas. Otra forma de hacerlo sería configurar manualmente la IPv4 de nuestro ordenador y hacer que la dirección IP sea estática (en la imagen es la última línea).

```
# Allow replication connections from localhost, by a user with the
# replication privilege.
local    replication    all                                     md5
host     replication    all         127.0.0.1/32             md5
host     replication    all         ::1/128               md5
host     replication    all         0.0.0.0/0              md5
```

#### 4.2.2 Configuración del servidor esclavo o standby

El archivo **standby.signal** es creado en nuestro cluster al hacer el backup.



Opcionalmente, podríamos definir el comando `archive_cleanup_command` para hacer que se borren los archivos de WAL para que no tengamos archivos innecesarios, pero como no vamos a insertar muchos datos lo dejaremos como está.

```
archive_cleanup_command = 'pg_archivecleanup
                          /Library/PostgreSQL/12/data/pg_wal/ %r'
```

Debemos añadir `primary_conninfo`, incluyendo la IP address del maestro, así como el usuario y la contraseña en caso de tenerla.

Isabel:

```
# - Standby Servers -

# These settings are ignored on a master server.

primary_conninfo = 'host=192.168.1.41 port=5450 user=postgres password=900799' # connection string to sending server
```

Laura:

```
# - Standby Servers -

# These settings are ignored on a master server.

primary_conninfo = 'host=192.168.1.132 port=5432 user=postgres password=299792458' # connection string to sending server
```

Una vez terminado de configurar, podemos iniciar el servidor.



```
C:\Program Files\PostgreSQL\12\bin>pg_ctl start -D "C:\Program Files\PostgreSQL\12\data"
esperando que el servidor se inicie...2020-06-04 15:12:44.191 CEST [1252] LOG: starting PostgreSQL 12.3, compiled by Visual C++ build 1914, 64-bit
2020-06-04 15:12:44.200 CEST [1252] LOG: listening on IPv6 address "::", port 5432
2020-06-04 15:12:44.202 CEST [1252] LOG: listening on IPv4 address "0.0.0.0", port 5432
2020-06-04 15:12:44.401 CEST [1252] LOG: redirecting log output to logging collector process
2020-06-04 15:12:44.401 CEST [1252] HINT: Future log output will appear in directory "log".
... listo
servidor iniciado
```

## 4.3 Pruebas de replicación

Para comprobar que una vez iniciado el servidor y hayamos abierto pgAdmin en ambos ordenadores, ejecutaremos la siguiente consulta, la cual nos devolverá información acerca de los clientes que están replicando nuestras bases de datos.

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

El resultado de la consulta es:

pid	username	application_name	client_addr	backend_start	client_port	sync_state	state	reply_time
integer	name	text	inet	timestamp with time zone	integer	text	text	timestamp with time zone
1	45067	postgres	walreceiver	192.168.1.50	2020-06-04 20:51:51.823709+02	61859	async	streaming
				2020-06-04 20:52:08.221762+02				

Query Editor Query History Explain Notifications													
1 select * from pg_stat_replication													
Data Output Messages													
pid	usesysid	username	application_name	client_addr	client_hostname	client_port	backend_start	backend_xmin	state	sent_lsn	write_lsn	flush_lsn	
integer	oid	name	text	inet	text	integer	timestamp with time zone	xid	text	pg_lsn	pg_lsn	pg_lsn	
1	14484	10 postgres	walreceiver	192.168.1.130	[null]	51000	2020-06-05 00:48:55.236565+02		[null]	streaming	10/AD000148	10/AD000148	10/AD000148

Podemos ver que aparece en modo replicación en Streaming un host con el correspondiente address 192.168.1.50.

También podemos comprobarlo insertando un registro desde el máster y comprobando que nos aparezca desde el esclavo por ejemplo:

Desde el master ejecutamos la siguiente consulta:

```
INSERT INTO public."Tienda" (
    "Id_tienda", "Nombre", "Ciudad", "Barrio", "Provincia")
VALUES (20202020, 'Prueba replicación', 'Insertando', 'Desde', 'Master Isa');
```

Y ahora desde el ordenador que utilizo como esclavo ejecuto:

```
SELECT * FROM "Tienda" WHERE "Id_tienda"=20202020;
```

Y podemos ver que efectivamente se nos muestra la consulta desde el esclavo:

	Id_tienda [PK] integer	Nombre text	Ciudad text	Barrio text	Provincia text
1	20202020	Prueba replic...	Insertando	Desde	Master Isa

## 4.4 Tipo de nodo maestro

Según la documentación, hemos podido comprobar que PostgreSQL nos proporciona dos métodos para realizar consultas a servidores externos:

### ❑ Dblink

Su ventaja es que es sencillo de implementar al requerir escasa configuración para poder conectarse a otras bases de datos. A diferencia de `postgres_fwd` que mencionaremos en el siguiente apartado, Dblink no soporta el modo de solo lectura, ni sigue el estándar SQL. Es compatible con las versiones antiguas de PostgreSQL y las conexiones sólo perduran durante la misma sesión.

### ❑ postgres\_fwd

A diferencia de Dblink, se implementó a partir de la versión 9.3 de PostgreSQL por lo que no es compatible con las versiones anteriores. Supera a Dblink ya que soporta el modo lectura y es más eficiente. Aunque como todo, tiene sus desventajas, que son la retrocompatibilidad y las conexiones existentes. Hemos decidido usar este método ya que es la opción que más ventajas aporta y creemos que podría ser la más eficiente. Ya hemos visto cómo usar este módulo en la realización de los apartados anteriores.

## 4.5 Tipo de nodo esclavo

Según la documentación de PostgreSQL, en el capítulo 26.1 hemos podido comprobar todos los tipos de nodo esclavo que podemos utilizar y sus diferencias:

### ❑ 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.

## 5. Operaciones en cada tipo de equipo red

### 5.1 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. Se replica en el esclavo.



```
INSERT INTO public."Tienda"(  
  "Id_tienda", "Nombre", "Ciudad", "Barrio", "Provincia")  
VALUES (20202020, 'Prueba replicación', 'Insertando', 'Desde', 'Master Isa');
```

Ahora vamos a probar a hacer un **update** desde el maestro. Ejecutamos la siguiente consulta. También se nos ejecuta correctamente y se replica al esclavo.

```
UPDATE public."Tienda"  
SET "Ciudad"='Actualizando'  
WHERE "Id_tienda"=20202020;
```

Nos falta hacer una **lectura** de los datos de la tabla. Ejecutamos la siguiente consulta, la cual se ejecuta sin ningún problema:

```
SELECT "Id_tienda", "Nombre"  
FROM "Tienda";
```

	 Id_tienda [PK] integer	 Nombre text
1	1	fUNCIONA
2	2	fUNCIONA2
3	20202020	Prueba replicación

Por último, comprobaremos si podemos **eliminar** registros. Ejecutamos la siguiente consulta y vemos que desaparece del esclavo.

```
DELETE FROM "Tienda" WHERE "Id_tienda" = 20202020;
```

Después de estas pruebas podemos ver que para el nodo maestro, podrá realizar cualquier operación sobre el conjunto de datos.

## 5.2 Operaciones desde el esclavo

Probamos a hacer una **lectura** de los datos desde el esclavo. Para ello ejecutamos la siguiente consulta:

```
SELECT "Id_tienda","Nombre" FROM "Tienda";
```

	Id_tienda [PK] integer	Nombre text
1		1 fUNCIONA
2		2 fUNCIONA2

Como podemos comprobar a raíz de lo deducido en las demostraciones de las operaciones del servidor maestro, se replica correctamente y el registro con ID 20202020 ha desaparecido.

La siguiente prueba que haremos será **insertar** un nuevo registro a la base de datos TIENDA1.

```
INSERT INTO public."Tienda" VALUES (212121, 'Inserto', 'Desde', 'El', 'Esclavo');
```

Ahora probaremos a **eliminar** un registro existente de la base de datos TIENDA1.

```
DELETE FROM "Tienda" WHERE "Id_tienda" = 1;
```

Y por último vamos a tratar de **actualizar** los datos de algún registro ya existente.

```
UPDATE "Tienda" SET "Nombre"='Esclavo' WHERE "Id_tienda" = 1;
```

Las tres últimas operaciones terminaban con el mismo resultado, un mensaje en el output que nos indica que la única operación que un servidor esclavo es capaz de realizar es la de lectura.

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

Después de estas pruebas podemos ver que para el nodo esclavo, las operaciones que puede realizar están limitadas, porque se restringe a los contenidos de las tablas.

## 6. Simulación de caídas

### 6.1 Caída nodo esclavo

Para simular la caída del nodo esclavo, basta simplemente con apagar el ordenador en el cual estuviéramos llevando a cabo la réplica.

Esta caída prácticamente no altera el funcionamiento del sistema ya que como hemos visto en las pruebas que acabamos de realizar, su única función es leer datos y replicar los WAL.

Para demostrar que ahora el esclavo no está replicando volvemos a aplicar el comando:

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

Pero en esta ocasión obtenemos el registro en blanco, ya que hemos provocado que el nodo caiga.

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

Habría un escenario aún peor, que sería el caso en que el nodo esclavo se caiga al mismo tiempo que el nodo maestro, ya que esto provocaría una pérdida de datos que en ocasiones puede ser irreparable. Aquí es donde entraría el uso del comando `restore_command` que lo usaríamos para actualizar los datos mediante el `recovery`.

### 6.2 Caída nodo maestro

Como tenemos dos nodos maestros conectados, para simularla tendremos que detener el servidor mediante la CMD usando

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

Haciendo esto conseguiremos mantener el esclavo. Como podemos ver el esclavo sigue funcionando correctamente y podríamos conseguir que comenzase el proceso de recuperación ante caídas si detecta fallos en las conexiones.

La parada de uno de los maestros supondrá que el otro maestro deja de tener acceso a los datos de la base de datos del otro maestro. Si intentamos acceder a una de las tablas foráneas del otro maestro lo podemos comprobar:

```
PECL4.Tienda/TIENDA2/postgres@PostgreSQL 12
Query Editor  Query History  Explain  Notifications
1  SELECT * FROM "PECL4"."Tienda"
2
Data Output  Messages
ERROR:  could not connect to server "server1"
DETAIL:  no se pudo conectar con el servidor: Connection refused (0x0000274D/10061)
         Est el servidor en ejecución en el servidor 81.35.76.6 y aceptando
         conexiones TCP/IP en el puerto 5450?
SQL state: 08001
```

Debemos empezar el proceso de recuperación. Para ello tenemos que convertir el esclavo en maestro. Para conseguirlo podemos usar `pg_ctl promote`, que podemos activar mediante la CMD.

Con esto conseguiremos que deje de tener el modo standby y tenga capacidad para realizar más operaciones que sólo lectura.

En caso de que el antiguo maestro se recuperase, habría que volver a invertir los papeles y hacer que el antiguo maestro volviese a ser el maestro y el actual maestro el esclavo.

Para conseguirlo deberíamos de volver a configurar sus respectivas direcciones IP de los `postgres_fwd` y de los `pg_hba`.

## 7. Resolución de consultas

### 7.1 Inserción de datos en las tablas

Datos a insertar en TIENDA1:

```
INSERT INTO public."Tienda"
VALUES (76543, 'Tienda1','Ciudad1', 'Barrio1', 'Provincia 1');

INSERT INTO public."Tienda"
VALUES (75555, 'Tienda2','Ciudad2', 'Barrio2', 'Provincia 2');

INSERT INTO public."Tienda"(
    "Id_tienda", "Nombre", "Ciudad", "Barrio", "Provincia")
VALUES (75000, 'Tienda3','Ciudad3', 'Barrio3', 'Provincia 3');

INSERT INTO public."Trabajador"(
    codigo_trabajador, "DNI", "Nombre", "Apellidos", "Puesto", "Salario", "Id_tienda_Tienda")
```

```

VALUES (111111, 111111, 'Nombre1', 'Apellido1', 'reponedor', 654, 76543);

INSERT INTO public."Trabajador"(
    codigo_trabajador, "DNI", "Nombre", "Apellidos", "Puesto", "Salario", "Id_tienda_Tienda")
VALUES (222222, 222222, 'Nombre2', 'Apellido2', 'cajero', 34, 75555);

INSERT INTO public."Trabajador"(
    codigo_trabajador, "DNI", "Nombre", "Apellidos", "Puesto", "Salario", "Id_tienda_Tienda")
VALUES (333333, 333333, 'Nombre3', 'Apellido3', 'reponedor', 567, 75000);

INSERT INTO public."Ticket"(
    "N_de_ticket", "Importe", fecha, "codigo_trabajador_Trabajador")
VALUES (8888888, 8754, '01-01-2020', 111111);

INSERT INTO public."Ticket"(
    "N_de_ticket", "Importe", fecha, "codigo_trabajador_Trabajador")
VALUES (666666, 87, '01-03-2020', 222222);

INSERT INTO public."Ticket"(
    "N_de_ticket", "Importe", fecha, "codigo_trabajador_Trabajador")
VALUES (444566, 387, '04-03-2020', 333333);

INSERT INTO public."Productos"(
    "Codigo_de_barras", "Nombre", "Tipo", "Descripcion", "Precio")
VALUES (00000, 'Jamon', 'Rico', 'Hacendado', 6);

INSERT INTO public."Ticket_Productos"(
    "Cantidad", "Codigo_de_barras_Productos", "N_de_ticket_Ticket")
VALUES (2, 00000, 666666);

INSERT INTO public."Tienda_Productos"(
    "Id_tienda_Tienda", "Codigo_de_barras_Productos", "Stock")
VALUES (76543, 00000, 30);

```

Datos a insertar en TIENDA2 generados con el código de java de la PECL2:

TIENDA	TICKET_PRODUCTO
1,Lmsc,Caf,Vkyah Caf,Leon	3,1,1
2,Opth,Gwvripup,Odnbc Gwvripup,Cuenca	5,2,11
3,Cghx,Jexfqic,Sela Jexfqic,Huelva	4,3,11
4,Wgli,Sfqi,Bpvmoibb Sfqi,Tarragona	3,9,1
5,Nynf,Rbmnu,Ngfomhr Rbmnu,A Corunia	5,2,2
	1,3,2
	7,3,12
	7,4,12
<b>TRABAJADOR</b>	
1,46882686,Erhju,Ovgaihbmpeov,jefe,2441,1	8,10,2
2,21984882,Devmgx,Ac Xihblvx,cajero,3767,3	1,4,2
3,62475963,Qxfedgth,Eesqh Pgb,reponedor,2283,7	4,7,2
4,39135785,Ddj,Gush Xbqi,dependiente,4476,5	1,3,13













5,78410976,Jhtr,Worfg Djdnjros,dependiente,1530,2	1,8,3
6,42747094,Hmbi,Yog Qbmmml,cajero,3913,4	7,3,3
7,44552859,Bxqxxgvf,Xqvik Gwpswbj,cajero,2087,5	2,9,14
8,85650954,Vdvsu,Bydohf Cybp,cajero,3058,3	1,7,4
9,99067005,Pim,Cnk Wmmwwg,reponedor,4106,2	7,3,15
10,6745606,Fk,Pdpl Nvanx,jefe,3478,1	2,8,15
	5,7,5
	9,4,5
<b>PRODUCTO</b>	9,3,5
1,Shvxfw,ropa,Made in Sapin,664	4,15,16
2,Httaup,comida,Made in Andorra,302	7,13,16
3,Jechwu,higiene,Made in China,220	9,14,6
4,Kngvfr,higiene,Made in China,590	7,7,6
5,Oymijd,higiene,Made in Sapin,944	4,11,6
6,Klwnpb,higiene,Made in Andorra,403	9,9,6
7,Rsefgo,ropa,Made in Andorra,514	2,7,16
8,Wpldds,deporte,Made in France,701	3,6,6
9,Aryslq,higiene,Made in Andorra,252	5,12,6
10,Ltkvyl,limpieza,Made in China,486	7,20,17
11,Qeupdm,limpieza,Made in Sapin,332	6,2,17
12,Qphvgl,deporte,Made in France,101	9,5,7
13,Cwrpos,ropa,Made in France,542	2,6,7
14,Mcqsfc,limpieza,Made in Andorra,112	4,16,7
15,Rffsgv,deporte,Made in China,99	7,19,18
16,Fmakeu,limpieza,Made in Andorra,173	3,14,8
17,Tvkrfy,higiene,Made in Sapin,808	4,1,18
18,Btdrfw,limpieza,Made in China,635	7,8,18
19,Qqaklu,higiene,Made in France,109	2,5,8
20,Kvejga,calzado,Made in France,413	5,13,8
	2,1,8
	7,2,19
	7,3,19
<b>TICKET</b>	5,4,9
1,9841,6-6-2019,1	4,5,9
2,3784,1-1-2019,2	4,8,20
3,1864,2-2-2019,6	6,10,10
4,8507,10-10-2019,8	5,11,10
5,2182,11-11-2019,2	1,12,10
6,111,1-1-2019,10	7,13,11
7,3167,4-4-2019,8	8,14,11
8,6874,5-5-2019,8	5,15,11

9,6003,10-10-2019,4	9,16,11
10,3752,7-7-2019,7	9,17,11
11,6399,2-2-2019,3	9,18,11
12,9010,3-3-2019,7	9,19,11
13,7720,3-3-2019,10	5,20,12
14,408,9-9-2019,1	
15,7752,10-10-2019,5	
16,4262,12-12-2019,10	<b>TIENDA_PRODUCTO</b>
17,6776,12-12-2019,9	1,1,42
18,429,9-9-2019,9	1,2,113
19,7924,8-8-2019,4	1,3,83
20,1477,4-4-2019,3	1,4,43
	2,5,80
	2,6,170
	2,7,120
	2,8,109
	3,9,167
	7,10,112
	3,11,59
	3,12,28
	4,13,21
	4,14,124
	4,15,108
	4,16,154
	5,17,44
	5,18,112
	5,19,34
	5,20,140
	7,1,62
	7,20,6

## 7.2 Consulta

Para la elaboración de esta consulta, tenemos que aclarar que al emplear *postgres\_fwd*, las tablas del otro maestro nos aparecen como *FOREIGN TABLES*, que contienen los datos que no se encuentran en su propio servidor. El problema es que esas tablas tienen que tener un nombre distinto que las del esquema, pero como en nuestro caso coincidía ya que habíamos ejecutado el mismo código, hemos creado un nuevo esquema (Schema) para almacenar las tablas llamado **PECL4**. Su funcionamiento es el mismo que las tablas normales.

- ▼  TIENDA1
  - >  Casts
  - >  Catalogs
  - >  Event Triggers
  - >  Extensions
  - >  Foreign Data Wrappers (1)
  - >  Languages
  - ▼  Schemas (2)
    - >  PECL4
    - >  public

Finalmente, la consulta es:

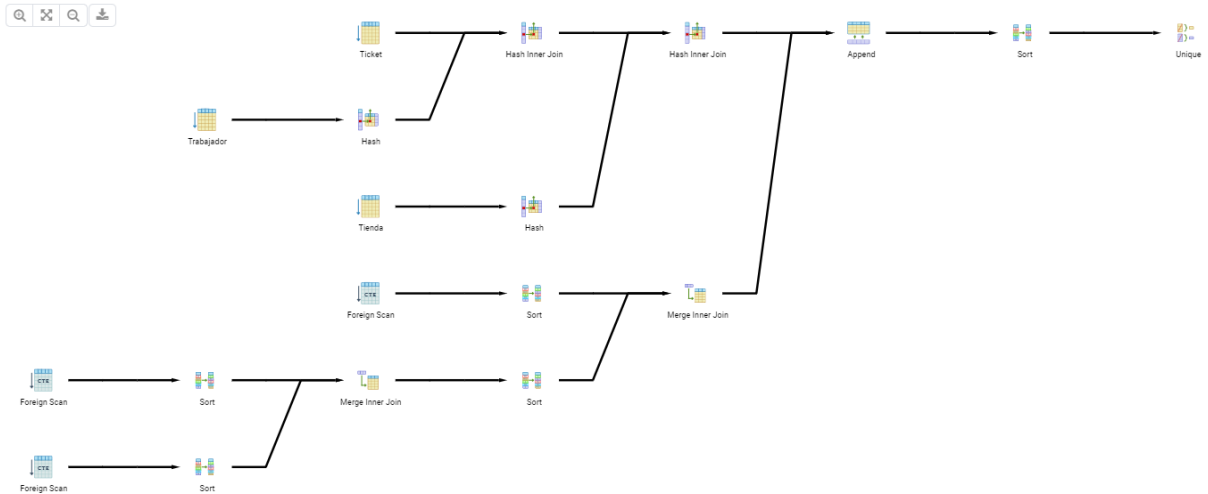
```
-- Realizar una consulta sobre el MAESTRO1 que permita obtener el nombre de todos los
-- trabajadores junto con su tienda en la que trabajan que hayan realizado por lo menos
-- una venta de algun producto en toda la base de datos distribuida (MAESTRO1 + MAESTRO2)
SELECT "Trabajador"."Nombre", "Tienda"."Nombre"
FROM public."Tienda" INNER JOIN public."Trabajador" ON "Id_tienda"="Id_tienda_Tienda"
INNER JOIN public."Ticket" ON "codigo_trabajador" = "codigo_trabajador_Trabajador"
UNION
SELECT "Trabajador"."Nombre", "Tienda"."Nombre"
FROM "PECL4"."Tienda" INNER JOIN "PECL4"."Trabajador" ON "Id_tienda"="Id_tienda_Tienda"
INNER JOIN "PECL4"."Ticket" ON "codigo_trabajador" = "codigo_trabajador_Trabajador";
```

El resultado de la consulta ejecutado desde Maestro1 es el siguiente:

	Nombre character varying	Nombre text
1	Bxqxgvf	Nynf
2	Ddj	Nynf
3	Devmgx	Cghx
4	Erhju	Lmsc
5	Fk	Lmsc
6	Hmbi	Wgli
7	Jhtr	Opth
8	Nombre1	Tienda1
9	Nombre2	Tienda2
10	Nombre3	Tienda3
11	Pim	Opth
12	Qxfedgth	Desde mi a mi
13	Vdvsu	Cghx

Comparamos y vemos que sale la misma solución en el Maestro2. Como podemos ver, funciona correctamente y nos añade los registros de cada una de las bases de datos, ya que, por ejemplo, desde Maestro1 habíamos insertado las tiendas *Tienda1*, *Tienda2* y *Tienda3* mientras que desde Maestro2 habíamos insertado las tiendas *Lmsc*, *Opth* y *Cghx* y somos capaces de verlas todas desde una misma consulta.

### El plan de ejecución:



Como vemos en el plan de ejecución, se puede destacar que hay un foreign scan, que consulta los datos de la otra base de datos. Lo mismo pasa con las otras tablas locales, que tendrán sus estadísticas y costes estimados.

## 8. Escenario maestro1 inservible

### 8.1 Acciones para volver al funcionamiento normal

Suponemos que el maestro de Isabel falla (vamos a detener el servidor):

```
[Isabels-MacBook-Pro:bin postgres$ ./pg_ctl stop -D /Library/PostgreSQL/12/data/  
waiting for server to shut down..... done  
server stopped
```

Ahora, el esclavo de Isabel empezará con los procesos de failover para dejar su función como standby y poder realizar todas las operaciones que el máster podía llevar a cabo, ya que si recordamos las pruebas previamente realizadas el esclavo sólo podía realizar transacciones de lectura.

Para convertir el esclavo de Isabel en maestro, nos vamos a la CMD y vamos a utilizar, según las indicaciones de la documentación:

```
pg_ctl promote [-D datadir] [-W] [-t seconds] [-s]
```

```
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
```

Como podemos ver el servidor ha sido promovido correctamente.

### 8.2 Configuración de nodos

En el pg\_hba.conf de Laura no hace falta añadir ningún rol más para dar permiso al nuevo maestro ya que las direcciones IPs que ya había puestas en este archivo permiten el acceso de cualquier IP. Si no hubiésemos utilizado este “comodín” tendríamos que haber puesto la dirección IP pública del dispositivo donde se encuentra el antiguo esclavo de Isabel, que ahora es su maestro.

#	TYPE	DATABASE	USER	ADDRESS	METHOD
# IPv4 local connections:					
host	all	all		0.0.0.0/0	md5
host	all	all		127.0.0.1/32	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		0.0.0.0/0	md5
host	replication	all		:::1/128	md5

En pg\_hba.conf del antiguo maestro de Isabel deberemos de indicarle que ya no es más el maestro, ya que suponiendo la situación en que el master fuese capaz de volver a ponerse en marcha. Para hacer esto vamos a eliminar el rol que habíamos

asignado de replication para el que ahora es el maestro. Por lo que vuelve a tener las mismas conexiones que al principio.

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

Desde el nuevo máster de Isabel debemos de añadir una línea de conexión para permitir la conexión entre ambos masters.

```
# IPv4 local connections:
host     all             all             127.0.0.1/32          md5
host     all             all             0.0.0.0/0             md5
```

Y por último, debemos volver a asignarle al puerto de nuestro router la IP privada de nuestro host, ya que anteriormente teníamos el antiguo master.

#### MIS PUERTOS

Modifica o elimina tus puertos.

<input type="checkbox"/>	Nombre	Dirección IP	Protocolo	Interno Inicio	Interno Fin	Externo Inicio	Externo Fin
<input type="checkbox"/>	Puerto 1	192.168.1.50	TCP+UDP	5432	5432	5450	5450

## 8.3 Pruebas del correcto funcionamiento

Para comprobar que no ha habido ningún problema y que todo marcha en orden, volvemos a ejecutar consultas para ver que funciona y no devuelve mensaje de error.

Desde el nuevo Maestro de Isabel vamos a probar a insertar el registro:

```
INSERT INTO public."Tienda" VALUES (444444, 'Inserto', 'Desde', 'El', 'Nuevo_Master');
```

```
INSERT 0 1
```

```
Query returned successfully in 198 msec.
```

Como podemos comprobar, ahora si que tenemos permisos que solo podía realizar el que fuese master y no servidor esclavo, ya que ahora tenemos más permisos que antes. Podemos insertar, modificar, eliminar y leer registros.

Ahora vamos a comprobar que las conexiones entre ambos maestros siguen funcionando. Para ello, desde el maestro de Laura ejecutamos la siguiente consulta:

PECL4.Tienda/TIENDA2/postgres@PostgreSQL 12						
Query Editor   Query History   Explain   Notifications						
<pre>1 SELECT * FROM "PECL4"."Tienda"</pre>						
Data Output   Messages						
	Id_tienda integer	Nombre text	Ciudad text	Barrio text	Provincia text	
1	1	FUNCIONA	sdfg	sdfg	knjbhvg	
2	2	FUNCIONA2	sdfg	sdfg	knjbhvg	
3	10	Desde master Laura a Isa	Master	Laura	dfg	
4	100	De Laura a Isa	Master	Laura	Porfa	
5	76543	Tienda1	Ciudad1	Barrio1	Provincia 1	
6	75555	Tienda2	Ciudad2	Barrio2	Provincia 2	
7	75000	Tienda3	Ciudad3	Barrio3	Provincia 3	
8	212121	Inserto	Desde	El	Nuevo_Master	
9	444444	Inserto	Desde	El	Nuevo_Master	

Podemos ver los nuevos registros que hemos insertado desde el nuevo maestro de Isabel, con ID 212121 y 444444 junto con los demás. La conexión entre ambos funciona por lo tanto correctamente.

## 9. Inconsistencias base de datos de maestro y esclavo

Según el método propuesto por PostgreSQL es posible que haya inconsistencias de datos entre la base de datos del nodo maestro y la base de datos del nodo esclavo. Podemos diferenciar dos tipos de inconsistencias:

- ☐ Lectura
- ☐ Escritura

### Tipo de replicación: Asíncrono

El maestro envía de forma asíncrona los datos del WAL según se escriben sin esperar a que el esclavo los realice. Puede incluso que ocurra que la situación de que el esclavo y el maestro vayan a la par.

En este caso las inconsistencias que se producirían son de **lectura**, porque puede que lea datos del nodo esclavo que resultan ser distintos a los del maestro debido a que pueden no haber sido actualizados

Sin embargo, también podrían producirse inconsistencias de **escritura**, porque puede que se produzca la situación en la que el archivo WAL no se haya escrito en el nodo esclavo, por lo que estarían desincronizados. En este caso, se produciría

una pérdida de datos en el caso de que el maestro no termine de enviar el archivo WAL al nodo esclavo y se produjese una caída del nodo maestro.

Como podemos ver, una de las desventajas que tiene el sistema de replicación asíncrono es que aunque la replicación se realiza mucho más rápido que en un sistema de replicación síncrona, es mucho **más probable la aparición de inconsistencias** en sus bases de datos.

#### Tipo de replicación: Síncrono

En el modo síncrono se espera a que se escriban los datos en el esclavo antes de enviar los datos. Aún siendo menores las probabilidades de aparición de inconsistencias que en el asíncrono, puede haber inconsistencias de **lectura**. Estas inconsistencias ocurren ya que no se asegura que el cambio haya sido aplicado en la base de datos del nodo esclavo y un cliente podría leer antes de que se aplicase la escritura en el esclavo. Mediante *synchronous\_commit* podemos asegurar la consistencia.

## 10. Conclusión

Centrándonos en la base de datos distribuida, esta aporta tres claros beneficios:

1. Mayor seguridad ante caídas, ya que la caída de un nodo no implica que deje de funcionar en su totalidad, si no que se pierde una parte de los datos.
2. Mayor seguridad ante ataques, pues no está toda la información almacenada en el mismo sitio. Por el contrario se encuentra “distribuida”, haciendo más difícil su obtención.
3. Mayor facilidad para atender a muchos clientes al no centrar todas las peticiones en un solo nodo, si no que podemos repartir esta carga, aumentando la velocidad y estableciendo equilibrio que evitará saturaciones.

En lo referente a la replicación de datos y su recuperación, hemos sido capaces de crear un servidor standby modificando los archivos de configuración. En caso de que el maestro se caiga, este servidor standby puede ponerse fácilmente en modo activo. Esto, sin lugar a duda, es de una gran importancia pues, en caso de caída, se puede dar soporte.

También nos hemos enfrentado a ciertos problemas con PostgreSQL, ya que cuando las bases de datos se encuentran en sistemas operativos distintos encontramos incompatibilidades en directorios, nombres, parámetros...



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

## 11. 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