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

Curso: **2019-2020. Convocatoria Ordinaria de Junio**

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

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

ALUMNO 1:

Nombre y Apellidos: Ana Cortés Cercadillo

DNI:

ALUMNO 2:

Nombre y Apellidos: Carlos Javier Hellín Asensio

DNI:

Fecha: 08/06/2020

Profesor Responsable: José Carlos Holgado

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`

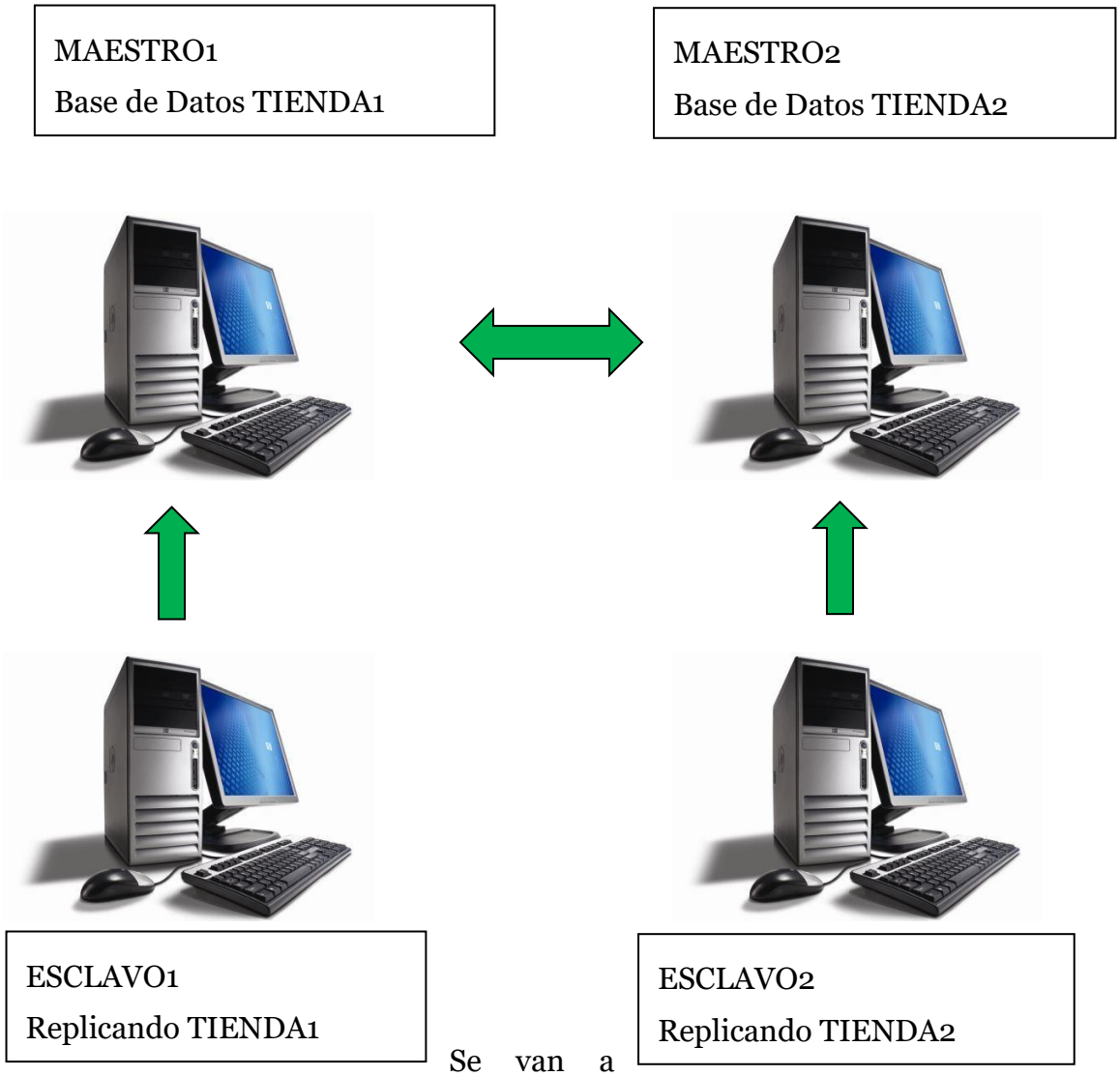
AMBOS ALUMNOS DEBEN ENTREGAR EL FICHERO EN LA PLATAFORMA.

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:



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.

Bibliografía

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

1.

La configuración de cada uno de los nodos maestros de la base de datos de TIENDA1 y TIENDA2 se realiza a través del módulo postgres_fdw. El módulo postgres_fdw se usa para acceder a los datos almacenados en servidores PostgreSQL externos. Para el

acceso remoto a través de postgres_fdw, se realizan los siguientes pasos en cada una de las dos bases de datos:

Se instala la extensión postgres_fdw.

```
tienda1=# CREATE EXTENSION postgres_fdw;
CREATE EXTENSION
```

Se crea el servidor externo con las características de la conexión que queremos realizar. Se especifica la IP del host, el puerto para la conexión y el nombre de la base de datos. En este caso, nos conectamos al servidor PostgreSQL en el host 88.10.183.6 que escucha en el puerto 5432. La base de datos a la que se realiza la conexión se llama tienda2 en el servidor remoto.

```
tienda1=# CREATE SERVER maestro2_server FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host '88.10.183.6', port '5432', dbna
me 'tienda2');
CREATE SERVER
```

Hemos creado el usuario que va a tener acceso a nuestro servidor remoto. Hay que crear una asignación de usuario con CREATE USER MAPPING para que ese usuario obtenga el acceso y para identificar la función que se utilizará en el servidor remoto. Se especifica el nombre del usuario y su contraseña.

```
tienda2=# CREATE USER maestro2_user WITH PASSWORD 'maestro2';
CREATE ROLE
tienda1=# CREATE USER MAPPING FOR postgres SERVER maestro2_server OPTIONS (user 'maestro2_user', password 'maestro2');
CREATE USER MAPPING
```

Le asignamos todos los permisos sobre todas las tablas de la base de datos remota al usuario creado.

```
tienda2=# GRANT ALL PRIVILEGES ON "Productos" TO maestro2_user;
GRANT
tienda2=#
tienda2=# GRANT ALL PRIVILEGES ON "Ticket" TO maestro2_user;
GRANT
tienda2=# GRANT ALL PRIVILEGES ON "Ticket_Productos" TO maestro2_user;
GRANT
tienda2=# GRANT ALL PRIVILEGES ON "Tienda" TO maestro2_user;
GRANT
tienda2=# GRANT ALL PRIVILEGES ON "Tienda_Productos" TO maestro2_user;
GRANT
tienda2=# GRANT ALL PRIVILEGES ON "Trabajador" TO maestro2_user;
GRANT
```

Se crea el esquema que se corresponde con la base de datos local.

```
tienda2=# CREATE SCHEMA maestro1;
CREATE SCHEMA
```

También importamos el esquema de la base de datos remota a la que se desea acceder.

```
tienda1=# IMPORT FOREIGN SCHEMA public FROM SERVER maestro2_server INTO maestro2;
IMPORT FOREIGN SCHEMA
```

Se modifica el archivo pg_hba.conf de cada una de las bases de datos añadiendo en ellas la línea correspondiente para aceptar conexiones de la base de datos y el usuario del otro MAESTRO. Luego se reinicia el servidor.

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
host    tienda2         maestro2_user   88.1.64.88/1            trust
```

En el archivo postgresql.conf se cambia el valor de listen_addresses para que quede: listen_addresses = '*'.

En las sesiones remotas abiertas por postgres_fdw, el parámetro search_path se establece solo en pg_catalog; inicialmente el search_path es public. Este no es un problema para las consultas generadas por postgres_fdw en la base de datos local. Sin embargo, esto puede representar un peligro para las consultas que se ejecutan en el servidor remoto. Para realizar consultas remotas, se recomienda cambiar el search_path al esquema que hemos creado anteriormente.

```
tienda1=# set search_path to 'maestro2';
SET
```

Con estos pasos, al realizar una consulta nos saldrán los registros almacenados en la base de datos remota a la que nos hemos conectado.

```
tienda1=# select * from "Productos" limit 10;
 Codigo de barras |  Nombre  |  Tipo  |  Descripcion  | Precio
------------------+----------+--------+---------------+--------
 codigo1          | nombre1  | tipo1  | descripcion1  |     97
 codigo2          | nombre2  | tipo2  | descripcion2  |    287
 codigo3          | nombre3  | tipo3  | descripcion3  |    335
 codigo4          | nombre4  | tipo4  | descripcion4  |    310
 codigo5          | nombre5  | tipo5  | descripcion5  |    461
 codigo6          | nombre6  | tipo6  | descripcion6  |    394
 codigo7          | nombre7  | tipo7  | descripcion7  |    847
 codigo8          | nombre8  | tipo8  | descripcion8  |    838
 codigo9          | nombre9  | tipo9  | descripcion9  |    233
 codigo10         | nombre10 | tipo10 | descripcion10 |    337
(10 filas)
```

2.

La configuración completa de los equipos para estar en modo de replicación depende de la aplicación y el hardware disponible, podría haber muchas formas diferentes de "guardar los datos en alguna parte"; en este caso la replicación se realiza en los ESCLAVOS, localizados cada uno en una máquina virtual dentro del propio

MAESTRO. Para proporcionar flexibilidad al administrador de la base de datos, PostgreSQL intenta no hacer suposiciones sobre cómo se realizará el archivado. En cambio, PostgreSQL le permite al administrador especificar un comando de shell que se ejecutará para copiar un archivo de segmento completado a donde sea necesario.

En la configuración del nodo MAESTRO, para habilitar el archivado WAL, se configura el parámetro de configuración wal_level en replica, archive_mode en on, y se especifica el comando de shell para usar en el parámetro de configuración archive_command. En la práctica, esta configuración siempre se colocará en el archivo postgresql.conf. En archive_command, %p es el nombre de la ruta del archivo a archivar, mientras que %f es simplemente el nombre del archivo (el nombre de la ruta es relativo al directorio de trabajo actual). Las modificaciones dentro del archivo postgresql.conf quedarían finalmente así: wal_level = replica, archive_mode = on y archive_command = 'copy "%p" "..\\archivedir\\%f"'. Después de realizar estos cambios, requiere el reinicio del servidor.

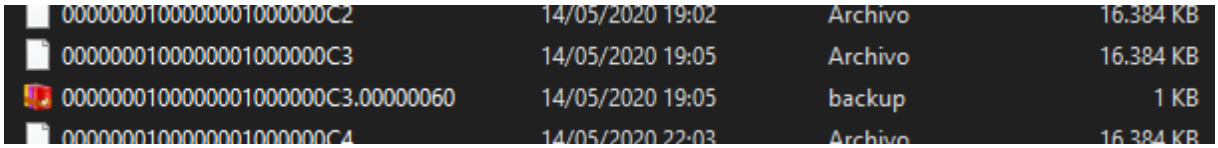
Es muy importante que se configuren los privilegios de acceso para la replicación de modo que sólo los usuarios de confianza puedan leer la secuencia WAL, porque es fácil extraer información privilegiada de ella. Los servidores en espera deben autenticarse en el primario como un superusuario o una cuenta que tenga el privilegio REPLICATION.

La autenticación del cliente para la replicación se controla mediante un registro pg_hba.conf que especifica la replicación en el campo de la base de datos.

```
# Allow replication connections from localhost, by a user with the
# replication privilege.
host    replication     all             127.0.0.1/32            trust
host    replication     all             192.168.37.129/1        trust
host    replication     all             ::1/128                 trust
```

La forma más fácil de realizar una copia de seguridad base es usar la herramienta pg_basebackup. Puede crear una copia de seguridad base ya sea como archivos normales o como un archivo tar. Se ejecuta pg_basebackup.exe de la siguiente forma: pg_basebackup.exe -D "..\backup" -p 5432. De esta forma, se copia el contenido de backup (que incluye postgresql.conf y pg_hba.conf) a la carpeta data de cada ESCLAVO y también se copia la carpeta archivedir del MAESTRO al ESCLAVO.

Para hacer uso de la copia de seguridad, deberá conservar todos los archivos del segmento WAL generados durante y después de la copia de seguridad del sistema de archivos. Para ayudar a conservar dichos archivos, el proceso crea un archivo del historial que se almacena inmediatamente en el área de archivo WAL. Este archivo lleva el nombre del primer archivo de segmento WAL que necesita para la copia de seguridad del sistema de archivos. En nuestro caso, el archivo WAL inicial es 000000010000000100000000C3, el archivo del historial de copia de seguridad se denomina como 000000010000000100000000C3.00000060.backup (la segunda parte del nombre del archivo representa una posición exacta dentro del archivo WAL, y normalmente se puede ignorar).



Una vez que haya archivado con seguridad la copia de seguridad del sistema de archivos y los archivos del segmento WAL utilizados durante la copia de seguridad (como se especifica en el archivo del historial de backup), todos los segmentos WAL archivados con nombres numéricamente menores ya no son necesarios para recuperar la copia de seguridad del sistema de archivos y se pueden eliminar. Sin embargo, debe considerar mantener varios conjuntos de copias de seguridad para estar absolutamente seguro de que puede recuperar sus datos.

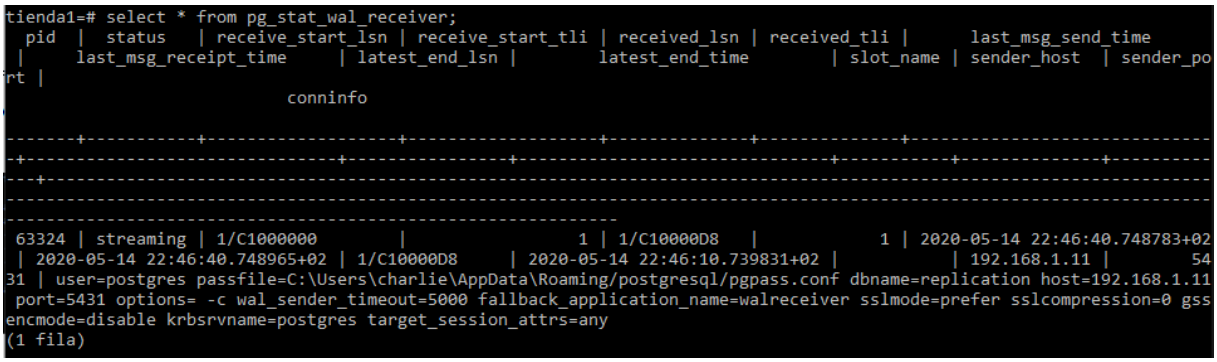
Para configurar el ESCLAVO, se restaura la copia de seguridad base tomada del servidor primario o MAESTRO. Se crea el archivo standby.signal en la carpeta data del ESCLAVO. En postgresql.conf se comprueba que estos parámetros son correctos de lo ya copiado desde la carpeta backup del MAESTRO (wal_level = replica, archive_mode = on y archive_command = 'copy "%p" "..\\archivedir\\%f"').

El tipo de replicación utilizada es streaming replication. Para configurarla se completa primary_conninfo con una cadena de conexión libpq, incluido el nombre de host (o la dirección IP) y los detalles adicionales necesarios para conectarse al servidor primario. Se establece restore_command en un comando simple para copiar archivos del archivo WAL. Por lo tanto, se modifican estos parámetros para el ESCLAVO en su postgresql.conf: primary_conninfo = 'host=192.168.1.11 port=5432 user=postgres options=" -c wal_sender_timeout=5000"' y restore_command = 'copy "..\\archivedir\\%f" "%p"'.

Se arrancan ambos servidores y se demuestra que la conexión entre MAESTRO1 y ESCLAVO1 funcionan. Para comprobar su correcto funcionamiento, se puede recuperar una lista de procesos del remitente WAL a través de la vista pg_stat_replication. El campo sent_lsn de la vista puede indicar que el servidor MAESTRO está bajo una gran carga. Se consulta pg_stat_replication en MAESTRO1:

```
tienda1=# select * from pg_stat_replication;
  pid  | usesysid | usename  | application_name | client_addr  | client_hostname | client_port |         backend_start
       | backend_xmin |   state   |  sent_lsn  | write_lsn  | flush_lsn  | replay_lsn | write_lag | flush_lag | replay_l
ag | sync_priority | sync_state |          reply_time
-------+----------+----------+------------------+--------------+-----------------+-------------+-------------------------------
-------+--------------+-----------+------------+------------+------------+------------+-----------+-----------+----------
---+---------------+------------+-------------------------------
 63336 |    16384 | postgres | walreceiver      | 192.168.1.11 |                 |       57451 | 2020-05-14 22:45:58.806
733+02 |              | streaming | 1/C10000D8 | 1/C10000D8 | 1/C10000D8 | 1/C10000D8 |           |           |
   |             0 | async      | 2020-05-14 22:47:13.257918+02
(1 fila)
```

En un modo de espera activo, el estado del proceso del receptor WAL se puede recuperar a través de la vista pg_stat_wal_receiver. La vista received_lsn indica que WAL se recibe más rápido de lo que se puede reproducir. Se consulta pg_stat_wal_receiver en ESCLAVO1:



Como se ha dicho anteriormente, el tipo de replicación elegido es streaming replication o replicación de transmisión. La replicación de transmisión permite que un servidor en espera se mantenga más actualizado de lo que es posible con el envío de registros basado en archivos. El que espera se conecta al primario, que transmite los registros WAL a medida que se generan, sin esperar a que se llene el archivo WAL. En este tipo de replicación hay un pequeño retraso entre la confirmación de una transacción en el primario y los cambios que se hacen visibles en el modo de espera. Sin embargo, este retraso es mucho menor que con el envío de registros basado en archivos, generalmente menos de un segundo, suponiendo que el modo de espera sea lo suficientemente potente como para mantenerse al día con la carga. La replicación de transmisión es asíncrona.

Otro tipo de replicación es replication slots o ranuras de replicación. Las ranuras de replicación proporcionan una forma automatizada de garantizar que el maestro no elimine los segmentos WAL hasta que hayan sido recibidos por todos los recursos en espera, y que el maestro no elimine las filas que podrían causar un conflicto de recuperación incluso cuando el dispositivo en espera está desconectado. Las ranuras de replicación retienen sólo el número de segmentos que se sabe que se necesitan. Una ventaja de estos métodos es que limitan el requisito de espacio para pg_wal; actualmente no hay forma de hacerlo utilizando ranuras de replicación.

La cascading replication o replicación en cascada permite que un servidor en espera acepte conexiones de replicación y transmita registros WAL a otros en espera, actuando como una parada. Esto se puede usar para reducir el número de conexiones directas al maestro y también para minimizar los gastos generales de ancho de banda entre sitios. Un modo de espera que actúa como receptor y como remitente se conoce como un modo de espera en cascada. Los servidores en espera que están más directamente conectados al maestro se conocen como servidores ascendentes, mientras que los servidores en espera más alejados son servidores descendentes. La replicación en cascada no establece límites en la cantidad o disposición de los servidores en sentido descendente, aunque cada modo de espera se conecta a un solo servidor en sentido ascendente que finalmente se vincula a un único servidor maestro o primario. Un modo de espera en cascada envía no solo los registros WAL recibidos del maestro sino también los restaurados del archivo. Por lo tanto, incluso si se termina la conexión de replicación en alguna conexión ascendente, la replicación de

transmisión continúa en sentido descendente mientras haya nuevos registros WAL disponibles. La replicación en cascada es actualmente asíncrona.

Uno de los tipos de replicación de PostgreSQL es synchronous replication o replicación sincrónica. La replicación sincrónica ofrece la capacidad de confirmar que todos los cambios realizados por una transacción se han transferido a uno o más servidores en espera sincrónicos. Esto extiende ese nivel estándar de durabilidad ofrecido por un compromiso de transacción. Al solicitar la replicación síncrona, cada confirmación de una transacción de escritura esperará hasta que se reciba la confirmación de que la confirmación se ha escrito en el registro de escritura anticipada en el disco tanto del servidor primario como del servidor en espera. La única posibilidad de que los datos se puedan perder es si tanto el primario como el en espera sufren bloqueos al mismo tiempo. Esto puede proporcionar un nivel de durabilidad mucho más alto, aunque solo si el administrador del sistema es cauteloso con respecto a la ubicación y administración de los dos servidores. Esperar la confirmación aumenta la confianza del usuario de que los cambios no se perderán en caso de fallas del servidor, pero también necesariamente aumenta el tiempo de respuesta para la transacción solicitada. El tiempo mínimo de espera es el tiempo de ida y vuelta entre primario y en espera. Las transacciones de solo lectura y las reversiones de transacciones no necesitan esperar respuestas de los servidores en espera. Las confirmaciones de subtransacción no esperan respuestas de los servidores en espera, solo las confirmaciones de nivel superior. Las acciones de ejecución prolongada, como la carga de datos o la creación de índices, no esperan hasta el último mensaje de confirmación. Todas las acciones de confirmación de dos fases requieren esperas de confirmación, incluidas la preparación y la confirmación.

Por último, también está el tipo de replicación Continuous Archiving in Standby o archivado continuo en espera. Cuando se utiliza el archivo WAL continuo en un modo de espera, hay dos escenarios diferentes: el archivo WAL se puede compartir entre el primario y el modo en espera, o el modo en espera puede tener su propio archivo WAL.

3.

Las operaciones que se pueden realizar en cada tipo de equipo de red son SELECT, INSERT, UPDATE y DELETE siempre que el usuario tenga los permisos adecuados y los nodos estén en funcionamiento.

Ahora provocaremos situaciones de caída de los nodos y observaremos los mensajes que aparecen y llegar así a acciones correctoras que se puedan realizar para volver el sistema a un estado consistente.

En el caso de caída del nodo maestro, no hay que hacer acciones correctoras ya que responde el esclavo, pero solo se permite transacciones de sólo lectura desde el otro maestro, por lo tanto solo se puede realizar SELECT.

```
tienda2=# select * from "Productos" limit 10;
 Codigo de barras |  Nombre  |  Tipo  |  Descripcion  | Precio
------------------+----------+--------+---------------+--------
 codigo1          | nombre1  | tipo1  | descripcion1  |    406
 codigo2          | nombre2  | tipo2  | descripcion2  |    302
 codigo3          | nombre3  | tipo3  | descripcion3  |    682
 codigo4          | nombre4  | tipo4  | descripcion4  |    221
 codigo5          | nombre5  | tipo5  | descripcion5  |     91
 codigo6          | nombre6  | tipo6  | descripcion6  |    479
 codigo7          | nombre7  | tipo7  | descripcion7  |    372
 codigo8          | nombre8  | tipo8  | descripcion8  |    957
 codigo9          | nombre9  | tipo9  | descripcion9  |    150
 codigo10         | nombre10 | tipo10 | descripcion10 |    616
(10 filas)


tienda2=# INSERT INTO "Productos" ("Codigo de barras", "Nombre", "Tipo", "Descripci
on", "Precio") VALUES ('codigo0', 'nombre0', 'tipo0', 'descripcion0', 50);
ERROR:  no se puede ejecutar INSERT en una transacción de sólo lectura
CONTEXTO:  remote SQL command: INSERT INTO public."Productos"("Codigo de barras", "
Nombre", "Tipo", "Descripcion", "Precio") VALUES ($1, $2, $3, $4, $5)
```

```
tienda2=# UPDATE "Productos" SET "Precio"=10 WHERE "Codigo de barras"='codigo0';
ERROR:  no se puede ejecutar UPDATE en una transacción de sólo lectura
CONTEXTO:  remote SQL command: UPDATE public."Productos" SET "Precio" = 10 WHERE ((
"Codigo de barras" = 'codigo0'::text))
tienda2=# DELETE FROM "Productos" WHERE "Codigo de barras"='codigo0';
ERROR:  no se puede ejecutar DELETE en una transacción de sólo lectura
CONTEXTO:  remote SQL command: DELETE FROM public."Productos" WHERE (("Codigo de ba
rras" = 'codigo0'::text))
```

En el caso de caída del nodo esclavo, a veces es necesario copiar la carpeta archivedir del maestro al esclavo para que el log no de los siguientes mensajes:

```
2020-05-15 19:36:57.773 CEST [9912] LOG:  iniciando el flujo de WAL desde el primario en 1/CC000000 en el timeline 1
2020-05-15 19:36:57.776 CEST [9912] FATAL:  no se pudo recibir datos desde el flujo de WAL: ERROR:  el segmento de WAL s
El sistema no puede encontrar el archivo especificado.
El sistema no puede encontrar el archivo especificado.
2020-05-15 19:37:02.791 CEST [1868] LOG:  iniciando el flujo de WAL desde el primario en 1/CC000000 en el timeline 1
2020-05-15 19:37:02.792 CEST [1868] FATAL:  no se pudo recibir datos desde el flujo de WAL: ERROR:  el segmento de WAL s
El sistema no puede encontrar el archivo especificado.
        1 archivo(s) copiado(s).
2020-05-15 19:37:07.655 CEST [4244] LOG:  se ha restaurado el archivo «000000010000000100000000CC» desde el área de archiv
El sistema no puede encontrar el archivo especificado.
2020-05-15 19:37:08.039 CEST [11560] LOG:  iniciando el flujo de WAL desde el primario en 1/CD000000 en el timeline 1
```

Otra posible solución es compartir en red la carpeta archivedir y que tanto el maestro como el esclavo tengan acceso a ella.

Si el maestro realiza operaciones como INSERT, UPDATE o DELETE, cuando el nodo esclavo vuelva a estar operativo no hace falta realizar nada ya que vuelve al estado de streaming replication.

4.

Se insertan datos distintos en cada una de las bases de datos del MAESTRO1 y del MAESTRO2. Se realiza la 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).

```
tienda1=# select "Nombre", "Id_tienda_Tienda" from "Trabajador" where exists (select * from "Ticket" where "Ticket"."cod
igo_trabajador_Trabajador" = "Trabajador".codigo_trabajador) UNION select "Nombre", "Id_tienda_Tienda" from maestro2."Tr
abajador" where exists (select * from maestro2."Ticket" where maestro2."Ticket"."codigo_trabajador_Trabajador" = maestro
2."Trabajador".codigo_trabajador);
 Nombre | Id_tienda_Tienda
--------+------------------
 n      |             4001
 n      |             4003
 a      |             1001
 m      |             4002
 a      |             1003
 b      |             1001
 b      |             1003
 m      |             4001
 a      |             1002
 m      |             4003
(10 filas)
```

Usando EXPLAIN podemos saber cómo se resuelve la consulta y su plan de ejecución.

```
tienda1=# explain select "Nombre", "Id_tienda_Tienda" from "Trabajador" where exists (select * from "Ticket" where "Tick
et"."codigo_trabajador_Trabajador" = "Trabajador".codigo_trabajador) UNION select "Nombre", "Id_tienda_Tienda" from maes
tro2."Trabajador" where exists (select * from maestro2."Ticket" where maestro2."Ticket"."codigo_trabajador_Trabajador" =
 maestro2."Trabajador".codigo_trabajador);
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=444.09..453.19 rows=910 width=36)
   Group Key: "Trabajador"."Nombre", "Trabajador"."Id_tienda_Tienda"
   ->  Append  (cost=37.63..439.54 rows=910 width=36)
         ->  Hash Join  (cost=37.63..57.45 rows=270 width=36)
               Hash Cond: ("Trabajador".codigo_trabajador = "Ticket"."codigo_trabajador_Trabajador")
               ->  Seq Scan on "Trabajador"  (cost=0.00..15.40 rows=540 width=40)
               ->  Hash  (cost=35.13..35.13 rows=200 width=4)
                     ->  HashAggregate  (cost=33.13..35.13 rows=200 width=4)
                           Group Key: "Ticket"."codigo_trabajador_Trabajador"
                           ->  Seq Scan on "Ticket"  (cost=0.00..28.50 rows=1850 width=4)
         ->  Hash Join  (cost=309.56..368.44 rows=640 width=36)
               Hash Cond: ("Trabajador_1".codigo_trabajador = "Ticket_1"."codigo_trabajador_Trabajador")
               ->  Foreign Scan on "Trabajador" "Trabajador_1"  (cost=100.00..148.40 rows=1280 width=40)
               ->  Hash  (cost=207.06..207.06 rows=200 width=4)
                     ->  HashAggregate  (cost=205.06..207.06 rows=200 width=4)
                           Group Key: "Ticket_1"."codigo_trabajador_Trabajador"
                           ->  Foreign Scan on "Ticket" "Ticket_1"  (cost=100.00..197.75 rows=2925 width=4)
(17 filas)
```

Hace un hash join y una lectura secuencial para la base de datos local (tienda1) y también un hash join y una lectura en las tablas foráneas de la base de datos tienda2. Después se unen las dos consultas, según se observa en la salida del EXPLAIN, a través de Append.

5.

En principio, si el MAESTRO1 queda inservible es el ESCLAVO1 quien se convierte en el nuevo MAESTRO1, por lo que hasta aquí no habría que tomar acciones. Pero si se quiere que el esclavo funcione como maestro debido a que MAESTRO1 ya es inservible, entonces en postgresql.conf habría que quitar los parámetros específicos para ser esclavo y quitar el archivo stanby.signal.

El ESCLAVO2 no sería necesario una nueva configuración porque no afecta a que el MAESTRO1 ya no funcione. En cambio, si el ESCLAVO1 pertenece a una subred distinta al MAESTRO1, sí sería necesario modificar la configuración del MAESTRO2 para que sigan manteniendo la conexión entre maestros.

6.

Según el método propuesto por PostgreSQL, sí podría haber inconsistencias en los datos entre la base de datos del nodo maestro y la base de datos del nodo esclavo. Si, por ejemplo, el MAESTRO1 se cae o falla y es el ESCLAVO1 quien toma el testigo, y posteriormente el MAESTRO1 vuelve a iniciarse, se debe hacer saber que él ya no es el MAESTRO.

Como PostgreSQL no ofrece un sistema para identificar que si el maestro falla, entonces hay que notificar al esclavo para que no crean que ambos nodos son el maestro, ya que provoca inconsistencias de datos.

7.

PostgreSQL, para implementar una base de datos distribuida, incluye módulos que permiten acceder a los datos almacenados en servidores PostgreSQL externos.

La replicación y la copia de seguridad son muy útiles en caso de caída de los servidores ya que permiten seguir accediendo a sus datos. Los nodos maestros son automáticamente reemplazados por nodos en modo replicación. PostgreSQL admite diferentes tipos de replicación según las necesidades de cada sistema.

Con esta configuración de los nodos que ofrece PostgreSQL, podemos tener en nuestro servidor tablas y esquemas correspondientes a otra base de datos perteneciente a otro servidor, es decir, importar sus tablas o sus esquemas para poder usar sus datos. De esta forma, se puede también obtener datos a través de una consulta de varias bases de datos con la misma estructura. Esto permite conseguir información asociada de servidores foráneos y de bases de datos locales al mismo tiempo.

En cambio, cuando uno de los servidores falle y un servidor esclavo se ponga en su lugar, encontramos el principal problema de PostgreSQL en este tipo de sistemas distribuidos. No tiene ningún sistema implementado que ayude a conocer cuando un servidor caído vuelve a funcionar y un servidor esclavo ya ha ocupado su puesto. Tener dos nodos principales ocasiona inconsistencias en los datos.