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

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

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

Práctica 3: Seguridad, Usuarios y Transacciones.

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 13 de Abril, Semana 20 de Abril y semana 27 de Abril.

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

Documento a entregar: Este mismo fichero con las respuestas a las cuestiones planteadas, con el código SQL utilizado en cada uno de los aparatos. Si se entrega en formato electrónico se entregará en un ZIP comprimido: **`DNI'sdelosAlumnos_PECL3.zip`**

AMBOS ALUMNOS DEBEN ENTREGAR EL FICHERO EN LA PLATAFORMA.

Introducción

El contenido de esta práctica versa sobre el manejo de las transacciones en sistemas de bases de datos, así como el control de la concurrencia y la recuperación de la base de datos frente a una caída del sistema. Las transacciones se definen como una unidad lógica de procesamiento compuesta por una serie de operaciones simples que se ejecutan como una sola operación. Entre las etiquetas BEGIN y COMMIT del lenguaje SQL se insertan las operaciones simples a realizar en una transacción. La sentencia ROLLBACK sirve para deshacer todos los cambios involucrados en una transacción y devolver a la base de datos al estado consistente en el que estaba antes de procesar la transacción. También se verá el registro diario o registro histórico del sistema de la base de datos (en PostgreSQL se denomina WAL: Write Ahead Loggin) donde se reflejan todas las operaciones sobre la base de datos y que sirve para recuperar ésta a un estado consistente si se produjera un error lógico o de hardware. La versión de postgres a utilizar deberá ser la versión 12.

Actividades y Cuestiones

En esta parte la base de datos **TIENDA** deberá de ser nueva y no contener datos. Además, consta de 5 actividades:

- Conceptos generales.
- Manejo de transacciones.
- Concurrencia.
- Registro histórico.
- Backup y Recuperación

Cuestión 1: Arrancar el servidor Postgres si no está y determinar si se encuentra activo el diario del sistema. Si no está activo, activarlo. Determinar cuál es el directorio y el archivo/s donde se guarda el diario. ¿Cuál es su tamaño? Al abrir el archivo con un editor de textos, ¿se puede deducir algo de lo que guarda el archivo?

El diario del sistema está almacenado en el directorio pg_wal dentro de la carpeta data, como ficheros segmentados, que suelen ocupar 16 MB. Cada segmento se divide en bloques, normalmente de 8 KB cada uno.

```
w:\uah\postgresql-12.2-4-windows-x64-binaries\pecl3\data\pg_wal>dir
 Volume in drive W is Datos
 Volume Serial Number is 325C-A4E6

 Directory of w:\uah\postgresql-12.2-4-windows-x64-binaries\pecl3\data\pg_wal

17/05/2020  23:27    <DIR>          .
17/05/2020  23:27    <DIR>          ..
17/05/2020  23:32        16.777.216 000000010000000000000001
17/05/2020  23:27    <DIR>          archive_status
               1 File(s)     16.777.216 bytes
               3 Dir(s)  528.599.691.264 bytes free
```

Al abrir el archivo no se puede deducir nada en claro, ya que se encuentra codificado en binario.

Cuestión 2: Realizar una operación de inserción de una tienda sobre la base de datos **TIENDA**. Abrir el archivo de diario ¿Se encuentra reflejada la operación en el archivo del sistema? ¿En caso afirmativo, por qué lo hará?

Sí, se encuentra en el diario al realizar la operación como se muestra en la siguiente captura de pantalla:

```
w:\uah\postgresql-12.2-4-windows-x64-binaries\pecl3\bin>pg_waldump.exe -p ..\data\pg_wal 000000010000000000000003 | grep
"COMMIT"
pg_waldump: fatal: error remng rr:e Transaction len (rec/tot):     34/    34, tx:        505, lsn:gistro  de WAL en 0/300
02E8: invalid record length0/03 0a0t0 10/3000320: wanted 24, got 0
58, prev 0/030000D8, desc: COMMIT 2020-05-18 01:07:07.112821 Romance Daylight Time
rmgr: Transaction len (rec/tot):     34/    34, tx:        506, lsn: 0/030002C0, prev 0/03000288, desc: COMMIT 2020-05-18
 01:08:19.586120 Romance Daylight Time
```

Al tratarse de una operación de inserción sin que se haya realizado una transacción previamente usando BEGIN, lo hace porque se registra como si fuera un COMMIT y poder usar posteriormente el log en caso de recuperaciones.

Cuestión 3: ¿Para qué sirve el comando pg_waldump.exe? Aplicarlo al último fichero de WAL que se haya generado. Obtener las estadísticas de ese fichero y comentar qué se está viendo.

pg_waldump.exe sirve para mostrar de una forma más legible el contenido de los ficheros WAL que anteriormente no se podía leer al abrirlo con un editor de texto.

Se obtiene las estadísticas del último fichero WAL:

```
w:\uah\postgresql-12.2-4-windows-x64-binaries\pecl3\bin>pg_waldump.exe -p ..\archivedir -z 000000010000000000000002
Type                                           N      (%)          Record size      (%)             FPI size      (%)
    Combined size      (%)
----                                           -      ---          -----------      ---             --------      ---
    -------------      ---
XLOG                                          17 (  1,44)                984 (  0,83)                960 (  0,75)
            1944 (  0,79)
Transaction                                   19 (  1,61)              10299 (  8,73)                  0 (  0,00)
           10299 (  4,19)
Storage                                       23 (  1,95)                966 (  0,82)                  0 (  0,00)
             966 (  0,39)
CLOG                                           0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
Database                                       1 (  0,08)                 42 (  0,04)                  0 (  0,00)
              42 (  0,02)
Tablespace                                     0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
MultiXact                                      0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
RelMap                                         0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
Standby                                       28 (  2,38)               1236 (  1,05)                  0 (  0,00)
            1236 (  0,50)
Heap2                                          0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
Heap                                         356 ( 30,22)              52648 ( 44,64)              52288 ( 40,88)
          104936 ( 42,68)
Btree                                        734 ( 62,31)              51766 ( 43,89)              74672 ( 58,37)
          126438 ( 51,43)
Hash                                           0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
Gin                                            0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
Gist                                           0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
Sequence                                       0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
SPGist                                         0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
BRIN                                           0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
CommitTs                                       0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
ReplicationOrigin                              0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
Generic                                        0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
LogicalMessage                                 0 (  0,00)                  0 (  0,00)                  0 (  0,00)
               0 (  0,00)
                                        --------                    --------                    --------
        --------
Total                                       1178                      117941 [47,97%]             127920 [52,03%]
          245861 [100%]
```

Lo que se puede ver son el número de registros (columna N), el tamaño de los registros (columna Record size) y el tamaño del bloque (columna FPI size) de cada tipo (XLOG, Transaction, etc…)

Cuestión 4: Determinar el identificador de la transacción que realizó la operación anterior. Aplicar el comando anterior al último fichero de WAL que se ha generado y mostrar los registros que se han creado para esa transacción. ¿Qué se puede ver? Interpretar los resultados obtenidos.

Para la operación anterior se ha generado los siguientes registros:

```
w:\uah\postgresql-12.2-4-windows-x64-binaries\pecl3\bin>pg_waldump.exe -p ..\data\pg_wal -x 506 000000010000000000000003
rmgr: Heap        len (rec/tot):     71/    71, tx:        506, lsn: 0/030001B8, prev 0/03000180, desc: INSERT off 2 flag
s 0x00, blkref #0: rel 1663/16385/16386 blk 0
rmgr: Btree       len (rec/tot):     53/   133, tx:        506, lsn: 0/03000200, prev 0/030001B8, desc: INSERT_LEAF off 2
, blkref #0: rel 1663/16385/16392 blk 1 FPW
rmgr: Transaction len (rec/tot):     34/    34, tx:        506, lsn: 0/030002C0, prev 0/03000288, desc: COMMIT 2020-05-18
 01:08:19.586120 Romance Daylight Time
pg_waldump: fatal: error en registro de WAL en 0/30003D0: invalid record length at 0/3000408: wanted 24, got 0
```

Se puede ver que hacer un INSERT a Heap, otro INSERT_LEAF para la hoja del índice B, ya que en tabla tienda el campo clave utiliza ese índice, y por el último se registra el COMMIT en el log.

Cuestión 5: Se va a crear un backup de la base de datos **TIENDA**. Este backup será utilizado más adelante para recuperar el sistema frente a una caída del sistema. Realizar solamente el backup mediante el procedimiento descrito en el apartado 25.3 del manual (versión 12 es *"Continous Archiving and point-in-time recovery (PITR)"*.

Para realizar el backup de la base de datos se usa pg_basebackup.exe y el resultado de la copia de seguridad se guarda en una carpeta llamada backup, tal como se muestra a continuación desde la línea de comandos:

pg_basebackup.exe -U postgres -p 5432 -D ..\backup\

Cuestión 6: Qué herramientas disponibles tiene PostgreSQL para controlar la actividad de la base de datos en cuanto a la concurrencia y transacciones? ¿Qué información es capaz de mostrar? ¿Dónde se guarda dicha información? ¿Cómo se puede mostrar?

PostgreSQL, para administrar el acceso concurrente a los datos, usa internamente un Control de Concurrencia (Multiversión Multiversion Concurrency Control, MVCC) que mantiene la consistencia de ellos. Este modelo de concurrencia se basa en mostrar una versión de la base de datos que puede no coincidir con la actual para evitar ver datos inconsistentes. Esto aporta independencia entre las transacciones y un mejor rendimiento en entornos multiusuarios por minimizar los bloqueos. Existen también diferentes tipos de bloqueos que no entran en conflicto entre ellos, por ejemplo, el adquirido para leer no colisiona con el bloqueo de escribir. PostgreSQL también dispone de bloqueos para tablas o registros para controlar conflictos específicos.

Para saber los bloqueos que están activos dentro de una base de datos, se pueden consultar a través de la vista del sistema pg_locks donde aparecen todos los tipos de bloqueos concedidos.

Esta vista contiene una fila por cada bloqueo activo. Las columnas más importantes que tiene la vista pg_locks son locktype (tipo del objeto que se bloquea), database (oid de la base de datos a la que pertenece), transactionid (identificador de la transacción), pid (identificador del proceso), mode (modo o tipo de bloqueo que quiere ese proceso) y granted (indica si se le ha concedido o no el bloqueo).

La información guardada en pg_locks se puede mostrar a través de una consulta: SELECT * FROM pg_locks;.

Cuestión 7: Crear dos usuarios en la base de datos que puedan acceder a la base de datos **TIENDA** identificados como usuario1 y usuario2 que tengan permisos de lectura/escritura a la base de datos tienda, pero que no puedan modificar su estructura. Describir el proceso seguido.

Se crea el usuario1 y el usuario con el siguiente comando SQL:

```
tienda=# CREATE USER usuario1;
CREATE ROLE
```

```
tienda=# CREATE USER usuario2;
CREATE ROLE
```

Se dan con GRANT los permisos de lectura/escritura a la base de datos y con REVOKE se le quita permisos para que no puedan modificar la estructura:

```
tienda=# GRANT SELECT, INSERT, UPDATE, DELETE ON "Productos", "Ticket", "Ticket_Productos", "Tienda", "Tienda_Productos"
, "Trabajador" to usuario1;
GRANT
tienda=# REVOKE CREATE ON DATABASE tienda FROM usuario1;
REVOKE
tienda=# GRANT SELECT, INSERT, UPDATE, DELETE ON "Productos", "Ticket", "Ticket_Productos", "Tienda", "Tienda_Productos"
, "Trabajador" to usuario2;
GRANT
tienda=# REVOKE CREATE ON DATABASE tienda FROM usuario2;
REVOKE
```

Cuestión 8: Abrir una transacción que inserte una nueva tienda en la base de datos (NO cierre la transacción). Realizar una consulta SQL para mostrar todas las tiendas de la base de datos dentro de esa transacción. Consultar la información sobre lo que se encuentra actualmente activo en el sistema. ¿Qué conclusiones se pueden extraer?

Se abre una transacción con BEGIN, se inserta una nueva tienda y se realiza una consulta SQL dentro de la transacción:

```
tienda=# BEGIN;
BEGIN
tienda=# INSERT INTO "Tienda" VALUES (2, 'n2', 'c2', 'b2', 'p2');
INSERT 0 1
tienda=# SELECT * FROM "Tienda";
 Id_tienda | Nombre | Ciudad | Barrio | Provincia
-----------+--------+--------+--------+-----------
         1 | n1     | c1     | b1     | p1
         2 | n2     | c2     | b2     | p2
(2 filas)
```

Con pg_locks se consulta la información de la actividad registrada en cuanto a transacciones:

```
tienda=# SELECT * FROM pg_locks;
   locktype    | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualt
ransaction |  pid   |       mode       | granted | fastpath
--------------+----------+----------+------+-------+------------+---------------+---------+-------+----------+---------
-----------+--------+------------------+---------+----------
 relation      |    16385 |    12143 |      |       |            |               |         |       |          | 3/16
          | 737492 | AccessShareLock  | t       | t
 relation      |    16385 |    16392 |      |       |            |               |         |       |          | 3/16
          | 737492 | AccessShareLock  | t       | t
 relation      |    16385 |    16386 |      |       |            |               |         |       |          | 3/16
          | 737492 | AccessShareLock  | t       | t
 relation      |    16385 |    16386 |      |       |            |               |         |       |          | 3/16
          | 737492 | RowExclusiveLock | t       | t
 virtualxid    |          |          |      |       | 3/16       |               |         |       |          | 3/16
          | 737492 | ExclusiveLock    | t       | t
 transactionid |          |          |      |       |            |           513 |         |       |          | 3/16
          | 737492 | ExclusiveLock    | t       | f
(6 filas)
```

Se puede sacar la conclusión de que existe un cerrojo de tipo transacción (transactionid) que es exclusivo (ExclusiveLock) y ha sido concedido (granted a true) ya que no se ha cerrado aún la transacción.

Cuestión 9: Cierre la transacción anterior. Utilizando pgAdmin o psql, abrir una transacción T1 en el usuario1 que realice las siguientes operaciones sobre la base de datos **TIENDA**. NO termine la transacción. Simplemente:

- Inserte una nueva tienda con ID_TIENDA 1000.
- Inserte un trabajador de la tienda anterior.
- Inserte un nuevo ticket del trabajador anterior con número 54321.

Se cierra la transacción anterior:

```
tienda=# COMMIT;
COMMIT
```

Se inicia la sesión de usuario1 con: psql -U usuario1 -d tienda
Se abre la transacción T1 y se la realiza las siguientes operaciones SQL:

```
tienda=> BEGIN;
BEGIN
tienda=> INSERT INTO "Tienda" VALUES (1000, 'n1000', 'c1000', 'b1000', 'p1000');
INSERT 0 1
tienda=> INSERT INTO "Trabajador" VALUES (1, '000000001', 'n1', 'a1', 'p1', 557, 1000);
INSERT 0 1
tienda=> INSERT INTO "Ticket" VALUES (54321, 25, '2017-03-14', 1);
INSERT 0 1
```

Cuestión 10: Realizar cualquier consulta SQL que muestre los datos anteriores insertados para ver que todo está correcto.

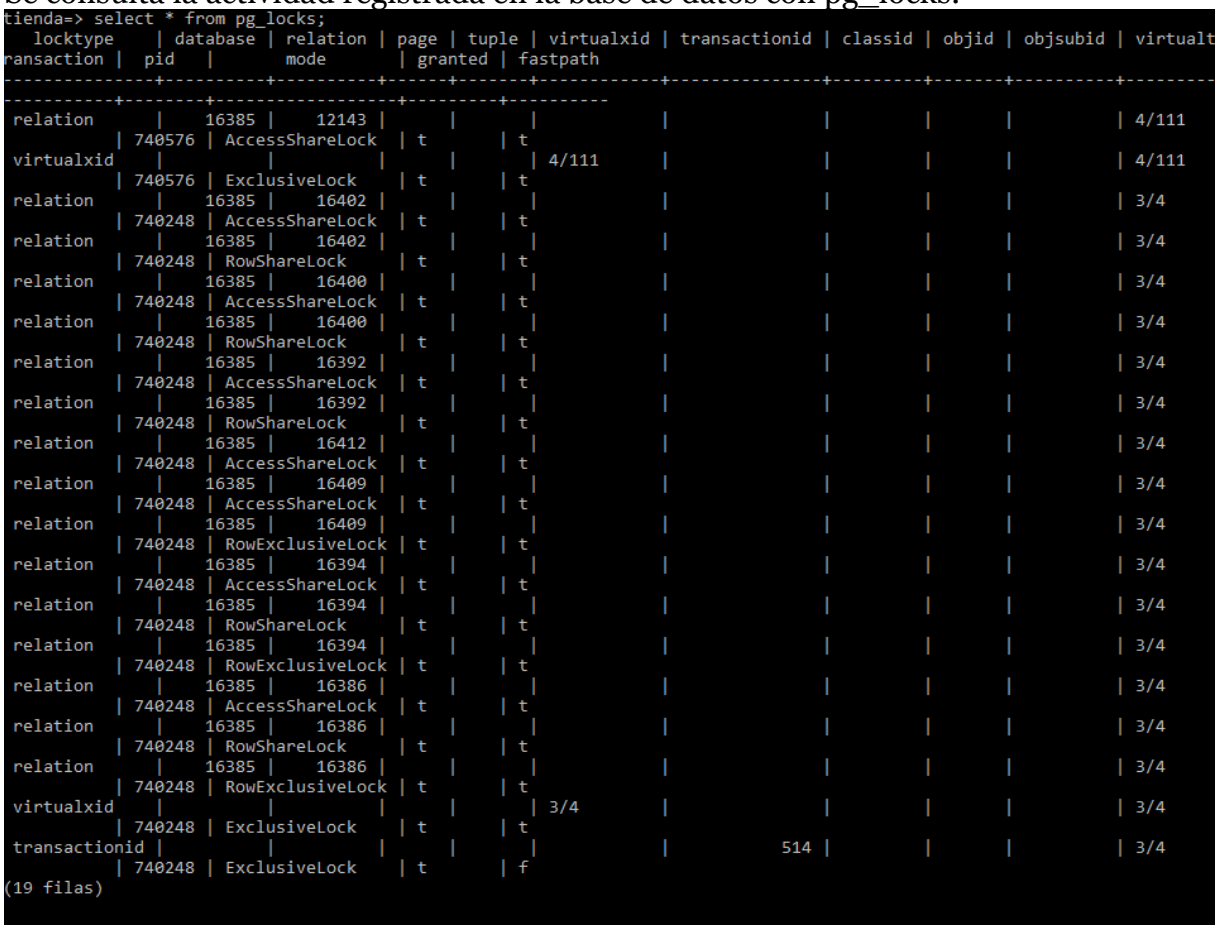
Se realiza la siguiente consulta SQL desde usuario1 que muestra todos los datos anteriores que se han insertado:

```
tienda=> SELECT * FROM "Tienda","Trabajador","Ticket" where "Tienda"."Id_tienda"=1000 and codigo_trabajador = 1 and "N╥
 de tickect" = 54321;
 Id_tienda | Nombre | Ciudad | Barrio | Provincia | codigo_trabajador |    DNI    | Nombre | Apellidos | Puesto | Salari
o | Id_tienda_Tienda | N╥ de tickect | Importe |   fecha    | codigo_trabajador_Trabajador
-----------+--------+--------+--------+-----------+-------------------+-----------+--------+-----------+--------+-------
--+------------------+----------------+---------+------------+---------------------------
      1000 | n1000  | c1000  | b1000  | p1000     |                 1 | 000000001 | n1     | a1        | p1     |     55
7 |             1000 |          54321 |      25 | 2017-03-14 |                           1
(1 fila)
```

Cuestión 11: Establecer una **nueva conexión** con pgAdmin o psql a la base de datos con el usuario2 (abrir otra sesión diferente a la abierta actualmente que pertenezca al usuario2) y realizar la misma consulta. ¿Se nota algún cambio? En caso afirmativo, ¿a qué puede ser debido el diferente funcionamiento en la base de datos para ambas consultas? ¿Qué información de actividad hay registrada en la base de datos en este momento?
Se inicia la sesión de usuario2 con: psql -U usuario2 -d tienda
y se realiza la misma consulta que en la cuestión 10:

```
tienda=> SELECT * FROM "Tienda","Trabajador","Ticket" where "Tienda"."Id_tienda"=1000 and codigo_trabajador = 1 and "N╥
 de tickect" = 54321;
 Id_tienda | Nombre | Ciudad | Barrio | Provincia | codigo_trabajador | DNI | Nombre | Apellidos | Puesto | Salario | Id
_tienda_Tienda | N╥ de tickect | Importe | fecha | codigo_trabajador_Trabajador
-----------+--------+--------+--------+-----------+-------------------+-----+--------+-----------+--------+---------+---
---------------+----------------+---------+-------+---------------------------
(0 filas)
```

Sí, se nota algún cambio. Esto es debido a que la transacción T1 todavía no se ha comprometido (no ha hecho COMMIT), por lo tanto cuando el usuario2 consulta estos datos no existen aún en la base de datos.
Se consulta la actividad registrada en la base de datos con pg_locks:



```
tienda=> select * from pg_locks;
   locktype    | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualtransaction |  pid   |       mode       | granted | fastpath
---------------+----------+----------+------+-------+------------+---------------+---------+-------+----------+--------------------+--------+------------------+---------+----------
 relation      |    16385 |    12143 |      |       |            |               |         |       |          | 4/111              | 740576 | AccessShareLock  | t       | t
 virtualxid    |          |          |      |       | 4/111      |               |         |       |          | 4/111              | 740576 | ExclusiveLock    | t       | t
 relation      |    16385 |    16402 |      |       |            |               |         |       |          | 3/4                | 740248 | AccessShareLock  | t       | t
 relation      |    16385 |    16402 |      |       |            |               |         |       |          | 3/4                | 740248 | RowShareLock     | t       | t
 relation      |    16385 |    16400 |      |       |            |               |         |       |          | 3/4                | 740248 | AccessShareLock  | t       | t
 relation      |    16385 |    16400 |      |       |            |               |         |       |          | 3/4                | 740248 | RowShareLock     | t       | t
 relation      |    16385 |    16392 |      |       |            |               |         |       |          | 3/4                | 740248 | AccessShareLock  | t       | t
 relation      |    16385 |    16392 |      |       |            |               |         |       |          | 3/4                | 740248 | RowShareLock     | t       | t
 relation      |    16385 |    16412 |      |       |            |               |         |       |          | 3/4                | 740248 | AccessShareLock  | t       | t
 relation      |    16385 |    16409 |      |       |            |               |         |       |          | 3/4                | 740248 | AccessShareLock  | t       | t
 relation      |    16385 |    16409 |      |       |            |               |         |       |          | 3/4                | 740248 | RowExclusiveLock | t       | t
 relation      |    16385 |    16394 |      |       |            |               |         |       |          | 3/4                | 740248 | AccessShareLock  | t       | t
 relation      |    16385 |    16394 |      |       |            |               |         |       |          | 3/4                | 740248 | RowShareLock     | t       | t
 relation      |    16385 |    16394 |      |       |            |               |         |       |          | 3/4                | 740248 | RowExclusiveLock | t       | t
 relation      |    16385 |    16386 |      |       |            |               |         |       |          | 3/4                | 740248 | AccessShareLock  | t       | t
 relation      |    16385 |    16386 |      |       |            |               |         |       |          | 3/4                | 740248 | RowShareLock     | t       | t
 relation      |    16385 |    16386 |      |       |            |               |         |       |          | 3/4                | 740248 | RowExclusiveLock | t       | t
 virtualxid    |          |          |      |       | 3/4        |               |         |       |          | 3/4                | 740248 | ExclusiveLock    | t       | t
 transactionid |          |          |      |       |            |           514 |         |       |          | 3/4                | 740248 | ExclusiveLock    | t       | f
(19 filas)
```

Se puede ver que existe una transacción en curso (la T1) con el identificador 514 que ha sido concedida y se mantiene en modo exclusivo.

Cuestión 12: ¿Se encuentran los nuevos datos físicamente en las tablas de la base de datos? Entonces, ¿de dónde se obtienen los datos de la cuestión 2.10 y/o de la 2.11?

No, no se encuentran los datos físicamente en las tablas de la base de datos como se ha podido comprobar anteriormente al hacer una cosulta con el usuario2. Para ello haría falta hacer COMMIT en T1.

Estos datos son obtenidos de la memoria global y no del disco duro.

Cuestión 13: Finalizar con éxito la transacción T1 y realizar la consulta de la cuestión 2.10 y 2.11 sobre ambos usuarios conectados. ¿Qué es lo que se obtiene ahora? ¿Por qué?

Se finaliza la transacción T1 con éxito y se realiza la misma consulta en usuario1:

```
tienda=> COMMIT;
COMMIT
tienda=> SELECT * FROM "Tienda","Trabajador","Ticket" where "Tienda"."Id_tienda"=1000 and codigo_trabajador = 1 and "NT
 de tickect" = 54321;
 Id_tienda | Nombre | Ciudad | Barrio | Provincia | codigo_trabajador |    DNI    | Nombre | Apellidos | Puesto | Salari
o | Id_tienda_Tienda | NT de tickect | Importe |   fecha    | codigo_trabajador_Trabajador
-----------+--------+--------+--------+-----------+-------------------+-----------+--------+-----------+--------+-------
--+------------------+---------------+---------+------------+---------------------------
      1000 | n1000  | c1000  | b1000  | p1000     |                 1 | 000000001 | n1     | a1        | p1     |     55
7 |             1000 |         54321 |      25 | 2017-03-14 |                           1
(1 fila)
```

y en usuario2:

```
tienda=> SELECT * FROM "Tienda","Trabajador","Ticket" where "Tienda"."Id_tienda"=1000 and codigo_trabajador = 1 and "NT
 de tickect" = 54321;
 Id_tienda | Nombre | Ciudad | Barrio | Provincia | codigo_trabajador |    DNI    | Nombre | Apellidos | Puesto | Salari
o | Id_tienda_Tienda | NT de tickect | Importe |   fecha    | codigo_trabajador_Trabajador
-----------+--------+--------+--------+-----------+-------------------+-----------+--------+-----------+--------+-------
--+------------------+---------------+---------+------------+---------------------------
      1000 | n1000  | c1000  | b1000  | p1000     |                 1 | 000000001 | n1     | a1        | p1     |     55
7 |             1000 |         54321 |      25 | 2017-03-14 |                           1
(1 fila)
```

Ahora ya sí se obtienen datos en ambos usuarios porque T1 se ha comprometido y al realizar el COMMIT se ha escrito los datos en el disco duro.

Cuestión 14: Sin ninguna transacción en curso, abrir una transacción en un usuario cualquiera y realizar las siguientes operaciones:

- Insertar una tienda nueva con ID_TIENDA a 2000.
- Insertar un trabajador de la tienda 2000.
- Insertar un ticket del trabajador anterior con número 54300.
- Hacer una modificación del trabajador para cambiar el número de tienda de 2000 a 1000.
- Cerrar la transacción.

¿Cuál es el estado final de la base de datos? ¿Por qué?

Se abre una nueva transacción y se realizan las siguientes operaciones:

```
tienda=> BEGIN;
BEGIN
tienda=> INSERT INTO "Tienda" VALUES (2000, 'n2000', 'c2000', 'b2000', 'p2000');
INSERT 0 1
tienda=> INSERT INTO "Trabajador" VALUES (2, '000000002', 'n2', 'a2', 'p2', 776, 2000);
INSERT 0 1
tienda=>  INSERT INTO "Ticket" VALUES (54300, 52, '2017-03-15', 2);
INSERT 0 1
tienda=> UPDATE "Trabajador" SET "Id_tienda_Tienda" = 1000 WHERE "Id_tienda_Tienda" = 2000;
UPDATE 1
tienda=> COMMIT;
COMMIT
tienda=> select * from pg_locks;
  locktype  | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualtran
saction |  pid   |      mode       | granted | fastpath
------------+----------+----------+------+-------+------------+---------------+---------+-------+----------+------------
--------+--------+-----------------+---------+----------
 relation   |    16385 |    12143 |      |       |            |               |         |       |          | 3/7
        | 740248 | AccessShareLock | t       | t
 virtualxid |          |          |      |       | 3/7        |               |         |       |          | 3/7
        | 740248 | ExclusiveLock   | t       | t
(2 filas)
```

Al consultar con pg_locks se puede ver que no hay ninguna transacción en curso (transactionid) por lo tanto los datos ya se encuentran insertados en el disco duro.

Cuestión 15: Repetir la cuestión 9 con otra tienda, trabajador y ticket. Realizar la misma consulta de la cuestión 10, pero ahora terminar la transacción con un ROLLBACK y repetir la consulta con los mismos dos usuarios. ¿Cuál es el resultado? ¿Por qué?

Se inicia sesión con el usuario1: psql -U usuario1 -d tienda
Se inicia la transacción, se insertan nuevos datos de la cuestión 9, se hace la consulta y se termina la transacción con ROLLBACK.

```
tienda=> BEGIN;
BEGIN
tienda=> INSERT INTO "Tienda" VALUES (3000, 'n3000', 'c3000', 'b3000', 'p3000');
INSERT 0 1
tienda=> INSERT INTO "Trabajador" VALUES (3, '000000003', 'n3', 'a3', 'p3', 553, 3000);
INSERT 0 1
tienda=> INSERT INTO "Ticket" VALUES (54303, 53, '2017-03-13', 3);
INSERT 0 1
tienda=> SELECT * FROM "Tienda","Trabajador","Ticket" where "Tienda"."Id_tienda"=3000 and codigo_trabajador = 3 and "N_T
 de tickect" = 54303;
 Id_tienda | Nombre | Ciudad | Barrio | Provincia | codigo_trabajador |    DNI    | Nombre | Apellidos | Puesto | Salari
o | Id_tienda_Tienda | N_T de tickect | Importe |   fecha    | codigo_trabajador_Trabajador
-----------+--------+--------+--------+-----------+-------------------+-----------+--------+-----------+--------+-------
--+------------------+----------------+---------+------------+----------------------------
      3000 | n3000  | c3000  | b3000  | p3000     |                 3 | 000000003 | n3     | a3        | p3     |     55
3 |             3000 |          54303 |      53 | 2017-03-13 |                           3
(1 fila)


tienda=> ROLLBACK;
ROLLBACK
tienda=> SELECT * FROM "Tienda","Trabajador","Ticket" where "Tienda"."Id_tienda"=3000 and codigo_trabajador = 3 and "N_T
 de tickect" = 54303;
 Id_tienda | Nombre | Ciudad | Barrio | Provincia | codigo_trabajador | DNI | Nombre | Apellidos | Puesto | Salario | Id
_tienda_Tienda | N_T de tickect | Importe | fecha | codigo_trabajador_Trabajador
-----------+--------+--------+--------+-----------+-------------------+-----+--------+-----------+--------+---------+---
---------------+----------------+---------+-------+----------------------------
(0 filas)
```

Por último, se hace la misma consulta después del ROLLBACK tanto para el usuario1 como para el usuario2.

Se inicia sesión con el usuario2: psql -U usuario2 -d tienda
Se realiza la consulta.

```
tienda=> SELECT * FROM "Tienda","Trabajador","Ticket" where "Tienda"."Id_tienda"=3000 and codigo_trabajador = 3 and "N_T
 de tickect" = 54303;
 Id_tienda | Nombre | Ciudad | Barrio | Provincia | codigo_trabajador | DNI | Nombre | Apellidos | Puesto | Salario | Id
_tienda_Tienda | N_T de tickect | Importe | fecha | codigo_trabajador_Trabajador
-----------+--------+--------+--------+-----------+-------------------+-----+--------+-----------+--------+---------+---
---------------+----------------+---------+-------+----------------------------
(0 filas)
```

El resultado es que antes del ROLLBACK se puede obtener los datos insertados que se encuentran en memoria, pero después no existen esos datos tanto para el usuario1 como para el usuario2. Esto es debido porque ROLLBACK lo que hace es abortar la transacción actual y se descartan todas las operaciones realizadas por la transacción como puedan ser los INSERT en este caso.

Cuestión 16: Cerrar todas las sesiones anteriores. Abrir una sesión con el usuario1 de la base de datos **TIENDA**. Insertar la siguiente información en la base de datos:

- Insertar una tienda con id_tienda de 31145.
- Insertar un trabajador que pertenezca a la tienda anterior y tenga un código de 45678.

Se inicia sesión de usuario1 con: psql -U usuario1 -d tienda
Y ser inserta la información que se pide:

```
tienda=> INSERT INTO "Tienda" VALUES (31145, 'n31145', 'c31145', 'b31145', 'p31145');
INSERT 0 1
tienda=> INSERT INTO "Trabajador" VALUES (45678, '000045678', 'n45678', 'a45678', 'p45678', 514, 31145);
INSERT 0 1
```

Cuestión 17: Abrir una sesión con el usuario2 a la base de datos **TIENDA**. Abrir una transacción T2 en este usuario2 y realizar una modificación de la tienda código 31145 para cambiar el nombre a "Tienda Alcalá". ¿Qué actividad hay registrada en la base de datos? ¿Cuál es la información guardada en la base de datos? ¿Por qué?

Se inicia sesión de usuario2 con: psql -U usuario2 -d tienda

Se abre la transacción T2 y se actualiza la tienda:

```
tienda=> BEGIN;
BEGIN
tienda=> UPDATE "Tienda" SET "Nombre" = 'Tienda Alcalá' WHERE "Id_tienda" = 31145;
UPDATE 1
tienda=> select * from pg_locks;
   locktype    | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualt
ransaction |  pid   |       mode       | granted | fastpath
---------------+----------+----------+------+-------+------------+---------------+---------+-------+----------+---------
-----------+--------+------------------+---------+----------
 relation      |    16385 |    12143 |      |       |            |               |         |       |          | 4/167
          | 740688 | AccessShareLock  | t       | t
 relation      |    16385 |    16392 |      |       |            |               |         |       |          | 4/167
          | 740688 | RowExclusiveLock | t       | t
 relation      |    16385 |    16386 |      |       |            |               |         |       |          | 4/167
          | 740688 | RowExclusiveLock | t       | t
 virtualxid    |          |          |      |       | 4/167      |               |         |       |          | 4/167
          | 740688 | ExclusiveLock    | t       | t
 transactionid |          |          |      |       |            |           519 |         |       |          | 4/167
          | 740688 | ExclusiveLock    | t       | f
(5 filas)


tienda=> select * from "Tienda" where "Id_tienda" = 31145;
 Id_tienda |    Nombre     | Ciudad | Barrio | Provincia
-----------+---------------+--------+--------+-----------
     31145 | Tienda Alcalá | c31145 | b31145 | p31145
(1 fila)
```

Al consultar la actividad registra de la base de datos se puede ver una transacción activa con id 519, en modo exclusivo y que ha sido concedido (true en granted).
Y la información guardada en la base de datos se encuentra con "Tienda Alcalá" porque la consulta se realiza dentro de la propia transacción T2.

Cuestión 18. Abra una transacción T1 en el usuario1. Haga una actualización del trabajador con número 45678 para cambiar el salario a 3000. ¿Qué actividad hay registrada en la base de datos? ¿Cuál es la información guardada en la base de datos? ¿Por qué?

Se inicia sesión de usuario1 con: psql -U usuario1 -d tienda
Se abre la transacción T1 y se actualiza el salario:

```
tienda=> BEGIN;
BEGIN
tienda=> UPDATE "Trabajador" SET "Salario" = 3000 WHERE codigo_trabajador = 45678;
UPDATE 1
tienda=> select * from pg_locks;
   locktype    | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualt
ransaction |  pid   |       mode       | granted | fastpath
--------------+----------+----------+------+-------+------------+---------------+---------+-------+----------+---------
-----------+--------+------------------+---------+----------
 relation     |    16385 |    12143 |      |       |            |               |         |       |          | 4/167
          | 740688 | AccessShareLock  | t       | t
 relation     |    16385 |    16392 |      |       |            |               |         |       |          | 4/167
          | 740688 | AccessShareLock  | t       | t
 relation     |    16385 |    16392 |      |       |            |               |         |       |          | 4/167
          | 740688 | RowExclusiveLock | t       | t
 relation     |    16385 |    16386 |      |       |            |               |         |       |          | 4/167
          | 740688 | AccessShareLock  | t       | t
 relation     |    16385 |    16386 |      |       |            |               |         |       |          | 4/167
          | 740688 | RowExclusiveLock | t       | t
 virtualxid   |          |          |      |       | 4/167      |               |         |       |          | 4/167
          | 740688 | ExclusiveLock    | t       | t
 relation     |    16385 |    12143 |      |       |            |               |         |       |          | 3/15
          | 741044 | AccessShareLock  | t       | t
 relation     |    16385 |    16402 |      |       |            |               |         |       |          | 3/15
          | 741044 | RowExclusiveLock | t       | t
 relation     |    16385 |    16400 |      |       |            |               |         |       |          | 3/15
          | 741044 | RowExclusiveLock | t       | t
 relation     |    16385 |    16394 |      |       |            |               |         |       |          | 3/15
          | 741044 | RowExclusiveLock | t       | t
 virtualxid   |          |          |      |       | 3/15       |               |         |       |          | 3/15
          | 741044 | ExclusiveLock    | t       | t
 transactionid |          |          |      |       |            |           520 |         |       |          | 3/15
          | 741044 | ExclusiveLock    | t       | f
 transactionid |          |          |      |       |            |           519 |         |       |          | 4/167
          | 740688 | ExclusiveLock    | t       | f
(13 filas)


tienda=> select * from "Trabajador" where codigo_trabajador = 45678;
 codigo_trabajador |    DNI    | Nombre | Apellidos | Puesto | Salario | Id_tienda_Tienda
-------------------+-----------+--------+-----------+--------+---------+------------------
             45678 | 000045678 | n45678 | a45678    | p45678 |    3000 |            31145
(1 fila)
```

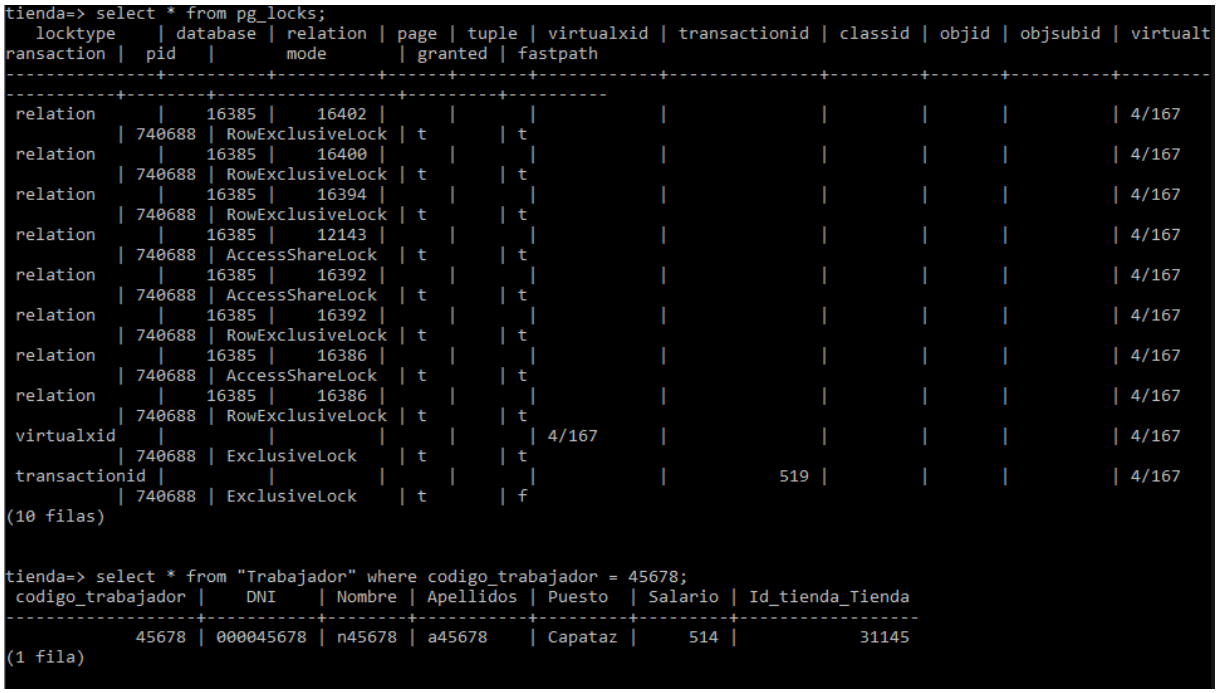
Al mirar la actividad registrada en la base de datos, se puede ver que existe una nueva transacción con id 520, que pide modo exclusivo y ha sido concedido.
En cuanto a la información guardada se puede ver que el Salario ha sido modificado a 3000 porque la consulta se realiza en la misma transacción T1.

Cuestión 19: En la transacción T2, realice una modificación del trabajador con código 45678 para cambiar el puesto a “Capataz”. ¿Qué actividad hay registrada en la base de datos? ¿Cuál es la información guardada en la base de datos? ¿Por qué?

Se realiza la modificación en la transacción T2 y se queda bloqueado:

```
tienda=> UPDATE "Trabajador" SET "Puesto" = ‘Capataz’ WHERE codigo_trabajador = 45678;
```

Una vez hecha la cuestión 20 se desbloquea la transacción y se procede a consultar la actividad registra y la información guardada en base de datos:



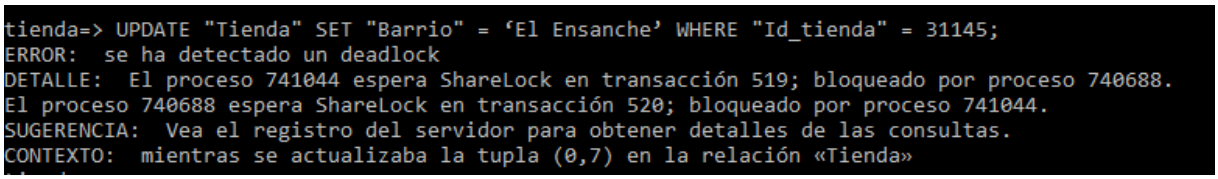
```
tienda=> select * from pg_locks;
   locktype    | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualt
ransaction |  pid   |       mode       | granted | fastpath
--------------+----------+----------+------+-------+------------+---------------+---------+-------+----------+---------
----------+--------+------------------+---------+----------
 relation      |    16385 |    16402 |      |       |            |               |         |       |          | 4/167
          | 740688 | RowExclusiveLock | t       | t
 relation      |    16385 |    16400 |      |       |            |               |         |       |          | 4/167
          | 740688 | RowExclusiveLock | t       | t
 relation      |    16385 |    16394 |      |       |            |               |         |       |          | 4/167
          | 740688 | RowExclusiveLock | t       | t
 relation      |    16385 |    12143 |      |       |            |               |         |       |          | 4/167
          | 740688 | AccessShareLock  | t       | t
 relation      |    16385 |    16392 |      |       |            |               |         |       |          | 4/167
          | 740688 | AccessShareLock  | t       | t
 relation      |    16385 |    16392 |      |       |            |               |         |       |          | 4/167
          | 740688 | RowExclusiveLock | t       | t
 relation      |    16385 |    16386 |      |       |            |               |         |       |          | 4/167
          | 740688 | AccessShareLock  | t       | t
 relation      |    16385 |    16386 |      |       |            |               |         |       |          | 4/167
          | 740688 | RowExclusiveLock | t       | t
 virtualxid    |          |          |      |       | 4/167      |               |         |       |          | 4/167
          | 740688 | ExclusiveLock    | t       | t
 transactionid |          |          |      |       |            |           519 |         |       |          | 4/167
          | 740688 | ExclusiveLock    | t       | f
(10 filas)


tienda=> select * from "Trabajador" where codigo_trabajador = 45678;
 codigo_trabajador |    DNI    | Nombre | Apellidos | Puesto  | Salario | Id_tienda_Tienda
-------------------+-----------+--------+-----------+---------+---------+------------------
             45678 | 000045678 | n45678 | a45678    | Capataz |     514 |            31145
(1 fila)
```

La transacción T2 con id 519 continua en modo exclusivo y concedido con la información guardada de la modificación que ha realizado T2 antes de quedar bloqueado.

Cuestión 20: En la transacción T1, realice una modificación de la tienda con código 31145 para modificar el barrio y poner “El Ensanche”. ¿Qué actividad hay registrada en la base de datos? ¿Cuál es la información guardada en la base de datos? ¿Por qué?

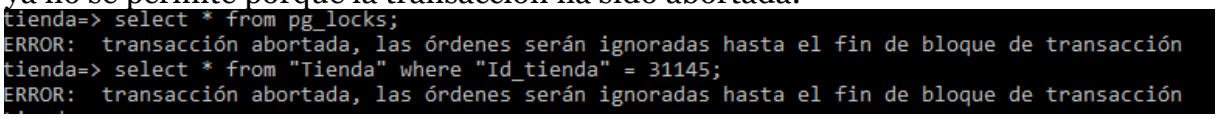
Se realiza la modificación en T1:



```
tienda=> UPDATE "Tienda" SET "Barrio" = 'El Ensanche' WHERE "Id_tienda" = 31145;
ERROR:  se ha detectado un deadlock
DETALLE:  El proceso 741044 espera ShareLock en transacción 519; bloqueado por proceso 740688.
El proceso 740688 espera ShareLock en transacción 520; bloqueado por proceso 741044.
SUGERENCIA:  Vea el registro del servidor para obtener detalles de las consultas.
CONTEXTO:  mientras se actualizaba la tupla (0,7) en la relación «Tienda»
```

Se produce un interbloqueo, se aborta la transacción T1 y, por ello, T2 continúa con su ejecución ya que no tiene que esperar a T1.

Al intentar consultar la actividad o la información guardada dentro de la transacción ya no se permite porque la transacción ha sido abortada:



```
tienda=> select * from pg_locks;
ERROR:  transacción abortada, las órdenes serán ignoradas hasta el fin de bloque de transacción
tienda=> select * from "Tienda" where "Id_tienda" = 31145;
ERROR:  transacción abortada, las órdenes serán ignoradas hasta el fin de bloque de transacción
```

Cuestión 21: Comprometa ambas transacciones T1 y T2. ¿Cuál es el valor final de la información modificada en la base de datos **TIENDA**? ¿Por qué?

T1 se compromete con COMMIT pero en realidad hace un ROLLBACK debido a que la transacción ha sido abortada.

```
tienda=> COMMIT;
ROLLBACK
```

T2 se compromete con COMMIT y lo realiza.

```
tienda=> COMMIT;
COMMIT
```

Al consultar el valor final de la información modificada en la base de datos se puede comprobar que las modificaciones de la transacción T1 no se ha hecho debido a que ha sido una transacción abortada:

```
tienda=> select * from "Trabajador" where codigo_trabajador = 45678;
 codigo_trabajador |    DNI    | Nombre | Apellidos | Puesto  | Salario | Id_tienda_Tienda
-------------------+-----------+--------+-----------+---------+---------+------------------
             45678 | 000045678 | n45678 | a45678    | Capataz |     514 |            31145
(1 fila)


tienda=> select * from "Tienda" where "Id_tienda" = 31145;
 Id_tienda |    Nombre     | Ciudad | Barrio | Provincia
-----------+---------------+--------+--------+-----------
     31145 | Tienda Alcalá | c31145 | b31145 | p31145
(1 fila)
```

Cuestión 22: Cerrar todas las sesiones anteriores. Abrir una sesión con el usuario1 de la base de datos **TIENDA**. Insertar en la tabla tienda una nueva tienda con código 6789. Abrir una transacción T1 en este usuario y realizar una modificación de la tienda con código 6789 y actualizar el nombre a “Mediamarkt”. No cierre la transacción.

Se inicia sesión de usuario1 con: psql -U usuario1 -d tienda

Se inserta una nueva tienda, se abre una transacción T1 y se actualiza tienda:

```
tienda=> INSERT INTO "Tienda" VALUES (6789, ‘n6789’, ‘c6789’, ‘b6789’, ‘p6789’);
INSERT 0 1
tienda=> BEGIN;
BEGIN
tienda=> UPDATE "Tienda" SET "Nombre" = ‘Mediamarkt’ WHERE "Id_tienda" = 6789;
UPDATE 1
```

Cuestión 23: Abrir una sesión con el usuario2 de la base de datos **TIENDA**. Abrir una transacción T2 en este usuario y realizar una modificación de la tienda con código 6789 y cambiar el nombre a “Saturn”. No cierre la transacción. ¿Qué es lo que ocurre? ¿Por qué? ¿Qué información se puede obtener de la actividad de ambas transacciones en el sistema? ¿Es lógica esa información? ¿Por qué?

Se inicia sesión de usuario2 con: psql -U usuario2 -d tienda
Se abre la transacción T2 y se modifica tienda:

```
tienda=> BEGIN;
BEGIN
tienda=> UPDATE "Tienda" SET "Nombre" = 'Saturn' WHERE "Id_tienda" = 6789;
```

En este momento la transacción T2 se ha bloqueado porque T1 ya tiene el modo exclusivo.

Se consulta la actividad de ambas transacciones:

```
tienda=> select * from pg_locks;
   locktype    | database | relation | page | tuple | virtualxid | transactionid | classid | objid | objsubid | virtualt
ransaction |  pid   |       mode       | granted | fastpath
--------------+----------+----------+------+-------+------------+---------------+---------+-------+----------+---------
-----------+--------+------------------+---------+----------
 relation     |    16385 |    16392 |      |       |            |               |         |       |          | 4/175
          | 743348 | RowExclusiveLock | t       | t
 relation     |    16385 |    16386 |      |       |            |               |         |       |          | 4/175
          | 743348 | RowExclusiveLock | t       | t
 virtualxid   |          |          |      |       | 4/175      |               |         |       |          | 4/175
          | 743348 | ExclusiveLock    | t       | t
 relation     |    16385 |    12143 |      |       |            |               |         |       |          | 3/36
          | 741012 | AccessShareLock  | t       | t
 relation     |    16385 |    16392 |      |       |            |               |         |       |          | 3/36
          | 741012 | RowExclusiveLock | t       | t
 relation     |    16385 |    16386 |      |       |            |               |         |       |          | 3/36
          | 741012 | RowExclusiveLock | t       | t
 virtualxid   |          |          |      |       | 3/36       |               |         |       |          | 3/36
          | 741012 | ExclusiveLock    | t       | t
 transactionid |          |          |      |       |            |           522 |         |       |          | 4/175
          | 743348 | ShareLock        | f       | f
 transactionid |          |          |      |       |            |           522 |         |       |          | 3/36
          | 741012 | ExclusiveLock    | t       | f
 tuple        |    16385 |    16386 |    0 |     9 |            |               |         |       |          | 4/175
          | 743348 | ExclusiveLock    | t       | f
 transactionid |          |          |      |       |            |           523 |         |       |          | 4/175
          | 743348 | ExclusiveLock    | t       | f
(11 filas)
```

No, no es lógica la información porque la transacción T1 con id 522 pide un bloqueo compartido que no se le concede, teniendo el exclusivo que sí está concedido. Y la transacción T2 con id 523 se le concede un bloqueo exclusivo a pesar de estar bloqueado por la transacción T1.

Cuestión 24: Comprometa la transacción T1, ¿Qué es lo que ocurre? ¿Por qué? ¿Cuál es el estado final de la información de la tienda con código 6789 para ambos usuarios? ¿Por qué?

T1 se compromete:

```
tienda=> COMMIT;
COMMIT
```

T2 continua porque T1 ha realizado el COMMIT y ha liberado su cerrojo exclusivo:

```
tienda=> UPDATE "Tienda" SET "Nombre" = 'Saturn' WHERE "Id_tienda" = 6789;
UPDATE 1
```

Al consultar el estado final de la información de la tienda con código 6789 con el usuario1 se ve la modificación que ha realizado este mismo usuario pero no la del usuario2 porque T2 todavía no se ha comprometido:

```
tienda=> select * from "Tienda" where "Id_tienda" = 6789;
 Id_tienda |   Nombre   | Ciudad | Barrio | Provincia
-----------+------------+--------+--------+-----------
      6789 | Mediamarkt | c6789  | b6789  | p6789
(1 fila)
```

En cambio el usuario2 no ve la modificación del usuario1 porque está viendo la modificación de su propia transacción:

```
tienda=> select * from "Tienda" where "Id_tienda" = 6789;
 Id_tienda | Nombre | Ciudad | Barrio | Provincia
-----------+--------+--------+--------+-----------
      6789 | Saturn | c6789  | b6789  | p6789
(1 fila)
```

Cuestión 25: Comprometa la transacción T2, ¿Qué es lo que ocurre? ¿Por qué? ¿Cuál es el estado final de la información de la tienda con código 6789? ¿Por qué?

T2 se compromete:

```
tienda=> COMMIT;
COMMIT
```

Al consultar el estado final de la información de la tienda se puede ver que se mantiene la modificación de T2 porque ha sido el último en comprometerse (escribir a disco)

```
tienda=> select * from "Tienda" where "Id_tienda" = 6789;
 Id_tienda | Nombre | Ciudad | Barrio | Provincia
-----------+--------+--------+--------+-----------
      6789 | Saturn | c6789  | b6789  | p6789
(1 fila)
```

Cuestión 26: Cerrar todas las sesiones anteriores. Abrir una sesión con el usuario1 de la base de datos **TIENDA**. Abrir una transacción T1 en este usuario y realizar una modificación del ticket con número 54321 para cambiar su código a 223560. Abra otro usuario diferente del anterior y realice una transacción T2 que cambie la fecha del ticket con número 54321 a la fecha actual. No cierre la transacción.

Se inicia sesión de usuario1 con: psql -U usuario1 -d tienda
Se abre la transacción T1 y se realiza la modificación:

```
tienda=> BEGIN;
BEGIN
tienda=> UPDATE "Ticket" SET "Nº de tickect" = 223560 WHERE "Nº de tickect" = 54321;
UPDATE 1
```

Se inicia sesión de usuario2 con: psql -U usuario1 -d tienda

Se realiza la transacción T2 para cambiar la fecha:

```
tienda=> BEGIN;
BEGIN
tienda=> UPDATE "Ticket" SET fecha = '2020-06-01' WHERE "Nº de tickect" = 54321;
```

En este momento, T2 se ha bloqueado.

Cuestión 27: Comprometa la transacción T1, ¿Qué es lo que ocurre? ¿Por qué? ¿Cuál es el estado de la información del ticket con código 54321 para ambos usuarios? ¿Por qué?

T1 se compromete

```
tienda=> COMMIT;
COMMIT
```

T2 continua ya que T1 ha hecho COMMIT y ha soltado el bloqueo exclusivo:

```
tienda=> UPDATE "Ticket" SET fecha = '2020-06-01' WHERE "N‖ de tickect" = 54321;
UPDATE 0
```

Pero no ha realizado ninguna actualización como se puede comprobar en la captura de pantalla que devuelve UPDATE 0.

Si se consulta los datos con el usuario1:

```
tienda=> select * from "Ticket" where "N‖ de tickect" = 54321;
 N‖ de tickect | Importe | fecha | codigo_trabajador_Trabajador
---------------+---------+-------+------------------------------
(0 filas)


tienda=> select * from "Ticket" where "N‖ de tickect" = 223560;
 N‖ de tickect | Importe |   fecha    | codigo_trabajador_Trabajador
---------------+---------+------------+------------------------------
        223560 |      25 | 2017-03-14 |                            1
(1 fila)
```

No existe ningún ticket con el número de ticket antiguo 54321 pero sí con el nuevo.

Con el usuario2 ocurre lo mismo:

```
tienda=> select * from "Ticket" where "N‖ de tickect" = 54321;
 N‖ de tickect | Importe | fecha | codigo_trabajador_Trabajador
---------------+---------+-------+------------------------------
(0 filas)


tienda=> select * from "Ticket" where "N‖ de tickect" = 223560;
 N‖ de tickect | Importe |   fecha    | codigo_trabajador_Trabajador
---------------+---------+------------+------------------------------
        223560 |      25 | 2017-03-14 |                            1
(1 fila)
```

Esto ocurre porque T1 se compromete antes que T2 y modifica el número de ticket, por lo que cuando T2 va a realizar la modificación usando el número de ticket antiguo 54321 ya no se realiza porque T1 lo ha cambiado antes.

Cuestión 28: Comprometa la transacción T2, ¿Qué es lo que ocurre? ¿Por qué? ¿Cuál es el estado final de la información del ticket con número 54321 para ambos usuarios? ¿Por qué?

T2 se compromete:

```
tienda=> COMMIT;
COMMIT
```

Se consulta para usuario1 y usuario2, dando el mismo resultado en ambos:

```
tienda=> select * from "Ticket" where "Nº de tickect" = 54321;
 Nº de tickect | Importe | fecha | codigo_trabajador_Trabajador
---------------+---------+-------+------------------------------
(0 filas)


tienda=> select * from "Ticket" where "Nº de tickect" = 223560;
 Nº de tickect | Importe |   fecha    | codigo_trabajador_Trabajador
---------------+---------+------------+------------------------------
        223560 |      25 | 2017-03-14 |                            1
(1 fila)
```

```
tienda=> select * from "Ticket" where "Nº de tickect" = 54321;
 Nº de tickect | Importe | fecha | codigo_trabajador_Trabajador
---------------+---------+-------+------------------------------
(0 filas)


tienda=> select * from "Ticket" where "Nº de tickect" = 223560;
 Nº de tickect | Importe |   fecha    | codigo_trabajador_Trabajador
---------------+---------+------------+------------------------------
        223560 |      25 | 2017-03-14 |                            1
(1 fila)
```

Como en la cuestión anterior, T2 a pesar de comprometerse no ha modificado la fecha del ticket con número 54321 porque antes T1 hizo COMMIT con la modificación del número de ticket a 223560.

Cuestión 29: ¿Qué es lo que ocurre en el sistema gestor de base de datos si dentro de una transacción que cambia el importe del ticket con número 223560 se abre otra transacción que borre dicho ticket? ¿Por qué?
Se abre una transacción que se modifica ticket y dentro de esa se abre otra que lo borra:

```
tienda=> BEGIN;
BEGIN
tienda=> UPDATE "Ticket" SET "Importe" = 46 WHERE "Nº de tickect" = 223560;
UPDATE 1
tienda=> BEGIN;
WARNING:  ya hay una transacción en curso
BEGIN
tienda=> DELETE FROM "Ticket" WHERE "Nº de tickect" = 223560;
DELETE 1
tienda=> SELECT * FROM "Ticket" WHERE "Nº de tickect" = 223560;
 Nº de tickect | Importe | fecha | codigo_trabajador_Trabajador
---------------+---------+-------+------------------------------
(0 filas)
```

Lo que ocurre es que el segundo BEGIN se ignora; aparece el WARNING diciendo que ya hay una transacción en curso. De esta forma es como si se estuviera realizando solamente una transacción que crea un ticket y lo borra y por eso la tabla aparece vacía.

Cuestión 30: Suponer que se produce una pérdida del cluster de datos y se procede a restaurar la instancia de la base de datos del punto 6. Realizar solamente la restauración (recovery) mediante el procedimiento descrito en el apartado 25.3 del

manual (versión, 12) *"Continous Archiving and point-in-time recovery (PITR).* ¿Cuál es el estado final de la base de datos? ¿Por qué?

La restauración mediante el procedimiento descrito en el manual de PostgreSQL sigue los siguientes pasos:

1. Se detiene el servidor.
2. Copiar el directorio de datos del clúster en una ubicación temporal. Guardar el contenido de pg_wal del clúster debido a que puede tener registros que no se archivaron.
3. Eliminar los archivos y subdirectorios del directorio de datos del clúster y en la raíz directorios de cualquier espacio de tabla que esté utilizando.
4. Restaurar los archivos de la base de datos desde la copia de seguridad del sistema de archivos.
5. Eliminar los archivos presentes en pg_wal.
6. Copiar en pg_wal los archivos de segmento WAL que están sin archivar y que se guardaron en el paso 2.
7. Poner en postgresql.conf la configuración de recuperación (restore_command = 'copy "..\\archivedir\\%f" "%p"') y crear un archivo recovery.signal.
8. Iniciar el servidor. Este entrará en modo recuperación y leerá los archivos guardados.
9. Asegurarse de que todo se ha recuperado correctamente.

Después de realizar estos pasos, comprobamos con consultas que la base de datos se ha recuperado completamente.

```
tienda=> SELECT * FROM "Tienda";
 Id_tienda |    Nombre     | Ciudad | Barrio | Provincia
-----------+---------------+--------+--------+-----------
         1 | n1            | c1     | b1     | p1
         2 | n2            | c2     | b2     | p2
      1000 | n1000         | c1000  | b1000  | p1000
      2000 | n2000         | c2000  | b2000  | p2000
     31145 | Tienda Alcalá | c31145 | b31145 | p31145
      6789 | Saturn        | c6789  | b6789  | p6789
(6 filas)
```

Podemos ver que las tuplas insertadas durante toda la práctica siguen estando en la tabla. Esto se debe a que la recuperación continua a través de todos los segmentos WAL disponibles, restaurando así la base de datos al punto actual en el tiempo (o lo más cerca posible dados los segmentos WAL disponibles).

Cuestión 31: A la vista de los resultados obtenidos en las cuestiones anteriores, ¿Qué tipo de sistema de recuperación tiene implementado postgreSQL? ¿Qué protocolo de gestión de la concurrencia tiene implementado? ¿Por qué? ¿Genera siempre planificaciones secuenciables? ¿Genera siempre planificaciones recuperables? ¿Tiene rollbacks en cascada? Justificar las respuestas.

El tipo de sistema de recuperación que tiene implementado PostgreSQL es diferido porque, como se puede observar en la cuestión 11, la transacción T1 todavía no se ha comprometido (no ha hecho COMMIT), por lo tanto cuando el usuario2 consulta estos datos no existen aún en la base de datos.

El protocolo de gestión de la concurrencia que tiene implementado es el de 2 fases riguroso refinado porque los bloqueos se sueltan después de hacer el COMMIT, como ocurre en la cuestión 24, y se puede pedir más de un tipo de bloqueo porque en la cuestión 20 las dos transacciones están pidiendo bloqueos compartidos, actualizando así los bloqueos que tenían antes que eran exclusivos.

No genera siempre planificaciones secuenciables porque, por ejemplo, en la cuestión 20, se produce un interbloqueo entre T1 y T2, lo que significa que en el grafo de precedencia hay bucles.

No tiene rollbacks en cascada ya que, en la cuestión 22, 23 y 24, la transacción T1 escribe en Tienda y hace COMMIT antes de que T2 lea de la tabla Tienda.

Genera siempre planificaciones recuperables debido a que no tiene rollbacks en cascada.

Bibliografía

- Capítulo 13: Concurrency Control.
- Capítulo 25: Backup and Restore.
- Capítulo 27: Monitoring Database Activity.
- Capítulo 29: Reliability and the Write-Ahead log.