TICS565 - Técnicas para Big Data

Sharding en PostgreSQL

1. Introducción

En esta guía vamos a estudiar como dividir una base de datos Postgres en dos servidores. En concreto, vamos a hacer *sharding* horizontal, en un caso de estudio que presentamos a continuación.

Supongamos que tenemos una universidad con sede en **Santiago** y **Viña del Mar**, y queremos guardar registro de los libros solicitados a cada una de las bibliotecas de las respectivas sedes. Así, nuestro modelo se ve originalmente de la siguiente forma:

```
Alumnos(id SERIAL PRIMARY KEY, name VARCHAR);

Libros(id SERIAL PRIMARY KEY, title VARCHAR);

Ejemplar_Libro(id SERIAL PRIMARY KEY, state VARCHAR, lendable VARCHAR,
    location VARCHAR, book_id INT REFERENCES book(id));

Prestamos(student_id INT REFERENCES student(id),
    booksample_id INT REFERENCES booksample(id), at DATE, returned_at DATE);
```

En donde estamos guardando alumnos y libros. Luego, tenemos la tabla `Ejemplar_Libro`, que guarda un ejemplar concreto del libro (esto es para representar que en la biblioteca puedes encontrar más de un ejemplar del mismo libro). Luego, guardamos los prestamos, en el que un alumno pide un ejemplar particular de un libro.

Ahora bien, suena razonable tener un servidor dedicado para la biblioteca en Santiago y otro para la de Viña del Mar. Por lo mismo, vamos a hacer *sharding* horizontal para los libros dependiendo si pertenecen a una ciudad o la otra. Las tablas que serán divididas son *Ejemplar_Libro* y `Prestamo`, ya que hace sentido que haya una separación por ciudad para estas tablas.

2. Creando las tablas

En este tutorial vamos a asumir que disponemos de dos servidores, **master** y **local**. El servidor master, en este caso el de Santiago, es el principal de la aplicación, donde definiremos todas las tablas. Mientras tanto el servidor local es el de Viña del Mar, donde serán alma-

cenados los datos de los ejemplares de libros y prestamos de aquella sede. Ahora partiremos con el servidor de Santiago (**master**).

2.1. Creando tablas en el servidor master (Santiago)

Primero creamos la base de datos:

```
CREATE DATABASE libros_test;
```

Recuerda que después te tienes que conectar a esta base de datos con:

```
\connect libros_test;
```

Ahora creamos nuestras tablas:

```
DROP TABLE IF EXISTS Alumnos CASCADE;
CREATE TABLE Alumnos(id SERIAL PRIMARY KEY, name VARCHAR);

DROP TABLE IF EXISTS Libros CASCADE;
CREATE TABLE Libros(id SERIAL PRIMARY KEY, title VARCHAR);

DROP TABLE IF EXISTS Ejemplar_Libro CASCADE;
CREATE TABLE Ejemplar_Libro(id SERIAL PRIMARY KEY, state VARCHAR,
    lendable VARCHAR, location VARCHAR, book_id INT REFERENCES Libros(id));

DROP TABLE IF EXISTS Prestamos CASCADE;
CREATE TABLE Prestamos(student_id INT REFERENCES Alumnos(id),
    booksample_id INT REFERENCES Ejemplar_Libro(id), at DATE, returned_at DATE);
```

Ahora vamos a crear una tabla mediante el comando `INHERITS` de Postgres, que nos sirve para hacer herencia en el contexto del modelo relacional. Esto lo haremos para guardar los datos relacionados a los ejemplares de libro y prestamos de Santiago.

```
DROP TABLE IF EXISTS EjemplarLibro_Stgo CASCADE;
CREATE TABLE EjemplarLibro_Stgo(CHECK(location='Santiago'))
    INHERITS(Ejemplar_Libro);

DROP TABLE IF EXISTS Prestamos_Stgo CASCADE;
CREATE TABLE IF NOT EXISTS Prestamos_Stgo() INHERITS(Prestamos);
```

Aquí la condición `CHECK(location='Santiago')` nos ayuda a que Postgres pueda optimizar de mejor forma las consultas. Ahora tenemos que crear las tablas en el servidor local.

2.2. Creando tablas en el servidor local (Viña del Mar)

Aquí también tenemos que crear la base de datos:

```
CREATE DATABASE libros_test;
```

Y debes recordar conectarte como lo hiciste anteriormente.

Ahora bien, como el servidor **master** debe acceder remotamente a este servidor, debemos abrir las conexiones. Para esto, tenemos que editar el archivo:

```
/etc/postgresql/XX/main/postgresql.conf
```

Donde `XX` es la versión de Postgres que tenemos instalada. Aquí editamos la variable `listen_addresses` y le damos el valor `'*'`. Ojo que esta línea viene comentada, por lo que tienes que asegurarte de descomentarla:

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

Luego, también debemos editar la configuración IPv4 en el archivo:

```
/etc/postgresql/XX/main/pg_hba.conf
```

para que se vea de la siguiente forma:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# IPv4 local connections:
host    all             all             all                     md5
```

Para que estos cambios tengan efecto, debes reiniciar Postgres. En general esto se hace con el comando:

```
sudo service restart postgresql
```

Luego, para saber si desde un servidor externo puedes acceder, puedes usar el comando `pg_isready`. Puedes buscar más información sobre esto en la web.

Ahora vamos a crear las tablas necesarias, que vendrían a ser las tablas locales en este servidor:

```
DROP TABLE IF EXISTS EjemplarLibro_Vinia CASCADE;
CREATE TABLE EjemplarLibro_Vinia(id INT, state VARCHAR,
    lendable BOOL DEFAULT FALSE, location VARCHAR, book_id INT);

DROP TABLE IF EXISTS Prestamos_Vinia CASCADE;
CREATE TABLE Prestamos_Vinia(student_id INT,
    booksample_id INT, at DATE, returned_at DATE);
```

Ahora, debemos hacer que el servidor **master** pueda insertar y acceder a estas tablas. Así que volvemos al servidor Santiago.

3. Accediendo a una tabla remota

Ahora en el servidor **master**, vamos a indicar que ciertos datos deben ser extraidos del servidor **local**. Para esto vamos a usar el comando `FOREIGN TABLE`. Para esto debemos instanciar la extensión `postgres_fdw`.

```
CREATE EXTENSION postgres_fdw;

CREATE SERVER master_server FOREIGN data wrapper postgres_fdw
OPTIONS (host '{ip address of the local postgres server}',
    port '5432' , dbname 'libros_test');

CREATE USER MAPPING FOR master_user SERVER master_server
OPTIONS (user '{our username on local server}',
    password '{our password on local server}');

ALTER SERVER master_server OWNER TO master_user;
```

En las instrucciones anteriores, necesitamos **reemplazar algunos valores**, por ejemplo, la ip del servidor **local** (Viña del Mar), el puerto (por defecto es 5432, pero puede cambiar). Además, `master_user` es el usuario de Postgres en el servidor **master**. Luego también hay que agregar el usuario y contraseña del usuario en el servidor **local**.

Ahora, tenemos que indicar la forma de acceder a las tablas externas:

```
DROP FOREIGN TABLE IF EXISTS EjemplarLibro_Vinia CASCADE;
CREATE FOREIGN TABLE EjemplarLibro_Vinia (CHECK(location='Viña del Mar'))
INHERITS(Ejemplar_Libro) SERVER master_server;

DROP FOREIGN TABLE IF EXISTS Prestamos_Vinia CASCADE;
CREATE FOREIGN TABLE Prestamos_Vinia()
INHERITS(Prestamos) SERVER master_server;
```

Ahora, para terminar, necesitamos que los datos sean insertados en las tablas correspondientes. Para esto, vamos a manejar la situación con *triggers*.

4. Manejando la inserción

Ahora vamos a crear dos *triggers* en el servidor **master** (Santiago). Para manejar la inserción en la tabla `Ejemplar_Libro`, tenemos que crear el siguiente *trigger*.

```
CREATE OR REPLACE FUNCTION booksample_trigger_fn() RETURNS TRIGGER AS
$$
BEGIN

    IF new.location = 'Santiago' THEN
        INSERT INTO EjemplarLibro_Stgo VALUES(new.*);
    ELSIF new.location = 'Viña del Mar' THEN
        INSERT INTO EjemplarLibro_Vinia values(new.*);
    END IF;

    RETURN null;
END
$$
language plpgsql;

DROP TRIGGER IF EXISTS booksample_trigger ON Ejemplar_Libro;
CREATE TRIGGER booksample_trigger BEFORE INSERT ON Ejemplar_Libro
FOR EACH ROW EXECUTE PROCEDURE booksample_trigger_fn();
```

Y para manejar la inserción en la tabla `Prestamos`:

```
CREATE OR REPLACE FUNCTION lend_trigger_fn() RETURNS TRIGGER AS
$$
DECLARE
    vbooksample Ejemplar_Libro%rowtype;
BEGIN
    SELECT * INTO vbooksample
    FROM Ejemplar_Libro
    WHERE id=new.booksample_id;

    IF vbooksample.location = 'Santiago' THEN
        INSERT INTO Prestamos_Stgo VALUES(new.*);
    ELSIF vbooksample.location = 'Viña del Mar' THEN
        INSERT INTO Prestamos_Vinia VALUES(new.*);
    END IF;

    RETURN null;
END
$$
language plpgsql;

DROP TRIGGER IF EXISTS lend_trigger ON Prestamos;
CREATE TRIGGER lend_trigger BEFORE INSERT ON Prestamos
FOR EACH ROW EXECUTE PROCEDURE lend_trigger_fn();
```

Así, estamos listos para poder utilizar nuestra base de datos distribuida. Ahora, vamos a probar cómo funciona.

5. Usando la base de datos distribuida

Primero vamos a insertar datos:

```
INSERT INTO Libros(title) VALUES('book#1');
INSERT INTO Alumnos(name) VALUES('std#1'),('std#2'), ('std#3');

INSERT INTO Ejemplar_Libro(state, lendable, location, book_id)
VALUES('old', true, 'Viña del Mar', 1),
  ('old', true, 'Santiago', 1),
  ('new', true 'Viña del Mar', 1);

INSERT INTO Prestamos(student_id , booksample_id ,at)
VALUES(1,1, now()), (2,2, now()), (3,3, now());
```

Si exploras las tablas, notarás que gracias a los *triggers* los datos son insertados en los servidores respectivos. Puedes ver cómo lo hace el sistema para reponder las consultas con el comando `EXPLAIN`.

```
EXPLAIN SELECT * FROM Ejemplar_Libro;

EXPLAIN SELECT * FROM Ejemplar_Libro WHERE location='Santiago';

EXPLAIN SELECT * FROM Ejemplar_Libro WHERE location='Viña del Mar';
```

Fuentes: Este tutorial fue realizado gracias a la documentación oficial de Postgres:

`https://www.postgresql.org/docs/9.1/ddl-partitioning.html`

Y gracias al siguiente tutorial:

`https://dev.to/aurelmegn/setting-up-distributed-database-architecture-with-postgresql-261`