Aurelio

Publicado el 20 oct 2019 • Actualizado el 22 oct 2019

Configuración de la arquitectura de base de datos distribuida con postgresql

#postgres #base de datos #sistemas distribuidos

¿Qué es una base de datos distribuida?

Bases de datos distribuidas son un conjunto de bases de datos divididas en diferentes ubicaciones que se comunican y brindan servicios a través de una red.

Una base de datos distribuida bien diseñada debe proporcionar:

- una transparencia de red: los usuarios finales no deben saber que la base de datos está dividida en diferentes ubicaciones, deben ejecutar consultas como lo hacen en una arquitectura de base de datos normal

- una transparencia de la arquitectura: los usuarios no conocen la arquitectura detrás de la base de datos

Pros y contras de las bases de datos distribuidas

/!\ no crea una arquitectura de base de datos distribuida porque lo desee, sino porque su proyecto lo necesita. Las bases de datos distribuidas proporcionan (cuando están bien diseñadas)

- escalar fácilmente toda su producción • replicación fácil de configurar para mantener la integridad de sus datos
- alta disponibilidad
  - conmutación por error

algunos contras son:

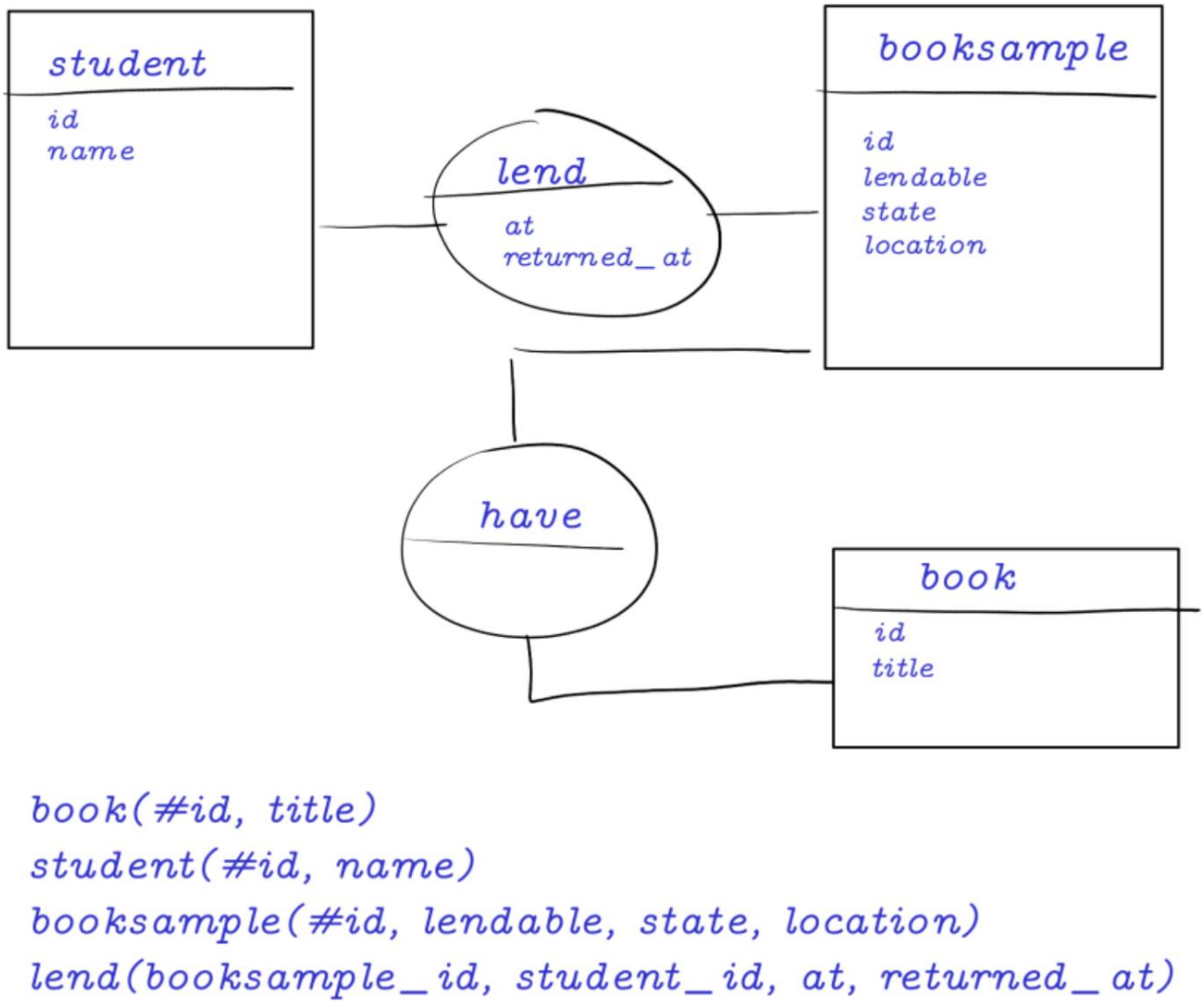
- no es fácil de mantener

- no es fácil de configurar

En esta publicación, usaremos postgresql como DBMS para configurar nuestra arquitectura:.

Caso de estudio

Consideremos este modelo relacional.



Describe un escenario en el que los estudiantes pueden prestar muestras de libros de una biblioteca. Supongamos que la biblioteca está dividida en dos ciudades: (París y Lagos). En las consultas generadas de nuestra aplicación, notamos que muchas consultas son de este tipo:

```
select booksample.state, booksample.lendable, booksample.ubicación from booksample select booksample.state, booksample.lendable, booksample.ubicación from booksample
```

Para acelerar la ejecución de consultas, decidimos configurar bases de datos distribuidas en dos sitios, París y Lagos.

dos sitios, París y Lagos.

Particionamiento

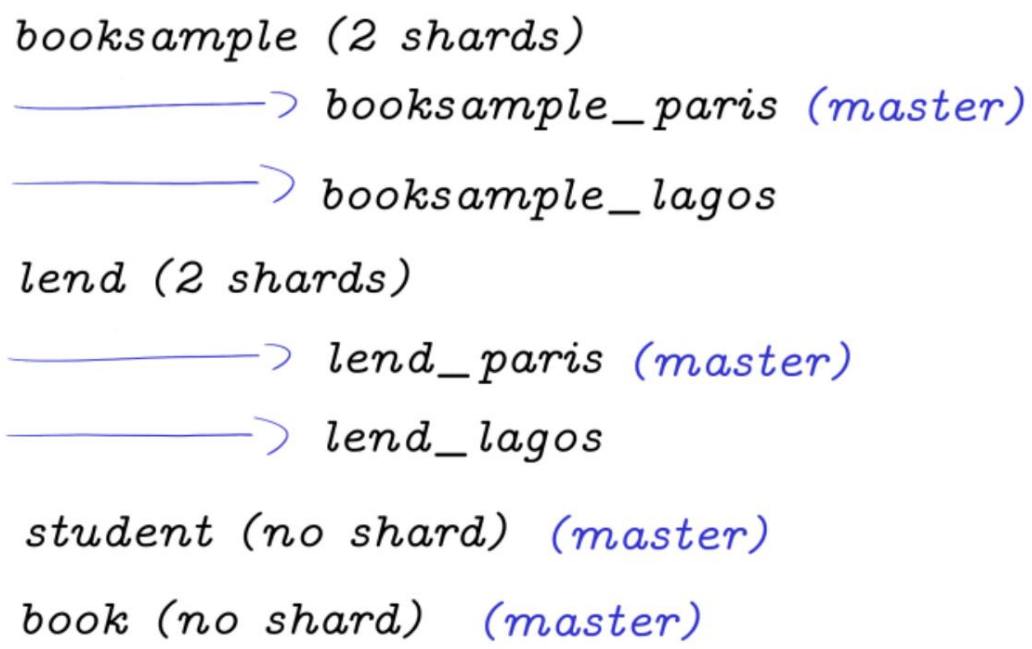
Existen muchos métodos de partición de tablas y se utilizan en bases de datos distribuidas.

- Particionamiento horizontal: se trata de dividir las filas según los valores de la atributos de la tabla.
- Partición vertical: se trata de dividir las columnas de la tabla en diferentes servidores.

En esta publicación, haremos una partición horizontal.

A partir de las consultas anteriores, podemos:

- Partición de la tabla de muestras de libros en dos tablas:
  - ÿ booksample_paris
  - ÿ booksample_lagos
- Partición de la tabla de préstamos en dos tablas: ÿ Lend_paris
  - ÿ prestar_lagos



/!\

/!\

• la intersección de las particiones de una tabla debe ser nula • la unión de las particiones de una tabla debe contener todas las filas de la tabla inicial

En el servidor maestro

Vamos a crear la tabla diferente en la base de datos maestra.

La base de datos maestra es la que está al frente del usuario, crearemos las tablas como mesa normal para empezar. Aquí usaríamos el servidor de París como maestro.

crear la base de datos:

```
crear base de datos dd_test;
```

crear las tablas estudiante y libro:

```
--estudiante de mesa
soltar tabla si existe cascada de estudiantes ; crear
estudiante de tabla ( clave principal de serie de identificación, nombre varchar); --libro
de mesa
soltar tabla si existe cascada de libros ; crear
libro de tabla ( clave principal de serie de identificación , título varchar);
```

Muestra de libro de mesa:

```
soltar la tabla si existe una cascada de muestras de libros;
crear tabla de muestra de libros ( clave principal de serie de identificación , varchar de estado , varchar prestable
```

préstamo de mesa:

```
soltar tabla si existe prestar cascada; crear tabla
prestar(student_id int referencias estudiante(id), booksample_id int referencias
```

En el servidor de París

En este ejemplo, como estamos usando el sitio de París como maestro, esta consulta debe ejecutarse en el mismo servidor de postgres que la anterior. Vamos a crear las tablas booksample_paris y lend_paris

```
-- tabla desplegable
del sitio de París si existe booksample_paris cascade; create
table muestra_libro_paris(check(ubicación='paris')) hereda(muestra_libro);

soltar tabla si existe Lend_paris cascade; crear tabla si
no existe lend_paris() hereda(prestamo);
```

El check (ubicación = 'paris') ayudará a postgres a buscar la fila de una manera "inteligente".

Utilizo la palabra clave heritage para decirle a postgres que la tabla booksample_paris es en realidad parte de la tabla booksample. Así, cuando emitimos un:

```
explicar seleccionar * de muestra de libros;
```

obtenemos

```
dd_test=> explicar select * from booksample;
                                PLAN DE CONSULTA
-------------------------------------------------------------------------
 Agregar (costo = 0.00..151.56 filas = 1251 ancho = 104)
   -> Seq Scan en muestra de libro (costo = 0.00...0.00 filas = 1 ancho = 104)
   -> Seq Scan en booksample_paris (costo=0.00...16.10 filas=610 ancho=104) (3 filas)
```

Significa que postgres intentará tomar las filas de la tabla booksample_paris en una selección.

En el servidor de Lagos

Creamos la base de datos

```
crear base de datos dd_test;
```

Aquí vamos a configurar el servidor postgres para escuchar las otras interfaces de red. Esto se hace modificando el archivo de configuración ubicado en (en la mayoría de los sistemas operativos Linux) /etc/postgresql/9.5/main/postgresql.conf

```
#------------------------------------------------ ------------------------------------------
# CONEXIONES Y AUTENTICACIÓN
```

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

# - Configuración de conexión -

escucha_direcciones = '*'                # en qué dirección(es) IP escuchar;
                                         # lista de direcciones separadas por comas;
                                         # por defecto es 'localhost'; use # (el cambio '*' para todos
                                         requiere reiniciar)
```

El siguiente paso es permitir que un usuario se conecte a través de las interfaces de red

modificando /etc/postgresql/9.5/main/pg_hba.conf

```
# TIPO BASE DE DATOS        USUARIO          DIRECCIÓN                MÉTODO

# Conexiones locales IPv4:
anfitrión    todos          usuario_prueba   todos                    md5
```

Vamos a crear ahora las tablas de particiones:

```
-- booksample_lagos
soltar la tabla si existe la cascada de booksample_lagos ;
crear tabla booksample_lagos(id int, state varchar, prestable bool default false

-- presta_lagos
soltar la tabla si existe la cascada de lend_lagos ;
crear tabla lend_lagos(student_id int, booksample_id int, en la fecha, devuelto_en
```

En el servidor maestro

Usaremos la función de tabla externa ** de postgres para poder acceder a la
**Tablas de la base de datos de Lagos de forma remota desde el servidor maestro. Para poder hacer esto, debemos crear la extensión postgres_fdw en nuestra base de datos. Esta acción debe ser hecho solo por un administrador, así que vamos a conectarnos a la base de datos como administrador usuario de postgres y hacer:

/!\ Hay muchas alternativas al uso de tablas foráneas, como el uso de tablas materializadas visualizaciones + disparadores o partición postgres rasgo.

```
crear extensión postgres_fdw;
```

```
crear servidor master_server contenedor de datos externos opciones postgres_fdw (host '{ip addr

crear asignación de usuarios para las opciones de master_server server master_user (usuario '{nuestro nombre de usuario

modificar el propietario del servidor master_server a master_user;
```

Primero, creamos la extensión postgres_fdw y después de un "servidor de datos extranjeros" en el servidor maestro de postgres. Ahora creamos un mapeo de usuarios para poder consultar el servidor de Lagos.

Ahora, creemos las tablas foráneas ubicadas en el servidor maestro que se asignan al fragmento en los servidores de Lagos.

```
suelte la tabla foránea si existe la cascada de booksample_lagos ; crear tabla
externa booksample_lagos (check(ubicación='lagos')) hereda(booksample soltar tabla externa si existe lend_lagos
cascade; crear tabla externa lend_lagos () hereda(prestar) servidor master_server;
```

En su estado actual, el servidor maestro llenará la tabla booksample y prestará cuando se ejecute una consulta como esta.

```
insertar en valores de muestra de libros (1, 'nuevo', 'paris', 1)
```

Este no es un buen comportamiento ya que las nuevas particiones que creamos no contendrán ningún dato. Para solucionar esta situación, utilizaremos "disparadores" para redirigir la fila a su destino normal.

El disparador a continuación es para redirigir la inserción de la muestra de libros a la partición correcta: ya sea booksample_lagos o booksample_paris según el valor de la ubicación del atributo:

```
-- disparador al insertar la función de
creación o reemplazo de la muestra del libro booksample_trigger_fn() devuelve el disparador como
$$
empezar

    si new.ubicación = 'paris' entonces inserte
        en los valores de booksample_paris(new.*); elsif
    nueva.ubicación = 'lagos' entonces
        insertar valores en booksample_lagos (nuevo .*);
```

```
            insertar valores en booksample_lagos (nuevo .*);
        terminara si;

        devolver nulo;
final
$$
lenguaje plpgsql;

suelte el activador si existe booksample_trigger en booksample; crear disparador
booksample_trigger antes de insertar en booksample para cada fila ejecutar pr
```

Ahora, nos gustaría redirigir las consultas en la tabla prestada a la partición correcta.

Aquí almacenamos la fila en el sitio al que pertenece la muestra del libro.

```
crear o reemplazar la función lend_trigger_fn() devuelve el disparador como
$$
declarar
        vbooksample booksample%rowtype; comenzar:
seleccione la muestra de libros a la que hace
        referencia booksample_id select * into vbooksample from booksample where
        id=new.booksample_id;

        -- obtener la ubicación a usar y guardar la fila si
        vbooksample.ubicación = 'paris' luego
            insertar en los valores de lend_paris (nuevo.*); elsif
        vbooksample.ubicación = 'lagos' luego inserte en los valores
            de lend_lagos (nuevo.*); terminara si;


        devolver nulo;
finalmente
$$
lenguaje plpgsql;

suelte el disparador si existe lend_trigger en préstamo; crear
desencadenante lend_trigger antes de insertar en préstamo para cada fila ejecutar procedimiento prestar
```

Nuestra base de datos ya es funcional.

Prueba

Ejecutemos una consulta para obtener una descripción general de las capacidades de nuestra base de datos:

Ejecutemos una consulta para obtener una descripción general de las capacidades de nuestra base de datos:

```
insertar en los valores del libro (título) ('libro # 1');
insertar en los valores del estudiante (nombre) ('std # 1'), ('std # 2'), ('std # 3');

-- gracias al activador que escribimos, la inserción se ejecutará en el servidor correcto insertar en booksample(estado, prestable, ubicación, book_id) valores('antiguo', verdadero

inserte en los valores de préstamo (student_id, booksample_id , at) (1,1, ahora ()), (2,2, ahora
```

Veamos qué sucede cuando tratamos de seleccionar todas las filas.

```
explicar seleccionar * de muestra de libros;

-- gracias a la restricción check(ubicación='{}'), la consulta de selección se ejecuta en una selección de explicación * de muestra de libros donde ubicación='lagos';
                              PLAN DE CONSULTA

 Agregar (costo = 0.00..118.08 filas = 4 ancho = 104)
    -> Seq Scan en muestra de libro (costo = 0.00...0.00 filas = 1 ancho = 104)
          Filtro: ((ubicación)::texto = 'lagos'::texto)
    -> Escaneo extranjero en booksample_lagos (costo=100.00..118.06 filas=3 ancho=104
(4 filas)

explique select * from booksample where location='paris';
                              PLAN DE CONSULTA

 Agregar (costo = 0.00..118.08 filas = 4 ancho = 104)
    -> Seq Scan en muestra de libro (costo = 0.00...0.00 filas = 1 ancho = 104)
          Filtro: ((ubicación)::texto = 'paris'::texto)
    -> Escaneo extranjero en booksample_paris (costo=100.00..118.06 filas=3 ancho=104
(4 filas)
```

Gracias a la restricción check(ubicación='{}') , la consulta de selección se ejecuta en un "forma inteligente. El DBMS escanea solo la tabla en el servidor de Paris o en el servidor de Lagos.

Gracias por su atención.

Comentarios principales (0)

ÿÿÿ La vida es demasiado corta para navegar sin el modo oscuro

Aurelio

un desarrollador de software al que le gustan los desafíos

UNIDO
12 abr 2018

Tendencia en la comunidad DEV ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

Estimados gerentes de proyecto: ¡Detengan la microgestión ahora!

productividad # profesional

Un teclado al 60% es bueno para ti

principiantes en # programación de productividad

Cómo optimizar su canalización de ETL para obtener la máxima eficiencia

aprendizaje # automático de la base # de datos de ciencia # de datos de python