User: root

Password: Cristian10

Server: localhost

**Backup base datos:**Texto

Descripción generada automáticamente

**Migrar base datos:**

mysqldump -u [usuario] -p [base\_de\_datos] --no-data > [archivo\_salida].sql

mysqldump -u root -p world –no-data > world-mysql.sql

perl mysql2pgsql.perl [opciones] sql\_formato\_mysql.sql sql\_formato\_postgresql.sql

perl mysql2pgsql.perl --nodrop world-mysql.sql world-postgresql.sql

psql -f world\_postgresql.sql -u usuario-db word nombrebd

**Comandos mysql:**

system cls

limpiar consola

show databases;

Mostrar BDS

use nombre\_BD;

Seleccionar BD

show tables;

Mostrar tablas de BD

describe nombre\_tabla;

desc nombre\_tabla;

Describir tabla

select nom\_tabla.atrib\_1, nom\_tabla.atrib\_2 / \*

from nombre\_tabla

where condiciones

order by nom\_tab.atrib\_# desc;

select nom\_tabla.atrib\_1, nom\_tabla.atrib\_2 / \*

from nombre\_tabla (izq)

inner join tabla2 (der)

on atributo tabla2

where condicionesorder by nom\_tab.atrib\_# desc;

create -> crear objetos de la bd

alter -> Modificar objeto de la bd

drop -> eliminar un objeto de la bd

truncate -> Vacia un objeto de la bd

alter table nombretabla

change nombreoriginal nombrenuevo tipo null/not null

alter table nombretabla

add constraint pk\_nombre

primary key(nombre); // Convertir atributo llave primaria

alter table nombretabla

drop primary key; // Eliminar llave primaria

alter table nombretabla

add constraint fk\_nombre

foreign key(fk\_nombrefk)

references nombretabla(nombre\_pk)

on delete no action

on update no action; // Convertir atributo llave foránea

insert into nombretabla(atributo1,atributo2,etc) values(4,'pepe','2021-11-25 14:55:45');

//agregar datos rapido

insert into nombre\_tabla values (datos), (datos2)

Agregar una columna: ADD “columna 1” “tipos de datos para columna 1”

Eliminar una columna: DROP “columna 1”

Cambiar el nombre de una columna: CHANGE “nombre antiguo de la columna” “nuevo nombre de la columna” “tipos de datos para la nueva columna".

Cambiar el tipo de datos para una columna: MODIFY “columna 1” “nuevo tipo de datos”

update nombre tabla

set atributo1='',atributo2=

where identificador de la fila=;

//consultas de agregación

count -> Total de filas

max -> Máximo valor de un campo

min -> Mínimo valor

sum -> Suma de un campo en específico

avg -> Devuelve promedio de un campo

**Comandos PostgreSQL**

\c nombredelabase -> para usar una bd

\dt -> mostrar relaciones de la bd

\l -> listar databases

\d nombretabla -> mostrar atributos de una tabla

alter table mi-tabla rename column nombre-antiguo to nombre-nuevo; -> Cambiar nombre de una columna

**-- Agregar datos**

alter table libros

add atributo tipodato unsigned not null,

atributo2 tipodato unsigned not null;

**-- Agregar un check**

alter table tabla

add constraint columna check (columna in (condicion1,condicion2));

create table usuarios(

nombre varchar(20),

clave varchar(10),

edad int not null check(edad;=18)

);

create table usuarios(

nombre varchar(20),

clave varchar(10),

edad int not null,

constraint chk\_usuario check

(edad;=18)

);

**-- Agregar PK**

alter table NOMBRETABLA

add constraint NOMBRECONSTRAINT

primary key (CAMPO,...);

create table usuarios(

nombre varchar(20),

clave varchar(10),

primary key(nombre)

);

Insert -> new

Update -> new y old

Delete -> old

**-- Trigger**

$$ -> significa que debe encontrar otro de estos para terminar la funcion

create or replace function insert\_trigger() returns trigger

as $$

declare

usuario varchar(45) := null;

fecha date := null;

begin

fecha = (select now());

usuario = (select usename from pg\_catalog.pg\_user

where usename= current user);

insert into "aud\_operando" values

(new.valor1,new.valor2,new.operacion,new.total, usuario, fecha);

return new;

end

$$

language plpgsql;

create trigger nuevo\_tr before insert

on operando

for each row

execute procedure insert\_trigger();

**-- if en postgre**

if new.operacion = 1 then

new.total = new.valor1 + new.valor2;

elsif new.operacion = 2 then

new.total = new.valor1 - new.valor2;

elsif new.operacion = 3 then

new.total = new.valor1 \* new.valor2;

else

new.total = new.valor1 / new.valor2;

end if;

**-- case en postgre**

case

when new.operacion = 1 then

new.total = new.valor1 + new.valor2;

when new.operacion = 2 then

new.total = new.valor1 - new.valor2;

when new.operacion = 3 then

new.total = new.valor1 \* new.valor2;

else

new.total = new.valor1 / new.valor2;

end case;

**-- Crear usuarios, administración y eliminación de roles y usuarios**

psql -U usuario nombre\_bd

create user/rol supervisor with privilegios password 'supervisor';

GRANT CONNECT ON DATABASE mydatabase TO readwrite;

grant privilegios on table/bd to user/rol;

revoke privilegios on table/bd to user/rol;

REASSIGN OWNED BY ryan TO postgres; -- or some other trusted role

DROP OWNED BY ryan;

-- repeat in ALL databases where the role owns anything or has any privileges!

DROP USER ryan;

Alter database nombrebd owner to nombreuser

**-- Privilegios**

SUPERUSER/NOSUPERUSER. Super usuario, privilegios para crear bases de datos y usuarios.

CREATEDB/NOCREATEDB. Permite crear bases de datos.

CREATEROLE/NOCREATEROLE. Permite crear roles.

CREATEUSER/NOCREATEUSER. Permite crear usuarios.

LOGIN/NOLOGIN. Este atributo hace la diferencia entre un rol y usuario. Ya que el usuario tiene permisos para acceder a la base de datos a traves de un cliente.

PASSWORD. Permite alterar la contraseña.

VALID UNTIL. Expiración de usuarios.

-- **Conectar bases de datos**

Cómo usar FDW para conectar dos bases de datos PostgreSQL remotas

Habilitamos la extensión fwd en la base de datos desde la que queremos acceder (en nuestro caso la local) y creamos opcionalmente un nuevo esquema para tenerlo todo más organizado:

CREATE EXTENSION postgres\_fdw;

CREATE SCHEMA esquema\_destino;

Opcionalmente nos aseguramos de eliminar la configuración al servidor remoto si ya la tuviésemos creada previamente.

DROP SERVER IF EXISTS servidor\_bd\_remota CASCADE;

Configuramos el acceso al servidor remoto:

CREATE SERVER servidor\_bd\_remota

FOREIGN DATA WRAPPER postgres\_fdw

OPTIONS (host 'xx.xx.xx.xx', dbname 'foo', port '5432');

Para consultar la base de datos remota es necesario saber qué usuario podrá realizar esta operación. Básicamente se trata de asignar un usuario local de nuestro servidor a un usuario del servidor remoto y para ello se crea una **asignación de usuarios**. En este caso el usuario es el mismo desde el que estamos operando (CURRENT\_USER), pero podría ser *postgres* o cualquier otro:

CREATE USER MAPPING FOR CURRENT\_USER SERVER servidor\_bd\_remota

OPTIONS (USER 'remote\_user', password 'admin');

Importamos el esquema remoto de tablas en el esquema de la base de datos local:

IMPORT FOREIGN SCHEMA esquema\_remoto

FROM SERVER servidor\_bd\_remota INTO esquema\_destino;

Finalmente, ya podemos consultar esta tabla externa como si estuviese en nuestra base de datos:

SELECT \* FROM tabla\_foranea;

Si quieres un ejemplo sencillo con el que poder practicar [aquí tienes el código](https://gist.github.com/egofer/39f4e5f1103a554e127d9d3f571480a1) para acceder mediante FDW a la base de datos de RNAcentral, una base de datos de acceso público de solo-lectura de secuencias de ácido ribonucleico (ARN).

Todo lo que hemos visto hasta ahora esta muy bien, pero sería muy interesante si pudiésemos utilizar estos contenedores de datos externos para acceder a geodatos.