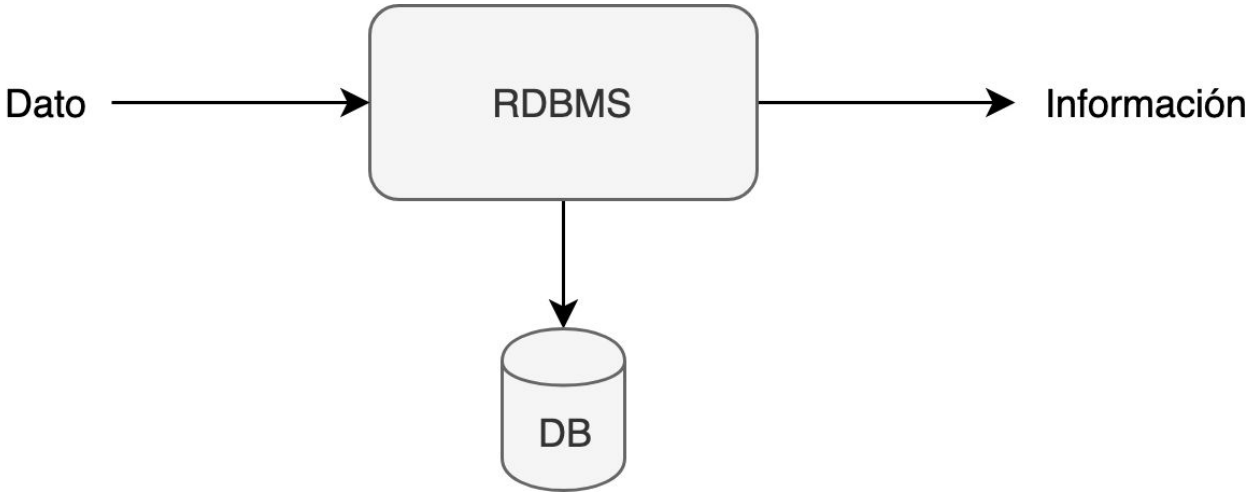
{desafío}
latam_

Modelamiento y gestión de bases de datos _

Introducción a las bases de datos

Bases de datos



¿Qué es SQL?

Structured **Q**uery **L**anguage (Lenguaje estructurado de consultas) es un lenguaje creado para definición y la manipulación de bases de datos relacionales.

Beneficio: Facilita la administración de datos almacenados.

Analicemos un caso cotidiano

Instalar y configurar PostgreSQL

¿Por qué PostgreSQL?

Ventajas:

- Licencia gratuita.
- Disponible para distintos sistemas operativos, como Windows, Linux y Unix, 32 y 64 bits.
- Bajo mantenimiento.
- Estabilidad, no se presentan caídas de bases de datos.
- Alto rendimiento
- Gran capacidad de almacenamiento
- Gran escalabilidad, se ajusta al número de CPU y memoria disponible de forma óptima, soportando gran cantidad de peticiones simultáneas

Desventajas:

- Alto consumo de recursos Algunos comando o sentencias pueden ser poco intuitivas
- No tiene soporte en línea.
- Tiene foros oficiales y una gran comunidad que responden a las dudas.

Administrar usuarios y bases de datos

Algunas características de PostgreSQL

- Todas las instrucciones terminan en ;
- No se distingue entre mayúscula y minúsculas

Algunos comandos:

- `ALIAS`
- `AND`
- `AS`
- `CREATE`
- `CREATEDB`
- `CREATEUSER`
- `DATABASE`
- `FROM`
- `INNER`
- `JOIN`
- `LARGE`
- `PASSWORD`
- `WHERE`

Administración de usuario en PostgreSQL

- Crear usuarios
- Eliminar usuarios
- Permisos para los usuarios

Comandos para la creación de usuarios

CREATE USER nombre_usuario WITH comando_opcional;

- PASSWORD
- ENCRYPTED PASSWORD
- UNENCRYPTED PASSWORD
- VALID UNTIL
- CREATEDB
- NOCREATEDB
- SUPERUSER
- NOSUPERUSER

Administración de Base de datos

- Crear base de datos
- Eliminar base de datos

Operaciones comunes a nivel de consola

Comando	Acción
\c nombre_base	Conectarse a una base de datos especifica
\l	Listar todas las bases de datos existentes
\du	Listar todos los usuarios en el motor
\d	Listar todas las relaciones (o tablas) existentes en una base de datos especifica
\q	Salir de la consola de PostgreSQL
\h	muestra la lista de comandos

Elementos de una base de datos

Ejemplo directorio telefónico

- nombre
- apellido
- numero_telefonico
- dirección
- edad

Tablas

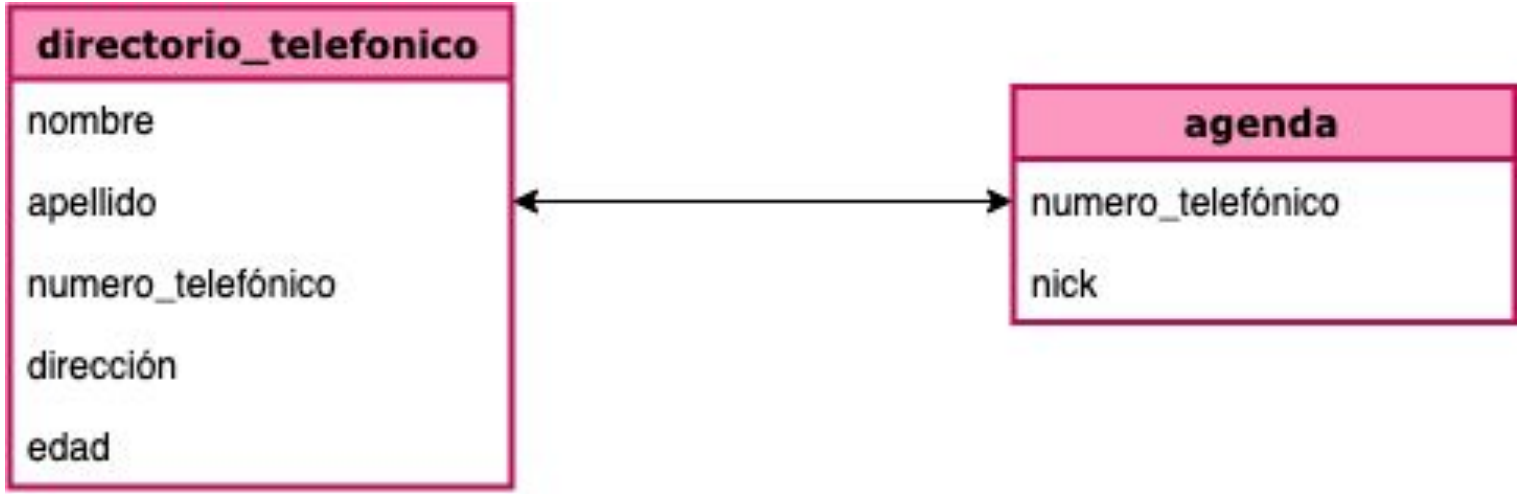
Cada tabla tiene 2 dimensiones:

- Filas, que representan a los registros en la tabla;
- Columnas, que van a representar los atributos ingresados en cada registro, definiendo el tipo de dato a ingresar.

Claves primarias y foráneas

- Clave primaria
- Clave foránea

Ejemplo directorio telefónico



Tipos de datos

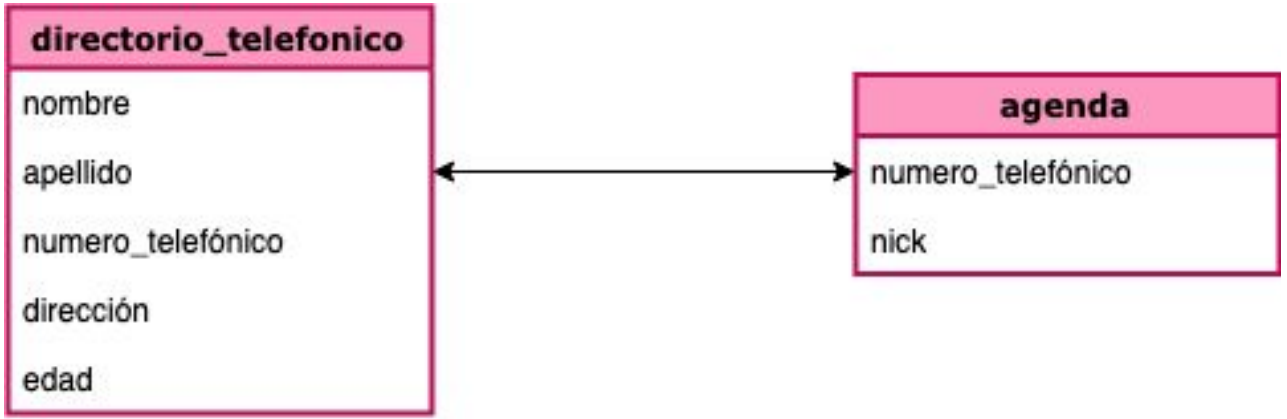
- INT
- SMALLINT
- BIGINT
- FLOAT
- DOUBLE
- CHAR
- VARCHAR
- DATE
- TIME
- TIMESTAMP
- BOOLEAN

Instrucciones de creación, inserción, actualización y eliminación de datos

Creación de tablas

```
CREATE TABLE nombre_tabla(
columna1 tipo_de_dato1;
columna2 tipo_de_dato2;
columna3 tipo_de_dato3;
)
```

Ejemplo directorio telefónico



Creando una tabla y su clave primaria

```
-- Creamos una tabla con el nombre directorio_telefonico
CREATE TABLE Directorio_telefonico(
-- Definimos el campo nombre con el tipo de dato cadena con un largo de 25
caracteres.
nombre VARCHAR(25),
-- Definimos el campo apellido con el tipo de dato cadena con un largo de 25
caracteres.
apellido VARCHAR(25),
-- Definimos el campo numeroTelefonico con el tipo de dato cadena con un largo
de 25 caracteres.
numero_telefonico VARCHAR(8),
-- Definimos el campo dirección con el tipo de dato cadena con un largo de 25
caracteres.
direccion VARCHAR(255),
-- Definimos el campo edad con el tipo de dato entero
edad INT,
-- Definimos que el campo numeroTelefonico representará la clave primaria de la
tabla.
PRIMARY KEY (numero_telefonico)
);
```

Creando una tabla con clave foráneas

```
-- Creamos una tabla con el nombre agenda
CREATE TABLE Agenda(
-- Denfinimos el campo nick con el tipo de dato cadena con un largo de 25
caracteres
nick VARCHAR(25),
-- Definimos el campo numero_telefonico con el tipo de dato cadena con un largo
de 8 caracteres.
numero_telefonico VARCHAR(8),
-- Vinculamos una clave foránea entre nuestra columna numeroTelefonico y su
simil en la tabla directorio_telefonico
FOREIGN KEY (numero_telefonico) REFERENCES
Directorio_telefonico(numero_telefonico)
);
```

Inserción de datos en una tabla

```
INSERT INTO nombre_tabla (columna1, columna2, columna3) VALUES (valor1,
valor2, valor3);
```

Actualización de registros

```
UPDATE nombre_tabla SET columna1=valor_nuevo WHERE condicion;
```

```
UPDATE Directorio_telefonico
SET direccion='Villa Los Leones'
WHERE nombre='Juan';
```

Eliminación de registros

Eliminando todos los registros de una tabla

```
DELETE FROM tabla;
```

Eliminando los registros que cumplen una condición

```
DELETE FROM tabla WHERE condicion;
```

Añadiendo o eliminado columnas

```
ALTER TABLE nombre_tabla
ADD nueva_columna tipo_de_dato;
```

```
ALTER TABLE nombre_tabla
DROP nueva_columna tipo_de_dato;
```

Restricciones

- NOT NULL
- UNIQUE
- SERIAL
- PRIMARY KEY
- FOREING KEY
- CHECK
- DEFAULT
- INDEX

Restricciones

```
-- Creamos una tabla
CREATE TABLE nombre_tabla(
-- Declaramos una serie de restricciones a cada campo de dato creado
columna1 tipo_de_dato1 restriccion,
columna2 tipo_de_dato2 restriccion,
columna3 tipo_de_dato3 restriccion
);
```

Restricciones a nivel de PRIMARY KEY y FOREIGN KEY

```
CREATE TABLE tabla1(
columna1 tipo_de_dato1,
columna2 tipo_de_dato2,
columna3 tipo_de_dato3,
PRIMARY KEY (columna1)
);

CREATE TABLE tabla2(
columna4 tipo_de_dato4,
columna5 tipo_de_dato5,
FOREIGN KEY (columna4) REFERENCES tabla1(columna1)
);
```

Cargar consultas desde un fichero

Creamos un fichero con extensión .sql

Usamos el comando:

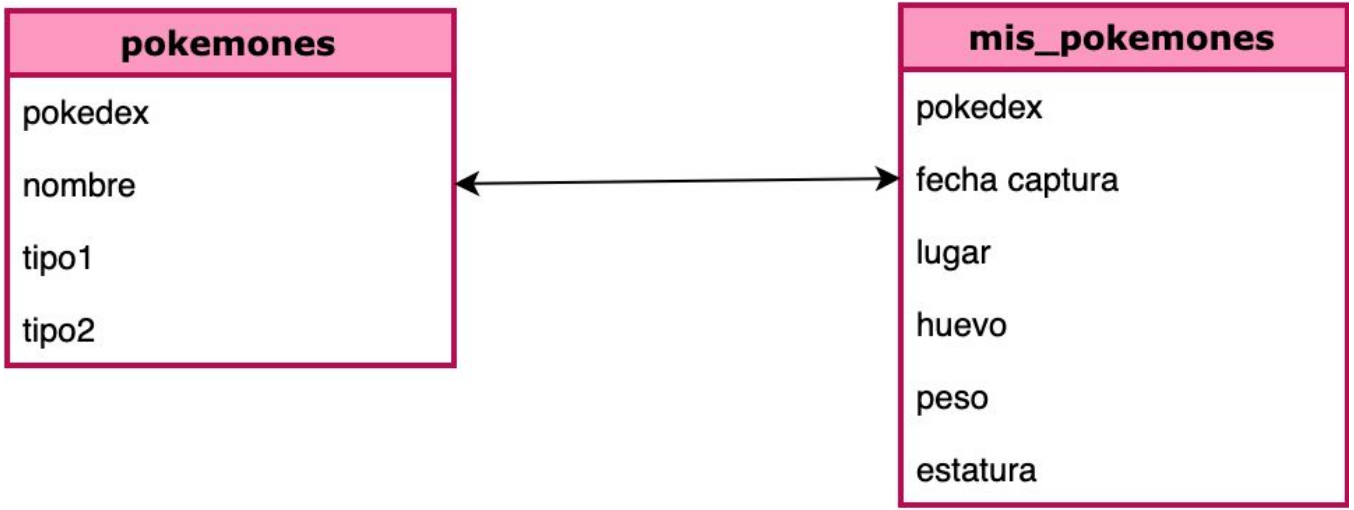
```
\i ubicación\nombre_fichero.sql
```

Realizando consultas

Importando datos de un .csv

```
\copy nombre_tabla FROM 'directorio/archivo.csv' csv [header];
```

Ejercicio Pokemones



Usando SELECT

Consultar por una columna

```
SELECT columna FROM nombre_tabla;
```

Consultar por más de una columna

```
SELECT columna1, columna2 FROM nombre_tabla;
```

Consultar por todas las columnas de una tabla

```
SELECT * FROM nombre_tabla;
```

Operaciones de unión entre Tablas

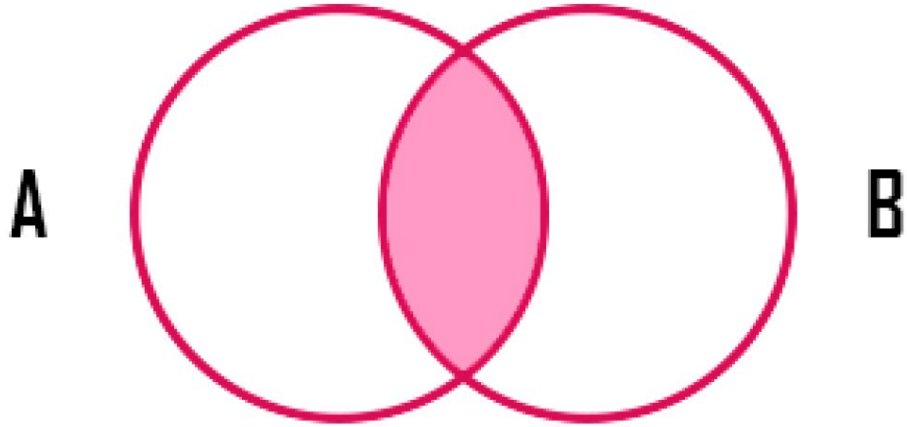
Caso hipotético

```
-- Seleccionamos las columnas desde la tabla1
SELECT columnas FROM tabla1
-- Posterior a la selección de la columna, indicamos que vamos a generar la unión
-- con la columna de la tabla2
JOIN tabla2 ON tabla1.columna=tabla2.columna
[WHERE condicion];
```

Tipos de unión

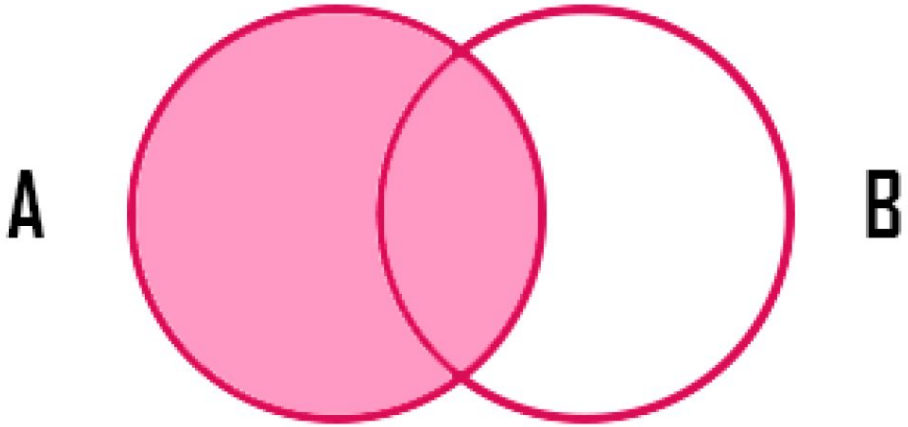
- INNER JOIN
- LEFT JOIN
- FULL OUTER JOIN

INNER JOIN



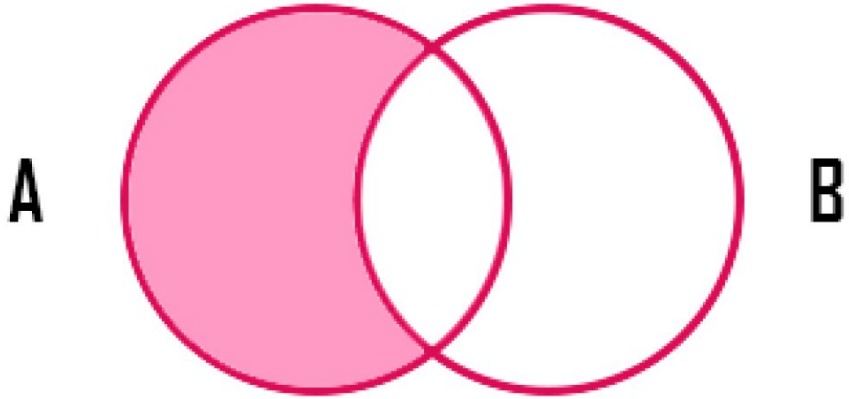

```
SELECT columnas
FROM A
INNER JOIN B
ON A.columna=B.columna
```

LEFT JOIN



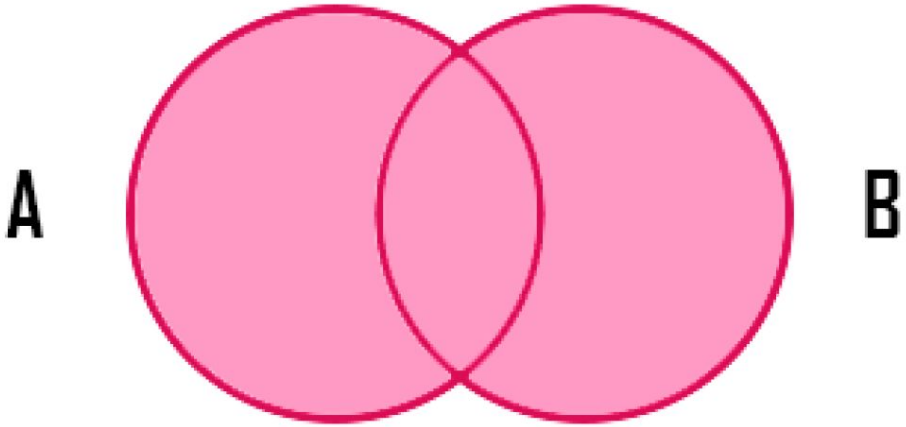

```
SELECT columnas
FROM A
LEFT JOIN B
ON A.columna=B.columna
```

LEFT JOIN



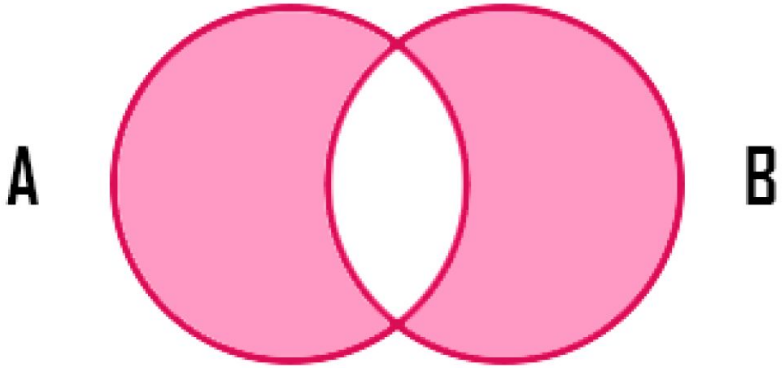

```
SELECT columnas
FROM A
LEFT JOIN B
ON A.columna=B.columna
WHERE B.columna IS NULL
```

FULL JOIN




```
SELECT columnas
FROM A
FULL OUTER JOIN B
ON A.columna=B.columna
```

FULL JOIN




```
SELECT columnas
FROM A
FULL OUTER JOIN B
ON A.columna=B.columna
WHERE A.columna IS NULL
OR B.columna IS NULL
```

Subquey - Operador WHERE

```
SELECT columna1,columna2,...
FROM nombre_tabla
WHERE columna1 IN
    (SELECT columna1
    FROM nombre_tabla2
    WHERE condicion);
```

Subquery - Operador FROM

```
SELECT x.columa1, xcoluma2, ...
FROM (
    SELECT columa1, columna2, ...
    FROM nombre_tabla2
    WHERE condicion
  ) as x
INNER JOIN nombre_tabla1 as y on x.columna1 = y.columna1;
```

Algunas reglas que deben seguir las subqueries

- Las consultas internas deben estar encapsuladas entre paréntesis.
- Una subquery puede tener sólo una columna especificada en SELECT , con la excepción de múltiples columnas definidas en la consulta principal.
- El comando ORDER BY no se puede utilizar en una consulta interna. La excepción es que esta instrucción si puede ser incluída en la consulta principal.
- Para obtener un resultado similar a ORDER BY dentro de una consulta interna, se puede implementar el comando GROUP BY .
- Aquellas consultas internas que retornen más de una fila sólo pueden ser utilizadas con operadores de múltiples valores como IN .

Transacciones

Transacciones

- Atomicidad
- Consistencia
- Aislamiento
- Durabilidad

Comandos para las transacciones

BEGIN: El sistema permite que se ejecuten todas las sentencias SQL que necesitemos.

COMMIT: Guarda los cambios de la transacción.

ROLLBACK: Retrocede los cambios realizados.

SAVEPOINT: Guarda el punto de partida al cual volver a la hora de aplicar ROLLBACK

SET TRANSACTION:: Le asigna nombre a la transacción.

Cargar una base de dato utilizando dump

Dump es una herramienta que nos permite de manera simple generar copias de nuestra base de datos, para así respaldarla o bien cargar datos de este respaldo en un momento determinado.

Dump para una base de datos

Para crear un archivo a partir de una base de datos, deberemos ejecutar el siguiente comando desde la terminal.

```
pg_dump -U nombre_usuario nombre_db > db.out
```

Dump para todas las bases de datos

Si queremos obtener el respaldo de todas las bases de datos debemos ejecutar el siguiente comando

```
$ sudo su - postgres
$ pg_dumpall > /directorio/dumpall.sql
```

Restaurar una base de datos

Para restaurar una base de datos, debemos utilizar el siguiente comando

```
psql -U postgres nombredb < archivo_restauracion.sql
```

Restaurar todas las bases de datos

Para restaurar una base de datos, debemos utilizar el siguiente comando

```
$ sudo su - postgres
$ psql -f /var/lib/pgsql/backups/dumpall.sql mydb
```

{desafío} latam_

Academia de talentos digitales

www.desafiolatam.com