Introducción a bases de datos

Objetivos

- [ ] Entender el por qué necesitamos bases de datos
- [ ] PostgreSQL
- [ ] Que son las tablas
- [ ] Tipos de datos
- [ ] Preparando nuestra primera consulta
- [ ] Realizar operaciones en una tabla
- [ ] Crear una base de datos
- [ ] Operaciones en una tabla

¿Por qué necesitamos bases de datos?

Las bases de datos son componentes fundamentales en la era de la información y desempeñan un papel crucial en una amplia variedad de aplicaciones en la vida cotidiana y en el mundo empresarial. Son sistemas organizados para recopilar, almacenar y gestionar datos de manera eficiente, lo que proporciona una serie de beneficios esenciales.

Algunas características de las bases de datos son:

Almacenamiento eficiente de datos: Las bases de datos permiten almacenar grandes cantidades de información de manera estructurada. En lugar de utilizar archivos separados para cada pieza de información, las bases de datos organizan los datos en tablas y relaciones, lo que facilita la gestión y recuperación de la información.

Acceso rápido y fácil: Con una base de datos, es posible recuperar datos específicos de manera rápida y precisa. Los motores de búsqueda y las consultas permiten encontrar información en cuestión de segundos, en lugar de buscar manualmente en montañas de documentos o archivos.

Consistencia de datos: Las bases de datos garantizan la integridad y la consistencia de los datos. Esto significa que los datos almacenados en la base de datos siguen un conjunto de reglas y restricciones, lo que ayuda a evitar errores y duplicaciones.

Seguridad de datos: Las bases de datos suelen contar con sistemas de seguridad avanzados para proteger la información. Los administradores pueden establecer permisos de acceso y cifrar datos confidenciales, lo que reduce el riesgo de pérdida o acceso no autorizado.

Compartir datos: Las bases de datos permiten compartir datos entre múltiples usuarios o sistemas de manera controlada. Esto es esencial en entornos empresariales donde varias personas necesitan acceder a la misma información.

Escalabilidad: Las bases de datos son escalables, lo que significa que pueden crecer para adaptarse a las necesidades cambiantes de una organización. Se pueden agregar más datos y funciones sin una reestructuración importante.

Análisis y toma de decisiones: Las bases de datos son fundamentales para el análisis de datos y la toma de decisiones informadas. Los datos almacenados pueden utilizarse para generar informes, estadísticas y análisis que ayuden a las organizaciones a comprender mejor su desempeño y planificar su futuro.

Integración con aplicaciones: Las bases de datos se integran con una amplia variedad de aplicaciones, lo que permite a las empresas y desarrolladores construir sistemas más complejos y funcionales.

las bases de datos son una herramienta esencial en el mundo moderno, ya que permiten la organización, gestión y acceso eficiente a grandes cantidades de datos. Esto no solo beneficia a las empresas, sino también a las personas en su vida cotidiana, ya que la mayoría de las aplicaciones y servicios que utilizamos dependen de bases de datos para funcionar de manera efectiva.

Existen distintos tipos de motores de bases de datos pero para este módulo solo usaremos El motor de base de datos relaciones PostgreSQL.

PostgreSQL

Es un sistema de gestión de bases de datos relacional (RDBMS, por sus siglas en inglés) de código abierto y de alto rendimiento. Es ampliamente considerado como uno de los sistemas de gestión de bases de datos más avanzados y poderosos disponibles en la actualidad. A continuación, te presento algunas características:

Base de datos relacional: PostgreSQL se basa en el modelo de base de datos relacional, lo que significa que organiza los datos en tablas con filas y columnas, permitiendo relaciones entre diferentes conjuntos de datos. Esto facilita la gestión y consulta de datos estructurados.

Código abierto: PostgreSQL es un software de código abierto, lo que significa que su código fuente está disponible públicamente para que cualquiera lo use, modifique y distribuya de acuerdo con los términos de la Licencia PostgreSQL (también conocida como la Licencia PostgreSQL, similar a la Licencia MIT). Esto lo hace accesible y personalizable para una amplia gama de aplicaciones y proyectos.

Alto rendimiento y escalabilidad: PostgreSQL está diseñado para ofrecer un alto rendimiento y es capaz de manejar grandes conjuntos de datos y cargas de trabajo intensivas. Además, es altamente escalable, lo que permite que crezca con las necesidades de una organización sin perder eficiencia.

Soporte para lenguaje SQL avanzado: PostgreSQL es compatible con SQL (Structured Query Language) y ofrece un amplio conjunto de características SQL, incluyendo soporte para consultas complejas, agregaciones, funciones y procedimientos almacenados.

Extensibilidad: Una de las características distintivas de PostgreSQL es su capacidad para extender sus capacidades a través de funciones definidas por el usuario, tipos de datos personalizados y lenguajes de programación, lo que permite a los desarrolladores personalizar la base de datos según sus necesidades específicas.

Características avanzadas: PostgreSQL incluye una amplia variedad de características avanzadas, como soporte para transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), índices avanzados, replicación de datos, particionamiento de tablas y una sólida seguridad de datos.

Plataformas y sistemas operativos compatibles: PostgreSQL es altamente portable y se ejecuta en una amplia variedad de sistemas operativos, incluyendo Linux, Windows, macOS y más.

Nota: Tiene soporte para datos no relacionales.

PostgreSQL es una poderosa y versátil base de datos relacional de código abierto que es adecuada para una amplia gama de aplicaciones, desde pequeños proyectos personales hasta grandes sistemas empresariales. Su robustez, escalabilidad y flexibilidad lo han convertido en una opción popular en la comunidad de desarrolladores y en el mundo empresarial.

¿Qué son las tablas?

En las bases de datos relacionales, una tabla es una estructura fundamental para organizar y almacenar datos. Las tablas son una de las formas más comunes de representar datos en un sistema de gestión de bases de datos relacional, como PostgreSQL, MySQL o Microsoft SQL Server.

Las tablas se componen de lo siguiente:

Estructura de tabla: Una tabla se compone de filas y columnas. Cada fila de la tabla representa una entidad o un registro individual, y cada columna define un atributo específico de esa entidad. Por ejemplo, si estás creando una tabla para almacenar

información de clientes, las filas representarían a clientes individuales, y las columnas podrían incluir campos como nombre, dirección, número de teléfono, etc.

Nombre y esquema: Cada tabla tiene un nombre único que la identifica dentro de la base de datos. Además, las tablas se organizan en un esquema, que es un espacio lógico que agrupa y organiza tablas relacionadas en la base de datos.

Datos estructurados: Las tablas se utilizan para almacenar datos estructurados, lo que significa que los datos en cada columna siguen un tipo de dato específico, como texto, números, fechas, etc. Esto permite un almacenamiento eficiente y una gestión más fácil de los datos.

Claves primarias: Las tablas suelen tener una columna o conjunto de columnas llamado "clave primaria" que tiene valores únicos para cada fila. La clave primaria se utiliza para identificar de manera única cada registro en la tabla y es esencial para establecer relaciones entre tablas.

Relaciones: Las bases de datos relacionales permiten establecer relaciones entre diferentes tablas. Esto significa que puedes conectar la información en una tabla con la de otra a través de las claves primarias y extranjeras. Por ejemplo, en una base de datos de ventas, podrías relacionar la tabla de clientes con la de pedidos mediante la clave del cliente.

Consultas y manipulación de datos: Las tablas son el lugar donde se realizan operaciones de consulta y manipulación de datos. Puedes insertar nuevos registros, actualizar información existente, eliminar registros y consultar datos utilizando instrucciones SQL (Structured Query Language).

Índices: Para acelerar la búsqueda de datos en una tabla, se pueden crear índices en columnas específicas. Los índices actúan como listas de referencia rápida que agilizan la búsqueda de datos.

En resumen, una tabla en una base de datos relacional es una estructura organizada en filas y columnas que se utiliza para almacenar datos estructurados. Estas tablas son esenciales para la organización, gestión y recuperación eficiente de datos en una base de datos y son la base de muchas aplicaciones y sistemas de información en el mundo de la informática.

Tipos de datos

Los tipos de datos son una categorización de los valores que una variable o campo puede contener en un programa de computadora o en una base de datos. Estos tipos definen el formato y las operaciones que se pueden realizar con los datos. Los tipos de datos son esenciales en la programación y la gestión de bases de datos, ya que ayudan a garantizar que los datos se almacenen, manipulen y presenten de manera adecuada. Aquí hay algunos ejemplos de tipos de datos comunes:

Enteros (Integer Types):

- **smallint**: Almacena números enteros pequeños (2 bytes).
- **integer**: Almacena números enteros más grandes (4 bytes).
- **bigint**: Almacena números enteros muy grandes (8 bytes).

Números de punto flotante (Floating-Point Types):

- **real**: Almacena números de punto flotante de precisión simple.
- **double precision**: Almacena números de punto flotante de doble precisión.

Caracteres y Cadenas de Texto (Character and Text Types):

- **character(n)**: Almacena una cadena de caracteres de longitud fija.
- **character varying(n):** Almacena una cadena de caracteres de longitud variable.
- **text**: Almacena texto de longitud variable sin límite definido.

Fecha y Hora (Date and Time Types):

- **date**: Almacena fechas (año, mes, día).
- **time**: Almacena horas del día.
- **timestamp**: Almacena fechas y horas.
- **interval**: Almacena un intervalo de tiempo.

Booleano (Boolean Type):

- **boolean**: Almacena valores de verdad (true o false).

Enumerados (Enumerated Types):

- **enum**: Almacena un conjunto predefinido de valores enumerados.

Arreglos (Array Types):

- **integer[], text[],** u otros tipos de datos, para almacenar arreglos de valores del mismo tipo.

JSON y JSONB (JSON Types):

- **json**: Almacena datos en formato JSON.
- **jsonb**: Almacena datos en formato binario JSON, que ofrece un rendimiento superior.

SERIAL:

- el tipo de dato SERIAL se utiliza para crear columnas que generan automáticamente valores enteros únicos cuando se inserta un nuevo registro en una tabla.
- Cuando defines una columna como tipo SERIAL, PostgreSQL se encarga de la generación automática de valores enteros únicos y crecientes. Esto es especialmente útil para crear claves primarias, ya que garantiza que cada fila en una tabla tenga un valor único en esa columna, y que los valores sean secuenciales y crecientes.

UUID:

- el tipo de dato UUID (Universally Unique Identifier). El tipo UUID es utilizado para almacenar identificadores únicos globales en una base de datos. Un UUID es una cadena de 36 caracteres que representa un valor único a nivel mundial y se genera de manera aleatoria.
- PostgreSQL admite varias funciones para trabajar con UUIDs, como **uuid-ossp**, que te permite generar UUIDs aleatorios o basados en el tiempo.
- PostgreSQL proporciona una función llamada uuid-ossp que te permite generar UUIDs automáticamente en una columna.

Estos son solo algunos ejemplos de los tipos de datos que PostgreSQL admite.

Preparando nuestra primera consulta

La instrucción SELECT

La instrucción SELECT es una de las instrucciones fundamentales y se utiliza para recuperar datos de una base de datos. Es una de las operaciones más comunes que se realizan en una base de datos y permite a los usuarios obtener información específica de una o varias tablas.

La sintaxis básica de la instrucción SELECT es la siguiente:

```
SELECT * FROM demo;
```

SELECT	Indica que la consulta a realizar será de selección
*	El asterisco es un comodín para indicar que se deben seleccionar todos los campos, o sea, todas las columnas de la tabla.
FROM	Indica de qué tabla específica se va a seleccionar
demo	Nombre de la tabla. En este caso esta viene precargada en sqliteonline
;	Una consulta termina con un punto y coma, de esta forma podemos separar varias instrucciones.

También podemos seleccionar algunas columnas de la tabla y solo traer la información necesaria, por ejemplo, tenemos una tabla que almacena datos de productos, pero solo queremos obtener mediante un select el nombre de los productos y el precio.

Nuestra consulta quedaría de la siguiente manera:

La cláusula WHERE

La cláusula WHERE es una cláusula fundamental en se utiliza en combinación con la instrucción SELECT (y otras instrucciones como UPDATE, DELETE, y INSERT) para filtrar los resultados de una consulta. La cláusula WHERE permite especificar una condición que debe cumplirse para que una fila sea incluida en el resultado de la consulta.

Sintaxis de una instrucción SELECT usando una condición.

```
SELECT columnas FROM tabla WHERE condicion;
```

En el caso anterior de la tabla productos podríamos usar la instrucción WHERE para obtener todos los productos que tengan el precio mayor a 10.000.

Nuestra consulta se vería así:

```
SELECT * FROM products WHERE price > 10000;
```

La cláusula ORDER BY

En muchos escenarios, los datos recuperados de una base de datos no se presentan en el orden deseado. Es aquí donde entra en juego la ordenación. La ordenación te permite organizar los resultados de una consulta de acuerdo con uno o más criterios, lo que facilita la interpretación y el análisis de los datos. La cláusula ORDER BY se utiliza para especificar la columna o las columnas por las cuales deseas ordenar los resultados, ya sea de forma ascendente o descendente.

Sintaxis de una instrucción SELECT usando WHERE y ORDER BY

```
SELECT columnas FROM tabla ORDER BY columna [ASC|DESC];
```

[ASC|DESC]: Opcionalmente, puedes especificar ASC (ascendente) para ordenar de menor a mayor o DESC (descendente) para ordenar de mayor a menor. De forma predeterminada, se utiliza el orden ascendente.

Puedes ordenar por una sola columna o por múltiples columnas, lo que te permite controlar la secuencia en la que se presentan los datos según tus necesidades.

Si seguimos con el mismo caso de la tabla productos, podemos ordenar los registros de varias formas, en el siguiente ejemplo vamos a ordenar los productos según su fecha de creación, mostrando los resultados de manera descendente, además seguiremos el ejemplo anterior y usaremos la cláusula WHERE para dar una condición.

```
SELECT * FROM products WHERE price > 10000 ORDER BY created_at DESC;
```

Si quieres ordenar por más de una columna puedes seguir la siguiente sintaxis;

```
SELECT * FROM tabla WHERE condicion ORDER BY columna_1, columna_2 DESC;
```

Por último si quieres ordenar datos de manera descendente y ascendente puedes usar la siguiente sintaxis;

```
SELECT * FROM tabla WHERE condicion ORDER BY columna_1 DESC columna_2;
```

La cláusula LIMIT

En situaciones en las que una consulta devuelve un gran número de registros y solo estás interesado en un subconjunto de ellos, la cláusula LIMIT te permite especificar cuántos registros deseas ver. Esto es especialmente útil para:

Paginación: Cuando se trata de mostrar resultados en páginas web, aplicaciones móviles u otros entornos, LIMIT se utiliza para dividir los resultados en páginas más pequeñas. Por ejemplo, puedes mostrar 10 resultados por página y luego permitir al usuario navegar a la siguiente página.

Optimización de consultas: Al limitar el número de filas devueltas, puedes mejorar el rendimiento de tu consulta, ya que procesar y transmitir grandes cantidades de datos puede ser costoso en términos de tiempo y recursos.

La sintaxis básica de la cláusula LIMIT es la siguiente:

```
SELECT * FROM tabla LIMIT numero de registros a mostrar;
```

Volviendo a nuestro caso anterior una consulta aplicando LIMIT a los productos quedaría de la siguiente forma:

```
SELECT * FROM products WHERE price > 10000 ORDER BY created_at DESC LIMIT 10;
```

Insertando datos

La inserción de datos es el proceso de agregar nuevos registros o filas a una tabla en una base de datos. Estos registros generalmente consisten en información que deseas almacenar y mantener para su posterior recuperación y uso. La operación de inserción es utilizada en una variedad de aplicaciones y escenarios, desde el registro de nuevos clientes en una base de datos de una empresa hasta la adición de nuevos productos a un catálogo en línea.

Para lograr la inserción usamos la instrucción INSERT INTO Esta instrucción es fundamental para la operación de inserción de datos y permite que los datos se almacenen de manera persistente en una base de datos relacional.

Sintaxis para la inserción de registros

```
INSERT INTO nombre_de_tabla (columna1, columna2, ...) VALUES (valor1, valor2, ...);
```

(columna1, columna2, ...): Aquí se enumeran las columnas de la tabla en las que deseas insertar valores. Puedes especificar las columnas en el mismo orden en que proporcionarás los valores, o puedes enumerar solo las columnas en las que deseas insertar datos.

VALUES (valor1, valor2, ...): Esta parte de la instrucción contiene los valores que deseas insertar en las columnas correspondientes. Debes asegurarte de que los valores estén en el mismo orden que las columnas especificadas.

Insertemos un registro nuevo en la tabla productos, supongamos que la tabla productos tiene las columnas name, price, created_at, description, category nuestra inserción de registros se vería de esta forma:

```
INSERT INTO products (name, price, created_at, description, category) VALUES ('lavadora', 100000, '2023/10/28', 'lavadora de carga frontal', 'linea blanca');
```

Nota:
los strings se ingresan con comillas simples, utilizar comillas dobles nos mostrara un error.

Si solo agregamos datos en algunas columnas el resto quedara con un valor de null (En el contexto de las bases de datos, un valor null se entiende como indefinido o desconocido.)

No es lo mismo null que ‘ ’ (un valor vacío se genera cuando uno de los campos de la tabla no contiene ningún carácter.)

Actualizando registros

La actualización de registros es una operación que te permite modificar uno o varios campos de una o varias filas en una tabla de PostgreSQL. Puedes utilizar esta operación para realizar cambios en los datos existentes sin necesidad de eliminar o reinsertar toda la fila. Es especialmente útil cuando necesitas corregir errores tipográficos, actualizar información obsoleta, cambiar el estado de un registro o realizar otras modificaciones en la base de datos.

Para hacer la actualización de un registro usamos la instrucción UPDATE

La sintaxis básica de la instrucción UPDATE en PostgreSQL es la siguiente:

```
UPDATE nombre_de_tabla SET columna1 = nuevo_valor1, columna2 = nuevo_valor2, ... WHERE condición;
```

SET columna1 = nuevo_valor1, columna2 = nuevo_valor2, ...: En esta parte de la instrucción, enumeras las columnas que deseas actualizar y les asignas los nuevos valores. Puedes actualizar una o varias columnas en la misma instrucción.

WHERE condición: La cláusula WHERE es opcional, pero es recomendable para especificar qué registros deseas actualizar. Sin una condición, la instrucción actualizará todos los registros de la tabla.

En el caso de nuestra tabla de productos podríamos actualizar el ultimo registro insertado cambiando su precio.

Nuestra actualización quedaría así

```
UPDATE products SET price = 200000 where id = 1;
```

Eliminando registros

La eliminación de registros es una operación que te permite borrar filas completas de una tabla en una base de datos. Es útil cuando necesitas eliminar datos que ya no son relevantes, que están duplicados, que son incorrectos o que violan restricciones de integridad. La eliminación de registros es esencial para mantener la base de datos actualizada y libre de información no deseada.

Para realizar la eliminación de registros usamos la instrucción DELETE

La sintaxis básica de la instrucción DELETE es la siguiente:

```
DELETE FROM nombre_de_tabla WHERE condición;
```

WARNING: **WHERE condición** Esta parte es opcional, pero se recomienda utilizarla para especificar qué registros deseas eliminar. **Sin una condición, la instrucción eliminará todos los registros de la tabla.**

Para finalizar con nuestro ejemplo eliminaremos el registro de la tabla productos

```
DELETE FROM products WHERE id = 1;
```

Creando nuestra primera base de datos

La creación de bases de datos es una etapa crítica en el desarrollo de sistemas de información. Implica definir la estructura, el esquema y las relaciones de datos que se almacenan en la base de datos. Esto incluye la creación de tablas para almacenar datos, la especificación de tipos de datos para cada columna, la definición de claves primarias y secundarias, y la creación de índices para optimizar la recuperación de datos.

Pasos típicos en la creación de bases de datos:

Diseño del modelo de datos: El primer paso es diseñar un modelo de datos que represente la estructura de la información que deseas almacenar. Esto incluye la identificación de las entidades, atributos y relaciones clave.

Elección del sistema de gestión de bases de datos (DBMS): Debes seleccionar el DBMS que mejor se adapte a tus necesidades. Ejemplos comunes incluyen PostgreSQL, MySQL, SQL Server, Oracle, entre otros.

Creación de la base de datos: Utilizando el DBMS seleccionado, creas una nueva base de datos vacía.

Definición de tablas: Creas las tablas que representarán las entidades en tu modelo de datos. Para cada tabla, especificas las columnas y los tipos de datos, así como las restricciones de clave primaria y secundaria.

Establecimiento de relaciones: Si tu modelo de datos incluye relaciones entre las tablas, defines estas relaciones, como las claves foráneas que conectan las tablas.

Creación de índices: Puedes crear índices en las columnas que se utilizan con frecuencia en las consultas para acelerar la recuperación de datos.

Inserción de datos iniciales: Insertas datos iniciales en las tablas si es necesario.

Configuración de la seguridad y los permisos: Definir quién tiene acceso a la base de datos y qué acciones pueden realizar es fundamental para proteger la información.
Para crear nuestra primera base de datos utilizaremos la instrucción CREATE DATABASE;

Sintaxis para la creación de una base de datos.

```
# CREATE DATABASE nombre_de_base_de_datos;
```

Como ejercicio crearemos una base de datos para la tienda Blockbuster

```
CREATE DATABASE blockbuster;
```

Para poder realizar operaciones en esta base de datos debemos seleccionarla usando el comando o instrucción \c más el nombre de la base de datos.

```
\c blockbuster;
```

Esto nos cambiara de la base de datos de postgres a nuestra nueva base de datos, ahora que estamos en ella comenzaremos creando nuestra primera tabla.

Creando nuestra primera tabla

La creación de tablas es un paso esencial en la gestión de bases de datos relacionales. Cada tabla en una base de datos representa una entidad o un conjunto de datos relacionados y está compuesta por filas y columnas. Cada columna define un atributo o campo de datos, y cada fila contiene un registro o instancia de datos.

Para crear una tabla en PostgreSQL, puedes utilizar la sentencia CREATE TABLE.

Sintaxis para la creación de una tabla.

```
CREATE TABLE nombre_de_tabla (
    nombre_columna1 tipo_de_dato1,
    nombre_columna2 tipo_de_dato2,
    ...,
    nombre_columnaN tipo_de_datoN
);
```

Siguiendo con nuestro nuevo caso para blockbuster crearemos la tabla movies la cual tendrá las siguientes columnas o atributos, id de tipo serial (puedes escoger uuid si lo prefieres),

title de tipo varchar, el año de estreno su nombre será premiere y su tipo será date, por ultimo un atributo description que sera de tipo varchar con una longitud de 140

Nuestra instrucción debería quedar así:

```
CREATE TABLE movies (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    premiere DATE NOT NULL,
    description TEXT NOT NULL,
);
```

Agregando columnas a una tabla

Agregar columnas en una tabla es una operación que te permite extender la estructura de la tabla existente para acomodar nuevos atributos o información sin tener que recrear la tabla desde cero. Esto es especialmente útil cuando necesitas realizar cambios en la base de datos sin perder los datos previamente almacenados.

La sintaxis para agregar columnas es la siguiente:

```
ALTER TABLE nombre_de_tabla ADD COLUMN nombre_columna tipo_de_dato [restricciones];
```

Para nuestra tabla movies podríamos agregar una columna adicional llamada director para almacenar quien dirigió la película nuestra instrucción quedaría así:

```
ALTER TABLE movies ADD COLUMN director VARCHAR(255) NOT NULL;
```

La instrucción DROP

La instrucción DROP se utiliza para eliminar objetos de la base de datos, lo que puede ser útil en situaciones como:

1. Eliminar tablas que ya no se necesitan.
2. Borrar bases de datos obsoletas.
3. Eliminar índices para optimizar el rendimiento de la base de datos.
4. Eliminar restricciones que ya no son necesarias.

El comando DROP se ejecuta mediante sentencias específicas, como DROP TABLE, DROP DATABASE, DROP INDEX entre otros, según el tipo de objeto que desees eliminar.

Precauciones al utilizar DROP:

Es importante recordar que el uso incorrecto o descuidado del comando DROP puede provocar la pérdida irreversible de datos. Aquí hay algunas precauciones importantes a tener en cuenta:

Verifica qué estás eliminando: Asegúrate de que estás eliminando el objeto correcto y que no haya datos importantes en él.

Realiza copias de seguridad: Antes de ejecutar un comando DROP, realiza copias de seguridad de los datos para que puedas restaurarlos en caso de que ocurra un error.

Otorga permisos adecuados: Asegúrate de tener los permisos adecuados para realizar la operación de eliminación. No todos los usuarios deben tener acceso para eliminar objetos.

Piensa dos veces antes de ejecutarlo: No uses DROP a menos que estés seguro de que es necesario. Puedes deshacer cambios en la base de datos, pero esto a menudo requiere un esfuerzo considerable y puede causar interrupciones en las aplicaciones que utilizan la base de datos.

DROP es una herramienta poderosa en SQL, pero debe manejarse con responsabilidad y cuidado para evitar la pérdida de datos no deseada.

Ahora que creamos nuestra primera tabla realizaremos unos ejercicios para practicar