

Proyecto de Base de Datos Grupal BD

Integrantes: Kevin Muñoz, José Vargas, Joshua Morocho

Instrucciones Generales

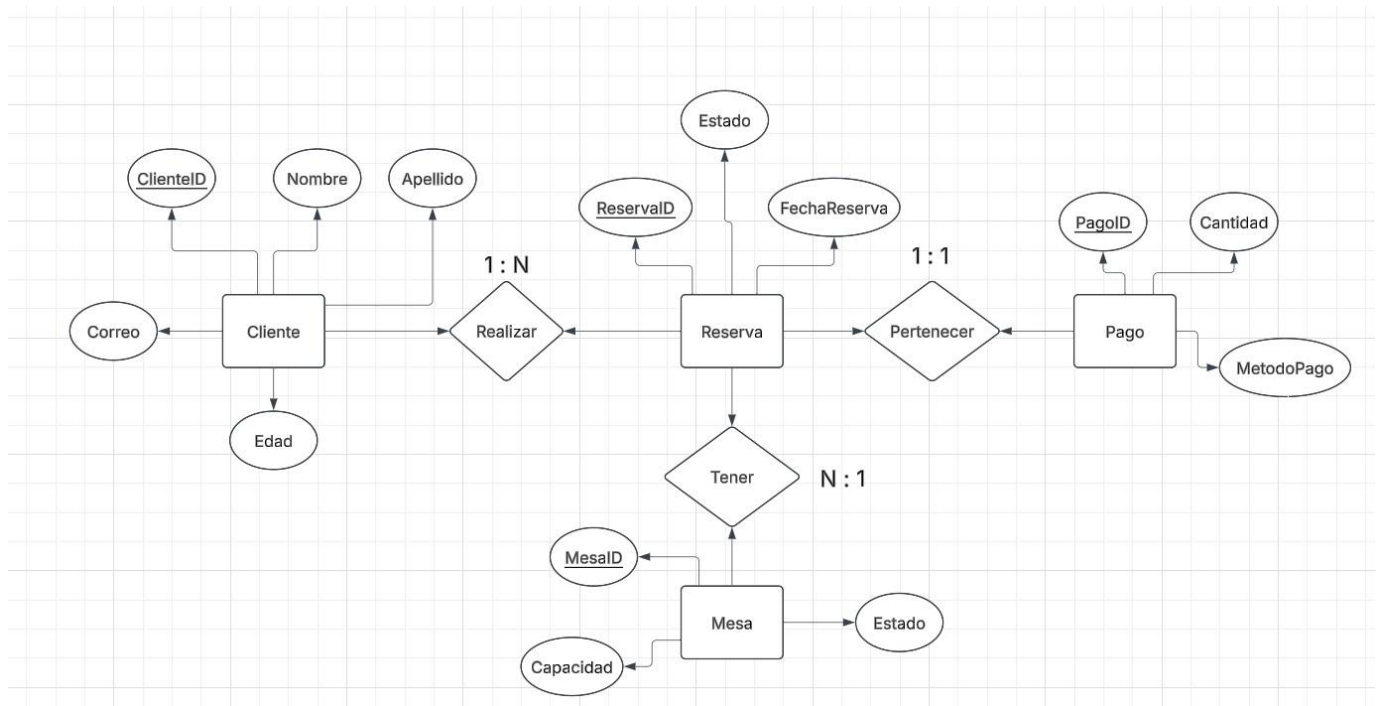
1. Modelado de Base de Datos y Diccionario de Datos

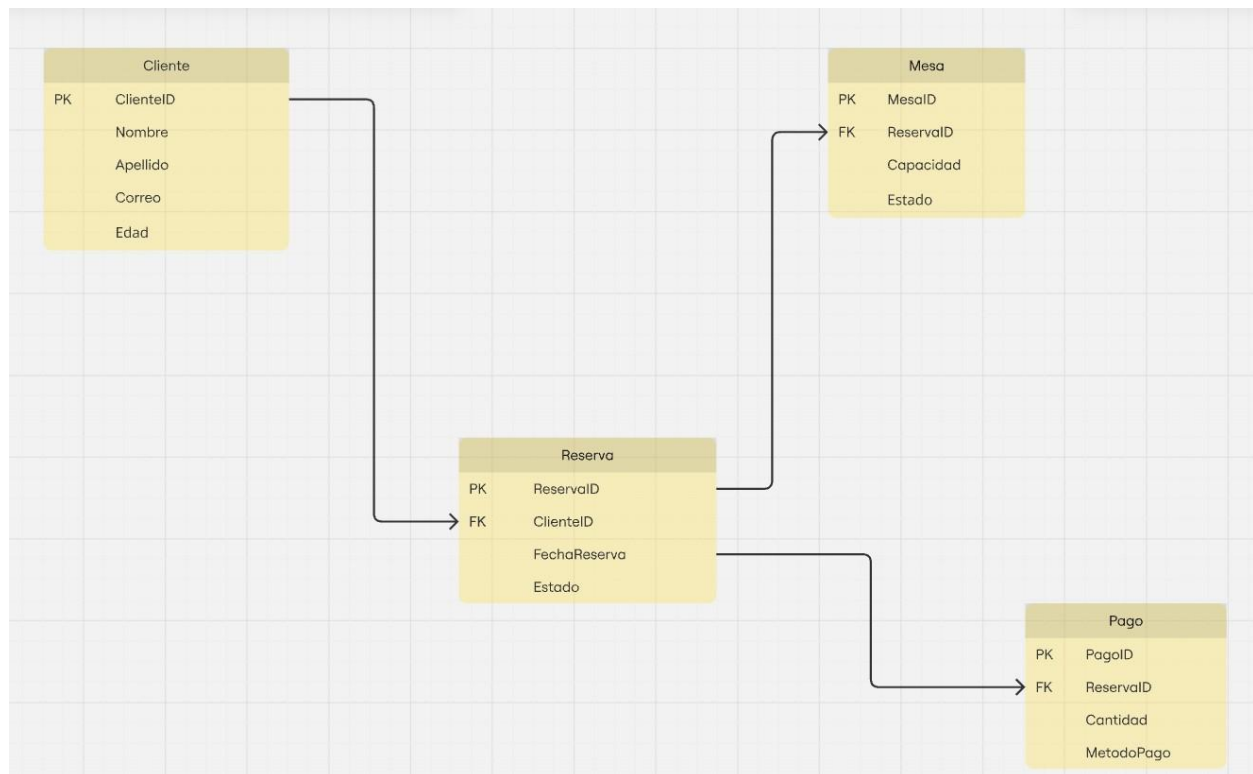
Objetivo: Crear un diseño eficiente y bien documentado para la base de datos, utilizando el modelado ER y un diccionario de datos completo.

Actividades:

Diseñar el modelo conceptual, lógico y físico.

Práctica: Crear un modelo entidad-relación que refleje las entidades clave (como Clientes, Vuelos, Reservas, Pagos) y sus relaciones.





Investigación: Buscar buenas prácticas sobre cómo hacer escalables los modelos de bases de datos para sistemas de reservas en aerolíneas.

Importancia del Conocimiento: Conocer cómo diseñar bases de datos adecuadas asegura la eficiencia y fácil mantenimiento de la aplicación.

1. Desarrollar un diccionario de datos detallado.

Práctica: Crear un diccionario que defina todas las tablas, sus campos, relaciones y restricciones.

Investigación: Investigar las mejores herramientas y métodos para generar diccionarios de datos y su uso en proyectos reales.

- Algunas herramientas son: Dataedo, SolarWinds Database Mapper, ApexSQL Doc, Redgate SQL Doc.
- Para generar diccionarios se puede seguir: Primero, antes de crear un diccionario de datos, es crucial establecer qué datos se documentarán, quiénes serán los usuarios y con qué frecuencia se actualizará. Luego, hay que utilizar una terminología coherente y una estructura organizada para facilitar la comprensión y el uso del diccionario de datos. Y finalmente, es esencial actualizar el diccionario de datos cada vez que haya cambios en la estructura o el uso de los datos para garantizar su precisión y relevancia.

Importancia del Conocimiento: Un diccionario de datos permite mantener la consistencia y facilita la colaboración entre desarrolladores.

Diccionario de Datos de la tabla Cliente

Campo	Tipo Dato	Longitud – Tamaño	Descripción	Restricciones	Ejemplo
ClienteID	INT	-	Identificador único del cliente	PRIMARY KEY, AUTO_INCREMENT	1
Nombre	VARCHAR	100	Nombre del cliente	NOT NULL	“Juan”
Apellido	VARCHAR	100	Apellido del cliente	NOT NULL	“Pérez”
Correo	VARCHAR	100	Correo único del cliente	NOT NULL, UNIQUE	"juan.perez@mail.com"
Edad	INT	1, 2	Edad del cliente en años	NOT NULL	30

Diccionario de Datos de la tabla Reserva

Campo	Tipo Dato	Longitud – Tamaño	Descripción	Restricciones	Ejemplo
ReservaID	INT	-	Identificador único de la reserva	PRIMARY KEY, AUTO_INCREMENT	101
ClienteID	INT	-	Referencia al cliente que realizó la reserva	NOT NULL, FOREIGN KEY a Cliente(ClienteID)	5
FechaReserva	TIMESTAMP	-	Fecha y hora de la reserva	DEFAULT CURRENT_TIMESTAMP	2025-01-31 10:00:00

Estado	ENUM	10	Estado de la reserva	NOT NULL, Valores: "Confirmado", "Cancelado"	"Confirmado"
--------	------	----	----------------------	--	--------------

Diccionario de Datos de la tabla Pago

Campo	Tipo Dato	Longitud – Tamaño	Descripción	Restricciones	Ejemplo
PagoID	INT	-	Identificador único del pago	PRIMARY KEY, AUTO_INCREMENT	500
ReservaID	INT	-	Referencia a la reserva asociada	NOT NULL, FOREIGN KEY a Reserva(ReservaID)	101
Cantidad	DECIMAL	(10, 2)	Monto del pago	NOT NULL	85.50
MetodoPago	ENUM	8	Método de pago	NOT NULL, Valores: "Tarjeta", "Efectivo"	"Tarjeta"

Diccionario de Datos de la tabla Mesa

Campo	Tipo Dato	Longitud – Tamaño	Descripción	Restricciones	Ejemplo
MesaID	INT	-	Identificador único de la mesa	PRIMARY KEY, AUTO_INCREMENT	1000
ReservaID	INT	-	Referencia a la reserva asociada	NOT NULL, FOREIGN KEY a Reserva(ReservaID)	500
Capacidad	INT	(10, 2)	Capacidad de la mesa en personas	NOT NULL	100

Estado	ENUM	7	Estado de la mesa	NOT NULL, Valores: "Libre", "Ocupado"	"Libre"
--------	------	---	-------------------	---	---------

Diccionario de Datos de la tabla CambiosReserva

Campo	Tipo Dato	Longitud – Tamaño	Descripción	Restricciones	Ejemplo
LogID	INT	-	Identificador único del cambio	PRIMARY KEY, AUTO_INCREMENT	200
ReservaID	INT	-	Referencia a la reserva asociada	-	102
ClienteID	INT	-	Referencia al cliente asociado	-	100
FechaReserva	TIMESTAMP	-	Fecha y hora de la reserva original Estado de la mesa	-	2025-01-31 10:00:00
EstadoAnterior	ENUM	10	Estado anterior de la reserva	Valores: "Confirmado", "Cancelado"	"Confirmado"
EstadoNuevo	ENUM	10	Nuevo estado de la reserva	Valores: "Confirmado", "Cancelado"	"Cancelado"
FechaCambio	TIMESTAMP	-	Fecha y hora del cambio	DEFAULT CURRENT_TIMESTAMP	2025-01-31 11:00:00
TipoCambio	ENUM	13	Tipo de cambio realizado	Valores: "Actualización", "Eliminación"	"Actualización"

Diccionarios de la tabla CambiosPago

Campo	Tipo Dato	Longitud – Tamaño	Descripción	Restricciones	Ejemplo
LogID	INT	-	Identificador único del cambio	PRIMARY KEY, AUTO_INCREMENT	304
PagoID	INT	-	Referencia al pago afectado	-	87
ReservaID	INT	-	Referencia a la reserva asociada	-	55
MontoAnterior	DECIMAL	(10, 2)	Monto del pago antes del cambio	-	85.50
MontoNuevo	DECIMAL	(10, 2)	Nuevo monto del pago	-	90.00
MetodoPagoAnterior	ENUM	8	Método de pago anterior	Valores: "Tarjeta", "Efectivo"	"Efectivo"
MetodoPagoNuevo	ENUM	8	Nuevo método de pago	Valores: "Tarjeta", "Efectivo"	"Tarjeta"
FechaCambio	TIMESTAMP	-	Fecha y hora del cambio	DEFAULT CURRENT_TIMESTAMP	2025-01-31 11:30:00

TipoCambio	ENUM	13	Tipo de cambio realizado	Valores: "Actualización", "Eliminación"	"Eliminación"
------------	------	----	--------------------------	---	---------------

1. Definir las restricciones de integridad referencial y eliminacion - update.

Práctica: Establecer claves primarias y foráneas entre las tablas, asegurando la coherencia de los datos (por ejemplo, ClienteID debe estar presente en las tablas relacionadas).

Eliminación – Update de cascada, set null, restrict, Constrains (NO ACTION)

Investigación: Investigar cómo la integridad referencial puede prevenir la pérdida de datos y mantener la consistencia.

Importancia del Conocimiento: Las restricciones de integridad aseguran que los datos no se corrompan.

```

CREATE TABLE Cliente(
    ClienteID INT AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(100) NOT NULL,
    Apellido VARCHAR(100) NOT NULL,
    Correo VARCHAR(100) UNIQUE NOT NULL,
    Edad INT NOT NULL
);

CREATE TABLE Reserva(
    ReservaID INT AUTO_INCREMENT PRIMARY KEY,
    ClienteID INT NOT NULL,
    FechaReserva DATE NOT NULL,
    Estado ENUM("Confirmado", "Cancelado") NOT NULL,
    CONSTRAINT fk_cliente_reserva FOREIGN KEY (ClienteID) REFERENCES Cliente(ClienteID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE Pago(
    PagoID INT AUTO_INCREMENT PRIMARY KEY,
    ReservaID INT NOT NULL,
    Cantidad DECIMAL(10, 2) NOT NULL,
    MetodoPago ENUM("Tarjeta", "Efectivo") NOT NULL,
    CONSTRAINT fk_pago_reserva FOREIGN KEY (ReservaID) REFERENCES Reserva(ReservaID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE Mesa(
    MesaID INT AUTO_INCREMENT PRIMARY KEY,
    ReservaID INT NOT NULL,
    Capacidad INT NOT NULL,
    Estado ENUM("Libre", "Ocupado") NOT NULL,
    CONSTRAINT fk_mesa_reserva FOREIGN KEY (ReservaID) REFERENCES Reserva(ReservaID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

```

2. Seguridad, Auditoría y Control de Acceso

Objetivo: Proteger los datos sensibles y controlar el acceso a la base de datos.

Actividades:

1. Implementar políticas de acceso y seguridad.

Práctica: Crear roles y permisos de usuario (por ejemplo, roles de Administrador, Usuario, Auditor) para controlar el acceso a las tablas y vistas.


```

1  -- 1) Creamos los roles que luego seran asignados un usuarios
2  • CREATE ROLE Administrador;
3  • CREATE ROLE Usuario;
4  • CREATE ROLE Auditor;
5  -- Asignamos todos los permisos al Administrador, permisos de escritura y consulta al usuario y permiso de leer datos a invitado
6  • GRANT ALL PRIVILEGES ON Restaurante.* TO Administrador;
7
8  • GRANT SELECT, INSERT, UPDATE ON Restaurante.Cliente TO Usuario;
9  • GRANT SELECT, INSERT, UPDATE ON Restaurante.Reserva TO Usuario;
10 • GRANT SELECT, INSERT, UPDATE ON Restaurante.Pago TO Usuario;
11 • GRANT select, insert, update on Restaurante.Mesa TO Usuario;
12 • GRANT SELECT ON Restaurante.Cliente TO Auditor;
13 • GRANT SELECT ON Restaurante.Reserva TO Auditor;
14 • GRANT SELECT ON Restaurante.Pago TO Auditor;
15 • GRANT SELECT ON Restaurante.Mesa TO Auditor;
16 -- 2) Creamos los usuarios y asignamos roles
17 • CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin123';
18 • CREATE USER 'empleado'@'localhost' IDENTIFIED BY 'empleado123';
19 • CREATE USER 'auditor'@'localhost' IDENTIFIED BY 'auditor123';
20
21 • GRANT Administrador TO 'admin'@'localhost';
22 • GRANT Usuario TO 'empleado'@'localhost';
23 • GRANT Auditor TO 'auditor'@'localhost';
24 -- 3) Aplicamos los roles a los usuarios creados
25 • SET DEFAULT ROLE Administrador to 'admin'@'localhost';
26 • SET DEFAULT ROLE Usuario to 'empleado'@'localhost';

```

✓	ID	TIME	QUERY
✓	12	20:13:10	GRANT SELECT, INSERT, UPDATE ON Restaurante.Cliente TO Usuario
✓	13	20:13:10	GRANT SELECT, INSERT, UPDATE ON Restaurante.Reserva TO Usuario
✓	14	20:13:10	GRANT SELECT, INSERT, UPDATE ON Restaurante.Pago TO Usuario
✓	15	20:13:10	GRANT select, insert, update on Restaurante.Mesa TO Usuario
✓	16	20:13:10	GRANT SELECT ON Restaurante.Cliente TO Auditor
✓	17	20:13:10	GRANT SELECT ON Restaurante.Reserva TO Auditor
✓	18	20:13:10	GRANT SELECT ON Restaurante.Pago TO Auditor
✓	19	20:13:10	GRANT SELECT ON Restaurante.Mesa TO Auditor
✓	20	20:13:10	CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin123'
✓	21	20:13:10	CREATE USER 'empleado'@'localhost' IDENTIFIED BY 'empleado123'
✓	22	20:13:10	CREATE USER 'auditor'@'localhost' IDENTIFIED BY 'auditor123'
✓	23	20:13:10	GRANT Administrador TO 'admin'@'localhost'
✓	24	20:13:10	GRANT Usuario TO 'empleado'@'localhost'
✓	25	20:13:10	GRANT Auditor TO 'auditor'@'localhost'
✓	26	20:13:10	SET DEFAULT ROLE Administrador to 'admin'@'localhost'
✓	27	20:13:10	SET DEFAULT ROLE Usuario to 'empleado'@'localhost'
✓	28	20:13:10	SET DEFAULT ROLE Auditor to 'auditor'@'localhost'

Investigación: Investigar sobre los mejores enfoques para la seguridad en bases de datos en entornos de alta disponibilidad.

Supongamos a una corporación o empresa que requiere encriptar datos para la seguridad de los mismos, entonces cual son sus opciones para mantener seguro a la base de datos.? Primero se debe tener en cuenta que las contraseñas no deben cifrarse, sino hashear con algoritmos diseñados para autenticación. Existen opciones mas allá de MySQL como:

- Argon2 (Resistente a ataques modernos)
- Bcrypt (Seguro y ampliamente usado)
- PBKDF2 (Seguro, pero más lento)

Y si se habla de softwares, por parte de Oracle se tiene Transparent Data Encryption (TDE) para cifrado automático en disco y por parte de MongoDB tenemos a Client-side Field Level Encryption (CSFLE) que es el cifrado en el cliente antes de llegar a la BD.

Importancia del Conocimiento: El control adecuado de acceso previene fugas de información y mejora la seguridad general.

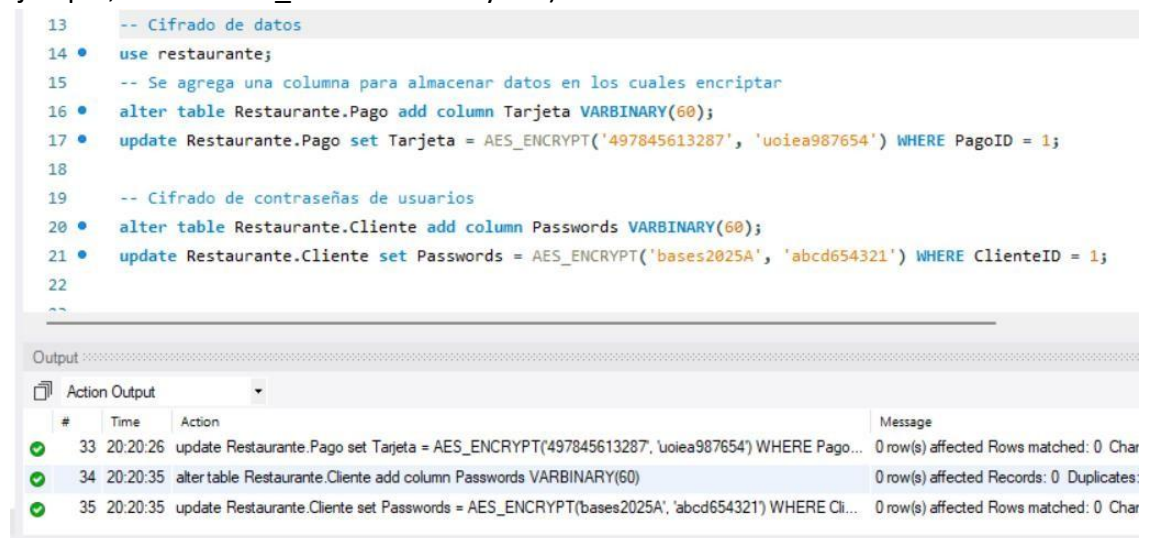
2. Cifrado de datos sensibles.

Práctica: Cifrar información sensible como contraseñas y detalles de pago (por ejemplo, usando AES_ENCRYPT en MySQL).

```

13  -- Cifrado de datos
14  • use restaurante;
15  -- Se agrega una columna para almacenar datos en los cuales encriptar
16  • alter table Restaurante.Pago add column Tarjeta VARBINARY(60);
17  • update Restaurante.Pago set Tarjeta = AES_ENCRYPT('497845613287', 'uoiea987654') WHERE PagoID = 1;
18
19  -- Cifrado de contraseñas de usuarios
20  • alter table Restaurante.Cliente add column Passwords VARBINARY(60);
21  • update Restaurante.Cliente set Passwords = AES_ENCRYPT('bases2025A', 'abcd654321') WHERE ClienteID = 1;
22
23  --

```



#	Time	Action	Message
33	20:20:26	update Restaurante.Pago set Tarjeta = AES_ENCRYPT('497845613287', 'uoiea987654') WHERE Pago...	0 row(s) affected Rows matched: 0 Char
34	20:20:35	alter table Restaurante.Cliente add column Passwords VARBINARY(60)	0 row(s) affected Records: 0 Duplicates:
35	20:20:35	update Restaurante.Cliente set Passwords = AES_ENCRYPT('bases2025A', 'abcd654321') WHERE Cli...	0 row(s) affected Rows matched: 0 Char

Investigación: Explorar algoritmos de cifrado y su impacto en el rendimiento de la base de datos.

- AES_ENCRYPT es más seguro para datos que necesitan descifrarse, pero impacta el rendimiento.
- SHA2 es más rápido y eficiente para contraseñas ya que no requiere descifrado.
- BCRYPT para mayor seguridad en contraseñas.

Importancia del Conocimiento: El cifrado es esencial para la protección de la información confidencial de los usuarios.

3. Habilitar auditoría y registrar eventos de base de datos.

Práctica: Activar los logs de acceso y auditoría para monitorear las actividades de los usuarios (por ejemplo, registrar quién accedió a qué datos).

Para activar los logs en este caso (binarios que son de solo lectura) es necesario modificar el archivo bin o conf, luego del reinicio del servidor se puede verificar el estado mediante:

```

1 • use restaurante;
2 -- Desde los privilegios de administrador activamos los logs binario para auditoria (requiere reinicio de servidor)
3 • set global log_bin= "ON";
4 • SHOW VARIABLES LIKE 'log_bin';
5

```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
Variable_name	Value		
log_bin	ON		

Luego se procede normalmente:

```

6 -- Se crea una tabla adicional para almacenar el registro de actualizacion
7 • create table Restaurante_Auditoria (
8     ID INT AUTO_INCREMENT PRIMARY KEY,
9     Usuario VARCHAR(50),
10    Accion VARCHAR(255),
11    Fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP
12 );
13 -- Proceso para crear el registro de actualizacion a los datos mediante un trigger
14 DELIMITER $$
15 • create trigger log_cliente_actualizar
16 after update on Restaurante.Cliente
17 FOR EACH ROW
18 BEGIN
19     INSERT INTO Restaurante_Auditoria (Usuario, Accion)
20     VALUES (CURRENT_USER(), CONCAT('Actualización en ClienteID: ', OLD.ClienteID));
21 END $$
22 DELIMITER ;
23

```

#	Time	Action	Message
41	20:34:24	SHOW VARIABLES LIKE 'log_bin'	1 row(s) returned
42	20:37:27	create table Restaurante_Auditoria (ID INT AUTO_INCREMENT PRIMARY KEY, Usuario VARC...	0 row(s) affected
43	20:37:33	create trigger log_cliente_actualizar after update on Restaurante.Cliente FOR EACH ROW BEGIN I...	0 row(s) affected

Investigación: Buscar cómo configurar herramientas de auditoría en MySQL o PostgreSQL.

En MySQL:

- General Log / Binlog = set global general_log = 'ON' En PostgreSQL:
- Logs de consulta = log_statement = 'all'

Importancia del Conocimiento: La auditoría permite rastrear cambios en los datos y detectar actividades sospechosas.

3. Respaldos y Recuperación de Datos

Objetivo: Asegurar la integridad y disponibilidad de los datos mediante técnicas de respaldo confiables.

Actividades:

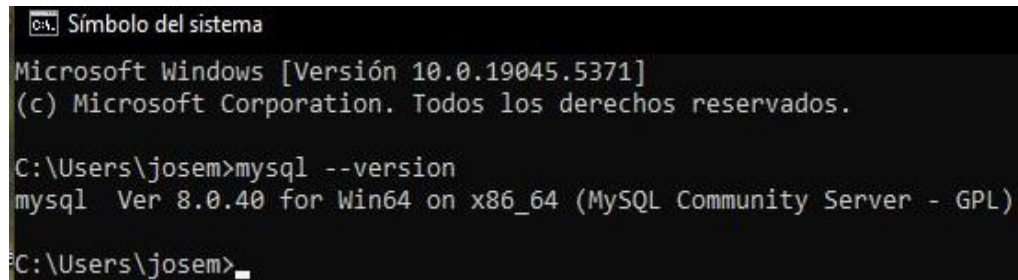
1. Crear respaldos completos (full backups).

Práctica: Utilizar mysqldump o herramientas similares para hacer respaldos completos de la base de datos.

Para poder realizar este proceso que tener MySQL en PATH, para esto hay que ir a configuraciones avanzadas del sistema, luego ir a variables de entorno y en la opción PATH agregar la ruta de instalación de MySQL como por ejemplo:

C:\Program Files\MySQL\MySQL Server 8.0\bin

Luego ejecutamos en el cmd el comando mysql --version y debe salir lo siguiente:

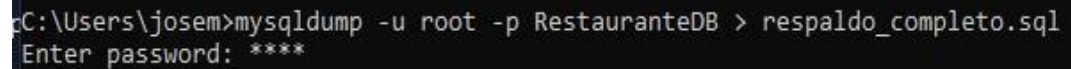


```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19045.5371]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\josem>mysql --version
mysql Ver 8.0.40 for Win64 on x86_64 (MySQL Community Server - GPL)

C:\Users\josem>
```

Luego usamos mysqldump para hacer el respaldo de toda la base de datos



```
C:\Users\josem>mysqldump -u root -p RestauranteDB > respaldo_completo.sql
Enter password: ****
```

Donde:

-u usuario: Indica el nombre del usuario que tiene privilegios para acceder a la base de datos (reemplázalo con el usuario adecuado).

-p: Esto te pedirá que ingreses la contraseña de MySQL.

RestauranteDB: Es el nombre de tu base de datos.

> respaldo_completo.sql: Esto especifica que el respaldo se guardará en un archivo SQL con el nombre respaldo_completo.sql.

Si se quiere incluir triggers y procedimientos de almacenado, ejemplo:

```
mysqldump -u root -p --databases RestauranteDB --routines --triggers --events >
restaurante_backup.sql
```

Para restaurar la base, ejemplo:

```
mysql -u root -p < restaurante_backup.sql
```

Investigación: Buscar estrategias de respaldo para bases de datos de gran tamaño y la mejor manera de gestionarlas.

Un ejemplo sería utilizar la comprensión para cuando se necesite respaldar bases de datos grandes, ahí se puede comprimir el archivo de respaldo para reducir el espacio de almacenamiento.

Se recomienda configurar una tarea programada, como ejemplo tenemos al programador de tareas en Windows para realizar respaldos periódicos que puede incluir respaldos diarios o semanales, dependiendo de la frecuencia con la que se actualicen los datos.

El comando sería:

```
mysqldump -u usuario -p RestauranteDB | gzip > respaldo_completo.sql.gz
```

El cual comprimirá el respaldo en formato gzip.

Importancia del Conocimiento: Los respaldos completos permiten restaurar toda la base de datos ante una falla.

Pero a su vez pueden consumir mucho tiempo y espacio de almacenamiento, especialmente con bases de datos grandes.

2. Configurar respaldos incrementales.

Práctica: Realizar respaldos incrementales para reducir el tiempo y espacio de almacenamiento.

Los respaldos incrementales solo respaldan los cambios que han ocurrido desde el último respaldo completo o incremental. Esto optimiza el uso de almacenamiento y tiempo, ya que no se respalda toda la base de datos cada vez.

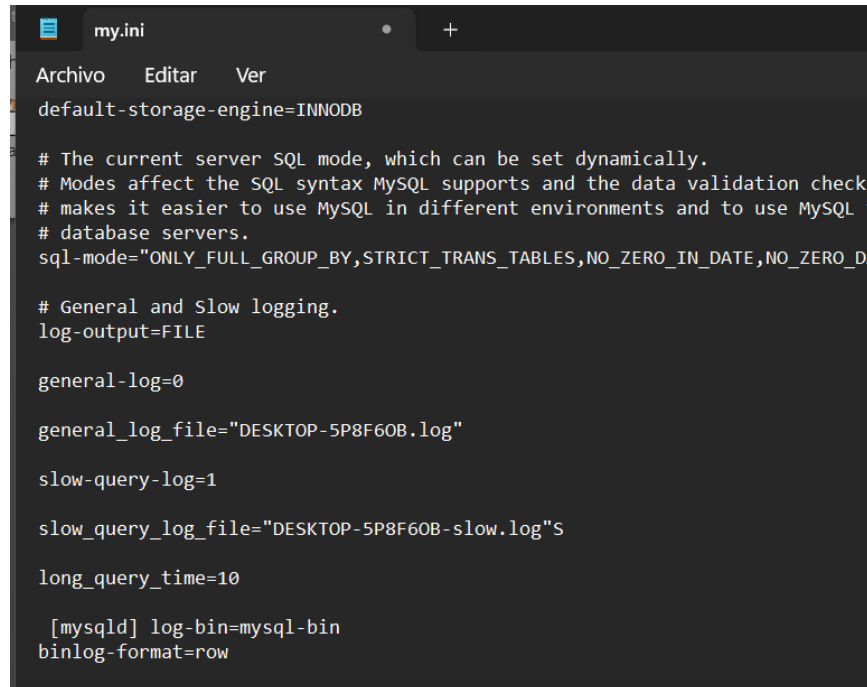
Investigación: Investigar cómo realizar respaldos incrementales y cuándo es más conveniente utilizarlos.

En la terminal cmd:

Hay que activar los binlogs en MySQL: Asegúrate de que los logs binarios estén habilitados en tu configuración de MySQL.

Para esto, se debe agregar lo siguiente en tu archivo my.cnf (en Linux) o my.ini (en Windows):

```
[mysqld] log-bin=mysql-bin  
binlog-format=row
```



```
my.ini  
Archivo  Editar  Ver  
default-storage-engine=INNODB  
  
# The current server SQL mode, which can be set dynamically.  
# Modes affect the SQL syntax MySQL supports and the data validation checks  
# makes it easier to use MySQL in different environments and to use MySQL t  
# database servers.  
sql-mode="ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DA  
  
# General and Slow logging.  
log-output=FILE  
  
general-log=0  
  
general_log_file="DESKTOP-5P8F60B.log"  
  
slow-query-log=1  
  
slow_query_log_file="DESKTOP-5P8F60B-slow.log"s  
  
long_query_time=10  
  
[mysqld] log-bin=mysql-bin  
binlog-format=row
```

Realizar el respaldo completo: Haz un respaldo completo de la base de datos, explicado en el punto 1.

Hacer respaldos incrementales utilizando los binlogs: Después del respaldo completo, puedes usar los binlogs para hacer un respaldo incremental de los cambios realizados. Para hacer esto, se utiliza el comando `mysqlbinlog` para aplicar los cambios registrados en los binlogs.

Ejemplo para aplicar un respaldo incremental:

```
mysqlbinlog /path/to/binlog.000001 > respaldo_incremental.sql
```

Es conveniente utilizar respaldos incrementales cuando se realizan cambios frecuentes en la base de datos, pero no se necesita hacer un respaldo completo en cada momento

También es recomendable usarlo en bases de datos grandes donde los respaldos completos podrían consumir mucho tiempo y espacio de almacenamiento.

Importancia del Conocimiento: Los respaldos incrementales permiten optimizar los recursos y acelerar los tiempos de recuperación.

3. Implementar respaldos en caliente (Hot Backups).

Práctica: Hacer respaldos sin interrumpir el servicio (por ejemplo, usando Percona XtraBackup).

Para realizar el proceso necesitamos instalar, Percona XtraBackup, se va a explicar en Linux, en cambio en Windows solo descargamos el instalador de la página oficial:

Instalar con el comando:

```
sudo yum install percona-xtrabackup
```

Realizar el respaldo en caliente:

```
xtrabackup --backup --target-dir=/ruta/del/respaldo
```

Preparar el respaldo para poder restaurar:

```
xtrabackup --prepare --target-dir=/ruta/del/respaldo
```

Para poder restaurar el respaldo:

```
xtrabackup --copy-back --target-dir=/ruta/del/respaldo
```

Investigación: Investigar cómo hacer respaldos sin detener la base de datos.

XtraBackup permite copiar los datos de MySQL sin interrumpir el acceso a las bases de datos mientras se realiza el respaldo. A pesar de que la base de datos sigue funcionando, XtraBackup asegura que el respaldo sea consistente. Usando

los binary logs que se agregaron a MySQL, asegura que los datos no comprometidos sean aplicados durante el proceso de respaldo y restauración.

Importancia del Conocimiento: Los respaldos en caliente son esenciales para bases de datos de producción que no pueden permitirse inactividad.

4. Optimización y Rendimiento de Consultas

Objetivo: Mejorar la eficiencia en la recuperación de datos mediante la optimización de consultas y el uso adecuado de índices.

Actividades:

1. Crear y gestionar índices.

Práctica: Implementar índices en las columnas más consultadas, como VueloID, ClienteID, etc.

Crear dos índices importantes

```
1  -- INDICES
2  -- índice en la columna ClienteID de la tabla Reserva
3  • CREATE INDEX indice_cliente ON Reserva(ClienteID);
4
5  -- índice en la columna ReservaID de la tabla Pago
6  • CREATE INDEX indice_pago ON Pago(ReservaID);
7
8  -- índice en la columna ReservaID de la tabla Mesa
9  • CREATE INDEX indice_mesa ON Mesa(ReservaID);
```

Luego comprobamos mediante la muestra de los índices en cada caso

11 • `show indexes in restaurante.reserva;`

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
reserva	0	PRIMARY	1	ReservaID	A	0	HULL	HULL		BTREE			YES	HULL
reserva	1	indice_cliente	1	ClienteID	A	0	HULL	HULL		BTREE			YES	HULL

11 • `show indexes in restaurante.pago;`

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
pago	0	PRIMARY	1	PagoID	A	0	HULL	HULL		BTREE			YES	HULL
pago	1	indice_pago	1	ReservaID	A	0	HULL	HULL		BTREE			YES	HULL

11 • `show indexes in restaurante.mesa;`

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
mesa	0	PRIMARY	1	MesaID	A	0	HULL	HULL		BTREE			YES	HULL
mesa	1	indice_mesa	1	ReservaID	A	0	HULL	HULL		BTREE			YES	HULL

Investigación: Investigar sobre los tipos de índices más adecuados para bases de datos transaccionales y cómo afectan el rendimiento.

- Los índices B-Tree y compuestos son los más utilizados debido a su versatilidad. Sin embargo si se da a elegir un índice, esta debe basarse en el tipo de consultas que se ejecutan con mayor frecuencia y en el equilibrio entre el rendimiento de lectura y el costo de mantenimiento. Un diseño cuidadoso de los índices es clave para optimizar el rendimiento de la base de datos.
- Otro tipo de índice se los puede denominar como “Índices Hash” que son usados para ubicar claves a ubicaciones específicas en una tabla. Es ideal para consultas de igualdad exacta.

Importancia del Conocimiento: Los índices son cruciales para acelerar las consultas y mejorar el rendimiento general de la base de datos.

2. Optimizar consultas SQL.

Práctica: Utilizar herramientas como EXPLAIN para identificar cuellos de botella en las consultas y optimizarlas.

Practica: Aplicación de 3 join

```

1  -- JOINS
2  -- información de reservas con detalles del cliente
3  • explain
4  select
5      C.Nombre,
6      C.Apellido,
7      R.FechaReserva,
8      R.Estado
9  from
10     Reserva R
11  join
12     Cliente C ON R.ClienteID = C.ClienteID;
13  -- Obtener detalles de pagos con información de la reserva y el cliente

```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	R	HULL	ALL	indice_cliente	HULL	HULL	HULL	1	100.00	HULL
1	SIMPLE	C	HULL	eq_ref	PRIMARY	PRIMARY	4	restaurante.R.ClienteID	1	100.00	HULL

```

13  -- Obtener detalles de pagos con información de la reserva y el cliente
14  • explain
15  select
16      C.Nombre,
17      C.Apellido,
18      R.FechaReserva,
19      P.Cantidad,
20      P.MetodoPago
21  from
22     Pago P
23  join
24     Reserva R ON P.ReservaID = R.ReservaID
25  join
26     Cliente C ON R.ClienteID = C.ClienteID;

```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	P	HULL	ALL	indice_pago	HULL	HULL	HULL	1	100.00	HULL
1	SIMPLE	R	HULL	eq_ref	PRIMARY,indice_cliente	PRIMARY	4	restaurante.P.ReservaID	1	100.00	HULL
1	SIMPLE	C	HULL	eq_ref	PRIMARY	PRIMARY	4	restaurante.R.ClienteID	1	100.00	HULL

```

28  -- información de mesas con detalles de la reserva y el cliente
29  • explain
30  select
31      C.Nombre,
32      C.Apellido,
33      R.FechaReserva,
34      M.Capacidad,
35      M.Estado
36  from
37     Mesa M
38  join
39     Reserva R ON M.ReservaID = R.ReservaID
40  join
41     Cliente C ON R.ClienteID = C.ClienteID;

```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	M	HULL	ALL	indice_mesa	HULL	HULL	HULL	1	100.00	HULL
1	SIMPLE	R	HULL	eq_ref	PRIMARY,indice_cliente	PRIMARY	4	restaurante.M.ReservaID	1	100.00	HULL
1	SIMPLE	C	HULL	eq_ref	PRIMARY	PRIMARY	4	restaurante.R.ClienteID	1	100.00	HULL

Investigación: Investigar cómo hacer uso eficiente de las uniones (JOIN), subconsultas, y optimizar las consultas complejas.

Si se desea una buena estructuración en cuanto a las relaciones y lectura de datos en MySQL, existen diferentes formas de organizar el código: Por

ejemplo, si se usa JOIN, se debe asegurar de que las columnas utilizadas estén bien indexadas, filtrar las filas antes de realizar el JOIN para reducir el tamaño de los datos que se unen.

Hay que tener en cuenta que las subconsultas correlacionadas (aquellas que dependen de la consulta externa) pueden ser lentas porque se ejecutan una vez por cada fila de la consulta externa por ello si hay la posibilidad se debe reemplazar por un JOIN y si es el caso contrario se debe filtrar las filas antes de realizar el JOIN para reducir el tamaño de los datos que se unen.

Importancia del Conocimiento: Las consultas optimizadas aseguran un sistema rápido y eficiente, especialmente en sistemas con alta demanda.

3. Utilizar particionamiento de tablas.

Práctica: Dividir tablas grandes, como Reservas, en particiones según una clave (por ejemplo, por fecha).

```
-- PARTICIONES
use restaurante;
-- Es necesario eliminar la tabla existente a particionar porque no se permite modificar la tabla y mucho menos particionarla
drop table Reserva;
-- Rehacemos otra tabla pero con la misma estructura u otro nombre
CREATE TABLE Reserva_Particionada (
  ReservaID INT AUTO_INCREMENT,
  ClienteID INT NOT NULL,
  FechaReserva DATE NOT NULL,
  Estado ENUM("Confirmado", "Cancelado") NOT NULL,
  PRIMARY KEY (ReservaID, FechaReserva) -- La llave debe contener a la columna de particion
);
```

Luego se crea las particiones y se verifica insertando datos y seleccionando la partición:

```
13 ALTER TABLE Reserva_Particionada
14 PARTITION BY RANGE (YEAR(FechaReserva) * 100 + MONTH(FechaReserva)) (
15   PARTITION part0 VALUES LESS THAN (202410), -- Particiones para meses anteriores a octubre 2024
16   PARTITION part1 VALUES LESS THAN (202411), -- Para octubre 2024
17   PARTITION part2 VALUES LESS THAN (202412), -- Para noviembre 2024
18   PARTITION part3 VALUES LESS THAN (202501), -- Para diciembre 2024
19   PARTITION part4 VALUES LESS THAN MAXVALUE -- Para meses fuera del año 2024
20 );
21 select version();
22 -- Para poner en uso las particiones podemos insertar datos para ver su distribucion em las particiones
23 INSERT INTO Reserva_Particionada (ClienteID, FechaReserva, Estado) VALUES
24 (1, '2024-09-15', 'Confirmado'), -- Irá a la partición part0
25 (2, '2024-10-01', 'Confirmado'), -- Irá a la partición part1
26 (3, '2024-11-10', 'Cancelado'), -- Irá a la partición part2
27 (1, '2024-12-25', 'Confirmado'), -- Irá a la partición part3
28 (2, '2025-01-05', 'Cancelado'); -- Irá a la partición part4
29
30 SELECT * FROM Reserva_Particionada PARTITION (part1);
```

Result Grid			
Filter Rows:			
ReservaID	ClienteID	FechaReserva	Estado
7	2	2024-10-01	Confirmado
NULL	NULL	NULL	NULL

Investigación: Investigar sobre los beneficios del particionamiento y cómo implementarlo en sistemas de bases de datos grandes.

La partición en bases de datos tiene como propósito mejorar el rendimiento y la gestión de bases de datos grandes. Consiste en dividir una tabla en partes más pequeñas y manejables, llamadas particiones, que se almacenan y gestionan de forma independiente.

La implementación de particiones en MySQL se divide por tipos:

- **Particionamiento por rango (RANGE):** Ya utilizada en la práctica, consiste en dividir los datos en función de un rango de valores (por ejemplo, fechas o números).
- **Particionamiento por lista (LIST):** Divide los datos en función de una lista de valores.
- **Particionamiento por hash (HASH):** Distribuye los datos en particiones basadas en una función hash.

Importancia del Conocimiento: El particionamiento de tablas mejora la escalabilidad y el rendimiento en bases de datos con gran volumen de datos.

5. Procedimientos Almacenados, Vistas y Triggers, Funciones (prácticas de cada uno)

Objetivo: Mejorar la eficiencia y automatizar tareas mediante el uso de procedimientos almacenados, vistas y triggers.

Actividades:

1. Crear procedimientos almacenados.

Práctica: Crear un procedimiento para calcular el precio total de una reserva, aplicando descuentos y cargos adicionales, aplicar 2 ejercicios y explicar comprensión al 100%

Investigación: Explorar cómo los procedimientos almacenados pueden mejorar la reutilización de código y la eficiencia.

- Los procedimientos almacenados nos permiten centralizar la lógica de negocio dentro de la base de datos. Esto significa que el código que se ejecuta con regularidad puede ser escrito una sola vez y luego reutilizado en múltiples aplicaciones, reduciendo así la duplicación de código. Al usar procedimientos almacenados, se minimiza la necesidad de enviar múltiples comandos SQL desde la aplicación a la base de datos.

Importancia del Conocimiento: Los procedimientos almacenados centralizan la lógica y pueden mejorar el rendimiento al ejecutarse directamente en el servidor.

DELIMITER \$\$

```
CREATE PROCEDURE CalcularPrecioTotal(  
    IN p_ReservaID INT,  
    IN p_DescuentoPorc DECIMAL(10, 2),  
    IN p_CargoAdicional DECIMAL(10, 2)  
)  
BEGIN  
    DECLARE v_Cantidad DECIMAL(10, 2);  
    DECLARE v_PrecioFinal DECIMAL(10, 2);  
  
    SELECT SUM(Cantidad) INTO v_Cantidad FROM Pago WHERE ReservaID = p_ReservaID;  
  
    SET v_PrecioFinal = v_Cantidad - (v_Cantidad * p_DescuentoPorc / 100) + p_CargoAdicional;  
  
    SELECT p_ReservaID AS ReservaID, v_Cantidad AS PrecioBase, p_DescuentoPorc AS Descuento,  
        p_CargoAdicional AS CargoAdicional, v_PrecioFinal AS PrecioFinal;  
  
END $$
```

DELIMITER ;

-- En caso de haber algun descuento o cargo adicional

CALL CalcularPrecioTotal(1, 20, 1.50);

-- En caso de no haber ninguno

CALL CalcularPrecioTotal(1, 0, 0);

68 • -- En caso de haber algun descuento o cargo adicional

69 CALL CalcularPrecioTotal(1, 20, 1.50);

--

Result Grid					
Filter Rows:					
Export: Wrap Cell Content:					
	ReservaID	PrecioBase	Descuento	CargoAdicional	PrecioFinal
▶	1	50.00	20.00	1.50	41.50

71 -- En caso de no haber ninguno

72 • CALL CalcularPrecioTotal(1, 0, 0);

--

Result Grid					
Filter Rows:					
Export: Wrap Cell Content:					
	ReservaID	PrecioBase	Descuento	CargoAdicional	PrecioFinal
▶	1	50.00	0.00	0.00	50.00

```

-- FUNCION

DELIMITER $$

• CREATE FUNCTION TotalReservasPorCliente(p_ClienteID INT)
  RETURNS INT
  DETERMINISTIC
  BEGIN
    DECLARE total_reservas INT;


    SELECT COUNT(*) INTO total_reservas FROM Reserva WHERE ClienteID = p_ClienteID;
    RETURN total_reservas;
  END$$

DELIMITER ;

• SELECT TotalReservasPorCliente(5) AS NumeroReservas;

--
60 • SELECT TotalReservasPorCliente(5) AS NumeroReservas;
61
--

```



NumeroReservas
1

2. Crear vistas para simplificar consultas complejas.

Práctica: Crear vistas que presenten información de varias tablas de manera unificada (por ejemplo, una vista que combine datos de Vuelos, Clientes y Reservas).

Investigación: Investigar las ventajas de usar vistas en lugar de consultas complejas repetitivas.

- Las vistas encapsulan consultas complejas dentro de un objeto de base de datos, lo que permite simplificar las consultas posteriores. En lugar de repetir la misma lógica compleja una y otra vez, se puede simplemente referirse a la vista, lo que mejora la legibilidad del código.

Importancia del Conocimiento: Las vistas ayudan a simplificar el acceso a datos complejos y pueden mejorar la seguridad al limitar el acceso directo a las tablas.

```
-- VISTA

• CREATE VIEW Vista_ClienteReservaMesa AS
SELECT
    c.ClienteID,
    CONCAT(c.Nombre, ' ', c.Apellido) AS ClienteNombre,
    c.Correo AS ClienteCorreo,
    c.Edad AS ClienteEdad,
    r.ReservaID,
    r.FechaReserva,
    r.Estado AS ReservaEstado,
    m.MesaID,
    m.Capacidad AS MesaCapacidad,
    m.Estado AS MesaEstado
FROM Cliente c JOIN Reserva r ON c.ClienteID = r.ClienteID JOIN Mesa m ON r.ReservaID = m.ReservaID;

• SELECT * FROM Vista_ClienteReservaMesa;

59 • SELECT * FROM Vista_ClienteReservaMesa;
--
```

ClienteID	ClienteNombre	ClienteCorreo	ClienteEdad	ReservaID	FechaReserva	ReservaEstado	MesaID	MesaCapacidad	MesaEstado
1	Juan Pérez	juan.perez@example.com	25	1	2025-01-30 19:47:53	Confirmado	1	4	Ocupado
2	Ana Gómez	ana.gomez@example.com	30	2	2025-01-30 19:47:53	Cancelado	2	2	Libre
3	Carlos López	carlos.lopez@example.com	22	3	2025-01-30 19:47:53	Confirmado	3	6	Ocupado
4	María Rodríguez	maria.rodriguez@example.com	28	4	2025-01-30 19:47:53	Confirmado	4	8	Libre
5	Pedro Martínez	pedro.martinez@example.com	35	5	2025-01-30 19:47:53	Cancelado	5	4	Ocupado
6	Lucía Hernández	lucia.hernandez@example.com	27	6	2025-01-30 19:47:53	Confirmado	6	2	Ocupado
7	Jorge Sánchez	jorge.sanchez@example.com	40	7	2025-01-30 19:47:53	Confirmado	7	6	Libre
8	Elena Ruiz	elena.ruiz@example.com	33	8	2025-01-30 19:47:53	Cancelado	8	8	Ocupado

3. Implementar triggers para auditoría y control de cambios.

Práctica: Crear triggers que registren cambios en las tablas de Reservas y Pagos cada vez que un registro se actualiza o elimina., 2 ejercicios conocimiento al 100%

Investigación: Investigar cómo utilizar triggers para mantener un historial de cambios en la base de datos.

- Crear una tabla de historial: Esta tabla almacenará los registros históricos de los cambios realizados en las tablas originales.
- Crear triggers para cada operación de interés (INSERT, UPDATE, DELETE).
- Dentro de cada trigger, capturar los valores antiguos y nuevos, para luego almacenarlos en la tabla de historial.

Importancia del Conocimiento: Los triggers permiten automatizar tareas como la auditoría y validación de datos.

Acciones a realizar de forma automática, es decir si desea aplicar un calcuo de descuento y cambio del iva que se debe hacer donde se pone esos valores y como se automatiza


```

CREATE TABLE CambiosReserva (
    LogID INT AUTO_INCREMENT PRIMARY KEY,
    ReservaID INT,
    ClienteID INT,
    FechaReserva TIMESTAMP,
    EstadoAnterior ENUM("Confirmado", "Cancelado"),
    EstadoNuevo ENUM("Confirmado", "Cancelado"),
    FechaCambio TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    TipoCambio ENUM("Actualización", "Eliminación")
);

CREATE TABLE CambiosPago (
    LogID INT AUTO_INCREMENT PRIMARY KEY,
    PagoID INT,
    ReservaID INT,
    MontoAnterior DECIMAL(10, 2),
    MontoNuevo DECIMAL(10, 2),
    MetodoPagoAnterior ENUM("Tarjeta", "Efectivo"),
    MetodoPagoNuevo ENUM("Tarjeta", "Efectivo"),
    FechaCambio TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    TipoCambio ENUM("Actualización", "Eliminación")
);

-- TRIGGER PARA VERIFICAR ACTUALIZACIONES EN LAS RESERVAS

DELIMITER $$

CREATE TRIGGER RegistroReservaUpdate
AFTER UPDATE ON Reserva
FOR EACH ROW
BEGIN
    INSERT INTO CambiosReserva(ReservaID, ClienteID, FechaReserva, EstadoAnterior, EstadoNuevo, TipoCambio)
    VALUES (OLD.ReservaID, OLD.ClienteID, OLD.FechaReserva, OLD.Estado, NEW.Estado, "Actualización");
END$$

DELIMITER ;

UPDATE Reserva SET Estado = "Confirmado" WHERE ReservaID = 16;
SELECT * FROM CambiosReserva;

```



```

-- TRIGGER PARA VERIFICAR ELIMINACIONES EN LAS RESERVAS

DELIMITER $$

CREATE TRIGGER RegistroReservaDelete
AFTER DELETE ON Reserva
FOR EACH ROW
BEGIN
    INSERT INTO CambiosReserva(ReservaID, ClienteID, FechaReserva, EstadoAnterior, EstadoNuevo, TipoCambio)
    VALUES (OLD.ReservaID, OLD.ClienteID, OLD.FechaReserva, OLD.Estado, NULL, "Eliminación");
END$$

DELIMITER ;

DELETE FROM Reserva WHERE ReservaID = 29;
SELECT * FROM CambiosReserva;

-- TRIGGER PARA VERIFICAR ACTUALIZACION EN LOS PAGOS

DELIMITER $$

CREATE TRIGGER RegistroPagoUpdate
AFTER UPDATE ON Pago
FOR EACH ROW
BEGIN
    INSERT INTO CambiosPago(PagoID, ReservaID, MontoAnterior, MontoNuevo, MetodoPagoAnterior, MetodoPagoNuevo, TipoCambio)
    VALUES (OLD.PagoID, OLD.ReservaID, OLD.Cantidad, NEW.Cantidad, OLD.MetodoPago, NEW.MetodoPago, "Actualización");
END $$

DELIMITER ;

UPDATE Pago SET Cantidad = 85.50, MetodoPago = "Efectivo" WHERE PagoID = 34;
SELECT * FROM CambiosPago;

-- TRIGGER PARA VERIFICAR ELIMINACIONES EN LOS PAGOS

DELIMITER $$

CREATE TRIGGER RegistroPagoDelete
AFTER DELETE ON Pago
FOR EACH ROW
BEGIN
    INSERT INTO CambiosPago(PagoID, ReservaID, MontoAnterior, MontoNuevo, MetodoPagoAnterior, MetodoPagoNuevo, TipoCambio)
    VALUES (OLD.PagoID, OLD.ReservaID, OLD.Cantidad, NULL, OLD.MetodoPago, NULL, "Eliminación");
END$$

DELIMITER ;

DELETE FROM Pago WHERE PagoID = 44;
SELECT * FROM CambiosPago;

81 • SELECT * FROM CambiosReserva;
--

```

Result Grid								
Filter Rows:								
Edit: Export/Import: Wrap Cell Content:								
LogID	ReservaID	ClienteID	FechaReserva	EstadoAnterior	EstadoNuevo	FechaCambio	TipoCambio	
1	16	16	2025-01-30 20:50:06	Cancelado	Confirmado	2025-01-30 21:07:56	Actualización	
2	29	29	2025-01-30 20:50:06	Confirmado	NULL	2025-01-30 21:08:13	Eliminación	
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	

115 • `SELECT * FROM CambiosPago;`

Result Grid									
Filter Rows:									
LogID	PagoID	ReservaID	MontoAnterior	MontoNuevo	MetodoPagoAnterior	MetodoPagoNuevo	FechaCambio	TipoCambio	
1	34	34	55.00	85.50	Efectivo	Efectivo	2025-01-30 21:09:07	Actualización	
2	44	44	50.00	NULL	Efectivo	NULL	2025-01-30 21:10:18	Eliminación	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	

6. Monitoreo y Optimización de Recursos

Objetivo: Controlar el rendimiento de la base de datos, identificando y solucionando problemas de recursos.

Actividades:

1. Monitorear el rendimiento de consultas.

Práctica: Usar herramientas como SHOW PROCESSLIST para detectar consultas lentas y optimizarlas.

Con SHOW PROCESSLIST

-- Monitorear el rendimiento

`SHOW PROCESSLIST;`

Result Grid								
Filter Rows:								
Id	User	Host	db	Command	Time	State	Info	
5	event_scheduler	localhost	NULL	Daemon	166843	Waiting on empty queue	NULL	
22	root	localhost:8090	basetrigger	Sleep	381		NULL	
23	root	localhost:8091	basetrigger	Query	0	init	SHOW	

Los datos proporcionados son:

- ✦ ID: El identificador del hilo.
- ✦ Usuario: El usuario que ejecuta la consulta.
- ✦ Host: Desde qué host se está ejecutando la consulta.
- ✦ Base de datos: La base de datos que está siendo accedida.
- ✦ Comando: El tipo de comando (por ejemplo, Query para consultas).
- ✦ Tiempo: Cuánto tiempo lleva ejecutándose la consulta.
- ✦ Estado: El estado actual de la consulta.
- ✦ Info: La consulta SQL que está siendo ejecutada.

Para activar el log de consultas lentas:

```
SET GLOBAL slow_query_log = 'ON';
```

```
SET GLOBAL long_query_time = 2; -- Registra las consultas que duren más de 2 segundos
```

```
SHOW PROCESSLIST;  
SET GLOBAL slow_query_log = 'ON';  
SET GLOBAL long_query_time = 2; -- Registra las consultas que duren más de 2 segundos
```

Investigación: Investigar las mejores prácticas para monitorear el rendimiento de las consultas en producción.

Usando el log de consultas lenta ya explicado que permite identificar consultas que toman más tiempo del esperado para ejecutarse.

Usando EXPLAIN que permite analizar el plan de ejecución de una consulta, proporcionando detalles sobre cómo MySQL planea ejecutar esa consulta es decir si usará índices, tipo de unión, lo que ayudará a identificar posibles mejoras de consulta

```
EXPLAIN SELECT * FROM RestaurantDB.Cliente WHERE Correo = 'example@gmail.com';
```

SHOW STATUS para obtener información sobre el estado de la base de datos y sus variables de configuración, lo que es útil para identificar posibles cuellos de botella en el rendimiento.

```
177 • SHOW STATUS LIKE 'Handler%';
```

Variable_name	Value
Handler_commit	7
Handler_delete	0
Handler_discover	0
Handler_external_lock	70
Handler_mrr_init	0
Handler_prepare	0

Result 3

Importancia del Conocimiento: El monitoreo proactivo puede identificar cuellos de botella antes de que afecten el rendimiento del sistema.

2. Realizar pruebas de carga.

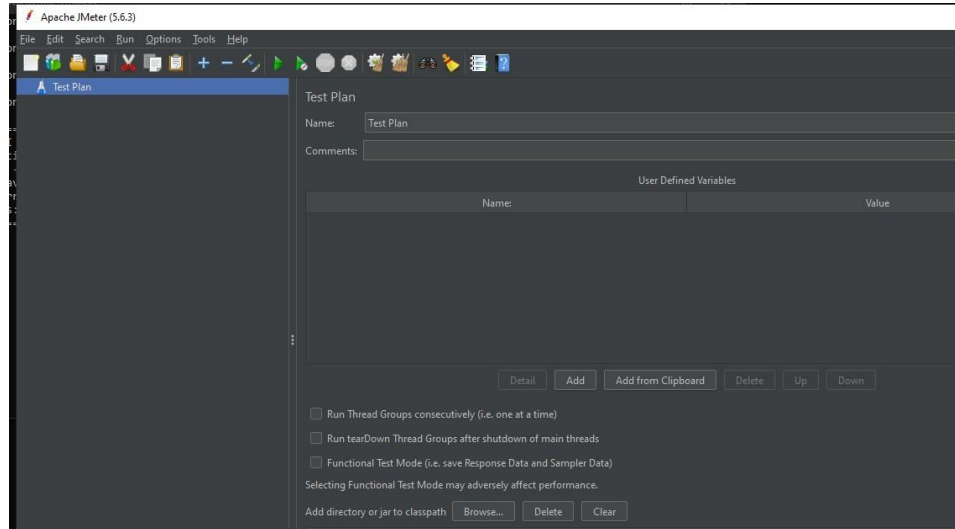
Práctica: Simular múltiples usuarios concurrentes usando herramientas como Apache JMeter para ver cómo responde la base de datos bajo alta carga.

JMeter permite simular un número de usuarios concurrentes interactuando con tu base de datos, lo que te permitirá medir cómo se comporta bajo carga.

Pasos

Instalar Apache JMeter.

Configura un **plan de prueba** que simule la carga en tu base de datos, mediante el uso de consultas SQL simuladas.



Ejecutar y monitorizar los resultados, observando tiempos de respuesta, tasa de éxito y posibles errores.

Investigación: Investigar cómo realizar pruebas de estrés y carga en bases de datos de alto rendimiento.

Con MySQLslap se puede simular diferentes cantidades de tráfico en la base de datos, lo que ayuda a identificar cómo se comporta la base de datos bajo alta carga.

Usando el comando:

```
mysqlslap --user=root --password=tu_contraseña --concurrency=5 --iterations=10 --create-schema=RestauranteDB --query="SELECT * FROM Cliente"
```

```
C:\WINDOWS\system32>mysqlslap --user=root --password=root --concurrency=5 --iterations=10 --create-schema=RestauranteDB
--query="SELECT * FROM Cliente"
mysqlslap: [Warning] Using a password on the command line interface can be insecure.
Benchmark
  Average number of seconds to run all queries: 0.006 seconds
  Minimum number of seconds to run all queries: 0.000 seconds
  Maximum number of seconds to run all queries: 0.016 seconds
  Number of clients running queries: 5
  Average number of queries per client: 1
```

Importancia del Conocimiento: Las pruebas de carga aseguran que el sistema sea capaz de manejar tráfico alto y crecimiento de datos.

3. Optimizar el uso de recursos y gestionar índices.

Práctica: Identificar índices no utilizados y eliminarlos para liberar recursos y mejorar la velocidad de las operaciones de escritura.

Se puede identificar los índices que no están siendo utilizados a través del Query Profiler o haciendo un análisis de las consultas que se ejecutan frecuentemente. Luego se los puede eliminar para reducir la sobrecarga de escritura.

Para mostrar todos los índices de la tabla, y ver cuáles se están utilizando y cuáles no, usamos:

```
SHOW INDEX FROM RestauranteDB.Cliente;
```

	Table	Non_uniq	Key_name	Seq_in_inde	Column_name	Collation	Cardinality	Sub_part	Packed	Nul	Index_ty
▶	cliente	0	PRIMARY	1	ClienteID	A	14	NULL	NULL		BTREE
	cliente	0	Correo	1	Correo	A	14	NULL	NULL		BTREE

Para eliminar los índices no utilizados usamos:

```
DROP INDEX indice_cliente ON RestauranteDB.Cliente;
```

Para crear índices

```
CREATE INDEX idx_cliente_correo ON RestauranteDB.Cliente(Correo);
```

	Table	Non_unique	Key_name	Seq_in_inde	Column_name	Collation	Cardinality	Sub_part	Packed	Nul	Index_type	Comment	Index_comment	Visible	Expression
▶	cliente	0	PRIMARY	1	ClienteID	A	14	NULL	NULL		BTREE			YES	NULL
	cliente	0	Correo	1	Correo	A	14	NULL	NULL		BTREE			YES	NULL

Investigación: Investigar cómo ajustar el número de índices según el tipo de consulta (lectura/escritura).

Cuando hay muchas operaciones de lectura, si la base de datos se usa principalmente para lecturas, crear índices en las columnas que se utilizan frecuentemente en las condiciones de búsqueda como WHERE, JOIN puede mejorar significativamente el rendimiento de las consultas. Pero si hay demasiados índices se pueden ralentizar las operaciones de inserción y actualización.

Por otro lado, si en las bases de datos con muchas escrituras hay un uso excesivo de índices, esto puede disminuir el rendimiento, ya que cada operación de escritura debe actualizar todos los índices. En este caso, es importante tener un número mínimo de índices, solo en las columnas que realmente los necesiten.

En el caso de índices compuestos, si se realiza muchas consultas con múltiples columnas en la cláusula WHERE, los índices compuestos nos benefician porque esto ayuda a que las consultas sean más rápidas y eficientes.

```
CREATE INDEX idx_cliente_name_email ON RestauranteDB.Cliente(Nombre, Correo);
```

Importancia del Conocimiento: La optimización de los recursos asegura un uso eficiente del hardware y mejora la escalabilidad.

7. Git y Control de Versiones

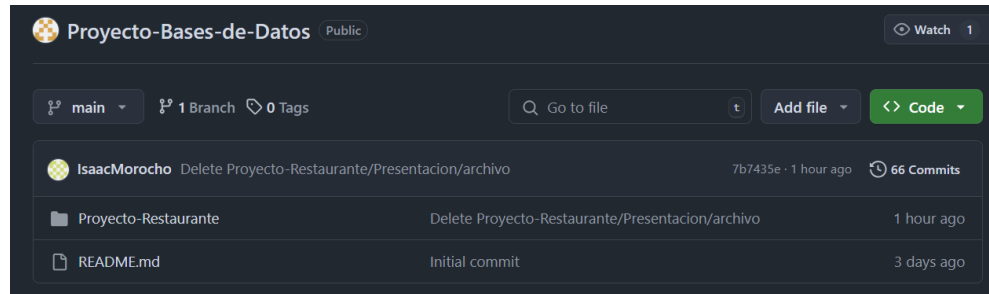
Objetivo: Asegurar que el código relacionado con la base de datos esté versionado y que el equipo pueda colaborar de manera eficiente.

Actividades:

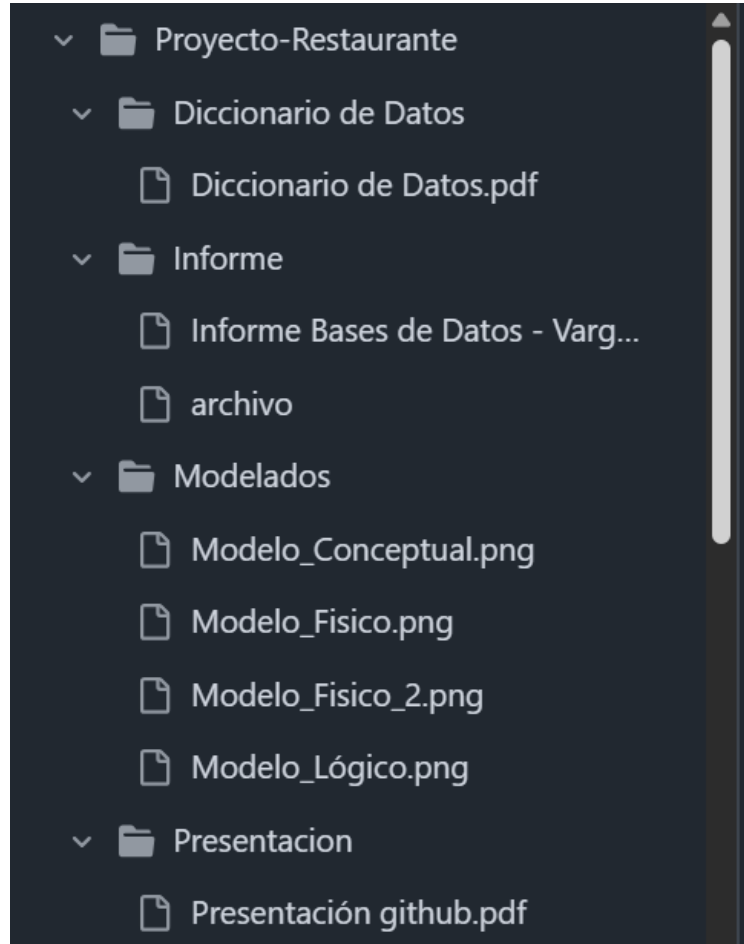
1. Configurar un repositorio de Git para el proyecto.

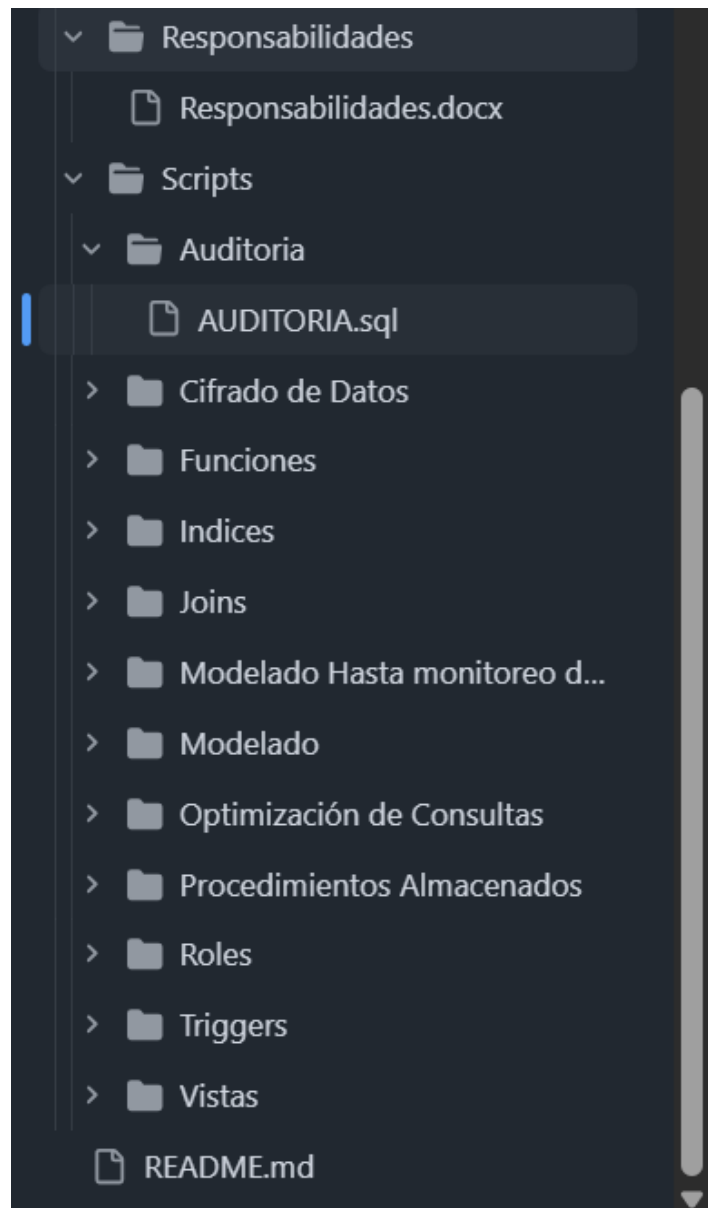
Práctica: Inicializar un repositorio en Git y subir los archivos de definición de la base de datos, scripts de SQL y procedimientos almacenados.

Carpeta Principal en el repositorio de GitHub



Ramas del proyecto y su distribución





Investigación: Investigar buenas prácticas de flujo de trabajo en Git (por ejemplo, uso de ramas, git merge).

Son esenciales para mantener un código organizado, colaborar eficientemente en equipos y garantizar la calidad del software.

Tipos:

- Ramas (branching)
- Estrategias de fusión (merge)
- Commits y mensajes claros
- Pull Requests (PR)
- Resolución de conflictos: Cuando dos ramas modifican el mismo archivo, Git puede generar conflictos.

Recomendaciones en la configuración de un repositorio GitHub

- Evita trabajar directamente en main: Siempre usa ramas para cambios.
- Documenta tu flujo de trabajo: Asegúrate de que todos en el equipo sigan las mismas prácticas.

Importancia del Conocimiento: Git permite la colaboración y el manejo eficiente de cambios en el código, especialmente cuando se trabaja en equipo.

2. Realizar commits frecuentes y con mensajes claros.

Práctica: Hacer commits regularmente, describiendo claramente los cambios realizados en los scripts SQL y la estructura de la base de datos.



Investigación: Investigar cómo utilizar git rebase y git pull para evitar conflictos.

Git Rebase:

Permite reescribir la historia de los commits al actualizar una rama con los cambios de otra.

Ejemplo:

```
git checkout main
```

```
git pull origin main
```

Git Pull:

Por defecto, git pull hace un merge cuando se trae la última versión del repositorio remoto, lo que puede generar conflictos en ramas con historial divergente.

Para evitar esto:

```
git pull --rebase origin main
```

Y para evitar escribir --rebase cada vez, se puede configurar Git globalmente:

```
git config --global pull.rebase true
```

Importancia del Conocimiento: Un flujo de trabajo claro en Git mejora la colaboración y la gestión de versiones.

3. Automatizar pruebas con GitHub Actions.

Práctica: Crear flujos de trabajo de CI/CD que automaticen las pruebas de las consultas SQL y otros scripts relacionados con la base de datos.

```
1  name: CI/CD Flujo
2
3  on:
4    push:
5      branches:
6        - main
7    pull_request:
8      branches:
9        - main
10
11  jobs:
12    test-db:
13      runs-on: ubuntu-latest
14      services:
15        mysql:
16          image: mysql:8
17          env:
18            MYSQL_ROOT_PASSWORD: 123456
19            MYSQL_DATABASE: restaurante
20
21      ports:
22        - 3306:3306
23      options: >-
24        --health-cmd="mysqladmin ping -h 127.0.0.1 -u root --password=123456"
25        --health-interval=10s
26        --health-timeout=5s
27        --health-retries=5
28
29  steps:
```

Use **Control + Shift + m** to toggle the **tab** key moving focus. Alternatively, use **esc** then **tab** to move to the next

Edit Preview

Spaces 2 No wrap

```
15  mysql:
16    image: mysql:8
17    env:
18      MYSQL_ROOT_PASSWORD: 123456
19      MYSQL_DATABASE: restaurante
20
21    ports:
22      - 3306:3306
23    options: >-
24      --health-cmd="mysqladmin ping -h 127.0.0.1 -u root --password=123456"
25      --health-interval=10s
26      --health-timeout=5s
27      --health-retries=5
28
29  steps:
30    - name: Clonar repositorio
31      uses: actions/checkout@v4
32
33    - name: Instalar MySQL Client
34      run: sudo apt-get update && sudo apt-get install -y mysql-client
35
36    - name: Esperar a que MySQL esté listo
37      run: sleep 20 # Ajustar si es necesario
38
39    - name: Crear estructura de base de datos
40      run: |
41        mysql -h 127.0.0.1 -u root -123456 restaurante < Scripts/Vistas/Script-Vistas.sql
42
43
```

Use **Control + Shift + m** to toggle the **tab** key moving focus. Alternatively, use **esc** then **tab** to move to the next

Investigación: Investigar sobre integración continua y cómo aplicarla en bases de datos con GitHub Actions.

Una práctica de desarrollo donde los cambios en el código se integran y prueban automáticamente en un entorno de prueba antes de fusionarse en la rama principal del repositorio.

Para bases de datos, esto implica:

- Verificar que los scripts SQL funcionan correctamente.
- Comprobar la estructura de la base de datos antes de hacer cambios.
- Ejecutar pruebas automatizadas en las consultas SQL.
- Validar migraciones para evitar inconsistencias.

Y para implementar en una base de datos se puede ver lo visto anteriormente.

Importancia del Conocimiento: Las pruebas automáticas aseguran que las bases de datos se mantengan consistentes y funcionales a lo largo del tiempo.

Enlace GitHub

<https://github.com/KevinMunoz15112004/Proyecto-Bases-de-Datos>