



ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS



BASES DE DATOS (TSDS)

ASIGNATURA:

Bases de Datos

PROFESOR:

Ing. Lorena Chulde

FECHA DE ENTREGA:

2025 - 02-05

PERÍODO ACADÉMICO:

2024-B

TALLER

(individual)

TÍTULO

PROYECTO FINAL



Estudiantes

Odaliz Balseca Valencia
Sebastian Chico Quezada

Resumen Ejecutivo

Objetivos

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:

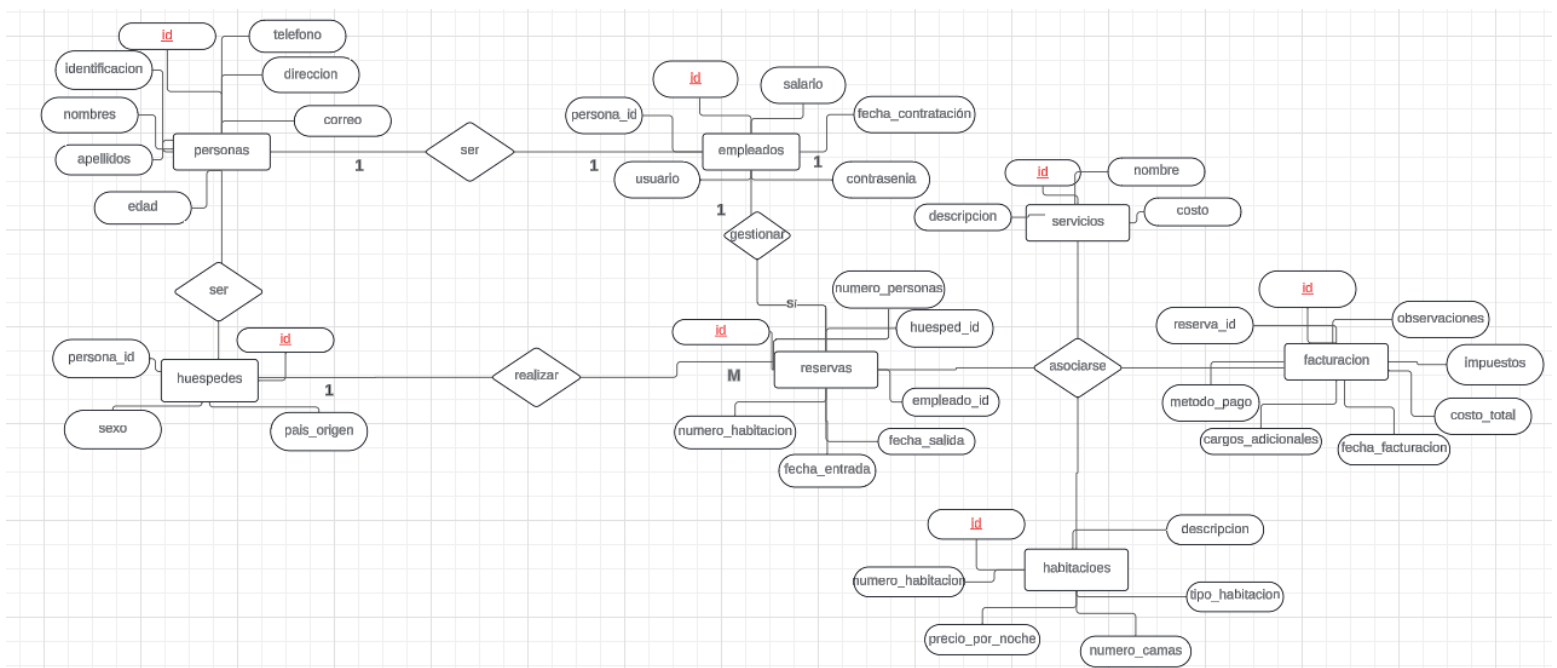
Diseñar el modelo conceptual, lógico y físico.

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

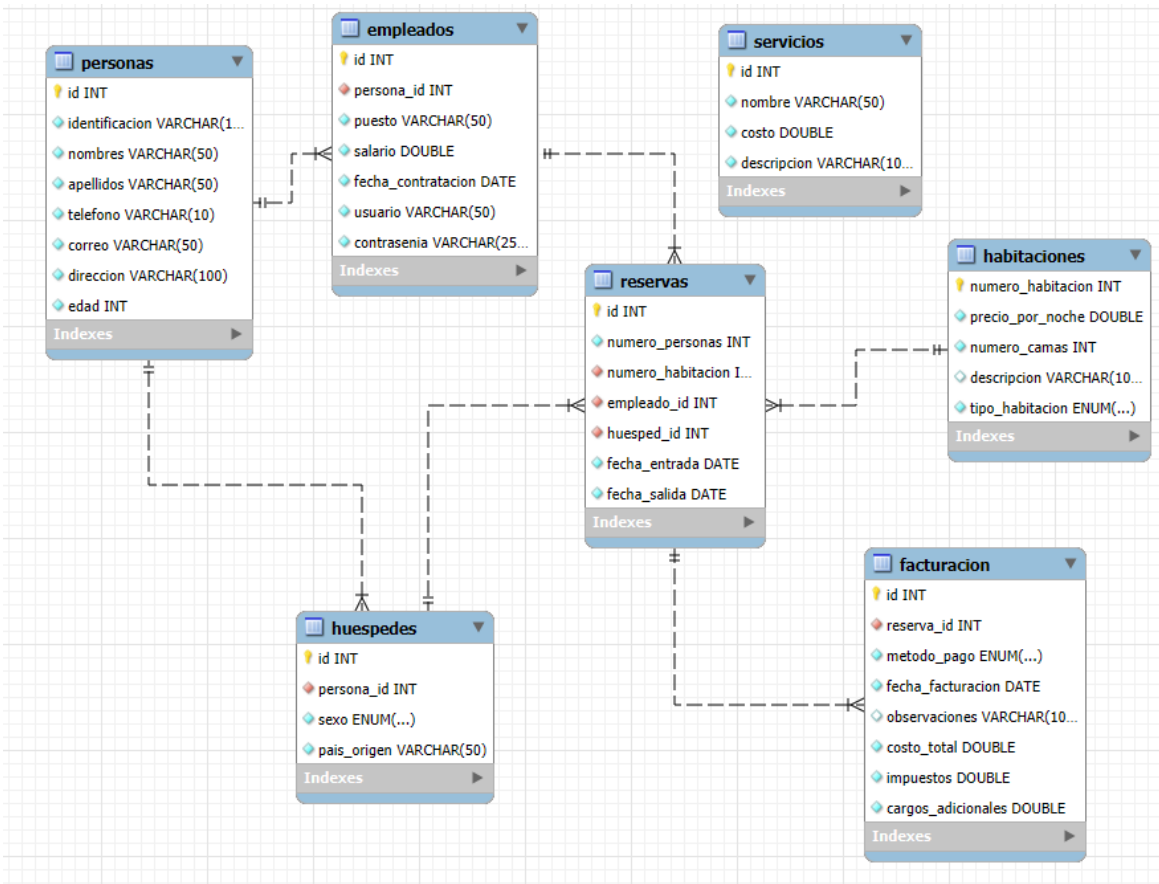
Investigación: Buscar buenas prácticas sobre cómo hacer escalables los modelos de bases de datos para sistemas de reservas en aerolíneas.

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

1. Diseño Conceptual



2. Diseño Lógico



3. Diseño Físico

-- Tabla personas (almacena información general de todas las personas)

```

create table personas (
    id int not null primary key auto_increment,
    identificacion varchar(10) not null unique,
    nombres varchar(50) not null,
    apellidos varchar(50) not null,
    telefono varchar(10) not null unique,
    correo varchar(50) not null unique,
    direccion varchar(100) not null,
    edad int not null check (edad >= 0)
);
    
```

-- Tabla empleados (relacionada con personas)

```
create table empleados (  
    id int not null primary key auto_increment,  
    persona_id int not null,  
    puesto varchar(50) not null,  
    salario double not null,  
    fecha_contratacion date not null,  
    usuario varchar(50) not null unique,  
    contrasenia varchar(255) not null,  
    foreign key (persona_id) references personas(id) on delete cascade  
);
```

-- Tabla huéspedes (relacionada con personas)

```
create table huespedes (  
    id int not null primary key auto_increment,  
    persona_id int not null,  
    sexo enum('Masculino', 'Femenino', 'Otro') not null,  
    pais_origen varchar(50) not null,  
    foreign key (persona_id) references personas(id) on delete cascade  
);
```

-- Tabla habitaciones

```
create table habitaciones (  
    numero_habitacion int not null primary key,  
    precio_por_noche double not null,  
    numero_camras int not null,  
    descripcion varchar(100),  
    tipo_habitacion enum('Suite', 'Junior suite', 'Gran suite', 'Normal', 'Habitación matrimonial') not null  
);
```

-- Tabla reservas (relacionada con huéspedes, empleados y habitaciones)

```
create table reservas (  
    id int not null primary key auto_increment,  
    numero_personas int not null,  
    numero_habitacion int not null,
```

```

    empleado_id int not null,
    huesped_id int not null,
    fecha_entrada date not null,
    fecha_salida date not null,
    foreign key (numero_habitacion) references habitaciones(numero_habitacion) on delete cascade,
    foreign key (empleado_id) references empleados(id) on delete cascade,
    foreign key (huesped_id) references huespedes(id) on delete cascade
);

```

-- Tabla servicios (almacena los servicios disponibles)

```

create table servicios (
    id int not null primary key auto_increment,
    nombre varchar(50) not null,
    costo double not null,
    descripcion varchar(100) not null
);

```

-- Tabla facturación (relacionada con reservas)

```

create table facturacion (
    id int not null primary key auto_increment,
    reserva_id int not null,
    metodo_pago enum('En efectivo', 'Con tarjeta de crédito', 'Con tarjeta de débito') not null,
    fecha_facturacion date not null,
    observaciones varchar(100),
    costo_total double not null,
    impuestos double not null,
    cargos_adicionales double not null,
    foreign key (reserva_id) references reservas(id) on delete cascade
);

```

Inserción de datos:

```

insert into personas (identificacion, nombres, apellidos, telefono, correo, direccion, edad) values
('1102567890', 'Odaliz', 'Balseca', '0987654321', 'odaliz.balseca@gmail.com', 'Quito', 25),

```

```
('1103578910', 'Sebastian', 'Chico', '0987123456', 'sebastian.chico@gmail.com', 'Guayaquil', 30),  
(1104589123', 'María', 'González', '0912345678', 'maria.gonzalez@gmail.com', 'Cuenca', 40),  
(1105698123', 'Carlos', 'Ramírez', '0923456789', 'carlos.ramirez@gmail.com', 'Loja', 45);
```

```
insert into empleados (persona_id, puesto, salario, fecha_contratacion, usuario, contrasenia) values  
(3, 'Recepcionista', 800, '2023-05-01', 'maria.gonzalez', '123456'),  
(4, 'Gerente', 1500, '2022-01-10', 'carlos.ramirez', '789012');
```

```
insert into huespedes (persona_id, sexo, pais_origen) values  
(1, 'Femenino', 'Ecuador'),  
(2, 'Masculino', 'Ecuador'),  
(3, 'Femenino', 'Colombia'),  
(4, 'Masculino', 'Perú');
```

```
insert into habitaciones (numero_habitacion, precio_por_noche, numero_camas, descripcion,  
tipo_habitacion) values  
(101, 50.00, 2, 'Vista al mar', 'Suite'),  
(102, 35.00, 1, 'Cómoda y acogedora', 'Normal'),  
(103, 45.00, 1, 'Amplia y luminosa', 'Habitación matrimonial');
```

```
insert into reservas (numero_personas, numero_habitacion, empleado_id, huesped_id, fecha_entrada,  
fecha_salida) values  
(2, 101, 1, 1, '2025-02-05', '2025-02-10'),  
(1, 102, 2, 2, '2025-02-07', '2025-02-12');
```

```
insert into servicios (nombre, costo, descripcion) values  
( 'Comida incluida', 20.00, 'Desayuno, almuerzo y cena'),  
( 'Limpieza diaria', 10.00, 'Servicio de limpieza todos los días'),  
( 'WiFi gratuito', 0.00, 'Acceso ilimitado a internet');
```

```
insert into facturacion (reserva_id, metodo_pago, fecha_facturacion, observaciones, costo_total,  
impuestos, cargos_adicionales) values  
(1, 'Con tarjeta de crédito', '2025-02-05', 'Sin observaciones', 250.00, 25.00, 10.00),  
(2, 'En efectivo', '2025-02-07', 'Pago en recepción', 175.00, 17.50, 5.00);
```

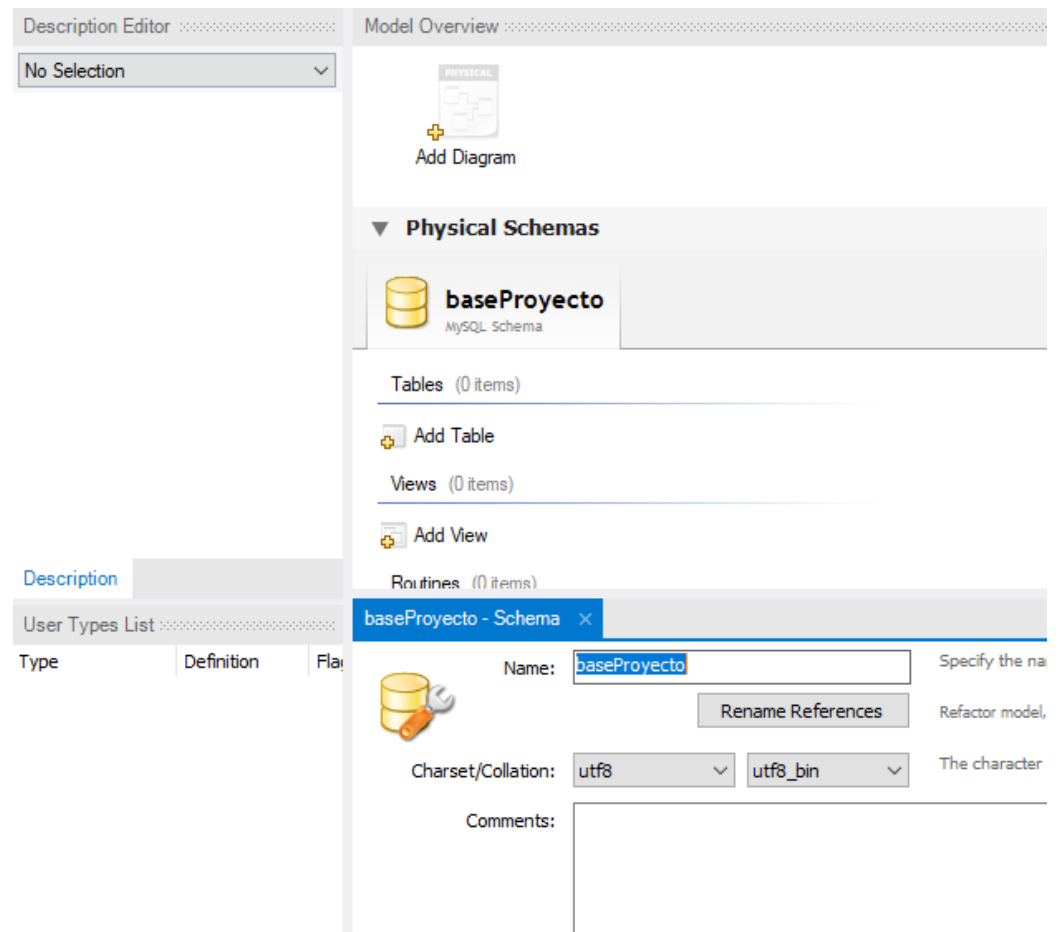

1. Desarrollar un diccionario de datos detallado.

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.

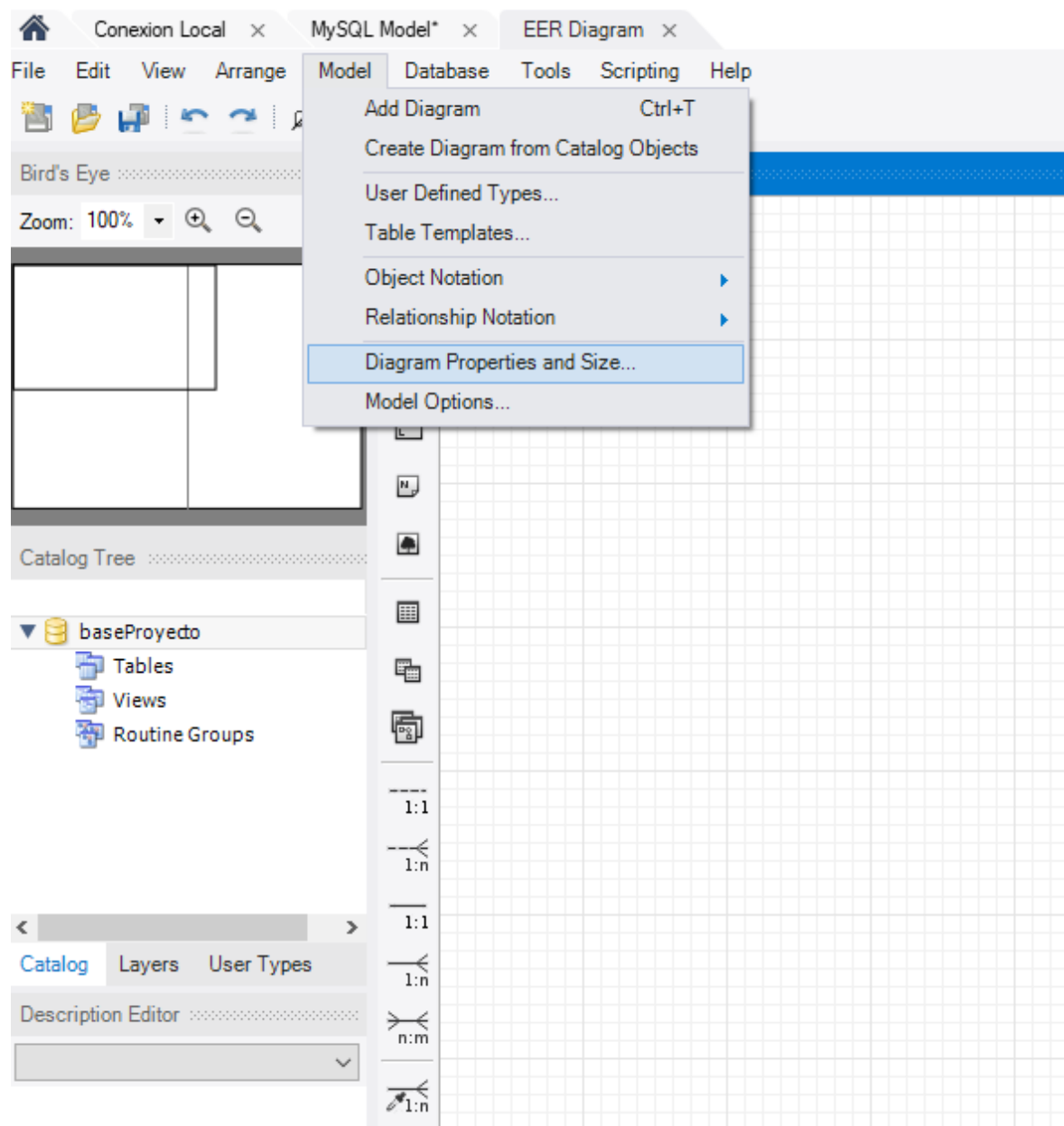
Importancia del Conocimiento: Un diccionario de datos permite mantener la consistencia y facilita la colaboración entre desarrolladores.

Primero creamos una base de datos e este caso el Proyecto Final, donde añadiremos un diagrama y posteriormente generaremos los diccionarios.



Se crea un nuevo modelo de relacional, hacer doble click sobre el nuevo modelo.

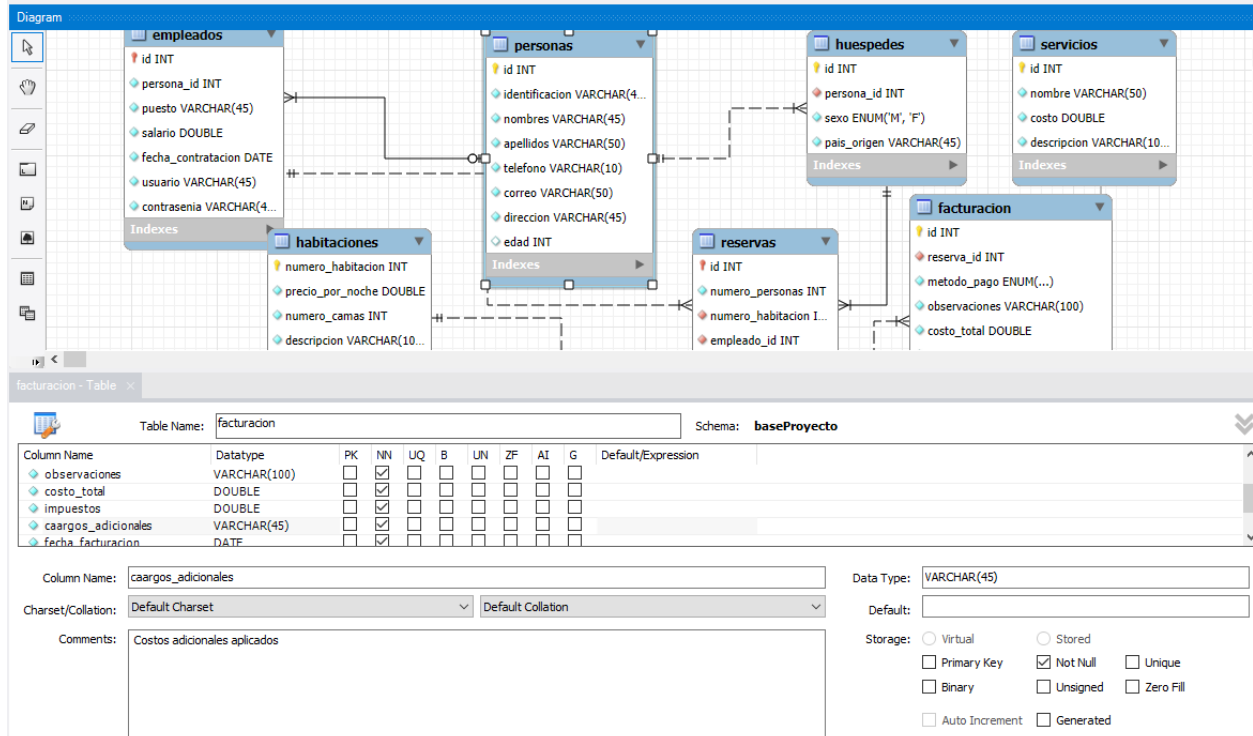
Se visualiza el espacio de trabajo del diseño. Estamos listos para trabajar en el diseño lógico de la base de datos. En el apartado de Diagram Properties and Size podemos modificar como su nombre lo dice el tamaño de la cuadrícula para nuestro modelo.



Para crear las tablas hacemos doble click sobre el elemento tabla.

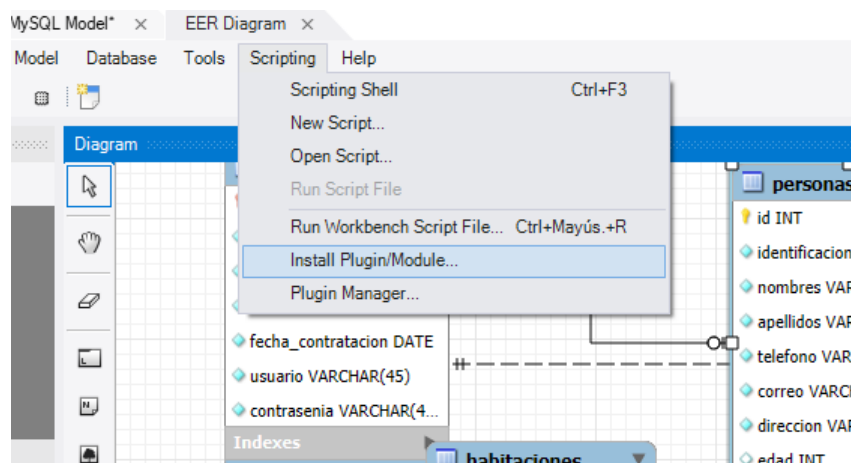
Agregamos los respectivos campos de cada una de las tablas con sus respectivos, nombres y tipos de datos.

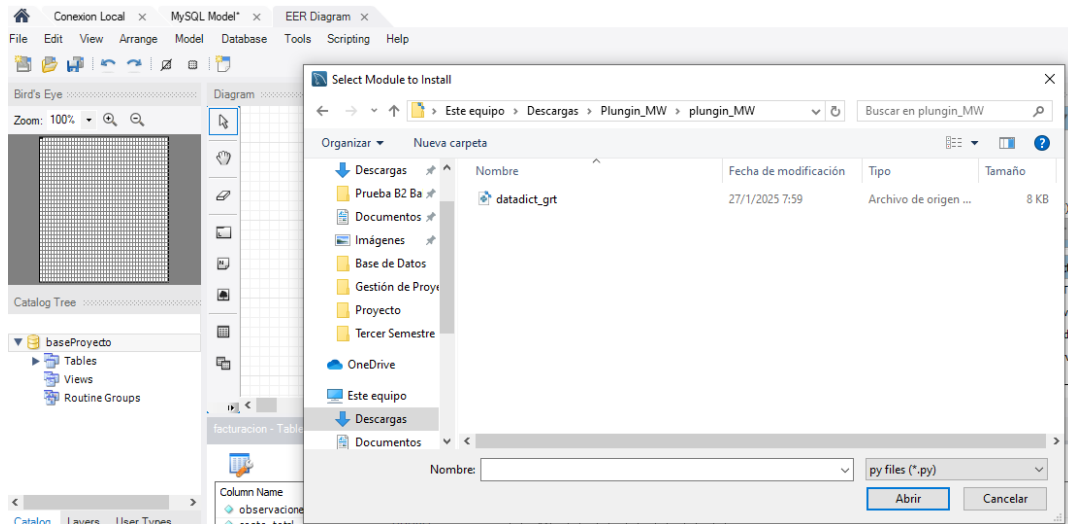
Indicar cuales son los valores foreign key y primary key.



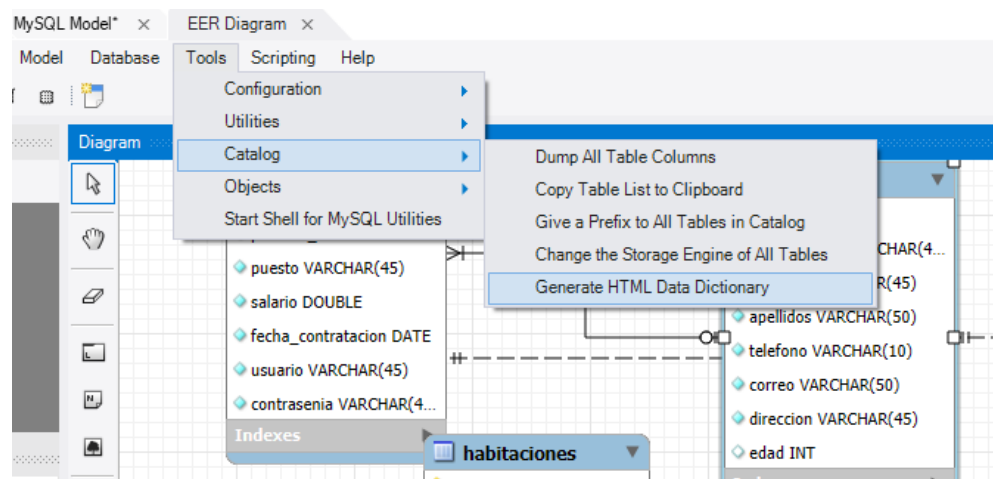
Posteriormente para generar automáticamente los diccionarios seguiremos los siguientes pasos:

1. Descargar el drive del diccionario desde el siguiente sitio web.
2. http://wiki.caballero.co/index.php?title=Generar_diccionario_de_datos_con_MySQL_Workbench
3. Una vez descargado, descomprimirlo.
4. Abrir workbench dirigirse a la pestaña scripting y seleccionar install login module.





Dirigirse en la parte superior del menú en Tools, Catálogo, Generar diccionario de datos con HTML



Agregar el nombre del archivo HTML con el fin de que se genere.

diccionario_proyecto	5/2/2025 0:29	Mi
ProyectoFinal	4/2/2025 19:55	SQ
ProyectoFinal.sql	4/2/2025 19:56	Mv

Diccionarios:

empleados											
Column name	DataType	PK	NN	UQ	BIN	UN	ZF	AI	Default	Comment	
id	INT	✓						✓		Identificador unico del empleado	
persona_id	INT		✓							Relacion con la tabla personas	
puesto	VARCHAR(45)		✓							Cargo o puesto del empleado	
salario	DOUBLE		✓							Salario del empleado	
fecha_contratacion	DATE		✓							Fecha de contratacion	
usuario	VARCHAR(45)		✓							Nombre del usuario del empleado	
contrasenia	VARCHAR(45)		✓							Contrasea de acceso del empleado	

facturacion												
Column name	DataType								PK	NN	UQ	BIN
id	INT								✓			
reserva_id	INT									✓		
metodo_pago	ENUM('Efectivo', 'Tarjeta de Credito', 'Tarjeta de Debito', 'Transferencia')									✓		
observaciones	VARCHAR(100)									✓		
costo_total	DOUBLE									✓		
impuestos	DOUBLE									✓		
caargos_adicionales	VARCHAR(45)									✓		
fecha_facturacion	DATE									✓		

habitaciones												
Column name	DataType								PK	NN	UQ	BIN
numero_habitacion	INT								✓			
precio_por_noche	DOUBLE									✓		
numero_camas	INT									✓		
descripcion	VARCHAR(100)									✓		
tipo_habitacion	ENUM('Suite', 'Junior suite', 'Gran suite', 'Normal', 'Habitacion matrimonial')									✓		

huespedes												
Column name	DataType	PK	NN	UQ	BIN	UN	ZF	AI	Default	Comment		
id	INT	✓						✓		Identificador unico de husped		
persona_id	INT		✓							Relacion con tabla personas		
sexo	ENUM('M', 'F')		✓							Genero del huesped		
pais_origen	VARCHAR(45)		✓							Pais de origen del huesped		

personas												
Column name	DataType	PK	NN	UQ	BIN	UN	ZF	AI	Default	Comment		
id	INT	✓						✓		Identificador unico de la persona.		
identificacion	VARCHAR(45)		✓							Numero de identificaci3n (c3dula o pasaporte).		
nombres	VARCHAR(45)		✓							Nombres de la persona		
apellidos	VARCHAR(50)		✓							Apellidos de la persona		
telefono	VARCHAR(10)		✓							Telefonode la persona		
correo	VARCHAR(50)		✓							Correo de la persona		
direccion	VARCHAR(45)		✓							Direccion de residencia de la persona		
edad	INT									Edad de la persona(Debe ser mayor de edad para gestionar las reservas)		

reservas												
Column name	DataType	PK	NN	UQ	BIN	UN	ZF	AI	Default	Comment		
id	INT	✓						✓		Identificador unico de la reserva		
numero_personas	INT		✓							Numero total de personas en la resrva		
numero_habitacion	INT		✓							Relacion con la tabla habitaciones		
empleado_id	INT		✓							Relacion con la tabla empleados (Quien gestiono la reserva)		
huesped_id	INT		✓							Relacion con la tabla huespedes (Quien hizo la reserva)		
fecha_entrada	DATE		✓							Fecha de ingreso al hotel		
fecha_salida	DATE		✓							Fecha de salida del hotel		

servicios												
Column name	DataType	PK	NN	UQ	BIN	UN	ZF	AI	Default	Comment		
id	INT	✓						✓		Identificador unico del servicio		
nombre	VARCHAR(50)				✓					Nombre del servicio		
costo	DOUBLE				✓					Costo del servicio		
descripcion	VARCHAR(100)				✓					Breve descripcion del servicio		

2. Definir las restricciones de integridad referencial.

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).

Se establece las claves primarias:

```
alter table personas add primary key (id);  
alter table empleados add primary key (id);  
alter table huespedes add primary key (id);  
alter table habitaciones add primary key (numero_habitacion);  
alter table reservas add primary key (id);  
alter table facturacion add primary key (id);  
alter table servicios add primary key (id);
```

Se establece las claves foraneas:

```
alter table empleados add constraint fk_empleado_persona foreign key (persona_id)  
references personas(id);
```

```
alter table huespedes add constraint fk_huesped_persona foreign key (persona_id)  
references personas(id);
```

```
alter table reservas add constraint fk_reserva_huesped foreign key (huesped_id)  
references huespedes(id);
```

```
alter table reservas add constraint fk_reserva_empleado foreign key (empleado_id)  
references empleados(id);
```

```
alter table reservas add constraint fk_reserva_habitacion foreign key  
(numero_habitacion) references habitaciones(numero_habitacion);
```

```
alter table facturacion add constraint fk_factura_reserva foreign key (reserva_id)  
references reservas(id);
```

Reglas de integridad referencial (ON DELETE / ON UPDATE)

Para manejar eliminaciones o cambios, podemos definir estas reglas:

- **ON DELETE CASCADE:** Si se borra un registro padre, se eliminan los hijos.
- **ON DELETE SET NULL:** Si se borra un padre, los hijos quedan con NULL.
- **ON DELETE RESTRICT:** Impide eliminar si hay hijos relacionados.

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

La **integridad referencial** previene errores como:

- **Registros huérfanos:** Evita que haya reservas sin huéspedes o facturas sin reservas.
- **Datos inconsistentes:** Impide que un huésped haga una reserva sin existir en la tabla huéspedes.
- **Errores de eliminación:** Controla qué pasa cuando se borra un registro relacionado.

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

- Evita corrupción de datos.
- Asegura coherencia entre las tablas.
- Facilita la integridad en bases de datos relacionales.

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.

El siguiente código tiene como propósito la creación de usuarios en MySQL y la asignación de permisos específicos sobre la base de datos `gestion_hoteles`, particularmente en la tabla `huespedes` y otras relacionadas.

Los permisos son otorgados a los usuarios mediante la sentencia `GRANT`, asegurando que cada rol tenga los privilegios adecuados sobre las tablas correspondientes:

administrador:

Recibe todos los privilegios (`ALL PRIVILEGES`) sobre la tabla `huespedes`.

Esto le permite realizar cualquier operación sobre esta tabla (`SELECT`, `INSERT`, `UPDATE`, `DELETE`, etc.).

empleado:

Puede realizar operaciones básicas de manipulación de datos (`SELECT`, `INSERT`, `UPDATE`, `DELETE`) en las tablas:

`huespedes`

`reservas`

`facturacion`

Esto le permite gestionar la información de los huéspedes, las reservas y la facturación.

cliente:

Solo tiene permisos de lectura (`SELECT`) en las tablas:

`huespedes`

`reservas`

`facturacion`

Esto le permite consultar información sobre su perfil, sus reservas y facturas, sin modificar datos.

```
create user 'administrador'@'localhost' identified by 'admin1234'; create user
'empleado'@'localhost' identified by 'empleado1234'; create user 'cliente'@'localhost'
identified by 'cliente1234'; /Permisos a los usuarios anteriormente creados/ grant all
privileges on gestion_hoteles.huespedes to 'administrador'@'localhost'; grant select, insert,
update, delete on gestion_hoteles.huespedes to 'empleado'@'localhost'; grant select,
insert, update, delete on gestion_hoteles.reservas to 'empleado'@'localhost'; grant select,
insert, update, delete on gestion_hoteles.facturacion to 'empleado'@'localhost'; grant
select on gestion_hoteles.huespedes to 'cliente'@'localhost'; grant select on
```


gestion_hoteles.reservas to 'cliente'@'localhost'; grant select on
gestion_hoteles.facturacion to 'cliente'@'localhost';

user	host
admin	%
administrador	localhost
chicoandres	localhost
chicosebas	localhost
cliente	localhost
empleado	localhost
mysql.infoschema	localhost
mysql.session	localhost
mysql.sys	localhost

17 • Show grants for 'administrador'@'localhost';

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
Grants for administrador@localhost			
GRANT USAGE ON *.* TO `administrador`@`localhost`			
GRANT ALL PRIVILEGES ON `gestion_hoteles`.`huespedes` TO `administrador`@`localhost`			

17 • Show grants for 'empleado'@'localhost';

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
Grants for empleado@localhost			
GRANT USAGE ON *.* TO `empleado`@`localhost`			
GRANT SELECT, INSERT, UPDATE, DELETE ON `gestion_hoteles`.`facturacion` TO `empleado`@`localhost`			
GRANT SELECT, INSERT, UPDATE, DELETE ON `gestion_hoteles`.`huespedes` TO `empleado`@`localhost`			
GRANT SELECT, INSERT, UPDATE, DELETE ON `gestion_hoteles`.`reservas` TO `empleado`@`localhost`			

17 • Show grants for 'cliente'@'localhost';

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
Grants for cliente@localhost			
GRANT USAGE ON *.* TO `cliente`@`localhost`			
GRANT SELECT ON `gestion_hoteles`.`facturacion` TO `cliente`@`localhost`			
GRANT SELECT ON `gestion_hoteles`.`huespedes` TO `cliente`@`localhost`			
GRANT SELECT ON `gestion_hoteles`.`reservas` TO `cliente`@`localhost`			

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

En entornos de alta disponibilidad, la seguridad de las bases de datos es un aspecto crucial, ya que estos sistemas deben estar operativos de manera continua y manejar grandes volúmenes de información crítica. Para garantizar la protección de los datos sin afectar la disponibilidad, se deben implementar estrategias que equilibren seguridad y rendimiento.

Uno de los enfoques fundamentales es la autenticación y control de acceso. Se recomienda el uso de autenticación multifactor (MFA) para accesos administrativos y la implementación del principio de privilegio mínimo, asegurando que cada usuario tenga solo los permisos estrictamente necesarios. Además, es clave emplear mecanismos de control de acceso basados en roles (RBAC) o atributos (ABAC) para gestionar los permisos de manera granular.

La cifrado de datos es otra medida esencial. Se deben cifrar tanto los datos en tránsito, mediante protocolos como TLS/SSL, como los datos en reposo, utilizando tecnologías como Transparent Data Encryption (TDE) o cifrado a nivel de aplicación. Esto protege la información ante accesos no autorizados, incluso en caso de una filtración.

El monitoreo y auditoría continua permite detectar y responder rápidamente a posibles amenazas. Se deben habilitar registros detallados de accesos y actividades mediante herramientas de auditoría, así como implementar sistemas de detección de intrusos (IDS) y análisis de comportamiento para identificar patrones sospechosos. Complementariamente, el uso de alertas automatizadas y soluciones SIEM (Security Information and Event Management) mejora la capacidad de respuesta ante incidentes.

2. Cifrado de datos sensibles.

Insertar datos con contraseñas hasheadas

Para insertar una contraseña hasheada, puedes usar la función SHA2() de MySQL. Por ejemplo, para hashear la contraseña "miContraseñaSegura123" con SHA-256:

```
INSERT INTO empleados (persona_id, puesto, salario, fecha_contratacion,
usuario, contrasenia) VALUES (1, 'Gerente', 50000.00, '2023-10-01',
'juan_perez', SHA2('miContraseñaSegura123', 256));
```

También anteriormente se insertaron dos registros pero sin encriptar las contraseñas, a continuación se puede visualizar los dos tipos de inserciones.

	id	persona_id	puesto	salario	fecha_contratacion	usuario	contrasenia
▶	1	3	Recepcionista	800	2023-05-01	maria.gonzalez	123456
	2	4	Gerente	1500	2022-01-10	carlos.ramirez	789012
	3	1	Gerente	50000	2023-10-01	juan_perez	8b0bf6345478bde3d8735f58cb5924241649297...
★	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

Entre los más utilizados están AES (Advanced Encryption Standard), que ofrece alta seguridad con claves de 128, 192 y 256 bits, y RSA (Rivest-Shamir-Adleman), empleado principalmente en la encriptación de claves debido a su alto costo computacional. Otros algoritmos como Blowfish y ChaCha20 son opciones eficientes para entornos donde el rendimiento es crítico.

El impacto en el rendimiento depende del tipo de cifrado aplicado. El cifrado en tránsito (TLS/SSL) tiene un impacto leve, mientras que el cifrado en reposo (TDE) puede generar sobrecarga en operaciones de lectura y escritura, especialmente en bases de datos de alto tráfico. El cifrado a nivel de aplicación, aunque más seguro, incrementa la latencia debido a la necesidad de desencriptar datos en cada consulta.

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

Tabla datosOperaciones:

```
create table datosOperaciones( id_datosOperaciones int not null
auto_increment primary key, usuario_operacion varchar(50),
fecha_operacion datetime, operacion_que_se_ejecuto varchar(2000)
default null, operacion_que_reviuy7poerte varchar(2000) default null );
```

1. Trigger para insertar huespedes:

```
DROP TRIGGER IF EXISTS despues_insertar_huespedes;
DELIMITER $$
CREATE TRIGGER despues_insertar_huespedes AFTER INSERT ON
huespedes
FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierte) VALUES (
CURRENT_USER(), NOW(), CONCAT('INSERT INTO huespedes (id,
persona_id, sexo, pais_origen) VALUES (', NEW.id, ', ', NEW.persona_id, ', ',
```

```
NEW.sexo, "", "", NEW.pais_origen, "");', CONCAT('DELETE FROM huespedes
WHERE id=', NEW.id, ');'); END$$ DELIMITER ;
```

```

403 • INSERT INTO huespedes (persona_id, sexo, pais_origen)
404 VALUES
405 (1, 'Masculino', 'Ecuador'),
406 (2, 'Femenino', 'Colombia'),
407 (3, 'Otro', 'Perú');
408 • select * from datosOperaciones;
409

```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
6	root@localhost	2025-02-05 08:21:14	INSERT INTO huespedes (id, persona_id, sexo,...	DELETE FROM huespedes WHERE id=5;
7	root@localhost	2025-02-05 08:21:14	INSERT INTO huespedes (id, persona_id, sexo,...	DELETE FROM huespedes WHERE id=6;
8	root@localhost	2025-02-05 08:21:14	INSERT INTO huespedes (id, persona_id, sexo,...	DELETE FROM huespedes WHERE id=7;
9	root@localhost	2025-02-05 08:22:01	DELETE FROM huespedes WHERE id=1;	INSERT INTO huespedes (id, persona_id, sexo,..

2. Trigger para eliminar huespedes:

```
DROP TRIGGER IF EXISTS despues_eliminacion_huespedes;
```

```
DELIMITER $$
```

```
CREATE TRIGGER despues_eliminacion_huespedes AFTER DELETE ON
huespedes FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierde) VALUES (
CURRENT_USER(), NOW(), CONCAT('DELETE FROM huespedes WHERE
id=', OLD.id, ';'), CONCAT('INSERT INTO huespedes (id, persona_id, sexo,
pais_origen) VALUES (' , OLD.id, ', ', OLD.persona_id, ', ', OLD.sexo, ', ',
OLD.pais_origen, ');'); END$$
```

```
DELIMITER ;
```

```

409 • DELETE FROM huespedes WHERE id = 1;
410 • select * from datosOperaciones;
411

```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
7	root@localhost	2025-02-05 08:21:14	INSERT INTO huespedes (id, persona_id, sexo,...	DELETE FROM huespedes WHERE id=6;
8	root@localhost	2025-02-05 08:21:14	INSERT INTO huespedes (id, persona_id, sexo,...	DELETE FROM huespedes WHERE id=7;
9	root@localhost	2025-02-05 08:22:01	DELETE FROM huespedes WHERE id=1;	INSERT INTO huespedes (id, persona_id, sexo,..

3. Trigger para actualizar huespedes:

```
CREATE TRIGGER despues_actualizacion_huespedes AFTER UPDATE ON
huespedes FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
```

```

operacion_que_se_ejecuto, operacion_que_revierde) VALUES (
CURRENT_USER(), NOW(), CONCAT( 'UPDATE huespedes SET persona_id
= ', NEW.persona_id, ', sexo = ', NEW.sexo, ', pais_origen = ',
NEW.pais_origen, ' WHERE id = ', NEW.id, ';' ), CONCAT( 'UPDATE
huespedes SET persona_id = ', OLD.persona_id, ', sexo = ', OLD.sexo, ',
pais_origen = ', OLD.pais_origen, ' WHERE id = ', OLD.id, ';' ) ); END$$

DELIMITER ;

```

```

411 • UPDATE huespedes
412 SET sexo = 'Femenino', pais_origen = 'Argentina'
413 WHERE id = 3;
414
415 • select * from datosOperaciones;
416

```

	id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
	8	root@localhost	2025-02-05 08:21:14	INSERT INTO huespedes (id, persona_id, sexo,...	DELETE FROM huespedes WHERE id=7;
	9	root@localhost	2025-02-05 08:22:01	DELETE FROM huespedes WHERE id=1;	INSERT INTO huespedes (id, persona_id, sexo,...
	10	root@localhost	2025-02-05 08:22:46	UPDATE huespedes SET persona_id = 3, sexo ...	UPDATE huespedes SET persona_id = 3, sexo ...
	NULL	NULL	NULL	NULL	NULL

4. Trigger para insertar habitaciones:

```

DROP TRIGGER IF EXISTS despues_insertar_habitaciones; DELIMITER $$

```

```

CREATE TRIGGER despues_insertar_habitaciones

```

```

AFTER INSERT ON habitaciones FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierde) VALUES (
CURRENT_USER(), NOW(), CONCAT( 'INSERT INTO habitaciones
(numero_habitacion, precio_por_noche, numero_camras, descripcion,
tipo_habitacion) VALUES (', NEW.numero_habitacion, ', ',
NEW.precio_por_noche, ', ', NEW.numero_camras, ', ', NEW.descripcion, ', ',
NEW.tipo_habitacion, ');' ), CONCAT('DELETE FROM habitaciones WHERE
numero_habitacion=', NEW.numero_habitacion, ');' ) ); END$$

```

```

DELIMITER ;

```

```

416 • INSERT INTO habitaciones (numero_habitacion, precio_por_noche, numero_camas, descripcion, tipo_habitacion)
417 VALUES
418 (106, 100.00, 2, 'Habitación con vista al mar', 'Normal'),
419 (107, 150.00, 1, 'Habitación individual', 'Habitación matrimonial'),
420 (108, 300.00, 3, 'Suite con jacuzzi', 'Suite');
421
422 • select * from datosOperaciones;
423

```

	id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
	11	root@localhost	2025-02-05 08:23:52	INSERT INTO habitaciones (numero_habitacion,...	DELETE FROM habitaciones WHERE numero_ha..
	12	root@localhost	2025-02-05 08:23:52	INSERT INTO habitaciones (numero_habitacion,...	DELETE FROM habitaciones WHERE numero_ha..
	13	root@localhost	2025-02-05 08:23:52	INSERT INTO habitaciones (numero_habitacion,...	DELETE FROM habitaciones WHERE numero_ha..
*	NULL	NULL	NULL	NULL	NULL

5. Trigger para eliminar habitaciones:

DROP TRIGGER IF EXISTS despues_eliminacion_habitaciones;

DELIMITER \$\$

```

CREATE TRIGGER despues_eliminacion_habitaciones AFTER DELETE ON
habitaciones FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierde) VALUES (
CURRENT_USER(), NOW(), CONCAT('DELETE FROM habitaciones WHERE
numero_habitacion=', OLD.numero_habitacion, ','), CONCAT('INSERT INTO
habitaciones (numero_habitacion, precio_por_noche, numero_camas,
descripcion, tipo_habitacion) VALUES (', OLD.numero_habitacion, ', ',
OLD.precio_por_noche, ', ', OLD.numero_camas, ', ', OLD.descripcion, ', ',
OLD.tipo_habitacion, ');' )); END$$

```

DELIMITER ;

```

422 • DELETE FROM habitaciones WHERE numero_habitacion = 101;
423
424 • select * from datosOperaciones;
425

```

	id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
	12	root@localhost	2025-02-05 08:23:52	INSERT INTO habitaciones (numero_habitacion,...	DELETE FROM habitaciones WHERE numero_ha..
	13	root@localhost	2025-02-05 08:23:52	INSERT INTO habitaciones (numero_habitacion,...	DELETE FROM habitaciones WHERE numero_ha..
	14	root@localhost	2025-02-05 08:24:31	DELETE FROM habitaciones WHERE numero_ha...	INSERT INTO habitaciones (numero_habitacion...
*	NULL	NULL	NULL	NULL	NULL

6. Trigger para actualizar habitaciones:

DROP TRIGGER IF EXISTS despues_actualizacion_habitaciones;
DELIMITER \$\$

```

CREATE TRIGGER despues_actualizacion_habitaciones AFTER UPDATE
ON habitaciones FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierde) VALUES (
CURRENT_USER(), NOW(), CONCAT('UPDATE habitaciones SET ', OLD.descripcion, ' = ', NEW.descripcion, ';'), CONCAT('INSERT INTO
habitaciones (numero_habitacion, precio_por_noche, numero_camas, descripcion, tipo_habitacion) VALUES (', OLD.numero_habitacion, ', ',
OLD.precio_por_noche, ', ', OLD.numero_camas, ', ', OLD.descripcion, ', ', OLD.tipo_habitacion, ');' )); END$$

```

```

operacion_que_se_ejecuto,    operacion_que_revierde)    VALUES    (
CURRENT_USER(), NOW(), CONCAT( 'UPDATE habitaciones SET
numero_habitacion = ', NEW.numero_habitacion, ', precio_por_noche = ',
NEW.precio_por_noche, ', numero_camas = ', NEW.numero_camas, ',
descripcion = ', NEW.descripcion, ', ', ', tipo_habitacion = ',
NEW.tipo_habitacion, ', ', ' WHERE numero_habitacion = ',
OLD.numero_habitacion, ', ' ), CONCAT( 'UPDATE habitaciones SET
numero_habitacion = ', OLD.numero_habitacion, ', precio_por_noche = ',
OLD.precio_por_noche, ', numero_camas = ', OLD.numero_camas, ',
descripcion = ', OLD.descripcion, ', ', ', tipo_habitacion = ',
OLD.tipo_habitacion, ', ', ' WHERE numero_habitacion = ',
NEW.numero_habitacion, ', ' ) ); END$$

```

DELIMITER ;

424	•	UPDATE habitaciones
425		SET precio_por_noche = 120.00, descripcion = 'Habitación con vista al jardín'
426		WHERE numero_habitacion = 102;
427	•	select * from datosOperaciones;
428		

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
13	root@localhost	2025-02-05 08:23:52	INSERT INTO habitaciones (numero_habitacion,...	DELETE FROM habitaciones WHERE numero_ha...
14	root@localhost	2025-02-05 08:24:31	DELETE FROM habitaciones WHERE numero_ha...	INSERT INTO habitaciones (numero_habitacion,...
15	root@localhost	2025-02-05 08:24:59	UPDATE habitaciones SET numero_habitacion =...	UPDATE habitaciones SET numero_habitacion =...
* NULL	NULL	NULL	NULL	NULL

7. Trigger para insertar servicios:

DROP TRIGGER IF EXISTS despues_insertar_servicios; DELIMITER \$\$

```

CREATE TRIGGER despues_insertar_servicios AFTER INSERT ON servicios
FOR      EACH      ROW      BEGIN      INSERT      INTO
datosOperaciones(usuario_operacion,      fecha_operacion,
operacion_que_se_ejecuto,    operacion_que_revierde)    VALUES    (
CURRENT_USER(), NOW(), CONCAT('INSERT INTO servicios(id, nombre,
costo, descripcion) VALUES (', NEW.id, ', ', NEW.nombre, ', ', NEW.costo, ', ',
NEW.descripcion, ');'), CONCAT('DELETE FROM servicios WHERE id=',
NEW.id, ');') ); END$$

```

DELIMITER ;

```

430 • INSERT INTO servicios (nombre, costo, descripcion)
431 VALUES
432 ('Desayuno buffet', 15.00, 'Desayuno con variedad de opciones'),
433 ('Lavandería', 20.00, 'Servicio de lavandería express'),
434 ('Spa', 50.00, 'Masajes relajantes y tratamientos');
435 • select * from datosOperaciones;

```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
16	root@localhost	2025-02-05 08:25:52	INSERT INTO servicios(id, nombre, costo, descr...	DELETE FROM servicios WHERE id=4;
17	root@localhost	2025-02-05 08:25:52	INSERT INTO servicios(id, nombre, costo, descr...	DELETE FROM servicios WHERE id=5;
18	root@localhost	2025-02-05 08:25:52	INSERT INTO servicios(id, nombre, costo, descr...	DELETE FROM servicios WHERE id=6;
•	NULL	NULL	NULL	NULL

8. Trigger para eliminar servicios:

DROP TRIGGER IF EXISTS despues_eliminacion_servicios; DELIMITER \$\$

```

CREATE TRIGGER despues_eliminacion_servicios AFTER DELETE ON
servicios FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierde) VALUES (
CURRENT_USER(), NOW(), CONCAT('DELETE FROM servicios WHERE id=',
OLD.id, ','), CONCAT('INSERT INTO servicios(id, nombre, costo, descripcion)
VALUES (, OLD.id, ', ', OLD.nombre, ', ', OLD.costo, ', ', OLD.descripcion, ');')
); END$$

```

DELIMITER ;

```

437 • DELETE FROM servicios WHERE id = 1;
438 • select * from datosOperaciones;

```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
17	root@localhost	2025-02-05 08:25:52	INSERT INTO servicios(id, nombre, costo, descr...	DELETE FROM servicios WHERE id=5;
18	root@localhost	2025-02-05 08:25:52	INSERT INTO servicios(id, nombre, costo, descr...	DELETE FROM servicios WHERE id=6;
19	root@localhost	2025-02-05 08:26:39	DELETE FROM servicios WHERE id=1;	INSERT INTO servicios(id, nombre, costo, descr...
•	NULL	NULL	NULL	NULL

9. Trigger para actualizar servicios:

DROP TRIGGER IF EXISTS despues_actualizacion_servicios; DELIMITER \$\$

```

CREATE TRIGGER despues_actualizacion_servicios AFTER UPDATE ON
servicios FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierde) VALUES (
CURRENT_USER(), NOW(), CONCAT('UPDATE servicios SET nombre = ',
NEW.nombre, ', costo = ', NEW.costo, ', descripcion = ', NEW.descripcion, '
WHERE id = ', NEW.id, ','), CONCAT('UPDATE servicios SET nombre = ',

```



```
OLD.nombre, '', costo = ', OLD.costo, ', descripcion = '', OLD.descripcion, ''
WHERE id = ', OLD.id, ''); END$$
```

```
DELIMITER ;
```

```

439 • UPDATE servicios
440     SET costo = 25.00, descripcion = 'Servicio de lavandería premium'
441     WHERE id = 2;
442 • select * from datosOperaciones;

```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
18	root@localhost	2025-02-05 08:25:52	INSERT INTO servicios(id, nombre, costo, descr...	DELETE FROM servicios WHERE id=6;
19	root@localhost	2025-02-05 08:26:39	DELETE FROM servicios WHERE id=1;	INSERT INTO servicios(id, nombre, costo, descr..
20	root@localhost	2025-02-05 08:27:14	UPDATE servicios SET nombre = 'Limpieza diaria...	UPDATE servicios SET nombre = 'Limpieza diaria.
NULL	NULL	NULL	NULL	NULL

10. Trigger para insertar empleados:

```
DROP TRIGGER IF EXISTS despues_insertar_empleados; DELIMITER $$
```

```
CREATE TRIGGER despues_insertar_empleados AFTER INSERT ON
empleados FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierde) VALUES (
CURRENT_USER(), NOW(), CONCAT('INSERT INTO empleados (id,
persona_id, puesto, salario, fecha_contratacion, usuario, contrasenia)
VALUES (, NEW.id, ', NEW.persona_id, ', ', NEW.puesto, ', ', NEW.salario, ',
', NEW.fecha_contratacion, ', ', NEW.usuario, ', ', NEW.contrasenia, ');'),
CONCAT('DELETE FROM empleados WHERE id=', NEW.id, ')); END$$
```

```
DELIMITER ;
```

```

390 • INSERT INTO empleados (persona_id, puesto, salario, fecha_contratacion, usuario, contrasenia)
391     VALUES
392     (1, 'Recepcionista', 1200.00, '2023-01-15', 'juan.perez', SHA2('password123', 256)),
393     (2, 'Gerente', 2500.00, '2022-05-10', 'maria.gomez', SHA2('securepass', 256)),
394     (3, 'Limpieza', 800.00, '2023-03-20', 'carlos.lopez', SHA2('clave456', 256));
395
396 • select * from datosOperaciones;
397

```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
1	root@localhost	2025-02-05 08:17:29	INSERT INTO empleados (id, persona_id, puest...	DELETE FROM empleados WHERE id=6;
2	root@localhost	2025-02-05 08:17:29	INSERT INTO empleados (id, persona_id, puest...	DELETE FROM empleados WHERE id=7;
3	root@localhost	2025-02-05 08:17:29	INSERT INTO empleados (id, persona_id, puest...	DELETE FROM empleados WHERE id=8;
NULL	NULL	NULL	NULL	NULL

11. Trigger para eliminar empleados:

```
CREATE TRIGGER despues_eliminacion_empleados AFTER DELETE ON
empleados FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierte) VALUES (
CURRENT_USER(), NOW(), CONCAT('DELETE FROM empleados WHERE
id=', OLD.id, ','), CONCAT('INSERT INTO empleados (id, persona_id, puesto,
salario, fecha_contratacion, usuario, contrasenia) VALUES (' , OLD.id, ' , ' ,
OLD.persona_id, ' , ' , OLD.puesto, ' , ' , OLD.salario, ' , ' ,
OLD.fecha_contratacion, ' , ' , OLD.usuario, ' , ' , OLD.contrasenia, ''));' ) );
END$$

DELIMITER ;
```

	id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
1		root@localhost	2025-02-05 08:17:29	INSERT INTO empleados (id, persona_id, puest...	DELETE FROM empleados WHERE id=6;
2		root@localhost	2025-02-05 08:17:29	INSERT INTO empleados (id, persona_id, puest...	DELETE FROM empleados WHERE id=7;
3		root@localhost	2025-02-05 08:17:29	INSERT INTO empleados (id, persona_id, puest...	DELETE FROM empleados WHERE id=8;
4		root@localhost	2025-02-05 08:19:33	DELETE FROM empleados WHERE id=1;	INSERT INTO empleados (id, persona_id, puest...
	NULL	NULL	NULL	NULL	NULL

```
DROP TRIGGER IF EXISTS despues_actualizacion_empleados; DELIMITER
$$
```

DELIMITER ;

```

398 • UPDATE empleados
399 SET puesto = 'Supervisor', salario = 1500.00, contrasenia = SHA2('newpassword', 256)
400 WHERE id = 2;
401
402 • select * from datosOperaciones;
403
404

```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
3	root@localhost	2025-02-05 08:17:29	INSERT INTO empleados (id, persona_id, puest...	DELETE FROM empleados WHERE id=8;
4	root@localhost	2025-02-05 08:19:33	DELETE FROM empleados WHERE id=1;	INSERT INTO empleados (id, persona_id, puest..
5	root@localhost	2025-02-05 08:20:18	UPDATE empleados SET persona_id = 4, puesto...	UPDATE empleados SET persona_id = 4, puesto..
6	NULL	NULL	NULL	NULL

13. Trigger para insertar reservas:

```

DROP TRIGGER IF EXISTS despues_insertar_reservas; CREATE TRIGGER
despues_insertar_reservas AFTER INSERT ON reservas FOR EACH ROW
INSERT INTO datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierde) VALUES (
CURRENT_USER(), NOW(), CONCAT("INSERT INTO reservas (id,
numero_personas, numero_habitacion, empleado_id, huesped_id,
fecha_entrada, fecha_salida) VALUES (", NEW.id, ", ",
NEW.numero_personas, ", ", NEW.numero_habitacion, ", ",
NEW.empleado_id, ", ", NEW.huesped_id, ", ", NEW.fecha_entrada, ", ",
NEW.fecha_salida, ");"), CONCAT("DELETE FROM reservas WHERE id=",
NEW.id, ";"));

```

```

445 ✖ INSERT INTO reservas (numero_personas, numero_habitacion, empleado_id, huesped_id, fecha_entrada, fecha_salida,
446 VALUES
447 (2, 108, 2, 4, '2023-10-01', '2023-10-05'),
448 (1, 102, 3, 2, '2023-11-15', '2023-11-20'),
449 (2, 103, 6, 3, '2023-12-10', '2023-12-15');
450 • select * from datosOperaciones;

```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
21	root@localhost	2025-02-05 08:31:35	INSERT INTO reservas (id, numero_personas, n...	DELETE FROM reservas WHERE id=9;
22	root@localhost	2025-02-05 08:31:35	INSERT INTO reservas (id, numero_personas, n...	DELETE FROM reservas WHERE id=10;
23	root@localhost	2025-02-05 08:31:35	INSERT INTO reservas (id, numero_personas, n...	DELETE FROM reservas WHERE id=11;
24	NULL	NULL	NULL	NULL

14. Trigger para eliminar reservas:

```

DROP TRIGGER IF EXISTS despues_eliminacion_reservas; CREATE
TRIGGER despues_eliminacion_reservas AFTER DELETE ON reservas FOR
EACH ROW INSERT INTO datosOperaciones(usuario_operacion,
fecha_operacion, operacion_que_se_ejecuto, operacion_que_revierde)
VALUES ( CURRENT_USER(), NOW(), CONCAT("DELETE FROM reservas

```

```
WHERE id=", OLD.id, ";"), CONCAT("INSERT INTO reservas (id,
numero_personas, numero_habitacion, empleado_id, huesped_id,
fecha_entrada, fecha_salida) VALUES (", OLD.id, ", ",
OLD.numero_personas, ", ", OLD.numero_habitacion, ", ", OLD.empleado_id,
", ", OLD.huesped_id, ", ", OLD.fecha_entrada, ", ", OLD.fecha_salida, ");") );
```

```
452 • DELETE FROM reservas WHERE id = 2;
453 • select * from datosOperaciones;
```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
22	root@localhost	2025-02-05 08:31:35	INSERT INTO reservas (id, numero_personas, n...	DELETE FROM reservas WHERE id=10;
23	root@localhost	2025-02-05 08:31:35	INSERT INTO reservas (id, numero_personas, n...	DELETE FROM reservas WHERE id=11;
24	root@localhost	2025-02-05 08:33:18	DELETE FROM reservas WHERE id=2;	INSERT INTO reservas (id, numero_personas, n...
NULL	NULL	NULL	NULL	NULL

15. Trigger para actualizar reservas:

```
DROP TRIGGER IF EXISTS despues_actualizacion_reservas; CREATE
TRIGGER despues_actualizacion_reservas AFTER UPDATE ON reservas
FOR EACH ROW INSERT INTO datosOperaciones(usuario_operacion,
fecha_operacion, operacion_que_se_ejecuto, operacion_que_revierde)
VALUES ( CURRENT_USER(), NOW(), CONCAT("UPDATE reservas SET
numero_personas=", NEW.numero_personas, ", numero_habitacion=",
NEW.numero_habitacion, ", empleado_id=", NEW.empleado_id, ",
huesped_id=", NEW.huesped_id, ", fecha_entrada=", NEW.fecha_entrada, ",
fecha_salida=", NEW.fecha_salida, " WHERE id=", NEW.id, ";"),
CONCAT("UPDATE reservas SET numero_personas=",
OLD.numero_personas, ", numero_habitacion=", OLD.numero_habitacion, ",
empleado_id=", OLD.empleado_id, ", huesped_id=", OLD.huesped_id, ",
fecha_entrada=", OLD.fecha_entrada, ", fecha_salida=", OLD.fecha_salida, "
WHERE id=", OLD.id, ";") );
```

```
454 • UPDATE reservas
455 SET numero_personas = 4, fecha_salida = '2023-12-20'
456 WHERE id = 9;
457 • select * from datosOperaciones;
```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
23	root@localhost	2025-02-05 08:31:35	INSERT INTO reservas (id, numero_personas, n...	DELETE FROM reservas WHERE id=11;
24	root@localhost	2025-02-05 08:33:18	DELETE FROM reservas WHERE id=2;	INSERT INTO reservas (id, numero_personas, n...
25	root@localhost	2025-02-05 08:34:17	UPDATE reservas SET numero_personas=4, nu...	UPDATE reservas SET numero_personas=2, nu...
NULL	NULL	NULL	NULL	NULL

16. Trigger para insertar facturacioens:

```
DROP TRIGGER IF EXISTS despues_insertar_facturacion; DELIMITER $$
CREATE TRIGGER despues_insertar_facturacion AFTER INSERT ON
facturacion FOR EACH ROW BEGIN INSERT INTO
```

```

datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierde) VALUES (
CURRENT_USER(), NOW(), CONCAT("INSERT INTO facturacion (id,
reserva_id, metodo_pago, fecha_facturacion, observaciones, costo_total,
impuestos, cargos_adicionales) VALUES (", NEW.id, ", ", NEW.reserva_id, "
", NEW.metodo_pago, ", ", NEW.fecha_facturacion, ", ",
NEW.observaciones, ", ", NEW.costo_total, ", ", NEW.impuestos, ", ",
NEW.cargos_adicionales, ");"), CONCAT("DELETE FROM facturacion
WHERE id=", NEW.id, ";") ); END $$ DELIMITER ;

```

```

458 • INSERT INTO facturacion (reserva_id, metodo_pago, fecha_facturacion, observaciones, costo_total, impuestos, cargo
459 VALUES
460 (9, 'En efectivo', '2023-10-05', 'Pago completo', 500.00, 50.00, 0.00),
461 (10, 'Con tarjeta de crédito', '2023-11-20', 'Pago con tarjeta', 750.00, 75.00, 10.00),
462 (11, 'Con tarjeta de débito', '2023-12-15', 'Pago con tarjeta', 1200.00, 120.00, 0.00);
463 • select * from datosOperaciones;

```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
26	root@localhost	2025-02-05 08:35:40	INSERT INTO facturacion (id, reserva_id, meto...	DELETE FROM facturacion WHERE id=6;
27	root@localhost	2025-02-05 08:35:40	INSERT INTO facturacion (id, reserva_id, meto...	DELETE FROM facturacion WHERE id=7;
28	root@localhost	2025-02-05 08:35:40	INSERT INTO facturacion (id, reserva_id, meto...	DELETE FROM facturacion WHERE id=8;
NULL	NULL	NULL	NULL	NULL

17. Trigger para eliminar facturacioens:

```

DROP TRIGGER IF EXISTS despues_eliminacion_facturacion; DELIMITER
$$ CREATE TRIGGER despues_eliminacion_facturacion AFTER DELETE
ON facturacion FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierde) VALUES (
CURRENT_USER(), NOW(), CONCAT("DELETE FROM facturacion WHERE
id=", OLD.id, ";"), CONCAT("INSERT INTO facturacion (id, reserva_id,
metodo_pago, fecha_facturacion, observaciones, costo_total, impuestos,
cargos_adicionales) VALUES (", OLD.id, ", ", OLD.reserva_id, ", ",
OLD.metodo_pago, ", ", OLD.fecha_facturacion, ", ", OLD.observaciones, "
", OLD.costo_total, ", ", OLD.impuestos, ", ", OLD.cargos_adicionales, ");") );
END $$ DELIMITER ;

```

```

464 • DELETE FROM facturacion WHERE id = 8;
465 • select * from datosOperaciones;

```

id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierde
27	root@localhost	2025-02-05 08:35:40	INSERT INTO facturacion (id, reserva_id, meto...	DELETE FROM facturacion WHERE id=7;
28	root@localhost	2025-02-05 08:35:40	INSERT INTO facturacion (id, reserva_id, meto...	DELETE FROM facturacion WHERE id=8;
29	root@localhost	2025-02-05 08:37:04	DELETE FROM facturacion WHERE id=8;	INSERT INTO facturacion (id, reserva_id, meto...
NULL	NULL	NULL	NULL	NULL

18. Trigger para actualizar facturacioens:

```

DROP TRIGGER IF EXISTS despues_actualizacion_facturacion; DELIMITER
$$ CREATE TRIGGER despues_actualizacion_facturacion AFTER UPDATE

```

```

ON facturacion FOR EACH ROW BEGIN INSERT INTO
datosOperaciones(usuario_operacion, fecha_operacion,
operacion_que_se_ejecuto, operacion_que_revierte) VALUES (
CURRENT_USER(), NOW(), CONCAT("UPDATE facturacion SET
reserva_id=", NEW.reserva_id, ", metodo_pago=", NEW.metodo_pago, ",
fecha_facturacion=", NEW.fecha_facturacion, ", observaciones=",
NEW.observaciones, ", costo_total=", NEW.costo_total, ", impuestos=",
NEW.impuestos, ", cargos_adicionales=", NEW.cargos_adicionales, "
WHERE id=", NEW.id, ";"), CONCAT("UPDATE facturacion SET reserva_id=",
OLD.reserva_id, ", metodo_pago=", OLD.metodo_pago, ",
fecha_facturacion=", OLD.fecha_facturacion, ", observaciones=",
OLD.observaciones, ", costo_total=", OLD.costo_total, ", impuestos=",
OLD.impuestos, ", cargos_adicionales=", OLD.cargos_adicionales, "
WHERE id=", OLD.id, ";")); END $$ DELIMITER ;

```

```

465
466 • UPDATE facturacion
467 SET metodo_pago = 'Con tarjeta de crédito', costo_total = 1300.00
468 WHERE id = 6;
469 • select * from datosOperaciones;

```

	id_datosOperaciones	usuario_operacion	fecha_operacion	operacion_que_se_ejecuto	operacion_que_revierte
	28	root@localhost	2025-02-05 08:35:40	INSERT INTO facturacion (id, reserva_id, meto...	DELETE FROM facturacion WHERE id=8;
	29	root@localhost	2025-02-05 08:37:04	DELETE FROM facturacion WHERE id=8;	INSERT INTO facturacion (id, reserva_id, meto...
	30	root@localhost	2025-02-05 08:38:01	UPDATE facturacion SET reserva_id=9, metodo...	UPDATE facturacion SET reserva_id=9, metodo...
•	NULL	NULL	NULL	NULL	NULL

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

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:

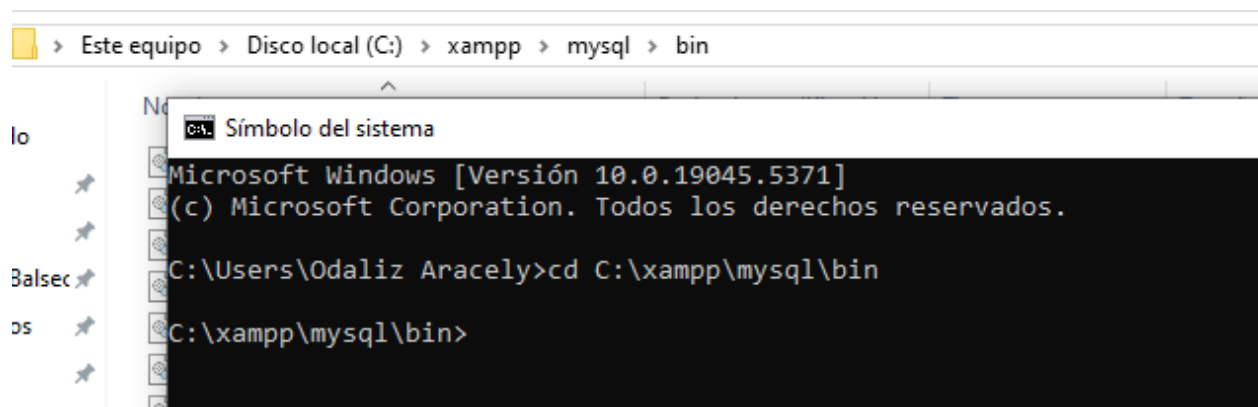
1. **Crear respaldos completos (full backups).**

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

Pasos realizados:

Informe: Procedimiento para Realizar una Copia de Seguridad en MySQL

1. Intento inicial de conexión a MySQL: Se intentó conectar a MySQL utilizando el siguiente comando:



The image shows a Windows File Explorer window with the address bar displaying the path: > Este equipo > Disco local (C:) > xampp > mysql > bin. Below the address bar, a Command Prompt window is open, showing the following text: "Símbolo del sistema", "Microsoft Windows [Versión 10.0.19045.5371]", "(c) Microsoft Corporation. Todos los derechos reservados.", "C:\Users\Odaliz Aracely>cd C:\xampp\mysql\bin", and "C:\xampp\mysql\bin>".

Sin embargo, se presentó un error de autenticación relacionado con el plugin caching_sha2_password, que impidió el acceso a la base de datos. El mensaje de error fue:


```
C:\xampp\mysql\bin>mysql -h localhost -u root -p
Enter password:
ERROR 1045 (28000): Plugin caching_sha2_password could not be loaded: No se puede encontrar el módulo especificado. Library path is 'caching_sha2_password.dll'
```

2. Inicio de MySQL en modo seguro: Para sortear el problema de autenticación, se inició MySQL en modo seguro, utilizando el parámetro `--skip-grant-tables` para omitir la verificación de permisos:

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '1234';
```

Este paso permitió ejecutar MySQL sin requerir autenticación, lo que facilitó la conexión posterior.

3. Conexión a MySQL sin autenticación: Se procedió a conectarse nuevamente a MySQL sin necesidad de introducir la contraseña, ejecutando el siguiente comando:

```
C:\xampp\mysql\bin>mysql -h localhost -u root -p
Enter password: ****
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 36
Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

Este comando permitió acceder al entorno de MySQL sin problemas y acceder al prompt de la base de datos.

4. Visualización de las bases de datos disponibles: Una vez dentro de MySQL, se ejecutó el comando `show databases;` para listar todas las bases de datos disponibles en el servidor:


```
show databases' at line 1
MySQL [(none)]> show databases;
+-----+
| Database |
+-----+
| bck_plibreria |
| ecommerce     |
| empresa1      |
| gestion_hoteles |
| information_schema |
| megamercado   |
| mysql         |
| performance_schema |
| prueba2       |
| sakila        |
| sistemalibreria |
| sys           |
| world         |
+-----+
13 rows in set (0.002 sec)

MySQL [(none)]>
```

Se obtuvo la siguiente lista de bases de datos:

- bck_plibreria
- ecommerce
- empresa1
- gestion_hoteles
- information_schema
- megamercado
- mysql
- performance_schema
- prueba2
- sakila
- sistemalibreria
- sys
- World

5. Selección de la base de datos gestion_hoteles: Se seleccionó la base de datos en la que se quería trabajar con el siguiente comando:

Esto permitió establecer gestion_hoteles como la base de datos activa.

```
MySQL [(none)]> use gestion_hoteles
Database changed
MySQL [gestion_hoteles]> show tables;
+-----+
| Tables_in_gestion_hoteles |
+-----+
| empleados                  |
| facturacion                 |
| habitaciones                |
| huéspedes                   |
| personas                   |
| reservas                   |
| reservas_old                |
| servicios                   |
+-----+
8 rows in set (0.002 sec)

MySQL [gestion_hoteles]>
```

6. Inspección de las tablas en gestion_hoteles: Se ejecutó el comando show tables; para ver las tablas contenidas en la base de datos seleccionada:

```

MySQL [(none)]> use gestion_hoteles
Database changed
MySQL [gestion_hoteles]> show tables;
+-----+
| Tables_in_gestion_hoteles |
+-----+
| empleados                 |
| facturacion                |
| habitaciones               |
| huéspedes                  |
| personas                   |
| reservas                   |
| reservas_old               |
| servicios                   |
+-----+
8 rows in set (0.002 sec)

MySQL [gestion_hoteles]>

```

Se obtuvo la siguiente lista de tablas dentro de la base de datos gestion_hoteles:

- empleados
- facturacion
- habitaciones
- huéspedes
- personas
- reservas
- reservas_old
- servicios

7. Inspección de la estructura de la tabla habitaciones: Se ejecutó el comando `describe habitaciones;` para ver la estructura de la tabla habitaciones:

La salida mostró los siguientes campos:

- numero_habitacion: int
- precio_por_noche: double
- numero_camapas: int
- descripcion: varchar(100)
- tipo_habitacion: enum con valores como 'Suite', 'Junior suite', 'Gran suite', 'Normal', 'Habitación matrimonial'

```
MySQL [gestion_hoteles]> describe habitaciones;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default |
+-----+-----+-----+-----+
| numero_habitacion | int | NO | PRI | NULL |
| precio_por_noche | double | NO | | NULL |
| numero_camas | int | NO | | NULL |
| descripcion | varchar(100) | YES | | NULL |
| tipo_habitacion | enum('Suite','Junior suite','Gran suite','Normal','Habitación matrimonial') | NO | | NULL |
+-----+-----+-----+-----+
5 rows in set (0.005 sec)
```

8. Cierre de la sesión de MySQL: Una vez completadas las inspecciones necesarias, se cerró la sesión de MySQL con el comando `bye`:

```
MySQL [gestion_hoteles]> bye
->
-> bye
-> BYE
-> Ctrl-C -- exit!
Bye
```

9. Realización de la copia de seguridad de la base de datos `gestion_hoteles`:

Finalmente, se ejecutó el comando `mysqldump` para crear una copia de seguridad de la base de datos `gestion_hoteles`:

```
mysqldump -h localhost -u root -p gestion_hoteles > copia.sql
```

La copia de seguridad se completó correctamente sin ningún error.

```
C:\xampp\mysql\bin>mysqldump -h localhost -u root -p gestion_hoteles > copia.sql
Enter password: ****
C:\xampp\mysql\bin>
```

← → ↕ ↶ ↷

Este equipo > Disco local (C:) > xampp > mysql > bin

Nombre	Fecha de modificación	Tipo	Tamaño
aria_read_log	30/10/2023 7:59	Aplicación	4.371 Ki
concr140.dll	11/10/2023 3:42	Extensión de la ap...	320 Ki
copia	5/2/2025 9:14	SQL Text File	11 Ki

✓ Acceso rápido

Escritorio

Descargas

copia: Bloc de notas

Archivo Edición Formato Ver Ayuda

-- MariaDB dump 10.19 Distrib 10.4.32-MariaDB, for Win64 (AMD64)

--

-- Host: localhost Database: gestion_hoteles

--

-- Server version 8.0.40

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;

/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;

/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;

/*!40101 SET NAMES utf8mb4 */;

/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;

/*!40103 SET TIME_ZONE='+00:00' */;

/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;

/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;

/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--

-- Table structure for table `empleados`

--

DROP TABLE IF EXISTS `empleados`;

/*!40101 SET @saved_cs_client = @@character_set_client */;

/*!40101 SET character_set_client = utf8 */;

CREATE TABLE `empleados` (

`id` int NOT NULL AUTO_INCREMENT,

`persona_id` int NOT NULL,

`puesto` varchar(50) NOT NULL,

`salario` double NOT NULL,

`fecha_contratacion` date NOT NULL,

`usuario` varchar(50) NOT NULL,

`contrasenia` varchar(255) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `usuario` (`usuario`),

KEY `persona_id` (`persona_id`),

CONSTRAINT `empleados_ibfk_1` FOREIGN KEY (`persona_id`) REFERENCES `personas` (`id`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

/*!40101 SET character_set_client = @saved_cs_client */;

--

-- Dumping data for table `empleados`

--

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

1. Respaldo Completo (Full Backup)

Un **respaldo completo** es una copia exacta de toda la base de datos, incluyendo todas las tablas, índices, registros y configuraciones. Esta es la forma más sencilla y segura de respaldo, ya que permite restaurar la base de datos a su estado exacto en el momento de la copia.

Ventajas:

- Sencillo de implementar y restaurar.
- Proporciona una imagen completa de la base de datos en un punto específico en el tiempo.

Desventajas:

- Requiere un gran espacio de almacenamiento.
- El tiempo de ejecución puede ser largo, especialmente con bases de datos grandes.

Mejores prácticas:

- Realizar respaldos completos de manera periódica (por ejemplo, semanalmente o mensualmente).
- Guardar estos respaldos en ubicaciones separadas, como servidores externos o servicios de almacenamiento en la nube.

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

- Realizar respaldos completos de manera periódica (por ejemplo, semanalmente o mensualmente).
- Guardar estos respaldos en ubicaciones separadas, como servidores externos o servicios de almacenamiento en la nube.

2. Respaldo Incremental

El **respaldo incremental** solo guarda los cambios realizados desde el último respaldo completo o incremental. Esto reduce significativamente el espacio de almacenamiento necesario y el tiempo de ejecución.

Ventajas:

- Menor consumo de espacio en disco en comparación con los respaldos completos.
- Más rápido que los respaldos completos, ya que solo se respaldan los cambios.

Desventajas:

- Para una restauración completa, es necesario restaurar primero el respaldo completo y luego aplicar cada uno de los respaldos incrementales, lo que puede ser más lento.

Mejores prácticas:

- Utilizar respaldos incrementales entre respaldos completos para minimizar el impacto en el rendimiento y almacenamiento.
- Asegurarse de que los respaldos incrementales se almacenen de manera segura, ya que la restauración depende de ellos.

3. Respaldo Diferencial

Un **respaldo diferencial** guarda todos los cambios realizados desde el último respaldo completo. A diferencia del respaldo incremental, no depende de los respaldos anteriores, lo que facilita la restauración de la base de datos.

Ventajas:

- Facilita la restauración, ya que solo es necesario restaurar el último respaldo completo y el último diferencial.
- Consume menos espacio que los respaldos completos, pero más que los incrementales.

Desventajas:

- A medida que pasa el tiempo, el tamaño de los respaldos diferenciales aumenta.

Mejores prácticas:

- Utilizar respaldos diferenciales de forma más frecuente que los completos, especialmente si la base de datos cambia rápidamente.
- Al igual que con los incrementales, almacenarlos en ubicaciones seguras.

4. Respaldo en Caliente (Hot Backup)

El **respaldo en caliente** permite realizar copias de seguridad de la base de datos sin interrumpir el servicio o la actividad de los usuarios. Esto es crucial en bases de datos de producción que no pueden permitirse tiempo de inactividad.

Ventajas:

- Permite la continuidad del servicio mientras se realiza el respaldo.
- Ideal para bases de datos de alta disponibilidad.

Desventajas:

- Requiere tecnologías avanzadas, como **Percona XtraBackup** o **MySQL Enterprise Backup**, que ofrecen soporte para bases de datos en caliente.
- Puede afectar ligeramente el rendimiento debido a la carga adicional de la copia de seguridad.

Mejores prácticas:

- Utilizar herramientas que soporten respaldos en caliente.
- Configurar alertas de rendimiento para garantizar que el respaldo en caliente no impacte negativamente el servicio.

5. Respaldo en Frío (Cold Backup)

El **respaldo en frío** se realiza cuando la base de datos está apagada o deshabilitada temporalmente. Este tipo de respaldo es menos común en entornos de alta disponibilidad debido a la interrupción que causa, pero es muy seguro.

Ventajas:

- Es más sencillo de implementar.
- No hay riesgo de realizar una copia inconsistente de la base de datos.

Desventajas:

- Causa tiempo de inactividad, lo que no es ideal en entornos de producción.

Mejores prácticas:

- Utilizar en entornos de baja demanda o para realizar respaldos completos en momentos planificados de inactividad.

6. Uso de Almacenamiento en la Nube

El almacenamiento en la nube es una opción excelente para almacenar respaldos de bases de datos grandes, debido a su escalabilidad y accesibilidad.

Ventajas:

- Accesibilidad desde cualquier lugar.
- Escalabilidad sin necesidad de gestionar hardware físico.
- Alta disponibilidad y redundancia.

Desventajas:

- Dependencia de la conectividad a internet.
- Costo a largo plazo.

Mejores prácticas:

- Utilizar soluciones de almacenamiento en la nube de proveedores confiables como AWS, Google Cloud o Azure.
- Realizar respaldos automáticos a la nube para asegurar que siempre haya una copia actualizada.

7. Herramientas y Técnicas Específicas para Bases de Datos Grandes

- **Percona XtraBackup:** Una herramienta popular para realizar respaldos en caliente de bases de datos MySQL y MariaDB sin interrumpir el servicio. Es ideal para bases de datos grandes debido a su capacidad para realizar respaldos rápidos y eficientes.
- **MySQL Enterprise Backup:** Ofrece una solución de respaldo en caliente, ideal para bases de datos grandes, pero requiere una licencia de pago.
- **Backup Incremental en ZFS o LVM:** Para grandes bases de datos en servidores Linux, los respaldos de nivel de sistema con ZFS o LVM pueden ser muy eficientes para realizar respaldos incrementales.

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 y gestionar Índices -- Índice en identificaciones para búsqueda rápida de personas

```
create index idx_identificacion on personas(identificacion);
```

-- Índice en nombre de servicios (si se usa mucho en consultas)

```
create index idx_servicio_nombre on servicios(nombre);
```

-- Índice en fechas de reserva (optimiza búsquedas por fechas)

```
create index idx_reserva_fecha on reservas(fecha_entrada, fecha_salida);
```

-- Índice en método de pago (si se usa mucho en consultas y reportes)

```
create index idx_facturacion_metodo on facturacion(metodo_pago);
```

Búsqueda rápida por Identificación

```
select * from personas where identificacion = '1234567890';
```

Verificar si un usuario existe antes de insertarlo:

```
select count(*) from personas where identificacion = '1234567890';
```

Usar el índice en una subconsulta optimizada:

```
select nombres, apellidos
```

```
from personas
```

```
where identificacion in (select identificacion from personas where identificacion like '123%');
```

```

130 -- PARTE 3
131 -- Búsqueda rápida por Identificación

```

```

132 • select * from personas where identificacion = '1102567890';

```

<								
Result Grid								
Filter Rows: <input type="text"/>								
Edit: Export/Import: Wrap Cell C								
	id	identificacion	nombres	apellidos	telefono	correo	direccion	edad
▶	1	1102567890	Odaliz	Balseca	0987654321	odaliz.balseca@gmail.com	Quito	25
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```

134 -- Verificar si un usuario existe antes de insertarlo:

```

```

135 • select count(*) from personas where identificacion = '1102567890';

```

<	
Result Grid	
Filter Rows: <input type="text"/>	
Export: Wrap Cell Content:	
	count(*)
▶	1

```

137 • explain select * from personas where identificacion = '1102567890';

```

<											
Result Grid											
Filter Rows: <input type="text"/>											
Export: Wrap Cell Content:											
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered
▶	1	SIMPLE	personas	NULL	const	identificacion,idx_identificacion	identificacion	42	const	1	100.00

```

139 -- Usar el índice en una subconsulta optimizada:

```

```

140 • select nombres, apellidos from personas
141 where identificacion in (select identificacion from personas where identificacion like '1102%');
142

```

<	
Result Grid	
Filter Rows: <input type="text"/>	
Export: Wrap Cell Content:	
	nombres apellidos
▶	Odaliz Balseca

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

B-Tree (predeterminado en MySQL): Bueno para búsquedas y rangos (BETWEEN, LIKE, etc.).

Full-text Index: Para búsquedas en texto largo (MATCH() AGAINST()).

Hash Index: Útil para búsquedas exactas en tablas MEMORY.

Índices compuestos: Cuando una consulta usa varias columnas (fecha_entrada, fecha_salida)

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

Acelera la recuperación de datos en tablas grandes.

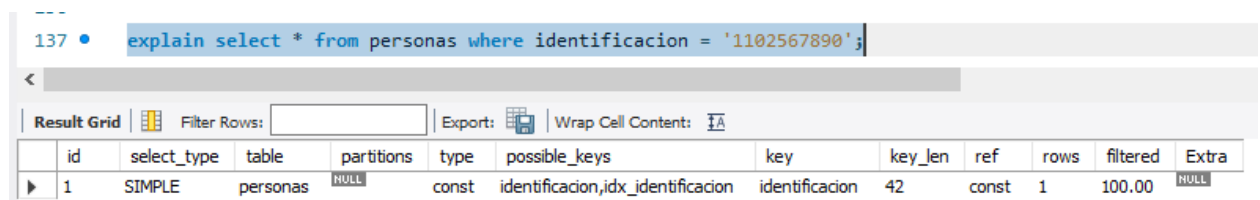
Reduce el **costo computacional** en SELECT.

Evita **full table scans** (escaneos completos de tabla).

2. Optimizar consultas SQL.

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

```
explain select * from reservas where fecha_entrada = '2025-02-10';
```



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	personas	NULL	const	identificacion,idx_identificacion	identificacion	42	const	1	100.00	NULL

Qué revisar en EXPLAIN:

type = ALL → Indica un escaneo completo (NO OPTIMIZADO).

possible_keys → Indica qué índices se pueden usar.

key → Indica si se está usando un índice.

Optimización de consultas con JOIN

Cuando combinamos tablas, es clave usar **índices en las claves foráneas** para mejorar rendimiento.

```
140 • select r.id, p.nombres, p.apellidos, hab.numero_habitacion, r.fecha_entrada
141 from reservas r
142 join huespedes h on r.huesped_id = h.id
143 join personas p on h.persona_id = p.id
144 join habitaciones hab on r.numero_habitacion = hab.numero_habitacion
145 where r.fecha_entrada >= '2025-02-01';
146
```

<

Result Grid | | Filter Rows: | Export: | Wrap Cell Content:

	id	nombres	apellidos	numero_habitacion	fecha_entrada
▶	1	Odaliz	Balseca	101	2025-02-05
	2	Sebastian	Chico	102	2025-02-07

Optimización:

Asegurar que huesped_id y persona_id tengan índices.

Usar WHERE en columnas indexadas.

Evitar SELECT * si no se necesitan todos los campos.

Investigación: Cómo mejorar consultas complejas

Usar JOIN en lugar de subconsultas cuando sea posible.

Evitar ORDER BY en grandes volúmenes de datos sin índice.

Limitar resultados con LIMIT cuando no se necesiten todos los datos.

Importancia

Asegura **respuestas rápidas** en bases de datos con muchos registros.

Evita bloqueos en bases de datos transaccionales.

Reduce **carga en el servidor** y optimiza el uso de recursos.

3. Utilizar particionamiento de tablas.

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

Práctica: Particionar la tabla reservas por fecha

Si reservas tiene millones de registros, podemos dividirla por año.

```
160 • create table reservas (
161     id int not null auto_increment,
162     numero_personas int not null,
163     numero_habitacion int not null,
164     empleado_id int not null,
165     huesped_id int not null,
166     fecha_entrada date not null,
167     fecha_salida date not null,
168     primary key (id, fecha_entrada) -- Incluir fecha_entrada en la clave primaria
169 ) partition by range (year(fecha_entrada)) (
170     partition p2024 values less than (2025),
171     partition p2025 values less than (2026),
172     partition p2026 values less than (2027),
173     partition p2027 values less than (2028)
174 );

177 -- INSERTAR UNA RESERVA
178 • insert into reservas (numero_personas, numero_habitacion, empleado_id, huesped_id, fecha_entrada, fecha_salida)
179     values (2, 101, 1, 1, '2025-06-10', '2025-06-15'); -- Se guardará en la partición `p2025`
180
181 • select * from reservas where year(fecha_entrada) = 2025;
182
183
```

Result Grid

	id	numero_personas	numero_habitacion	empleado_id	huesped_id	fecha_entrada	fecha_salida
▶	1	2	101	1	1	2025-02-05	2025-02-10
	1	2	101	1	1	2025-06-10	2025-06-15
	2	1	102	2	2	2025-02-07	2025-02-12
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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

El **particionamiento de bases de datos** es una técnica que divide grandes conjuntos de datos en unidades más pequeñas y manejables, conocidas como particiones. Esta estrategia mejora la escalabilidad, el rendimiento y la gestión de bases de datos con volúmenes significativos de información.

Beneficios del particionamiento:

1. **Mejora del rendimiento de consultas:** Al dividir los datos en particiones más pequeñas, las consultas que acceden a una fracción específica de los datos pueden ejecutarse más rápidamente, ya que se reduce la cantidad de datos a procesar.
2. **Escalabilidad eficiente:** El particionamiento permite distribuir los datos en múltiples servidores o nodos, facilitando la expansión de la capacidad de almacenamiento y procesamiento sin afectar el rendimiento general del sistema.

3. **Mantenimiento simplificado:** Gestionar particiones individuales es más sencillo que manejar una base de datos monolítica. Las operaciones de mantenimiento, como respaldos o actualizaciones, pueden realizarse en particiones específicas sin interrumpir el acceso a otras partes de la base de datos.
4. **Alta disponibilidad y tolerancia a fallos:** Distribuir las particiones en diferentes servidores o ubicaciones geográficas reduce el riesgo de pérdida de datos y mejora la disponibilidad del sistema. Si una partición falla, las demás pueden seguir operando normalmente.

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

Implementación del particionamiento en bases de datos grandes:

1. **Análisis de patrones de acceso a datos:** Antes de implementar el particionamiento, es crucial entender cómo se accede a los datos. Identificar columnas frecuentemente consultadas o utilizadas en filtros puede ayudar a determinar la estrategia de particionamiento más adecuada.
2. **Selección de la estrategia de particionamiento:** Existen varias técnicas, como:
 - a. **Por rango:** Divide los datos según rangos de valores, como fechas o números.
 - b. **Por lista:** Asigna datos a particiones basándose en valores específicos de una columna.
 - c. **Por hash:** Distribuye los datos de manera uniforme utilizando una función hash.

La elección depende de los patrones de acceso y las necesidades específicas de la aplicación.

3. **Implementación técnica:** La mayoría de los sistemas de gestión de bases de datos (DBMS) ofrecen mecanismos para particionar tablas. Por ejemplo, en

MySQL, se puede utilizar la cláusula `PARTITION BY` al crear una tabla para definir las particiones.

4. **Monitoreo y ajuste continuo:** Es esencial supervisar el rendimiento y el uso de las particiones, ajustando la estrategia según sea necesario para mantener la eficiencia y la escalabilidad del sistema.

5. Procedimientos Almacenados, Vistas y Triggers

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

Actividades:

1. **Crear procedimientos almacenados.**

A continuación, el código para crear un procedimiento almacenado en MySQL para calcular el precio total de una reserva, aplicando descuentos y cargos adicionales. Este procedimiento tomará como entrada el ID de la reserva, un porcentaje de descuento y un monto de cargos adicionales, y devolverá el precio total actualizado.

```
DELIMITER //
CREATE PROCEDURE CalcularPrecioTotalReserva( IN reserva_id INT,
IN descuento DECIMAL(5,2),
IN cargos_adicionales DECIMAL(10,2) ) BEGIN DECLARE
precio_habitacion DECIMAL(10,2); DECLARE noches INT;
DECLARE precio_total DECIMAL(10,2);
DECLARE precio_final DECIMAL(10,2);
```

```

-- Obtener el precio por noche de la habitación y el número de noches
SELECT
    h.precio_por_noche,
    DATEDIFF(r.fecha_salida, r.fecha_entrada) INTO precio_habitacion,
    noches
FROM
    reservas r
JOIN habitaciones h ON r.numero_habitacion = h.numero_habitacion
WHERE
    r.id = reserva_id;

-- Calcular el precio total sin descuentos ni cargos
SET precio_total = precio_habitacion * noches;

-- Aplicar descuento (si existe)
IF descuento > 0 THEN
    SET precio_total = precio_total - (precio_total * (descuento / 100));
END IF;

-- Aplicar cargos adicionales (si existen)
IF cargos_adicionales > 0 THEN
    SET precio_total = precio_total + cargos_adicionales;
END IF;

-- Devolver el precio final
SELECT
    precio_total AS Precio_Final;

END //
DELIMITER ;

```

Explicación del Procedimiento

Entradas:

reserva_id: El ID de la reserva para la cual se calculará el precio.

descuento: El porcentaje de descuento a aplicar (por ejemplo, 10.00 para un 10%).

cargos_adicionales: El monto adicional a sumar al precio total.

Ejemplo de Uso

Supongamos que tienes una reserva con ID 9, y deseas aplicar un descuento del 10% y cargos adicionales de 50.00. Puedes llamar al procedimiento de la siguiente manera:

42 • `CALL CalcularPrecioTotalReserva(9, 10.00, 50.00);`

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	Precio_Final
▶	21650.00

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

Los procedimientos almacenados son bloques de código SQL predefinidos que se almacenan y ejecutan en el servidor de la base de datos. Permiten encapsular operaciones complejas, lo que facilita la reutilización de código y mejora la eficiencia en el manejo de consultas repetitivas o lógicas complejas.

Uno de los beneficios clave de los procedimientos almacenados es la reutilización de código. Al almacenar procedimientos en el servidor, se pueden invocar múltiples veces desde diferentes aplicaciones o usuarios, lo que evita la duplicación de consultas o lógica en diversas partes del sistema. Esto simplifica el mantenimiento y reduce el riesgo de errores.

En términos de eficiencia, los procedimientos almacenados pueden mejorar el rendimiento al ejecutarse directamente en el servidor de base de datos, lo que minimiza la transferencia de datos entre el servidor y el cliente. Al procesar operaciones de manera interna, se reducen los tiempos de ejecución y se optimiza el uso de recursos del servidor. Además, los procedimientos almacenados permiten optimizar el uso de índices y la lógica de ejecución de las consultas, lo que puede mejorar el rendimiento en bases de datos grandes.

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

2. Crear vistas para simplificar consultas complejas.

1. Vista: Información completa de huéspedes y sus reservas

```
CREATE VIEW VistaInfoHuespedes AS
SELECT
    p.identificacion,
    CONCAT(p.nombres, ' ', p.apellidos) AS nombre_completo,
    p.telefono,
```

```

p.correo,
h.sexo,
h.pais_origen,
r.numero_habitacion,
r.fecha_entrada,
r.fecha_salida
FROM personas p
JOIN huespedes h ON p.id = h.persona_id
LEFT JOIN reservas r ON h.id = r.huesped_id;

```

Descripción:

Combina datos de huéspedes (huespedes), personas (personas) y reservas (reservas), mostrando información personal, país de origen y detalles de reservas asociadas.

93 • `SELECT * FROM VistaInfoHuespedes;`

identificacion	nombre_completo	telefono	correo	sexo	pais_origen	numero_habitacion	fecha_entrada	fecha_salida
1103578910	Sebastian Chico	0987123456	sebastian.chico@gmail.com	Masculino	Ecuador	102	2023-11-15	2023-11-20
1104589123	María González	0912345678	maria.gonzalez@gmail.com	Femenino	Argentina	103	2023-12-10	2023-12-15
1105698123	Carlos Ramírez	0923456789	carlos.ramirez@gmail.com	Masculino	Perú	108	2023-10-01	2023-12-20
1102567890	Odaliz Balseca	0987654321	odaliz.balseca@gmail.com	Masculino	Ecuador	NULL	NULL	NULL
1103578910	Sebastian Chico	0987123456	sebastian.chico@gmail.com	Femenino	Colombia	NULL	NULL	NULL
1104589123	María González	0912345678	maria.gonzalez@gmail.com	Otro	Perú	NULL	NULL	NULL

2. Vista: Detalles de reservas con datos de empleados y habitaciones

```

CREATE VIEW VistaDetallesReservas AS SELECT r.id AS reserva_id,
r.numero_personas, hab.precio_por_noche, hab.tipo_habitacion,
CONCAT(pe.nombres, ' ', pe.apellidos) AS empleado_responsable,
CONCAT(ph.nombres, ' ', ph.apellidos) AS huesped, r.fecha_entrada,
r.fecha_salida FROM reservas r JOIN habitaciones hab ON r.numero_habitacion =
hab.numero_habitacion JOIN empleados e ON r.empleado_id = e.id JOIN
personas pe ON e.persona_id = pe.id -- Persona del empleado JOIN huespedes h
ON r.huesped_id = h.id JOIN personas ph ON h.persona_id = ph.id; -- Persona del
huésped

```

Descripción:

Muestra información completa de reservas, incluyendo:

Datos de la habitación.

Nombre del empleado que gestionó la reserva.

Nombre del huésped.

Fechas y tipo de habitación.

95 • `SELECT * FROM VistaDetallesReservas WHERE tipo_habitacion = 'Suite';`

reserva_id	numero_personas	precio_por_noche	tipo_habitacion	empleado_responsable	huesped	fecha_entrada	fecha_salida
9	4	300	Suite	Carlos Ramírez	Carlos Ramírez	2023-10-01	2023-12-20

3. Vista: Facturación con datos de reservas y costos calculados

```
CREATE VIEW VistaFacturacionCompleta AS SELECT f.id AS factura_id,
r.numero_habitacion, DATEDIFF(r.fecha_salida, r.fecha_entrada) AS noches,
hab.precio_por_noche, (hab.precio_por_noche * DATEDIFF(r.fecha_salida,
r.fecha_entrada)) AS costo_base, f.cargos_adicionales, f.impuestos, f.costo_total
AS total_pagado, f.metodo_pago, f.fecha_facturacion FROM facturacion f JOIN
reservas r ON f.reserva_id = r.id JOIN habitaciones hab ON r.numero_habitacion =
hab.numero_habitacion;
```

Descripción:

Combina datos de facturación, reservas y habitaciones, mostrando:

Noches reservadas.

Costo base calculado.

Impuestos y cargos adicionales.

Método de pago y total final.

97 • `SELECT`
98 `factura_id,`
99 `noches,`
100 `costo_base,`
101 `total_pagado`
102 `FROM VistaFacturacionCompleta`
103 `WHERE fecha_facturacion BETWEEN '2023-10-01' AND '2023-12-31';`

factura_id	noches	costo_base	total_pagado
6	80	24000	1300
7	5	600	750

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

Las vistas en bases de datos son consultas almacenadas que presentan datos de una o más tablas de manera estructurada, como si fueran tablas virtuales. Usarlas en lugar de realizar consultas complejas repetitivas tiene varias ventajas importantes.

Una de las principales ventajas es la simplificación del código. Las vistas permiten encapsular consultas complicadas, lo que facilita la reutilización y hace que el código sea más limpio y comprensible. En lugar de escribir la misma consulta compleja cada vez que se necesita acceder a la información, se puede crear una vista con la consulta y luego invocar la vista, lo que ahorra tiempo y esfuerzo. Las vistas también mejoran el rendimiento en algunos casos, especialmente cuando las consultas involucradas son complejas y se repiten frecuentemente. Al ejecutar la consulta compleja solo una vez al crear la vista, se puede mejorar la eficiencia en comparación con la ejecución repetida de la misma consulta. Sin embargo, el rendimiento dependerá del tipo de vista (materializada o no materializada) y de la frecuencia con que se actualicen los datos.

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.

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.

Usaremos el siguiente comando que muestra las 10 consultas más costosas en términos de tiempo de ejecución.

```
SELECT * FROM  
performance_schema.events_statements_summary_by_digest  
ORDER BY SUM_TIMER_WAIT DESC LIMIT 10;
```

5 • `SELECT * FROM performance_schema.events_statements_summary_by_digest`

	SCHEMA_NAME	DIGEST	DIGEST_TEXT	COUNT_STAR	SUM_TIMER_WAIT	MIN_TIMER
	NULL	5ef58d36226b79536b1f6fe1ceea1cdd4c39d0c2...	SHOW FULL TABLES FROM `gestion_hoteles`	2	953215700000	9630170000
	gestion_hoteles	d53fd041feb16e031258285b378e80fb8d4295a...	CREATE SYSTEM_USER ? @@ IDENTIFIED BY ?	3	861070100000	1225330000
	gestion_hoteles	d4fe8706bad26e257a1d6caaa82018c447d248b...	INSERT INTO `personas` (`identificacion`, `n...	4	365099000000	2250000000
	NULL	6935f3922a42d8845977364d3d83d2ad4e11a5...	SHOW SESSION VARIABLES LIKE ?	27	351569500000	1467500000
	NULL	b31742f7d23c9cd7e41466837a9bf59 b31742f7d23c9cd7e41466837a9bf595201427497601718ac456e68040146a54			305437000000	562600000
	gestion_hoteles	2b0b912bcd08a8cde97930d2abe40a7da74efa1...	CREATE TABLE `datosOperaciones` (`id_dato...	2	297061700000	8714400000
	NULL	cbc301da493d6b465340bd346910730ee04459...	SHOW PROCEDURE STATUS WHERE `Db` = ?	2	276793800000	3265300000
	gestion_hoteles	d5b4144f7b7b6f97c251841a1b63388c5472d8b...	INSERT INTO `empleados` (`persona_id`, `p...	1	254558000000	2545580000
	gestion_hoteles	dfb76a06443b640a7d8d0c792e34b28d5a6ccaf...	GRANT SELECT , INSERT , UPDATE , DELETE O...	2	217164700000	5845060000
	gestion_hoteles	0c74c069a01c0c72d0bf49c9c71e6bc15112edb...	INSERT INTO `reservas` (`numero_personas` ...	3	174313600000	2108520000

INSERT INTO personas

- Aparece con un SUM_TIMER_WAIT alto (~3.65e+12).
- El AVG_TIMER_WAIT es 912,747,000,000, lo que indica que cada inserción toma un tiempo considerable.

Puede ser optimizada verificando índices y usando BULK INSERT si se insertan múltiples registros.

INSERT INTO empleados

- Tiene un SUM_TIMER_WAIT de 2.54e+12, con un AVG_TIMER_WAIT similar (~2.55e+11).

Si está relacionada con personas, se debe revisar si los FOREIGN KEY afectan el rendimiento.

INSERT INTO reservas

- También tiene un SUM_TIMER_WAIT elevado (1.74e+12).

Puede estar afectada por restricciones de integridad referencial (FOREIGN KEY con empleados y huéspedes).

Entonces analizaremos estas consultas con EXPLAIN para identificar si el índice de identificación no está en uso o la consulta muestra ALL en el campo type, entonces falta un índice.

10 • `EXPLAIN SELECT * FROM personas WHERE identificacion = '1234567890';`

11

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
	1	SIMPLE	personas	NULL	const	identificacion	identificacion	42	const	1	100.00	NULL

- const significa que MySQL ha optimizado la búsqueda porque identificacion es UNIQUE y la consulta devolverá como máximo una fila.

- Se está usando correctamente el índice en identificación.

No necesitas hacer ningún cambio aquí. Todo está optimizado.

```
12 • EXPLAIN SELECT * FROM personas WHERE id = 2;
13
```

Result Grid Filter Rows: Export: Wrap Cell Content:												
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	personas	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL

- const indica que la búsqueda es extremadamente eficiente porque id es la clave primaria (PRIMARY KEY).
- MySQL sabe que id es único, así que devuelve el resultado rápidamente.

No necesitas hacer ningún cambio aquí. Todo está optimizado.

```
14 • EXPLAIN SELECT * FROM reservas WHERE numero_habitacion = 102;
```

Result Grid Filter Rows: Export: Wrap Cell Content:												
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	reservas	NULL	ref	numero_habitacion	numero_habitacion	4	const	1	100.00	NULL

- ref indica que MySQL está usando un índice (numero_habitacion) para filtrar los resultados, lo cual es eficiente.
- rows: 1 indica que MySQL espera encontrar solo una fila con ese número de habitación.

No necesitas hacer ningún cambio aquí. Todo está optimizado.

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

Monitorear el rendimiento de las consultas en bases de datos en producción es esencial para garantizar la eficiencia y detectar posibles cuellos de botella que puedan afectar el desempeño. A continuación se detallan algunas de las mejores prácticas para lograr un monitoreo efectivo:

1. Uso de Herramientas de Monitoreo y Profiling

Implementar herramientas de monitoreo especializadas, como MySQL Enterprise Monitor, pgAdmin para PostgreSQL o SQL Server Profiler para

Microsoft SQL Server, permite obtener métricas detalladas sobre el rendimiento de las consultas. Estas herramientas pueden identificar consultas lentas, consumo de recursos, y otros problemas de rendimiento en tiempo real, proporcionando alertas cuando se superan ciertos umbrales.

2. Monitoreo de Consultas Lentas

Una práctica fundamental es habilitar el registro de consultas lentas en la base de datos. Esto ayuda a identificar qué consultas están tardando más tiempo del esperado. Con el análisis de las consultas más lentas, se pueden aplicar optimizaciones específicas, como la creación de índices adecuados, el ajuste de la consulta o la refactorización de su lógica.

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.

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

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.

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

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.

 chico20200 Creación de sqls de todo el proyecto		575e7f9 · 1 minute ago	 3 Commits
 Creacion de tablas e insercion de datos.sql	Creación de sqls de todo el proyecto	1 minute ago	
 README.md	Initial commit	3 days ago	
 Seguridad-Auditoría y Control de Acceso.sql	Creación de sqls de todo el proyecto	1 minute ago	
 monitoreo y optimizacion.sql	Creación de sqls de todo el proyecto	1 minute ago	
 procedimientos vistas y triggers.sql	Creación de sqls de todo el proyecto	1 minute ago	

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.

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

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

Conclusiones

CONSIDERACIONES

Sugerencia para mejorar el trabajo en equipo y habilidades blandas:

Para optimizar la colaboración, sugiero crear una **tabla de responsabilidades y capacitación**. Esta tabla permitirá monitorear quién es responsable de cada tema, qué actividades se han realizado para capacitar a los compañeros y cuándo se realizaron. Esto fomenta la responsabilidad individual y la transparencia en el equipo.

Ejemplo de tabla de seguimiento:

Responsable	Tema Asignado	Fecha de asigación	Fecha de culminación	Fecha de Capacitación	Capacitación a Compañeros	Observaciones
Guissela Franco	Consultas			01/12/2025	Índices y optimización	Uso de EXPLAIN

Danna López	Seguridad			03/12/2025	Cifrado y control de acceso	Implementación
-------------	-----------	--	--	------------	-----------------------------	----------------

Mejoras en habilidades blandas:

Comunicación efectiva: Promover reuniones de retroalimentación para que todos los miembros intercambien ideas y soluciones. Fomentar la participación activa en las discusiones y evitar el trabajo mecánico.

Investigación y curiosidad: Incentivar a los miembros a investigar profundamente sobre los temas, identificando problemas no documentados y buscando soluciones innovadoras.

Colaboración activa: Fomentar un ambiente de colaboración, promoviendo sesiones de brainstorming y revisiones entre compañeros, y asegurando que todos estén alineados con el progreso del proyecto.

Responsabilidad colectiva: Asegurar que los miembros del equipo no solo sean responsables de sus tareas individuales, sino también del éxito global del proyecto. Esto incluye apoyar a los compañeros en su aprendizaje.

Temáticas Disponibles

- El grupo es libre de elegir una temática sin repetirse con los demás grupos, seguir el ejemplo indicado

Entregables

```
/Project-AEROLINEAS
/Presentaciones
  Proyecto.pptx
/Informe
  Informe_Proyecto.pdf
/Modelados
  Modelo_ER_Conceptual.png
  Modelo_ER_Logico.png
  Modelo_ER_Fisico.png
/Diccionario_De_Datos
  diccionario_datos.xlsx
/Responsabilidades
  responsabilidades_equipo.xlsx
/Scripts
  /Modelado
    crear_tablas.sql
    relaciones_integridad.sql
  /Seguridad
    crear_rols.sql
    cifrado_datos.sql
  /Auditoria
    activar_auditoria.sql
  /Optimización
    crear_indices.sql
    optimizar_consultas.sql
README.md
```

EXPLICACION

Presentación (PPT):

Crear una carpeta llamada Presentaciones donde se suba el archivo .ppt o .pptx correspondiente a la explicación del proyecto, los objetivos, las actividades, y los resultados alcanzados.

Documento Informe:

- Subir el informe detallado del proyecto en formato .docx o .pdf, incluyendo:
Resumen ejecutivo
Descripción de cada fase del proyecto
Resultados obtenidos
Conclusiones

Modelados (ER):

Crear una carpeta llamada Modelados para almacenar los diagramas de modelado ER. Estos pueden estar en formatos como .png, .jpg, .pdf.

- Incluir versiones del modelo conceptual, lógico y físico.

Diccionario de Datos:

Subir un archivo en formato .xlsx o .csv que contenga el diccionario de datos.

Este debe incluir detalles como:

- Nombre de la tabla
- Descripción de la tabla
- Campos (nombre, tipo de datos, restricciones, etc.)
- Relación con otras tablas

Responsabilidades:

Subir un archivo que detalle las responsabilidades de cada miembro del equipo, indicando qué tareas corresponden a cada uno. Este archivo puede ser una tabla en formato .xlsx o .docx.

Script Actividades a Realizar:

Subir los scripts de las actividades, como la creación de la base de datos, la implementación de procedimientos almacenados, vistas, triggers, etc. Estos archivos pueden ser .sql o .sh (si son scripts de shell para automatizar tareas). Estos scripts deben estar organizados en carpetas según la actividad, por ejemplo, Scripts/Modelado, Scripts/Seguridad, Scripts/Auditoría, etc.

RUBRICA

Criterio	Descripción	Puntos
1. Modelado de Base de Datos y Diccionario de Datos		8
Diseño del Modelo Conceptual, Lógico y Físico	El modelo ER refleja las entidades y relaciones correctamente.	4
Diccionario de Datos	El diccionario de datos es detallado, claro y bien estructurado, incluye tablas, campos, relaciones y restricciones.	2

Restricciones de Integridad Referencial	Se definen correctamente las claves primarias y foráneas entre las tablas.	1
Escalabilidad y Mejores Prácticas	El modelo incluye prácticas recomendadas para la escalabilidad y la integración de sistemas de reservas.	1
2. Seguridad, Auditoría y Control de Acceso		8
Políticas de Acceso y Seguridad	Roles y permisos bien definidos para controlar el acceso a las tablas y vistas.	3
Cifrado de Datos Sensibles	Se implementa correctamente el cifrado de datos sensibles, como contraseñas y detalles de pago.	2
Auditoría y Registro de Eventos	Se habilitan herramientas de auditoría para monitorear el acceso y las actividades de los usuarios en la base de datos.	3
3. RespalDOS y Recuperación de Datos		5
RespalDOS Completos e Incrementales	Los respaldos completos e incrementales están implementados y explicados correctamente.	3
RespalDOS en Caliente	Se implementa correctamente el respaldo sin interrumpir el servicio (hot backups).	2
4. Optimización y Rendimiento de Consultas		5
Índices y Optimización de Consultas	Se implementan índices apropiados y se optimizan consultas SQL con herramientas como EXPLAIN.	3
Particionamiento de Tablas	El particionamiento de tablas está correctamente implementado y explicado.	2

5. Procedimientos Almacenados, Vistas y Triggers		5
Procedimientos Almacenados	Se crean procedimientos almacenados para cálculos y tareas recurrentes.	2
Vistas y Simplificación de Consultas	Se crean vistas para simplificar consultas complejas y mejorar el acceso a datos.	2
Triggers para Auditoría y Control de Cambios	Se implementan triggers para mantener un historial de cambios y automatizar tareas.	1
6. Monitoreo y Optimización de Recursos		2
7. Git y Control de Versiones		2
Uso de Git y Control de Versiones	El repositorio está correctamente configurado, con commits claros y frecuentes.	1
Colaboración y Flujo de Trabajo en Git	Se siguen buenas prácticas en el flujo de trabajo (uso de ramas, fusión, etc.).	1
Total		35

Link al diagrama entidades relaciones con atributos