Escuela Politécnica Nacional

Escuela de Formación de Tecnólogos

Proyecto de Base de Datos Grupal

Integrantes: Danna Lopez, Handel Manobanda, Andrew Paillacho, Cristian Tambaco

Fecha de entrega: 5 febrero 2025

Tema Elegido: Gestión de Salas de Conferencias en Empresas y Centros de Eventos

1. Descripción de la problemática a solucionar

Las empresas y centros de eventos suelen enfrentar dificultades en la reserva y administración de sus salas de conferencias. Los problemas comunes incluyen reservas duplicadas, conflictos de horarios, falta de control sobre el uso de las salas, y poca optimización de los recursos disponibles (equipos audiovisuales, capacidad de personas, disponibilidad de catering, etc.).

Sin un sistema adecuado, los empleados pierden tiempo tratando de coordinar reuniones, y los administradores tienen problemas para hacer un seguimiento eficiente del uso de las salas.

2. Objetivo

Diseñar e implementar una base de datos que permita gestionar de manera eficiente la reserva, disponibilidad y uso de salas de conferencias, evitando conflictos y optimizando la planificación de reuniones y eventos.

3. 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.

Importancia en el proyecto:

El diseño lógico, conceptual y físico de una base de datos (BDD) es importante porque permite comprender cómo se estructuran los datos y cómo se almacenan en las tablas de la base de datos.

El diseño conceptual de una base de datos es importante porque permite crear un esquema que sirva como base para la creación de la base de datos, proporcionando una visión de alto nivel de las entidades, atributos y relaciones de la base de datos. El diseño lógico permite visualizar las relaciones entre los datos, mientras que el diseño físico describe cómo se organizan los datos en las tablas.

Para el diseño de la base de datos, se han considerado las siguientes entidades, atributos y relaciones.

3.1 Definición de las entidades y atributos de la base de datos

Entidad: Usuarios (Personas que reservan salas o administran el sistema)

Atributos:

- ID_Usuario (PK)
- nombres_completos
- correo
- Teléfono
- Rol (Administrador, Asistente(Registrado), Organizador, Invitado (Usuario no Registrado))
- Contraseña (Cifrada)

Entidad: Salas (Espacios disponibles para conferencias)

Atributos:

- id_sala (PK)
- nombre_sala
- capacidad
- ubicación
- equipamiento (Proyector, sonido, micrófono, etc.)
- disponibilidad (Sí/No)

Entidad: Eventos (Reservas de salas para conferencias o reuniones)

Atributos:

- ID_Evento (PK)
- nombre
- fecha
- descripción
- fecha
- hora_inicio
- hora_fin
- Estado (Pendiente, Confirmado, Cancelado)
- SalaID (Fk)
- OrganizadorID (FK)

En este caso, OrganizadorID es la clave foránea relacionada a la tabla ‘Usuarios’.

Entidad: Pagos (Si el sistema incluye reservas pagadas)

Atributos:

- id_Pago (PK)
- monto
- fecha_pago
- metodo_Pago (Tarjeta, PayPal, Transferencia)

Entidad: Reserva (Registro de reservas de salas por usuarios)

Atributos:

- ID_Reserva (PK)
- Evento_ID (FK)
- Asistente_ID (FK)
- Pago_ID (FK)
- Estado (pendiente, confirmada, cancelada)
- Fecha_Reserva

Entidad: Usuarios_Salas

Atributos:

- ID PK
- id_usuario
- id_sala
- trabajo

3.2 Definición de las relaciones entre entidades en la base de datos

- **Usuarios y Reservas (1:M):** Un usuario puede hacer varias reservas, pero cada reserva pertenece a un solo usuario, siempre y cuando se hagan en fechas y horas diferentes.
- **Salas y Eventos (1:M):** Una sala puede albergar varios eventos en distintos momentos, por eso la relación es 1:M. En la tabla Eventos, la columna Sala_ID actúa como clave foránea (FK), conectando cada evento con una sala específica.
- **Salas y Reservas (1:M):** Una sala puede tener muchas reservas en diferentes fechas, pero una reserva se hace para una sola sala.
- **Eventos y Reservas (1:1 o 1:M)**: Un evento puede estar asociado a una sola reserva o varias, si se permite reservar múltiples salas.
- **Reservas y Pagos (1:1)**: Una reserva puede estar asociada a un solo pago.
- **Usuarios y Eventos 1:M:** Un usuario puede organizar varios eventos, pero un evento solo tiene un organizador).
- **Usuarios y Salas (si son administradores) (M:M):** Como cada sala tiene responsables o administradores específicos, entonces la relación entre Usuarios y Salas sería M:M puesto que, un usuario puede ser responsable de varias salas y

una sala puede tener varios responsables. La solución es crear una tabla intermedia Usuarios_Salas

Si permites que un usuario pueda reservar más de un boleto para eventos que ocurren en la misma fecha y hora, puede haber problemas de solapamiento, porque en la realidad, no puede estar en dos lugares al mismo tiempo. La solución es restringir solapamientos en la base de datos.

Antes de confirmar la reserva, el sistema puede verificar si el usuario ya está registrado en otro evento a la misma hora.

Lo que necesitamos es una tabla Usuarios_Eventos que registre la relación entre un usuario y un evento y que, además, evite solapamientos de reservas. Para esto, podemos hacer que la combinación de fecha y hora de inicio sea única por usuario.

```
CREATE TABLE Usuarios_Eventos (
    ID_Usuario INT,
    ID_Evento INT,
    Fecha DATE,
    Hora_Inicio TIME,
    Hora_Fin TIME,
    PRIMARY KEY (ID_Usuario, Fecha, Hora_Inicio),  -- Restricción de unicidad
    FOREIGN KEY (ID_Usuario) REFERENCES Usuarios(ID_Usuario),
    FOREIGN KEY (ID_Evento) REFERENCES Eventos(ID_Evento)
);
```

Consulta para verificar solapamientos

```
SELECT * FROM Usuarios_Eventos
WHERE ID_Usuario = ?
AND Fecha = ?
AND (
    (Hora_Inicio < ? AND Hora_Fin > ?)  -- Evento que comienza antes y termina después
    OR
```

```
    (Hora_Inicio < ? AND Hora_Fin > ?)  -- Otro evento que se solapa
);
```

Si la consulta retorna resultados, significa que el usuario ya tiene una reserva en ese horario, por lo que se le debe bloquear la segunda inscripción.

Restricción a nivel de base de datos

Puedes evitar estos conflictos utilizando una restricción de unicidad en la tabla Usuarios_Eventos

```
ALTER TABLE Usuarios_Eventos
ADD CONSTRAINT unico_evento_usuario
UNIQUE (ID_Usuario, Fecha, Hora_Inicio);
```

Ejemplo de cómo quedaría la estructura:

Esta tabla registra sólo a los usuarios que realmente asisten a un evento, mientras que Invitados_Eventos maneja invitados no registrados.

```
CREATE TABLE Usuarios_Eventos (
    ID_Usuario INT,
    ID_Evento INT,
    PRIMARY KEY (ID_Usuario, ID_Evento),
    FOREIGN KEY (ID_Usuario) REFERENCES Usuarios(ID_Usuario),
    FOREIGN KEY (ID_Evento) REFERENCES Eventos(ID_Evento)
);
```

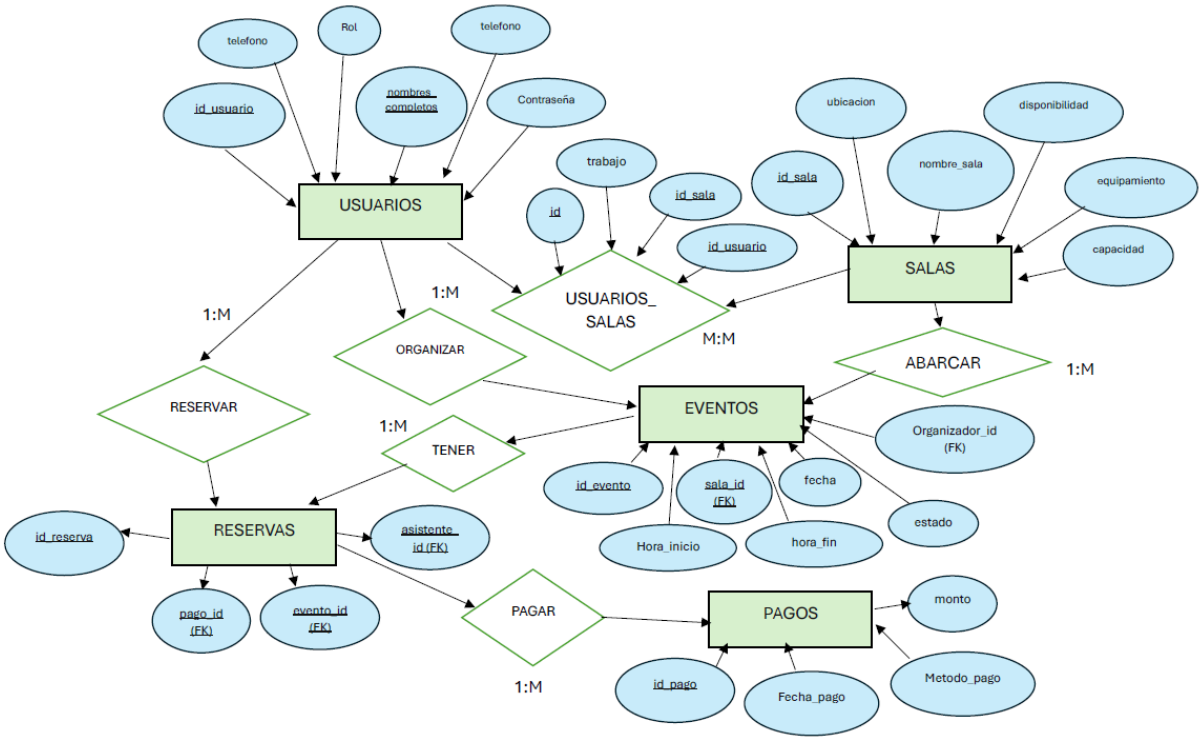
Ejemplo: Juan compra 3 boletos, pero solo él es usuario registrado.

Sus amigos Pedro y Ana no tienen cuenta en el sistema.

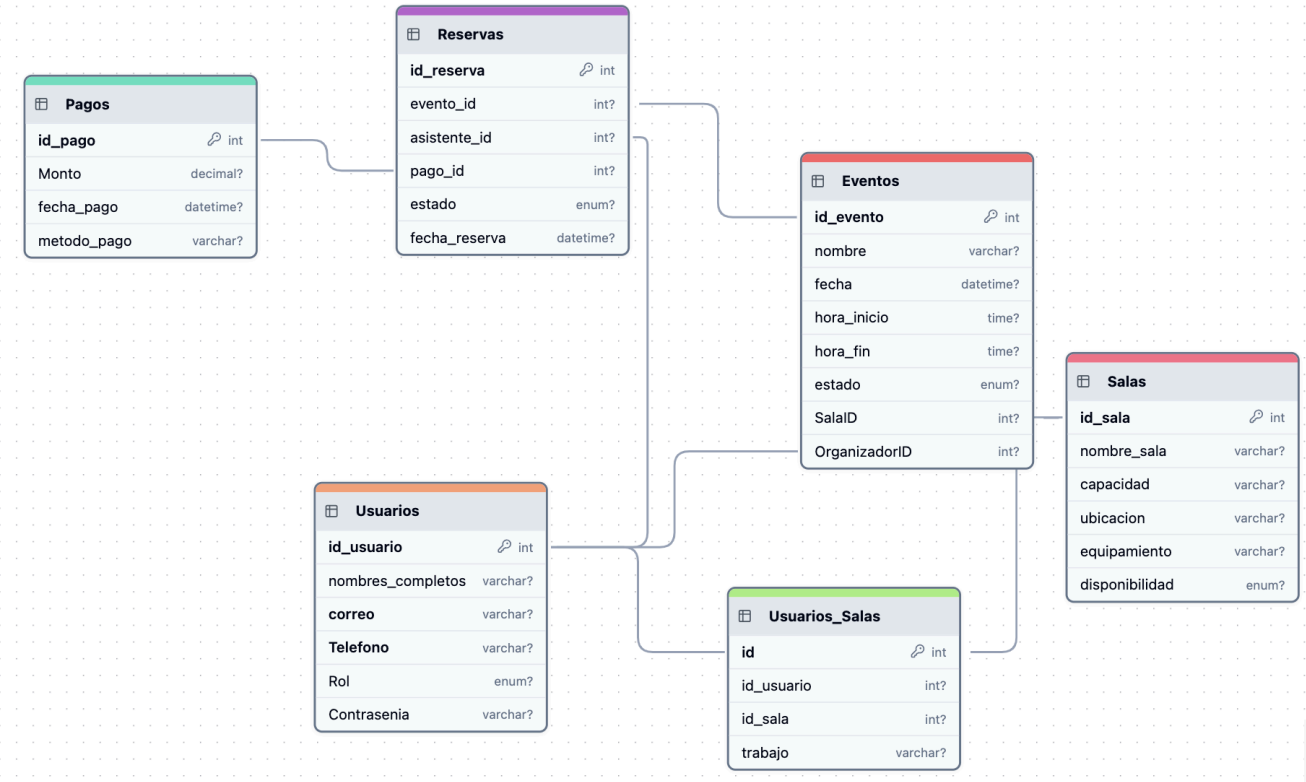
Juan entra en Usuarios_Eventos y Pedro/Ana en Invitados_Eventos.

Permite diferenciar entre usuarios registrados y simples invitados

3.1 Diseño del modelo entidad-relación.



3.2 Diseño del modelo lógico.



3.3 Diseño del modelo físico.

```sql
create database gestion_conferencias;
use gestion_conferencias;
-- Crear la tabla para usuarios con diferentes roles
create table Usuarios(id_usuario int auto_increment primary key,
                      nombres_completos varchar(100) NOT NULL,
                      correo varchar(100) UNIQUE NOT NULL,
                      Telefono varchar(50) UNIQUE NOT NULL,        -- El correo y el telefono deberán ser unicos para cada usuario
                      Rol ENUM('Invitado', 'Organizador', 'Asistente', 'Administrador') NOT NULL,
                      Contrasenia varchar(255) NOT NULL);
-- Crear la tabla para salas
create table Salas(id_sala int auto_increment primary key,
                      nombre_sala varchar(60) NOT NULL,
                      capacidad varchar(100) NOT NULL,
                      ubicacion varchar(50) NOT NULL,
                      equipamiento varchar(50) NOT NULL,       -- Describir los equipos con los que cuenta la sala, EJ: PROYECTORES
                      disponibilidad ENUM('Si', 'No') NOT NULL);                      -- Verificar si la sala estará libre u ocupada
```

```sql
-- Crear la tabla para Eventos
create table Eventos(id_evento int auto_increment primary key,
                      nombre varchar(50) NOT NULL,       -- Nombre del evento
                      descripcion varchar(50) NOT NULL,                          -- Descripcion del evento
                      fecha datetime NOT NULL,                                    -- Fecha en la que sera llevada a cabo el evento
                      hora_inicio time NOT NULL,
                      hora_fin time NOT NULL,
                      Estado enum('Pendiente', 'Cancelado', 'Confirmado') NOT NULL,
                      SalaID int,
                      OrganizadorID int,
                      foreign key (OrganizadorID) references Usuarios(id_usuario));           -- Relacionar la columna OrganizadorID

-- Crear la tabla Pagos
CREATE TABLE Pagos (
    id_pago INT PRIMARY KEY,  -- ID único para el pago
    Monto DECIMAL(10, 2),  -- Monto del pago
    Fecha_Pago DATE,  -- Fecha en que se realizó el pago
    Metodo_Pago VARCHAR(50)  -- Método de pago (tarjeta, transferencia, etc.)
);
```

```sql
-- Crear la tabla para reservas
create table Reservas(id_reserva int auto_increment primary key,
                      evento_id int NOT NULL,    -- Clave foranea con la tabla Eventos
                      asistente_id int NOT NULL,   -- clave foranea con la tabla Usuarios
                      id_pago int NOT NULL,
                      estado enum('Pendiente', 'Cancelado', 'Confirmado') NOT NULL,
                      fecha_reserva datetime,
                FOREIGN KEY (evento_ID) REFERENCES Eventos(id_evento),
                FOREIGN KEY (asistente_id) REFERENCES Usuarios(id_usuario),
                FOREIGN KEY (id_Pago) REFERENCES Pagos(id_pago)  -- Relación con el pago
);

-- Crear una tabla intermedia Usuarios_Salas para que un usuario este a cargo de varias salas
CREATE TABLE Usuarios_Salas (
    id int auto_increment primary key,
    id_usuario INT,
    id_sala INT,
    trabajo VARCHAR(50),  -- Ejemplo: "Administrador", "Soporte técnico"
    FOREIGN KEY (id_usuario) REFERENCES Usuarios(id_usuario),
    FOREIGN KEY (id_sala) REFERENCES Salas(id_sala)
);
```

3.4 Investigación: Buenas prácticas para hacer escalables los modelos de bases de datos para sistemas de Gestión de Salas de Conferencias en Empresas y Centros de Eventos.

- Es importante estructurar las tablas de manera que se minimice la duplicidad de información.
- En este tipo de sistemas, las relaciones entre salas, eventos, usuarios y equipos son fundamentales. Se utilizan claves foráneas correctamente para mantener la integridad referencial sin sacrificar la flexibilidad. Esto ayudará a gestionar los datos de forma coherente y reducirá la duplicidad de registros.
- Crear índices en las columnas que se usan con frecuencia en filtros y ordenamientos (como fechas de eventos, ID de salas, etc.). Un mal uso de los índices puede afectar el rendimiento, así que se debe tener cuidado con su cantidad y tipo.
- Diseña el modelo de datos de manera modular, de forma que si se agregan nuevas funcionalidades, puedas hacerlo sin reestructurar toda la base de datos.

3.5 Diseño del diccionario de datos.

Importancia en la base de datos:

Un diccionario de datos trata de documentar los metadatos más ligados a su almacenamiento en la base de datos. Es decir, incluye aspectos técnicos como el tipo de dato, formato, longitud, posibles valores que puede tomar e, incluso, transformaciones sufridas, sin olvidar la definición de cada campo. (Mañes, 2021)

Estos metadatos ayudan a los usuarios a entender los datos desde el punto de vista técnico para poder explotarlos adecuadamente. Por este motivo, cada base de datos debería contar con su diccionario de datos asociado.

3.5.1 Diseño del diccionario de datos en la base de datos

TABLA USUARIOS

Campo	Tipo de Dato	Descripción	Tamaño del carácter	Restricciones
id_usuario	Entero (int0	Código único para cada usuario	4 bytes	Clave primaria, autoincrementable, se acepta que el campo sea nulo
nombres_completos	varchar	Nombres completos del usuario	100 caracteres	Entrada de datos obligatoria
correo	varchar	Correo electronico del usuario	100 caracteres	Entrada de datos obligatoria y única, no se puede repetir
Teléfono	varchar	Numero de celular del usuario	50 caracteres	Entrada de datos obligatoria y única
Rol	enum	Rol de usuario (Administrador-Invitado		Entrada de datos obligatoria

		-Asistente-Organizador)		
Contraseña	varchar	Contraseña del usuario para acceder al sistema	255 caracteres	Entrada de datos obligatoria

TABLA SALAS

Campo	Tipo de Dato	Descripción	Tamaño del carácter	Restricciones
id_sala	Entero (int0	Código único para identificar cada sala	4 bytes	Clave primaria, autoincrementable, se acepta que el campo sea nulo
nombre_sala	varchar	Nombre de la sala donde será el evento	60 caracteres	Entrada de datos obligatoria
capacidad	varchar	Capacidad máxima de personas en una sala	100 caracteres	Entrada de datos obligatoria
ubicación	Entero (int)	Numero de celular del usuario	100 caracteres	Entrada de datos obligatoria
equipamiento	varchar	Los equipos con los que contará la sala (micrófonos, proyectores, etc)	50 caracteres	Entrada de datos obligatoria
disponibilidad	enum	Disponibilidad de la sala (Si/No)		Entrada de datos obligatoria

TABLA EVENTOS

Campo	Tipo de Dato	Descripción	Tamaño del carácter	Restricciones
id_evento	Entero (int0	Código único para identificar cada evento	4 bytes	Clave primaria, autoincrementable, se acepta que sea un campo nulo

Nombre	varchar	Nombre del evento al que asistirán los usuarios	50 caracteres	Entrada de datos obligatoria
descripción	varchar	Resumen de lo que va a tratar el evento	50 caracteres	Entrada de datos obligatoria
fecha	datetime	Fecha y hora en la que ocurrirá el evento	19 caracteres (AAAA-MM-DD HH:MM:SS)	Entrada de datos obligatoria
hora_inicio	time	Hora exacta a la que empieza el evento	10 caracteres (HH:MM:SS)	Entrada de datos obligatoria
hora_fin	time	Hora a la que finaliza el evento	10 caracteres (HH:MM:SS)	Entrada de datos obligatoria
Estado	ENUM	Si el evento está confirmado, pendiente o cancelado		Entrada de datos obligatoria
SalaID	Entero(int)	El código único de la sala donde será llevado a cabo el evento	4 bytes	Clave foránea con entrada de datos obligatoria
OrganizadorID	Entero(int)	El código único del organizador a cargo del evento	4 bytes	Clave foránea con entrada de datos obligatoria

TABLA PAGOS

Campo	**Tipo de Dato**	**Descripción**	**Tamaño del carácter**	**Restricciones**
id_pago	Entero (int0	Código único para identificar cada evento	4 bytes	Clave primaria, autoincrementable, se acepta que sea un campo nulo
Monto	decimal(10,2)	Valor que cancelarán los usuarios por el evento	Máximo 10 dígitos con 2 decimales	Entrada de datos obligatoria

Fecha_Pago	datetime	Fecha y hora en la que se realizó el pago	19 caracteres (AAAA-MM-DD HH:MM:SS)	Entrada de datos obligatoria
Estado	ENUM	Si el evento está confirmado, pendiente o cancelado		Entrada de datos obligatoria
SalaID	Entero(int)	El código único de la sala donde será llevado a cabo el evento	4 bytes	Clave foránea con entrada de datos obligatoria
OrganizadorID	Entero(int)	El código único del organizador a cargo del evento	4 bytes	Clave foránea con entrada de datos obligatoria

TABLA RESERVAS

Campo	Tipo de Dato	Descripción	Tamaño del carácter	Restricciones
Id_Reserva	Entero (int0	Código único para identificar cada evento	4 bytes	Clave primaria, autoincrementable, se acepta que sea un campo nulo
asistente_id	int	Código del usuario que asistirá al evento	4 bytes	Clave foránea con entrada de datos obligatoria
pago_id	int	Código del pago que le pertenece al usuario	4 bytes	Clave foránea con entrada de datos obligatoria
fecha_reserva	datetime	Fecha y hora en la que ocurrirá el evento	19 caracteres (AAAA-MM-DD HH:MM:SS)	Entrada de datos obligatoria

TABLA INTERMEDIA USUARIOS_SALAS

Campo	Tipo de Dato	Descripción	Tamaño del carácter	Restricciones
id	Entero (int0	Código único para identificar los registros	4 bytes	Clave primaria, autoincrementable, se acepta que sea un campo nulo
id_usuario	int	Código del usuario administrador de cada sala	4 bytes	Clave foránea con entrada de datos obligatoria
id_sala	int	Código de la sala que le corresponde	4 bytes	Clave foránea con entrada de datos obligatoria
trabajo	varchar	Descripción del trabajo que hace el usuario, si es encargado de mantenimiento técnico	50 caracteres	Entrada de datos obligatoria

Investigación: Herramientas y métodos para generar diccionarios de datos y su uso en proyectos reales.

4.6 Definición de las restricciones de integridad referencial (eliminación - update).

Importancia: La integridad referencial garantiza que solo se produzcan las modificaciones, adiciones o eliminaciones requeridas a través de reglas implantadas en la estructura de la base de datos sobre cómo se utilizan las claves externas. (*Astera*, 2024) Estas reglas pueden incluir condiciones que eliminen registros de datos duplicados para garantizar que los datos sean precisos y prohíban el registro de datos inadecuados.

Práctica: Establecer claves primarias y foráneas entre las tablas, asegurando la coherencia de los datos.

Eliminación – Update: ON CASCADE, set null, restrict, Constraint (NO ACTION)

4.6.1 Usuarios - Eventos: ON DELETE SET NULL, ON UPDATE CASCADE

- Si un usuario (organizador) es eliminado, su evento asociado tendrá el campo OrganizadorID establecido a NULL (no se elimina el evento).
- Si un usuario es actualizado, el OrganizadorID en los eventos relacionados se actualizará automáticamente.

```
-- Definición de las restricciones de integridad referencial (eliminación - update).

-- 1. Usuarios - Eventos: ON DELETE SET NULL, ON UPDATE CASCADE
-- Si un usuario (organizador) es eliminado, su evento asociado tendrá el campo OrganizadorID
-- establecido a NULL (no se elimina el evento).
-- Si un usuario es actualizado, el OrganizadorID en los eventos relacionados se actualizará automáticamente.

-- Modificar la tabla Eventos
ALTER TABLE Eventos
DROP FOREIGN KEY eventos_ibfk_1,
ADD CONSTRAINT fk_organizador_evento FOREIGN KEY (OrganizadorID) REFERENCES Usuarios(id_usuario)
ON DELETE SET NULL
ON UPDATE CASCADE;
```

4.6.2 Eventos - Reservas: ON DELETE CASCADE, ON UPDATE CASCADE

- Si un evento es eliminado, todas las reservas asociadas a él se eliminarán automáticamente.
- Si un evento es actualizado, las reservas asociadas se actualizarán automáticamente.

```
-- 2. Eventos - Reservas: ON DELETE CASCADE, ON UPDATE CASCADE
-- Si un evento es eliminado, todas las reservas asociadas a él se eliminarán automáticamente.
-- Si un evento es actualizado, las reservas asociadas se actualizarán automáticamente.

-- Modificar la tabla Reservas
ALTER TABLE Reservas
DROP FOREIGN KEY reservas_ibfk_1,
ADD CONSTRAINT fk_evento_reserva FOREIGN KEY (evento_id) REFERENCES Eventos(id_evento)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

4.6.3 Usuarios - Reservas: ON DELETE CASCADE, ON UPDATE CASCADE

- Si un usuario (asistente) es eliminado, sus reservas asociadas también serán eliminadas.

- Si un usuario es actualizado, las reservas asociadas se actualizarán automáticamente.

```
-- 3. Usuarios - Reservas: ON DELETE CASCADE, ON UPDATE CASCADE
-- Si un usuario (asistente) es eliminado, sus reservas asociadas también serán eliminadas.
-- Si un usuario es actualizado, las reservas asociadas se actualizarán automáticamente.

-- Modificar la tabla Reservas
ALTER TABLE Reservas
DROP FOREIGN KEY reservas_ibfk_2,
ADD CONSTRAINT fk_asistente_reserva FOREIGN KEY (asistente_id) REFERENCES Usuarios(id_usuario)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

4.6.4 Pagos - Reservas: ON DELETE RESTRICT, ON UPDATE CASCADE

- No se puede eliminar un pago si está asociado a alguna reserva.
- Si un pago es actualizado, la reserva relacionada se actualizará automáticamente.

```
-- 4. Pagos - Reservas: ON DELETE RESTRICT, ON UPDATE CASCADE
-- No se puede eliminar un pago si está asociado a alguna reserva.
-- Si un pago es actualizado, la reserva relacionada se actualizará automáticamente.

-- Modificar la tabla Reservas
ALTER TABLE Reservas
DROP FOREIGN KEY reservas_ibfk_3,
ADD CONSTRAINT fk_pago_reserva FOREIGN KEY (id_pago) REFERENCES Pagos(id_pago)
ON DELETE RESTRICT
ON UPDATE CASCADE;
```

4.6.5 Usuarios - Usuarios_Salas: ON DELETE CASCADE, ON UPDATE CASCADE

- Si un usuario es eliminado, todas sus asociaciones con salas (en la tabla intermedia Usuarios_Salas) serán eliminadas.
- Si un usuario es actualizado, las asociaciones en Usuarios_Salas se actualizarán automáticamente.

```sql
-- 5. Usuarios - Usuarios_Salas: ON DELETE CASCADE, ON UPDATE CASCADE
-- Si un usuario es eliminado, todas sus asociaciones con salas (en la tabla intermedia Usuarios_Salas) serán eliminadas.
-- Si un usuario es actualizado, las asociaciones en Usuarios_Salas se actualizarán automáticamente.

-- Modificar la tabla Usuarios_Salas
ALTER TABLE Usuarios_Salas
DROP FOREIGN KEY usuarios_salas_ibfk_1,
ADD CONSTRAINT fk_usuario_sala FOREIGN KEY (id_usuario) REFERENCES Usuarios(id_usuario)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

6. Salas - Usuarios_Salas: ON DELETE CASCADE, ON UPDATE CASCADE

- Si una sala es eliminada, todas las asociaciones de usuarios con esa sala (en la tabla Usuarios_Salas) también serán eliminadas.
- Si una sala es actualizada, las asociaciones en Usuarios_Salas se actualizarán automáticamente.

```sql
-- 6. Salas - Usuarios_Salas: ON DELETE CASCADE, ON UPDATE CASCADE
-- Si una sala es eliminada, todas las asociaciones de usuarios con esa sala (en la tabla Usuarios_Salas) también serán eliminadas.
-- Si una sala es actualizada, las asociaciones en Usuarios_Salas se actualizarán automáticamente.

-- Modificar la tabla Usuarios_Salas
ALTER TABLE Usuarios_Salas
DROP FOREIGN KEY usuarios_salas_ibfk_2,
ADD CONSTRAINT fk_sala_usuario FOREIGN KEY (id_sala) REFERENCES Salas(id_sala)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

¿Cómo la integridad referencial previene la pérdida de datos?

- Cuando la tabla 1 tiene una clave foránea que hace referencia a la clave primaria de la tabla 2, la integridad referencial impide que se eliminen o modifiquen registros en la tabla 2 si existen registros en la tabla 1 que dependen de esos datos. Esto ayuda a prevenir la pérdida de datos al asegurarse de que los datos relacionados no se pierdan o queden huérfanos.
- En la restricción de eliminación si se intenta eliminar un registro de una tabla que está referenciado en otra, la base de datos no permitirá la eliminación, a menos que se use una acción específica como la "eliminación en cascada" (que borra todos los registros relacionados).

- En la restricción de actualización, cuando se actualiza una clave primaria en una tabla, la integridad referencial garantiza que todos los registros que referencian esa clave primaria en otras tablas también se actualicen automáticamente, evitando que se queden obsoletos o erróneos.

¿Cómo la integridad referencial mantiene la consistencia de los datos?

- Mediante la opción de actualización en cascada o eliminación en cascada, se puede mantener la consistencia de los datos. Si se cambia o elimina un valor en una tabla primaria, las tablas que contienen claves foráneas se actualizan o eliminan de manera automática, asegurando que los datos permanezcan consistentes.
- La integridad referencial evita la creación de datos huérfanos, es decir, registros que referencian valores inexistentes. Esto asegura que las relaciones entre entidades sean siempre válidas, lo que es importante para mantener la coherencia de los datos en una base de datos.

4. Seguridad, Auditoría y Control de Acceso

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

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

5.1 Implementación de políticas de acceso y seguridad.

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

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

En esta parte se establecieron cuatro roles los cuales son los siguientes:

```
CREATE ROLE 'Invitado';
CREATE ROLE 'Organizador';
CREATE ROLE 'Asistente';
CREATE ROLE 'Administrador';
```

Para la siguiente parte se procedió a establecer los permisos para cada rol, con sus respectivas restricciones o sus respectivos permisos:

```
-- El rol de Invitado: Solo le permite ver eventos y salas
GRANT SELECT ON gestion_conferencias.Eventos TO 'Invitado';
GRANT SELECT ON gestion_conferencias.Salas TO 'Invitado';

-- el rol de Organizador: le permite crear, modificar y eliminar eventos, y gestionar salas
GRANT ALL PRIVILEGES ON gestion_conferencias.Eventos TO 'Organizador';
GRANT ALL PRIVILEGES ON gestion_conferencias.Salas TO 'Organizador';

-- el rol de sistente: le permite realizar reservar eventos y ver su información
GRANT SELECT, INSERT, UPDATE ON gestion_conferencias.Reservas TO 'Asistente';
GRANT SELECT ON gestion_conferencias.Eventos TO 'Asistente';

-- y por ultimo el rol de dministrador: no tiene restricciones, tiene todos los permisos..
GRANT ALL PRIVILEGES ON gestion_conferencias.* TO 'Administrador';
```

1. Medidas Fundamentales de Seguridad

- **Cifrado de datos:** Nos permite proteger nuestros datos mediante el cifrado, cumpliendo con las normas y evitando así posibles fugas de datos.
- **Autenticación y autorización:** Mediante la implementación de contraseñas que sean robustas y seguras, y aplicando autentificación, se puede aplicar la asignación de roles y permisos a cada usuario
- **Firewall y segmentación:** Mediante la restricción de acceso haciendo uso de firewalls, la red puede ser aislada y reducir posibles riesgos en nuestros datos.
- **Gestión de vulnerabilidades:** Mantener los softwares actualizados y con sus respectivos parches, y realizar análisis con frecuencia.

5.2 Estrategias Avanzadas para Alta Disponibilidad

- **Replicación y failover:** Mediante la configuración de replicación de copias de seguridad en tiempo real. Permite la conmutación por error causado por algun fallo en el servidor principal.
- **Clustering:** Distribuye la carga de trabajo entre varios servidores. Aumenta la disponibilidad y evita interrupciones del servicio.
- **Balanceo de carga:** Distribuye el tráfico entre los servidores de la base de datos. Evita la sobrecarga y mejora el rendimiento.
- **Copias de seguridad y recuperación:** Realiza copias de seguridad periódicas. Prueba el proceso de recuperación para garantizar la restauración de datos.

5.3 Seguridad en la Replicación y el Failover

- **Cifrado de las comunicaciones:** Cifra las comunicaciones entre el servidor principal y las réplicas. Protege los datos en tránsito durante la replicación.
- **Autenticación mutua:** Implementa la autenticación mutua entre servidores. Asegura que solo los servidores autorizados puedan comunicarse.
- **Failover seguro:** Asegura que el proceso de failover sea seguro. Evita la pérdida de datos y el acceso no autorizado durante la conmutación.
- **Monitoreo y auditoría:** Monitorea la replicación y el failover. Registra todas las acciones para facilitar la investigación de incidentes.

5.4 **Medidas de Protección contra Ataques Específicos**

- **Inyección SQL:** Implementa medidas para prevenir ataques de inyección SQL. Valida las entradas y utiliza consultas parametrizadas.
- **Denegación de servicio (DoS):** Utiliza herramientas de protección contra DoS. Mitiga los ataques que intentan sobrecargar la base de datos.
- **Ataques de fuerza bruta:** Implementa políticas de contraseñas robustas. Bloquea las cuentas después de varios intentos fallidos.
- **Malware:** Utiliza software antivirus y antimalware. Protege la base de datos contra software malicioso.

5.5 Seguridad en el Nivel de Aplicación

- **Validación de entrada:** Valida todas las entradas de usuario. Evita ataques de inyección SQL y otros tipos de ataques.
- **Manejo de errores seguro:** Evita mostrar información sensible en los mensajes de error. Protege la información confidencial.
- **Seguridad de las API:** Protege las API con mecanismos de autenticación y autorización. Asegura el acceso seguro a la base de datos.
- **Conexiones seguras:** Utiliza conexiones cifradas para acceder a la base de datos. Protege los datos en tránsito.
-

5.6 Otros Aspectos a Considerar

- **Plan de respuesta a incidentes:** Desarrolla un plan para saber cómo actuar en caso de un ataque. Ten un protocolo claro y definido.
- **Formación y concienciación:** Forma a usuarios y administradores en seguridad. Crea una cultura de seguridad en la organización.

- **Auditoría:** Habilita la auditoría para registrar todas las acciones. Facilita la investigación de incidentes y el cumplimiento normativo.
- **Evaluación continua:** Evalúa y actualiza tus medidas de seguridad de forma regular. Mantente al día con las últimas amenazas.

5.6 Cifrado de datos sensibles usando AES_ENCRYPT en MySQL.

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

```
-- Implementación del cifrado de contraseñas
-- Se crea una columna en la tabla
ALTER TABLE Usuarios ADD COLUMN Contrasenia_encriptada VARBINARY(255);

-- Insertar un usuario con contraseña cifrada
INSERT INTO Usuarios (nombres_completos, correo, Telefono, Rol, Contrasenia, Contrasenia_encriptada)
VALUES ('Juan Pérez', 'juan.perez@mail.com', '123456789', 'Organizador', 'mi_password', AES_ENCRYPT('mi_password', 'clave_secreta'));

-- Recuperar y descifrar la contraseña para validación
SELECT nombres_completos, correo,
       AES_DECRYPT(Contrasenia_encriptada, 'clave_secreta') AS Contrasenia_Descifrada
FROM Usuarios;

-- Convertir el resultado a texto
-- Cambia la consulta para que MySQL convierta el resultado a CHAR:
SELECT nombres_completos, correo,
       CONVERT(AES_DECRYPT(Contrasenia_encriptada, 'clave_secreta') USING utf8) AS Contrasenia_Descifrada
FROM Usuarios;

-- Esta es la consulta general que muestra contraseña no encriptada y encriptada
SELECT id_usuario, nombres_completos, correo, Telefono, Rol, Contrasenia, Contrasenia_encriptada FROM Usuarios;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id_usuario	nombres_completos	correo	Telefono	Rol	Contrasenia	Contrasenia_encriptada
1	Juan Pérez	juan.perez@mail.com	123456789	Organizador	mi_password	BLOB
NULL	NULL	NULL	NULL	NULL	NULL	NULL

5.3 Habilitar auditoría y registrar eventos.

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

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.

```
SET GLOBAL general_log = 'ON';

SET GLOBAL slow_query_log = 'ON';
SET GLOBAL long_query_time = 2; -- Registra consultas que tarden más de 2 segundos

CREATE TABLE Auditoria (
   id_auditoria INT AUTO_INCREMENT PRIMARY KEY,
   id_usuario INT,
   accion VARCHAR(100),
   tabla_afectada VARCHAR(100),
   fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (id_usuario) REFERENCES Usuarios(id_usuario)
);

DELIMITER //
CREATE TRIGGER after_insert_eventos
AFTER INSERT ON Eventos
FOR EACH ROW
BEGIN
   INSERT INTO Auditoria (id_usuario, accion, tabla_afectada)
   VALUES (NEW.OrganizadorID, 'INSERT', 'Eventos');
END//
DELIMITER ;

DELIMITER //
CREATE TRIGGER after_update_eventos
AFTER UPDATE ON Eventos
FOR EACH ROW
```

```
BEGIN
   INSERT INTO Auditoria (id_usuario, accion, tabla_afectada)
   VALUES (NEW.OrganizadorID, 'UPDATE', 'Eventos');
END//
DELIMITER ;

DELIMITER //

CREATE TRIGGER after_delete_eventos
AFTER DELETE ON Eventos
FOR EACH ROW
BEGIN
   INSERT INTO Auditoria (id_usuario, accion, tabla_afectada)
   VALUES (OLD.OrganizadorID, 'DELETE', 'Eventos');
END//
DELIMITER ;
```

La configuración de herramientas de auditoría en MySQL y PostgreSQL es un aspecto fundamental para garantizar la seguridad y el cumplimiento normativo de bases de datos:

MySQL ofrece diversas herramientas y métodos para configurar la auditoría, adaptándose a tus necesidades específicas:

1. **MySQL Enterprise Audit:**
    - Esta es la solución más completa y robusta para la auditoría en MySQL.
    - Permite registrar una amplia gama de eventos, desde inicios de sesión y consultas hasta modificaciones de datos y cambios en la estructura de la base de datos.
    - Ofrece una configuración granular, permitiéndote seleccionar qué eventos auditar y cuáles ignorar.

- Los registros de auditoría se almacenan en un formato estandarizado y pueden ser analizados con herramientas específicas o integrados en tu sistema de gestión de logs.

Para habilitar MySQL Enterprise Audit, sigue estos pasos:

1. **Instala el plugin:** Asegúrate de que el plugin audit_log esté instalado en tu servidor MySQL.

2. **Configura el archivo de configuración:** Edita el archivo de configuración de MySQL (usualmente my.cnf o my.ini) y añade las siguientes líneas:

```
[mysqld]
audit_log_plugin=audit_log
audit_log_format=JSON
```

3. **Reinicia el servidor MySQL:** Para que los cambios surtan efecto.

4. **Define reglas de auditoría:** Utiliza sentencias SQL para especificar qué eventos deseas auditar. Por ejemplo, para auditar todas las consultas a la tabla usuarios:

```
CREATE AUDIT POLICY usuarios_policy
FOR SELECT ON usuarios;
```

5. **Habilita la política de auditoría:**

   SET AUDIT POLICY usuarios_policy ENABLE;

6. **Consulta los registros de auditoría:** Utiliza un visor de archivos de texto o una herramienta especializada para analizar los registros.

2. **Auditoría basada en triggers:**
   - Esta técnica te permite crear triggers que se ejecutan en respuesta a eventos específicos en la base de datos.
   - Puedes personalizar los triggers para registrar información detallada sobre los eventos, como el usuario que realizó la acción, la hora y los datos modificados.
   - Esta opción es más flexible que MySQL Enterprise Audit, pero requiere más configuración y mantenimiento.
3. **Log de consultas generales:**
   - Esta opción registra todas las consultas que se ejecutan en el servidor MySQL.
   - Es útil para depurar problemas y analizar el rendimiento, pero no es ideal para auditoría de seguridad debido a la gran cantidad de información que genera.

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

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

2. **Configurar respaldos incrementales.**
   Práctica: Realizar respaldos incrementales para reducir el tiempo y espacio de almacenamiento.
   Investigación: Investigar cómo realizar respaldos incrementales y cuándo es más conveniente utilizarlos.
   Importancia del Conocimiento: Los respaldos incrementales permiten optimizar los recursos y acelerar los tiempos de recuperación.
3. **Implementar respaldos en caliente (Hot Backups).**
   Práctica: Hacer respaldos sin interrumpir el servicio (por ejemplo, usando Percona XtraBackup).
   Investigación: Investigar cómo hacer respaldos sin detener la base de datos.
   Importancia del Conocimiento: Los respaldos en caliente son esenciales para bases de datos de producción que no pueden permitirse inactividad.

4. Optimización y Rendimiento de Consultas

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

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

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

1. Índice Hash

Los índices hash se basan en funciones hash que asignan valores de la clave a ubicaciones específicas de la tabla. Son extremadamente rápidos para búsquedas exactas, pero no son eficientes para consultas con rangos.

Ventajas:

- Rendimiento en búsquedas exactas: Las búsquedas basadas en una clave única son extremadamente rápidas (O(1) en promedio).
- Eficiencia en operaciones de igualdad: Cuando las consultas requieren encontrar un valor exacto, este índice es muy eficiente.

Desventajas:

- Sin soporte para rangos: No puede ser utilizado para consultas que requieran rangos de valores (como BETWEEN o >, <).
- Colisiones: En caso de una mala función hash o datos distribuidos de manera ineficiente, las colisiones pueden disminuir el rendimiento.

Es ideal para consultas de búsqueda exacta de claves, especialmente cuando no se requieren rangos.

2. Índice B-Tree (Árbol-B)

Es el tipo más común en bases de datos transaccionales y está basado en un árbol balanceado. Permite búsquedas, inserciones y eliminaciones rápidas, lo que lo convierte en una excelente opción para bases de datos con un alto volumen de transacciones.

Ventajas:

- **Eficiencia en las búsquedas:** Debido a su estructura balanceada, las búsquedas de valores específicos son rápidas.
- **Acceso a rangos:** Es ideal para consultas que buscan un rango de valores, ya que mantiene los datos en orden.
- **Actualizaciones rápidas:** Las inserciones y eliminaciones son eficientes y no requieren reestructuraciones costosas.

Desventajas:

- **Rendimiento en escrituras concurrentes:** Aunque es eficiente en escrituras, bajo condiciones de alta concurrencia, los bloqueos en el árbol B pueden afectar el rendimiento.

Uso:

Ideal para consultas de búsqueda por claves primarias o consultas con condiciones de rango (mayor que, menor que).

3. Índice de Árbol B+

El índice B+ es una variante del B-Tree, pero con la diferencia de que en el B+ el almacenamiento de los datos reales se realiza solo en las hojas del árbol, mientras que los nodos internos solo contienen claves.

Ventajas:

- **Búsquedas eficientes:** Al igual que los B-Trees, permite búsquedas eficientes en O(log n), pero también ofrece mejoras en la eficiencia de lectura debido a la estructura de solo hojas.
- **Lectura secuencial eficiente:** Las hojas están enlazadas, lo que facilita la lectura secuencial de los datos.

Desventajas:

- **Rendimiento en escrituras:** En algunos casos, el proceso de reequilibrado de los árboles puede afectar el rendimiento en operaciones de inserción y eliminación.

Uso:

Ideal para bases de datos que requieren búsquedas de rango y donde la eficiencia en la lectura secuencial es crítica.

4. Índice Compuesto (Índice Multicolumna)

Un índice compuesto utiliza varias columnas para crear una clave única de índice. Es útil cuando las consultas frecuentemente utilizan más de una columna en la cláusula WHERE.

Ventajas:

- **Optimiza consultas complejas:** Mejora el rendimiento de las consultas que filtran o ordenan por múltiples columnas.
- **Reducción de la necesidad de múltiples índices:** Puede reemplazar la necesidad de índices separados para cada columna.

Desventajas:

- **Costos de mantenimiento:** Si las columnas combinadas en el índice cambian con frecuencia, el índice puede ser costoso de mantener.
- **No siempre eficiente:** Si las consultas no utilizan todas las columnas en el índice, puede no ser tan eficiente como se espera.

Uso:

Ideal para consultas que involucran varias columnas en las cláusulas WHERE o en los criterios de ordenación.

5. Índice de Texto Completo

Los índices de texto completo se utilizan en bases de datos para consultas que requieren búsquedas en grandes bloques de texto, como búsqueda de palabras clave, coincidencia de frases o coincidencias parciales.

Ventajas:

- **Optimiza las búsquedas de texto completo:** Ideal para realizar búsquedas de cadenas o palabras dentro de campos de texto largo.
- **Soporta operadores avanzados:** Permite consultas como búsquedas de "aproximación", "coincidencias parciales", etc.

Desventajas:

- **Consumo de espacio:** Requiere más espacio en disco para almacenar los índices invertidos y puede afectar el rendimiento de las escrituras.
- **Complejidad en el mantenimiento:** Mantener este tipo de índice actualizado puede ser costoso si se realizan muchas modificaciones en los datos.

Uso:

Ideal para bases de datos que manejan grandes volúmenes de datos textuales, como blogs, artículos o documentos completos.

6. Índice de GiST (Generalized Search Tree)

Los índices GiST son una estructura de datos generalizada que puede ser utilizada para diversas aplicaciones, como búsquedas espaciales o consultas más complejas. Son muy flexibles y se utilizan, por ejemplo, para la indexación de datos geoespaciales.

Ventajas:

- **Flexibilidad:** Puede ser usado en una variedad de aplicaciones, desde búsqueda de texto hasta consultas geoespaciales.
- **Optimización en búsquedas complejas:** Permite crear índices personalizados para casos específicos.

Desventajas:

- **Complejidad:** La implementación y mantenimiento son más complejos comparados con otros índices.
- **Costo de espacio y rendimiento:** Dependiendo de la configuración, pueden consumir mucho espacio y afectar el rendimiento de las escrituras.

Uso:

Ideal para aplicaciones que requieren consultas complejas, como datos geoespaciales, textos o multidimensionales.

Impacto en el Rendimiento

1. Los índices mejoran la velocidad de lectura, pero pueden ralentizar las operaciones de escritura debido a que deben actualizarse cada vez que los datos cambian. Es especialmente importante en bases de datos transaccionales con un alto volumen de inserciones, eliminaciones o actualizaciones.
2. **Consumo de espacio:** Los índices requieren almacenamiento adicional. Es fundamental considerar el costo de almacenamiento, especialmente en bases de datos con grandes volúmenes de datos.
3. **Concurrencia:** En sistemas con alta concurrencia, los índices pueden afectar el rendimiento debido a los bloqueos y la necesidad de sincronización. Algunas estrategias, como el uso de índices de solo lectura o el ajuste de la granularidad de los bloqueos, pueden ayudar a mitigar este problema.
4. **Elección del índice:** Elegir el índice adecuado depende de las consultas que más se realizan en la base de datos. Un índice mal elegido puede no proporcionar beneficios y puede incluso degradar el rendimiento.

4.1 Optimización de consultas SQL

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

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

Práctica: Aplicación de 3 join

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

1. Uso eficiente de las uniones (JOIN)
Seleccionar el tipo adecuado de JOIN, ya que existen varios tipos de uniones (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN, etc.).

- Se usa INNER JOIN cuando se necesite combinar solo las filas que tienen coincidencias en ambas tablas.

- Se usa LEFT JOIN (o RIGHT JOIN) solo si se necesitan los registros de la tabla izquierda (o derecha) aunque no haya coincidencia en la otra tabla.
- Se debe evitar usar FULL OUTER JOIN a menos que realmente sea necesario, ya que puede ser costoso si las tablas son grandes.
- En el orden de las tablas en JOIN se asegura de colocar primero la tabla con menos filas (cuando sea posible) para reducir el número de comparaciones que hace el sistema de gestión de bases de datos.

2. Uso de Subconsultas

- **Subconsultas en el SELECT**: Si se necesita calcular un valor para cada fila (por ejemplo, un promedio o una suma) y no se puede hacerlo en una sola tabla, se usa una subconsulta en el SELECT. Sin embargo, se debe evitar subconsultas en columnas si estas pueden reemplazarse por JOINs o funciones agregadas.
- **Subconsultas en el WHERE**: Las subconsultas en el WHERE pueden ser útiles para restringir resultados basados en un conjunto de datos. Pero si es posible, se usa JOINs en lugar de subconsultas en el WHERE, ya que las subconsultas pueden ser más lentas.
- **Subconsultas en el FROM**: Si la subconsulta se usa para generar una tabla temporal que se une con otras tablas, es más eficiente hacerlo en el FROM que en el WHERE.

3. Optimización de consultas complejas

- Asegurarse de tener índices en las columnas que se usan para filtrar (WHERE), ordenar (ORDER BY), y unir tablas (JOIN). Esto puede reducir drásticamente el tiempo de ejecución de la consulta.
- Evita usar SELECT *, ya que puede devolver más columnas de las que realmente se necesita. Siempre se especifica las columnas necesarias.
- Si solo se necesita un número limitado de resultados, se usa LIMIT o TOP para restringir la cantidad de filas devueltas.
- Si se tiene consultas complejas que se repiten, se considera crear vistas. Esto permite simplificar la consulta y, en algunos casos, mejorar el rendimiento.
- Si el rendimiento es crítico y las tablas son muy grandes, a veces es útil desnormalizar los datos. Sin embargo, esto puede llevar a redundancia y a más trabajo de mantenimiento.
- Si una subconsulta se puede reemplazar con un JOIN o si la subconsulta devuelve muchos resultados que no se necesitan, puede causar que la consulta se ejecute más lentamente.

1. **Utilizar particionamiento de tablas.**

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

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

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

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

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

Actividades:

1. **Crear procedimientos almacenados.**

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

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

   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.**

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

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

   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.

3. **Implementar triggers para auditoría y control de cambios.**

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

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

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

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

6. Monitoreo y Optimización de Recursos

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

Actividades:

1. **Monitorear el rendimiento de consultas.**
   Práctica: Usar herramientas como SHOW PROCESSLIST para detectar consultas lentas y optimizarlas.
   Investigación: Investigar las mejores prácticas para monitorear el rendimiento de las consultas en producción.
   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.

7.1 Link del repositorio de GitHub:

https://github.com/dannamishelle23/PROYECTO-FINAL-BDD.git

7.2 Commits en Github:

1. **Automatización de pruebas con GitHub Actions.**

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

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

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

```
/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_roles.sql
      cifrado_datos.sql
    /Auditoria
      activar_auditoria.sql
    /Optimización
      crear_indices.sql
      optimizar_consultas.sql
  README.md
```

EXPLICACIÓN

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**