Universidad Peruana Los Andes
Ingeniería de Sistemas y Computación
Implementación de una base de datos

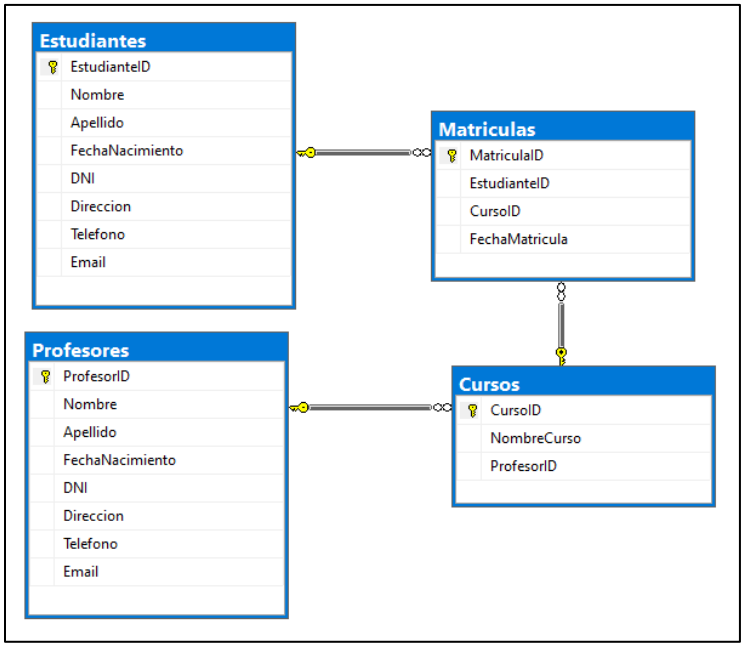
Curso: Base de Datos II
Docente: Raul Enrique Fernandez Bejarano
Alumno: Mosquera Zevallos Valerio

Descripción de la Base de Datos

La base de datos está diseñada para gestionar de manera eficiente la información de estudiantes, profesores, cursos e inscripciones en un instituto educativo. Esta base de datos resuelve el problema de **centralizar y organizar los datos académicos**, facilitando el seguimiento de las matrículas, cursos y profesores.

1. **Estudiantes**: Almacena los datos personales de los estudiantes, como nombre, DNI, dirección, teléfono y correo electrónico.
2. **Profesores**: Contiene los datos personales de los profesores y su vínculo con los cursos que dictan.
3. **Cursos**: Registra los cursos ofrecidos y su relación con los profesores encargados.
4. **Matrículas**: Gestiona las inscripciones de los estudiantes a los cursos, incluyendo la fecha de matrícula.

Diagrama



---

1) Vista con tres tablas

Crear una vista que combine información de tres tablas (Estudiantes, Profesores y Cursos). Consulta SQL que extraiga los datos relevantes de cada tabla y luego usarla como una vista.

```
CREATE VIEW VistaEstudiantesCursos AS
SELECT
    e.EstudianteID,
    e.Nombre AS EstudianteNombre,
    e.Apellido AS EstudianteApellido,
    e.DNI AS EstudianteDNI,
    c.NombreCurso,
    p.Nombre AS ProfesorNombre,
    p.Apellido AS ProfesorApellido,
    m.FechaMatricula
FROM
    Matriculas m
JOIN
    Estudiantes e ON m.EstudianteID = e.EstudianteID
JOIN
    Cursos c ON m.CursoID = c.CursoID
JOIN
    Profesores p ON c.ProfesorID = p.ProfesorID;
```

Explicación:

El código combina información de tres tablas: Estudiantes, Cursos y Profesores.

- **Objetivo**: Mostrar estudiantes, los cursos en los que están matriculados y los profesores a cargo.
- **JOINs**: Relaciona las tablas mediante claves foráneas:
  - Matriculas se une con Estudiantes y Cursos.
  - Cursos se une con Profesores.
- **Resultado**: Devuelve una lista con el nombre del estudiante, curso, profesor y la fecha de matrícula.

---

2) Procedimiento Almacenado para Insertar Datos

Vamos a crear un procedimiento almacenado que facilite la inserción de datos en la tabla de **estudiantes**.

```sql
-- Procedimiento almacenado para insertar un estudiante
CREATE PROCEDURE InsertarEstudiante
    @Nombre NVARCHAR(100),
    @Apellido NVARCHAR(100),
    @FechaNacimiento DATE,
    @DNI NVARCHAR(20),
    @Direccion NVARCHAR(200),
    @Telefono NVARCHAR(15),
    @Email NVARCHAR(100)
AS
BEGIN
    INSERT INTO Estudiantes (Nombre, Apellido, FechaNacimiento, DNI, Direccion, Telefono, Email)
    VALUES (@Nombre, @Apellido, @FechaNacimiento, @DNI, @Direccion, @Telefono, @Email);
END;
GO

-- Procedimiento almacenado para insertar un profesor
CREATE PROCEDURE InsertarProfesor
    @Nombre NVARCHAR(100),
    @Apellido NVARCHAR(100),
    @FechaNacimiento DATE,
    @DNI NVARCHAR(20),
    @Direccion NVARCHAR(200),
    @Telefono NVARCHAR(15),
    @Email NVARCHAR(100)
AS
BEGIN
    INSERT INTO Profesores (Nombre, Apellido, FechaNacimiento, DNI, Direccion, Telefono, Email)
    VALUES (@Nombre, @Apellido, @FechaNacimiento, @DNI, @Direccion, @Telefono, @Email);
END;
GO
```

```sql
-- Procedimiento almacenado para insertar un curso
CREATE PROCEDURE InsertarCurso
    @NombreCurso NVARCHAR(100),
    @ProfesorID INT
AS
BEGIN
    INSERT INTO Cursos (NombreCurso, ProfesorID)
    VALUES (@NombreCurso, @ProfesorID);
END;
GO

-- Procedimiento almacenado para insertar una matrícula
CREATE PROCEDURE InsertarMatricula
    @EstudianteID INT,
    @CursoID INT,
    @FechaMatricula DATE
AS
BEGIN
    INSERT INTO Matriculas (EstudianteID, CursoID, FechaMatricula)
    VALUES (@EstudianteID, @CursoID, @FechaMatricula);
END;
GO
```

Explicación:

✓ **InsertarEstudiante**:

Permite insertar un nuevo registro en la tabla Estudiantes. Los parámetros @Nombre, @Apellido, @FechaNacimiento, @DNI, @Direccion, @Telefono, y @Email se pasan al procedimiento y luego se usan para insertar los valores en la tabla.

✓ **InsertarProfesor**:

Inserta un nuevo profesor en la tabla Profesores con los mismos parámetros que el procedimiento anterior, pero para los datos del profesor.

✓ **InsertarCurso**:

Inserta un nuevo curso en la tabla Cursos. Este procedimiento toma el nombre del curso y el ProfesorID (que debe existir en la tabla Profesores) para asignar el curso a un profesor.

✓ **InsertarMatricula**:

Permite insertar una matrícula de un estudiante a un curso. Toma el EstudianteID, CursoID y FechaMatricula como parámetros y crea un nuevo registro en la tabla Matriculas.

---

3) Procedimiento Almacenado para Eliminar Datos

Vamos a crear un procedimiento almacenado que facilite la **eliminación** de datos en la tabla de **estudiantes**.

```
-- Procedimiento almacenado para eliminar un estudiante
CREATE PROCEDURE EliminarEstudiante
    @EstudianteID INT
AS
BEGIN
    -- Eliminar las matrículas del estudiante antes de eliminar el estudiante
    DELETE FROM Matriculas WHERE EstudianteID = @EstudianteID;

    -- Eliminar el estudiante
    DELETE FROM Estudiantes WHERE EstudianteID = @EstudianteID;
END;
GO

-- Procedimiento almacenado para eliminar un profesor
CREATE PROCEDURE EliminarProfesor
    @ProfesorID INT
AS
BEGIN
    -- Eliminar los cursos asignados al profesor antes de eliminar el profesor
    DELETE FROM Cursos WHERE ProfesorID = @ProfesorID;

    -- Eliminar el profesor
    DELETE FROM Profesores WHERE ProfesorID = @ProfesorID;
END;
GO
```

```sql
-- Procedimiento almacenado para eliminar un curso
CREATE PROCEDURE EliminarCurso
    @CursoID INT
AS
BEGIN
    -- Eliminar las matrículas asociadas al curso antes de eliminar el curso
    DELETE FROM Matriculas WHERE CursoID = @CursoID;

    -- Eliminar el curso
    DELETE FROM Cursos WHERE CursoID = @CursoID;
END;
GO

-- Procedimiento almacenado para eliminar una matrícula
CREATE PROCEDURE EliminarMatricula
    @MatriculaID INT
AS
BEGIN
    -- Eliminar la matrícula
    DELETE FROM Matriculas WHERE MatriculaID = @MatriculaID;
END;
GO
```

Explicación:

1. **EliminarEstudiante**:
    - Este procedimiento elimina un estudiante de la tabla Estudiantes.
    - **Paso 1**: Antes de eliminar al estudiante, se eliminan las matrículas asociadas en la tabla Matriculas utilizando el EstudianteID.
    - **Paso 2**: Luego, se elimina el registro del estudiante en la tabla Estudiantes.

2. **EliminarProfesor**:
    - Este procedimiento elimina un profesor de la tabla Profesores.
    - **Paso 1**: Antes de eliminar al profesor, se eliminan los cursos que estén asignados a ese profesor en la tabla Cursos.
    - **Paso 2**: Luego, se elimina el registro del profesor en la tabla Profesores.

3. **EliminarCurso**:
    - Este procedimiento elimina un curso de la tabla Cursos.
    - **Paso 1**: Se eliminan las matrículas asociadas al curso en la tabla Matriculas utilizando el CursoID.

- **Paso 2**: Luego, se elimina el curso de la tabla Cursos.

4. **EliminarMatricula**:
   - Este procedimiento elimina una matrícula de la tabla Matriculas.
   - Utiliza el MatriculaID para identificar y eliminar el registro correspondiente.

---

4) Procedimiento Almacenado para Actualizar Datos

Vamos a crear un procedimiento almacenado que facilite la **actualizacion** de datos en la tabla de **estudiantes**.

```
-- Procedimiento almacenado para actualizar los datos de un estudiante
CREATE PROCEDURE ActualizarEstudiante
    @EstudianteID INT,
    @Nombre NVARCHAR(100),
    @Apellido NVARCHAR(100),
    @FechaNacimiento DATE,
    @DNI NVARCHAR(20),
    @Direccion NVARCHAR(200),
    @Telefono NVARCHAR(15),
    @Email NVARCHAR(100)
AS
BEGIN
    UPDATE Estudiantes
    SET
        Nombre = @Nombre,
        Apellido = @Apellido,
        FechaNacimiento = @FechaNacimiento,
        DNI = @DNI,
        Direccion = @Direccion,
        Telefono = @Telefono,
        Email = @Email
    WHERE EstudianteID = @EstudianteID;
END;
GO
```

```sql
-- Procedimiento almacenado para actualizar los datos de un profesor
CREATE PROCEDURE ActualizarProfesor
    @ProfesorID INT,
    @Nombre NVARCHAR(100),
    @Apellido NVARCHAR(100),
    @FechaNacimiento DATE,
    @DNI NVARCHAR(20),
    @Direccion NVARCHAR(200),
    @Telefono NVARCHAR(15),
    @Email NVARCHAR(100)
AS
BEGIN
    UPDATE Profesores
    SET
        Nombre = @Nombre,
        Apellido = @Apellido,
        FechaNacimiento = @FechaNacimiento,
        DNI = @DNI,
        Direccion = @Direccion,
        Telefono = @Telefono,
        Email = @Email
    WHERE ProfesorID = @ProfesorID;
END;
GO
```

```sql
-- Procedimiento almacenado para actualizar los datos de un curso
CREATE PROCEDURE ActualizarCurso
    @CursoID INT,
    @NombreCurso NVARCHAR(100),
    @ProfesorID INT
AS
BEGIN
    UPDATE Cursos
    SET
        NombreCurso = @NombreCurso,
        ProfesorID = @ProfesorID
    WHERE CursoID = @CursoID;
END;
GO

-- Procedimiento almacenado para actualizar los datos de una matrícula
CREATE PROCEDURE ActualizarMatricula
    @MatriculaID INT,
    @EstudianteID INT,
    @CursoID INT,
    @FechaMatricula DATE
AS
```

```
BEGIN
    UPDATE Matriculas
    SET
        EstudianteID = @EstudianteID,
        CursoID = @CursoID,
        FechaMatricula = @FechaMatricula
    WHERE MatriculaID = @MatriculaID;
END;
GO
```

Explicación:

1. **ActualizarEstudiante**:
    - Este procedimiento permite actualizar los datos de un estudiante en la tabla Estudiantes.
    - Los parámetros del procedimiento son: @EstudianteID, @Nombre, @Apellido, @FechaNacimiento, @DNI, @Direccion, @Telefono, y @Email.
    - La instrucción UPDATE actualiza los datos del estudiante con el EstudianteID correspondiente.
2. **ActualizarProfesor**:
    - Este procedimiento permite actualizar los datos de un profesor en la tabla Profesores.
    - Los parámetros son: @ProfesorID, @Nombre, @Apellido, @FechaNacimiento, @DNI, @Direccion, @Telefono, y @Email.
    - Se utiliza UPDATE para modificar los valores en la tabla Profesores basándose en el ProfesorID.
3. **ActualizarCurso**:
    - Este procedimiento permite actualizar los datos de un curso en la tabla Cursos.
    - Los parámetros son: @CursoID, @NombreCurso, y @ProfesorID (la relación entre curso y profesor).
    - El UPDATE modifica los valores de NombreCurso y ProfesorID en el curso correspondiente.
4. **ActualizarMatricula**:
    - Este procedimiento permite actualizar los datos de una matrícula en la tabla Matriculas.
    - Los parámetros son: @MatriculaID, @EstudianteID, @CursoID, y @FechaMatricula.
    - La instrucción UPDATE actualiza los registros de matrícula según el MatriculaID dado.

5) Procedimiento almacenado para realizar cálculos matemáticos de una columna de su BD.

En este caso, el procedimiento almacenado calculará la edad de un estudiante utilizando su fecha de nacimiento, que está almacenada en la columna FechaNacimiento de la tabla Estudiantes.

```sql
-- Procedimiento almacenado para calcular la edad de un estudiante
CREATE PROCEDURE CalcularEdadEstudiante
    @EstudianteID INT
AS
BEGIN
    DECLARE @FechaNacimiento DATE;
    DECLARE @Edad INT;

    -- Obtener la fecha de nacimiento del estudiante
    SELECT @FechaNacimiento = FechaNacimiento
    FROM Estudiantes
    WHERE EstudianteID = @EstudianteID;

    -- Calcular la edad del estudiante basado en la fecha de nacimiento
    SET @Edad = DATEDIFF(YEAR, @FechaNacimiento, GETDATE())
                - CASE
                    WHEN MONTH(@FechaNacimiento) > MONTH(GETDATE())
                         OR (MONTH(@FechaNacimiento) = MONTH(GETDATE()) AND DAY(@FechaNacimiento) > DAY(GETDATE()))
                    THEN 1
                    ELSE 0
                END;

    -- Devolver el resultado
    SELECT @Edad AS Edad;
END;
GO
```

Explicación del script

1. **Parámetros del Procedimiento**:
   - El procedimiento CalcularEdadEstudiante recibe un parámetro @EstudianteID, que es el identificador único del estudiante cuya edad queremos calcular.
2. **Obtener la Fecha de Nacimiento**:
   - Se usa una consulta SELECT para obtener la fecha de nacimiento (FechaNacimiento) del estudiante que coincide con el EstudianteID proporcionado.
3. **Cálculo de la Edad**:
   - El cálculo de la edad se realiza utilizando la función DATEDIFF(YEAR, @FechaNacimiento, GETDATE()), que calcula la diferencia en años entre la fecha de nacimiento y la fecha actual.
   - Sin embargo, para ajustar correctamente la edad cuando la fecha de cumpleaños aún no ha ocurrido este año, se usa una cláusula CASE para restar 1 si el cumpleaños aún no ha pasado.
4. **Devolver el Resultado**:
   - El resultado del cálculo de la edad se devuelve con la instrucción SELECT @Edad AS Edad.

6) Disparador para ingresar un registro automáticamente en una tabla de su BD

Paso 01: Crear la Tabla de Auditoría

```
CREATE TABLE AuditoriaEstudiantes (
    AuditoriaID INT PRIMARY KEY IDENTITY(1,1),
    EstudianteID INT,
    Nombre NVARCHAR(100),
    Apellido NVARCHAR(100),
    FechaInscripcion DATE,
    FechaRegistro DATETIME DEFAULT GETDATE()
);
GO
```

Paso 02: Crear el Disparador (Trigger)

```
-- Crear un disparador para insertar un registro en la tabla de auditoría
CREATE TRIGGER Trigger_AuditoriaEstudiantes
ON Estudiantes
AFTER INSERT
AS
BEGIN
    -- Insertar un registro en la tabla AuditoriaEstudiantes después de que se inserte un nuevo estudiante
    INSERT INTO AuditoriaEstudiantes (EstudianteID, Nombre, Apellido, FechaInscripcion)
    SELECT EstudianteID, Nombre, Apellido, FechaNacimiento
    FROM inserted;
END;
GO
```

Explicación

1. **Creación d e la tabla de auditoría**:
    - La tabla AuditoriaEstudiantes almacena el EstudianteID, Nombre, Apellido, FechaInscripcion (la fecha de nacimiento del estudiante en este caso) y FechaRegistro (la fecha en que el registro fue insertado en la tabla de auditoría).
2. **Disparador Trigger_AuditoriaEstudiantes**:
    - Este disparador se ejecuta **después de** que se inserte un nuevo registro en la tabla Estudiantes (AFTER INSERT).
    - Dentro del disparador, utilizamos la tabla inserted, que es una tabla virtual que contiene las filas recién insertadas en la tabla Estudiantes.
    - Los valores de las columnas EstudianteID, Nombre, Apellido, y FechaNacimiento se extraen de la tabla inserted y se insertan en la tabla AuditoriaEstudiantes.

---

7) Disparador para ELIMINE un registro AUTOMÁTICAMENTE en una tabla de su base de datos

Se implementará un disparador que elimina automáticamente un registro relacionado en la tabla Matriculas cuando se elimina un estudiante de la tabla Estudiantes. Este enfoque garantiza la consistencia referencial y evita registros huérfanos.

```
-- 7) Crear un disparador que elimina automáticamente un registro relacionado en la tabla Matriculas
CREATE TRIGGER EliminarMatriculasAlBorrarEstudiante
ON Estudiantes
AFTER DELETE
AS
BEGIN
    -- Eliminar las matrículas relacionadas con los estudiantes eliminados
    DELETE FROM Matriculas
    WHERE EstudianteID IN (SELECT EstudianteID FROM DELETED);
END;
GO
```

Explicación del disparador:

1. **Evento activador**:
    - Se activa **después de que se elimine** un registro en la tabla Estudiantes.
2. **Uso de la tabla DELETED**:
    - SQL Server crea una tabla virtual llamada DELETED que contiene los registros eliminados durante la operación DELETE.
    - El disparador accede a esta tabla para identificar los estudiantes eliminados (EstudianteID).
3. **Elimina registros relacionados**:
    - Utiliza una consulta DELETE en la tabla Matriculas para eliminar todas las matrículas donde el EstudianteID coincide con los estudiantes eliminados.

---

8) Disparador para ACTUALICE un registro automáticamente en una tabla de su BD.

Implementaremos un disparador que **actualice automáticamente un registro en otra tabla** cuando se actualice un registro en la tabla Estudiantes. En este caso, asumimos que existe una tabla AuditoriaEstudiantes donde se almacenan registros históricos de cambios.

```sql
--8) Disparador que actualice automáticamente un registro en otra
--    tabla cuando se actualice un registro en la tabla Estudiantes
-- Crear la tabla AuditoriaEstudiantes para registrar actualizaciones
CREATE TABLE AuditoriaEstudiantes (
    AuditoriaID INT PRIMARY KEY IDENTITY(1,1),
    EstudianteID INT,
    NombreAnterior NVARCHAR(100),
    ApellidoAnterior NVARCHAR(100),
    NombreNuevo NVARCHAR(100),
    ApellidoNuevo NVARCHAR(100),
    FechaActualizacion DATETIME
);
GO
```

```sql
-- Crear el disparador para actualizar la tabla AuditoriaEstudiantes
CREATE TRIGGER ActualizarAuditoriaEstudiantes
ON Estudiantes
AFTER UPDATE
AS
BEGIN
    -- Insertar un registro en la tabla AuditoriaEstudiantes después de una actualización
    INSERT INTO AuditoriaEstudiantes (EstudianteID, NombreAnterior, ApellidoAnterior, NombreNuevo, ApellidoNuevo, FechaActualizacion)
    SELECT
        DELETED.EstudianteID,
        DELETED.Nombre AS NombreAnterior,
        DELETED.Apellido AS ApellidoAnterior,
        INSERTED.Nombre AS NombreNuevo,
        INSERTED.Apellido AS ApellidoNuevo,
        GETDATE() AS FechaActualizacion
    FROM
        DELETED
    INNER JOIN
        INSERTED ON DELETED.EstudianteID = INSERTED.EstudianteID;
END;
GO
```

Explicación del disparador

1. **Evento activador**:
    - Este disparador se ejecuta **después de que se actualice un registro** en la tabla Estudiantes.
2. **Uso de las tablas virtuales INSERTED y DELETED**:
    - La tabla DELETED contiene los valores antes de la actualización.
    - La tabla INSERTED contiene los valores después de la actualización.
3. **Inserción en AuditoriaEstudiantes**:
    - Cada vez que se actualiza un estudiante, se registra el cambio en la tabla AuditoriaEstudiantes, guardando:
        - El ID del estudiante (EstudianteID).

- El nombre y apellido antes y después de la actualización.
- La fecha de la actualización (GETDATE()).

---

9) Disparador para VERIFICAR el control de datos.

Implementación de un disparador que **verifique el control de datos** que garantizará que al intentar insertar un registro en la tabla Matriculas, se cumplan las siguientes condiciones:

1. **Validación de la existencia del estudiante y el curso**:

   Antes de insertar un registro, se verifica si tanto el EstudianteID como el CursoID existen en las tablas Estudiantes y Cursos, respectivamente.

2. **Restricción de duplicados**:

   Se evita que un estudiante se matricule más de una vez en el mismo curso.

```
--9) Crear el disparador para verificar el control de datos en la tabla Matriculas
CREATE TRIGGER VerificarMatriculas
ON Matriculas
INSTEAD OF INSERT
AS
BEGIN
    -- Verificar que el estudiante y el curso existan
    IF EXISTS (
        SELECT 1
        FROM INSERTED AS I
        LEFT JOIN Estudiantes AS E ON I.EstudianteID = E.EstudianteID
        LEFT JOIN Cursos AS C ON I.CursoID = C.CursoID
        WHERE E.EstudianteID IS NULL OR C.CursoID IS NULL
    )
    BEGIN
        RAISERROR ('El EstudianteID o CursoID no existe.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;
```

```
    -- Verificar que no exista una matrícula duplicada
    IF EXISTS (
        SELECT 1
        FROM Matriculas AS M
        INNER JOIN INSERTED AS I ON M.EstudianteID = I.EstudianteID AND M.CursoID = I.CursoID
    )
    BEGIN
        RAISERROR ('El estudiante ya está matriculado en este curso.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    -- Insertar el registro si todas las validaciones pasan
    INSERT INTO Matriculas (EstudianteID, CursoID, FechaMatricula)
    SELECT EstudianteID, CursoID, FechaMatricula
    FROM INSERTED;
END;
GO
```

Explicación del disparador

1. **Evento activador**:

Se ejecuta cuando se intenta insertar un nuevo registro en la tabla Matriculas.

2. **Validación de existencia**:

- Utiliza un LEFT JOIN para comprobar si el EstudianteID y el CursoID existen en las tablas correspondientes.
- Si alguna de estas claves no existe, se genera un error con RAISERROR y se cancela la transacción con ROLLBACK.

3. **Control de duplicados**:

- Se verifica si ya existe una matrícula para el mismo EstudianteID y CursoID.
- Si se encuentra un duplicado, se lanza un error y se cancela la transacción.

4. **Inserción final**:

- Solo si ambas validaciones son exitosas, se realiza la inserción en la tabla Matriculas.

---

10) Utilizando Script Crear 03 usuarios con nombres de sus compañeros y uno suyo

Implementación de un script que crea tres usuarios para una base de datos SQL Server utilizando los nombres: Luis Aquino (Mi compañero), Andy Aimituma (Mi compañero), Valerio Mosquera (Yo)

```sql
--10) Crear 03 usuarios
-- Crear inicios de sesión a nivel del servidor para los usuarios
CREATE LOGIN LuisAquino WITH PASSWORD = 'ContraseñaSegura123!';
CREATE LOGIN AndyAimituma WITH PASSWORD = 'ContraseñaSegura123!';
CREATE LOGIN ValerioMosquera WITH PASSWORD = 'ContraseñaSegura123!';
GO

-- Cambiar al contexto de la base de datos ColegioDB
USE ColegioDB;
GO

-- Crear usuarios en la base de datos asignados a los inicios de sesión
CREATE USER LuisAquino FOR LOGIN LuisAquino;
CREATE USER AndyAimituma FOR LOGIN AndyAimituma;
CREATE USER ValerioMosquera FOR LOGIN ValerioMosquera;
GO

-- Opcional: Asignar permisos específicos a los usuarios
-- Por ejemplo, dar permisos de lectura y escritura
ALTER ROLE db_datareader ADD MEMBER LuisAquino;
ALTER ROLE db_datawriter ADD MEMBER LuisAquino;

ALTER ROLE db_datareader ADD MEMBER AndyAimituma;
ALTER ROLE db_datawriter ADD MEMBER AndyAimituma;

ALTER ROLE db_datareader ADD MEMBER ValerioMosquera;
ALTER ROLE db_datawriter ADD MEMBER ValerioMosquera;
GO
```

Explicación:

1. **Crear inicios de sesión (CREATE LOGIN)**:
   - Los inicios de sesión son entidades a nivel de servidor que permiten autenticar a los usuarios.
   - Cada usuario tiene una contraseña segura (en este caso, ContraseñaSegura123!).
2. **Cambiar al contexto de la base de datos (USE ColegioDB)**:
   - Esto asegura que los usuarios creados se asocien a la base de datos ColegioDB.
3. **Crear usuarios en la base de datos (CREATE USER)**:
   - Los usuarios de la base de datos se enlazan a los inicios de sesión creados previamente.
4. **Asignar roles a los usuarios (ALTER ROLE)**:
   - db_datareader: Permite leer datos de todas las tablas.
   - db_datawriter: Permite insertar, actualizar y eliminar datos de todas las tablas.

```sql
SELECT name AS Usuario, type_desc AS Tipo
FROM sys.database_principals
WHERE type IN ('S', 'U') AND name NOT LIKE '##%';
```

106 %

Results | Messages

	Usuario	Tipo
1	dbo	WINDOWS_USER
2	guest	SQL_USER
3	INFORMATION_SCHEMA	SQL_USER
4	sys	SQL_USER
5	NT AUTHORITY\SYSTEM	WINDOWS_USER
6	LuisAquino	SQL_USER
7	AndyAimituma	SQL_USER
8	ValerioMosquera	SQL_USER

11) Utilizando un script, copiar la base de datos (creada anteriormente) y compartir en cada uno de los usuarios

11.1 Crear una Copia de la Base de Datos ColegioDB

El primer paso consiste en crear una copia exacta de la base de datos original (ColegioDB). Este proceso utiliza las instrucciones BACKUP y RESTORE para realizar una copia llamada ColegioDB_Copy.

```
--11) Copiar la Base de Datos y Compartirla con Usuarios
-- Crear una copia de la base de datos ColegioDB como ColegioDB_Copy
USE master;
GO

-- Verificar si ya existe la base de datos de copia
IF DB_ID('ColegioDB_Copy') IS NOT NULL
BEGIN
    DROP DATABASE ColegioDB_Copy; -- Eliminar si ya existe
END
GO
```

```
-- Crear la copia de la base de datos mediante backup y restore
BACKUP DATABASE ColegioDB
TO DISK = 'C:\Temp\ColegioDB.bak';
GO

RESTORE DATABASE ColegioDB_Copy
FROM DISK = 'C:\Temp\ColegioDB.bak'
WITH MOVE 'ColegioDB' TO 'C:\Temp\ColegioDB_Copy.mdf',
     MOVE 'ColegioDB_log' TO 'C:\Temp\ColegioDB_Copy.ldf',
     REPLACE;
GO
```

Explicación:

1. **Verificar existencia de la base de datos:** El comando IF DB_ID('ColegioDB_Copy') IS NOT NULL asegura que no exista una copia previa para evitar conflictos.

2. **Respaldo de la base original:** BACKUP DATABASE genera un archivo .bak que contiene una copia de seguridad.

3. **Restauración de la base de datos:** RESTORE DATABASE crea la copia ColegioDB_Copy usando los archivos de respaldo generados.

11.2 Configurar Usuarios y Permisos

Después de crear la copia de la base de datos, se deben asociar los usuarios con la base de datos copiada y asignarles permisos para leer y escribir.

```
-- Crear usuarios para la base de datos copia
CREATE USER LuisAquino FOR LOGIN LuisAquino;
CREATE USER AndyAimituma FOR LOGIN AndyAimituma;
CREATE USER ValerioMosquera FOR LOGIN ValerioMosquera;
GO

-- Asignar roles y permisos
-- Permitir leer y escribir datos en la base de datos copia
ALTER ROLE db_datareader ADD MEMBER LuisAquino;
ALTER ROLE db_datawriter ADD MEMBER LuisAquino;

ALTER ROLE db_datareader ADD MEMBER AndyAimituma;
ALTER ROLE db_datawriter ADD MEMBER AndyAimituma;

ALTER ROLE db_datareader ADD MEMBER ValerioMosquera;
ALTER ROLE db_datawriter ADD MEMBER ValerioMosquera;
GO
```

Explicación del Script:

1. **Asociar usuarios:** Se crean los usuarios LuisAquino, AndyAimituma y ValerioMosquera vinculándolos con los inicios de sesión previamente creados en el servidor.

2. **Asignar roles:** Los roles db_datareader y db_datawriter permiten a los usuarios leer y modificar datos dentro de la base de datos ColegioDB_Copy.

11.3 Verificar Usuarios y Permisos

Usuarios en la base de datos copiada:

```
-- Listar usuarios de la base de datos ColegioDB_Copy
USE ColegioDB_Copy;
GO

SELECT name AS Usuario, type_desc AS Tipo
FROM sys.database_principals
WHERE type IN ('S', 'U') AND name NOT LIKE '##%';
```

Permisos asignados a cada usuario:

```
-- Consultar los roles de los usuarios en ColegioDB_Copy
SELECT dp.name AS Usuario,
       dr.name AS Rol
FROM sys.database_principals dp
JOIN sys.database_role_members drm
ON dp.principal_id = drm.member_principal_id
JOIN sys.database_principals dr
ON drm.role_principal_id = dr.principal_id
WHERE dp.type = 'S';
```

---

12) Utilizando un script, generar una copia de seguridad de la base de datos y compartir a cada uno de los usuarios

12.1 Generar una Copia de Seguridad de la Base de Datos

Para generar una copia de seguridad de la base de datos ColegioDB, se utiliza el comando BACKUP DATABASE. El archivo generado puede compartirse con los usuarios designados.

```
-- Generar una copia de seguridad de la base de datos ColegioDB
BACKUP DATABASE ColegioDB
TO DISK = 'C:\Temp\ColegioDB.bak'
WITH FORMAT,
     NAME = 'Backup de la Base de Datos ColegioDB',
     DESCRIPTION = 'Copia de seguridad completa de ColegioDB';
GO
```

Explicación:

1. **Ruta de almacenamiento:** El archivo de copia de seguridad se guarda en C:\Temp\ColegioDB.bak.

2. **Parámetros opcionales:**

- WITH FORMAT: Sobrescribe cualquier copia existente en el archivo especificado.
- NAME y DESCRIPTION: Proporcionan información descriptiva sobre el respaldo.

12.2 Compartir la Copia con Usuarios

Después de generar la copia de seguridad, cada usuario puede restaurarla localmente en sus propias instancias de SQL Server.

```
-- Restaurar la base de datos desde el archivo de respaldo
RESTORE DATABASE ColegioDB_Luis
FROM DISK = 'C:\Temp\ColegioDB.bak'
WITH MOVE 'ColegioDB' TO 'C:\Temp\ColegioDB_Luis.mdf',
     MOVE 'ColegioDB_log' TO 'C:\Temp\ColegioDB_Luis.ldf',
     REPLACE;
GO

RESTORE DATABASE ColegioDB_Andy
FROM DISK = 'C:\Temp\ColegioDB.bak'
WITH MOVE 'ColegioDB' TO 'C:\Temp\ColegioDB_Andy.mdf',
     MOVE 'ColegioDB_log' TO 'C:\Temp\ColegioDB_Andy.ldf',
     REPLACE;
GO

RESTORE DATABASE ColegioDB_Valerio
FROM DISK = 'C:\Temp\ColegioDB.bak'
WITH MOVE 'ColegioDB' TO 'C:\Temp\ColegioDB_Valerio.mdf',
     MOVE 'ColegioDB_log' TO 'C:\Temp\ColegioDB_Valerio.ldf',
     REPLACE;
GO
```

Explicación del Script:

1. **Restaurar bases personalizadas:**

- Se restauran copias personalizadas con nombres únicos (ColegioDB_Luis, ColegioDB_Andy, ColegioDB_Valerio).

2. **Rutas de archivos:**

- Los archivos .mdf y .ldf se asignan rutas diferentes para cada usuario.

12.3 Configurar Accesos para Cada Usuario

Una vez restaurada la base de datos, se asignan permisos específicos a cada usuario. Esto garantiza que solo puedan acceder y modificar sus propias bases.

```
-- Conceder permisos a Luis
USE ColegioDB_Luis;
GO
CREATE USER LuisAquino FOR LOGIN LuisAquino;
ALTER ROLE db_owner ADD MEMBER LuisAquino;
GO

-- Conceder permisos a Andy
USE ColegioDB_Andy;
GO
CREATE USER AndyAimituma FOR LOGIN AndyAimituma;
ALTER ROLE db_owner ADD MEMBER AndyAimituma;
GO

-- Conceder permisos a Valerio
USE ColegioDB_Valerio;
GO
CREATE USER ValerioMosquera FOR LOGIN ValerioMosquera;
ALTER ROLE db_owner ADD MEMBER ValerioMosquera;
GO
```

Explicación:

- **Usuarios y roles:** Cada base restaurada tiene un usuario asociado con permisos de db_owner, otorgándole control total sobre su copia.

Verificación Final

1. **Listar bases restauradas:**

```
SELECT name AS NombreBaseDeDatos, state_desc AS Estado
FROM sys.databases
WHERE name LIKE 'ColegioDB_%';
```

2. **Verificar usuarios y roles:**

```
SELECT dp.name AS Usuario, dr.name AS Rol
FROM sys.database_principals dp
JOIN sys.database_role_members drm ON dp.principal_id = drm.member_principal_id
JOIN sys.database_principals dr ON drm.role_principal_id = dr.principal_id
WHERE dp.type = 'S';
```

13) Utilizando un script, encriptar una de las tablas para que no se puedan ver los datos

En SQL Server, para encriptar datos en una tabla y evitar que sean visibles directamente, se utiliza **Transparent Data Encryption (TDE)** o funciones de encriptación basadas en columnas con **EncryptByKey**.

13.1 Crear una Clave Maestra y un Certificado

```
-- Crear la clave maestra
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'ContraseñaSegura123!';
GO

-- Crear un certificado
CREATE CERTIFICATE CertificadoEncriptacion
WITH SUBJECT = 'Certificado para encriptar datos de la tabla';
GO

-- Crear una clave simétrica basada en el certificado
CREATE SYMMETRIC KEY ClaveSimetrica
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE CertificadoEncriptacion;
GO
```

13.2 Encriptar los Datos de la Tabla

```
-- Abrir la clave simétrica
OPEN SYMMETRIC KEY ClaveSimetrica
DECRYPTION BY CERTIFICATE CertificadoEncriptacion;
GO

-- Encriptar los nombres al insertarlos
UPDATE Estudiantes
SET Nombre = EncryptByKey(Key_GUID('ClaveSimetrica'), Nombre);
GO

-- Cerrar la clave simétrica
CLOSE SYMMETRIC KEY ClaveSimetrica;
GO
```

13.3 Visualizar los Datos Desencriptados

```
-- Abrir la clave simétrica
OPEN SYMMETRIC KEY ClaveSimetrica
DECRYPTION BY CERTIFICATE CertificadoEncriptacion;
GO

-- Seleccionar los datos desencriptados
SELECT
    EstudianteID,
    CONVERT(NVARCHAR(100), DecryptByKey(Nombre)) AS NombreDesencriptado,
    Apellido, FechaNacimiento, DNI
FROM Estudiantes;
GO

-- Cerrar la clave simétrica
CLOSE SYMMETRIC KEY ClaveSimetrica;
GO
```

- ✓ **Clave y certificado:**
  La clave y el certificado deben gestionarse cuidadosamente. Si se pierden, los datos encriptados no se podrán recuperar.

- ✓ **Encriptación por columna:**
  Este método encripta únicamente las columnas deseadas, permitiendo que otras queden visibles.

- ✓ **Evitar acceso directo:**
  Restrinja permisos a usuarios para prevenir que ejecuten comandos que abran la clave simétrica.

```sql
SELECT * FROM Estudiantes;
```

117 %

Results | Messages

	EstudianteID	Nombre	Apellido	FechaNacimiento	DNI	Direccion	Telefono	Email
1	1	㈠姮笑窕朶븝睎鄹	Pérez	2005-06-15	12345678A	Calle Ficticia 123, Ciudad	+34 600123456	juan.perez@colegio.com
2	2	㈠姮笑窕朶븝睎鄹	Gómez	2006-08-22	23456789B	Avenida Libertad 456, Ciudad	+34 600234567	ana.gomez@colegio.com
3	3	㈠姮笑窕朶븝睎鄹	Martínez	2004-03-18	34567890C	Calle Primavera 789, Ciudad	+34 600345678	luis.martinez@colegio.com
4	4	㈠姮笑窕朶븝睎鄹	Sánchez	2005-11-05	45678901D	Calle del Sol 101, Ciudad	+34 600456789	marta.sanchez@colegio.com
5	5	㈠姮笑窕朶븝睎鄹	Rodríguez	2005-01-13	56789012E	Avenida Marítima 202, Ciudad	+34 600567890	carlos.rodriguez@colegio.com
6	6	㈠姮笑窕朶븝睎鄹	Fernández	2006-06-30	67890123F	Calle de la Paz 303, Ciudad	+34 600678901	lucia.fernandez@colegio.com
7	7	㈠姮笑窕朶븝睎鄹	López	2004-09-25	78901234G	Calle de la Luna 404, Ciudad	+34 600789012	david.lopez@colegio.com
8	8	㈠姮笑窕朶븝睎鄹	González	2005-04-07	89012345H	Calle de los Pinos 505, Ciudad	+34 600890123	isabel.gonzalez@colegio.com
9	9	㈠姮笑窕朶븝睎鄹	Álvarez	2006-12-20	90123456I	Calle Real 606, Ciudad	+34 600901234	pedro.alvarez@colegio.com

---

14) Utilizando un script, aplique la seguridad a nivel de columna, restringiendo el acceso a la columna que contiene la clave primaria de una de las tablas de su base de datos

14.1 Crear un Usuario Restringido: Primero, creamos un usuario que tendrá acceso limitado.

```sql
-- Crear un nuevo usuario para pruebas
CREATE LOGIN UsuarioRestringido
WITH PASSWORD = 'ContraseñaSegura123!';
GO

CREATE USER UsuarioRestringido
FOR LOGIN UsuarioRestringido;
GO
```

14.2 Restringir el Acceso a la Columna Clave Primaria

```sql
-- Denegar acceso a la columna EstudianteID para el usuario restringido
DENY SELECT ON OBJECT::Estudiantes(EstudianteID) TO UsuarioRestringido;
GO
```

14.3 Verificar la Seguridad

- Conectarse como el usuario restringido: Cambiar de sesión o ejecutar como el usuario restringido.

```sql
EXECUTE AS USER = 'UsuarioRestringido';
GO
```

- Intentar consultar la tabla completa