Universidad Peruana Los Andes
Facultad de Ingeniería
Escuela Profesional de Ingeniería de Sistemas y Computación

Examen Final

Administración de Base de Datos

Apellidos y Nombres **GEYSON CAJAHUARINGA SAMANIEGO** Código: **R01035K**
Ciclo: V Salón: A1

Enunciado 01:

De acuerdo con la **base de datos** desarrollada en **Microsoft SQL Server**, responda las siguientes preguntas:

1) Explique qué problema soluciona su base de datos

La base de datos creada tiene como objetivo **solucionar el problema de la gestion académica dentro de una Universidad Peruana Los Andes.** Esto incluye tareas como el seguimiento y la administración de docentes, cursos, asignaciones y los diferentes permisos y roles dentro de la universidad. Con esta base de datos, es posible gestionar de manera eficiente la asignación de cursos a los docentes, mantener un control sobre la información personal de los mismos, gestionar la facultad a la que pertenecen y realizar un seguimiento adecuado de las calificaciones y otros aspectos relevantes de su desempeño académico.

En resumen, la base de datos busca:

1. Organizar la información de los docentes, sus cursos, y las asignaciones de manera estructurada, sílabos y carpetas de los docentes.
2. Mejorar la gestión administrativa dentro de la universidad.
3. Facilitar la consulta de la información de manera eficiente para apoyar la toma de decisiones.

2) Implemente un Script para crear una **vista** para crear utilizando tres tablas

```
55 CREATE VIEW vista_asignaciones_docentes AS
56 SELECT
57     d.nombre_docente,
58     d.facultad,
59     c.nombre_curso,
60     c.codigo_curso,
61     a.semestre
62 FROM
63     docentes d
64 JOIN
65     asignaciones a ON d.id_docente = a.id_docente
66 JOIN
67     cursos c ON a.id_curso = c.id_curso;
68 GO -- Separador de lote de consultas
69
70 SELECT * FROM vista_asignaciones_docentes;
```

Results | Messages

nombre_docente	facultad	nombre_curso	codigo_curso	semestre
Carlos Mendoza	Sistemas	Programación Avanzada	CS201	2024-1
Laura Ruiz	Arquitectura	Diseño Estructural	AR102	2024-1
Juan Pérez	Medio Ambiente	Ecosistemas y Biodiversidad	MA301	2024-2
Ana Gómez	Civil	Mecánica de Materiales	CI205	2024-1
Roberto López	Industrial	Gestión de Producción	II307	2024-2

3) Implemente un Script para crear un **procedimiento almacenado** para modificar el ingreso de datos en forma secuencial

```
CREATE PROCEDURE ModificarDatosSecuencial
@id_docente INT,
@nombre_docente VARCHAR(100),
@celular VARCHAR(15),
@correo_electronico VARCHAR(100),
@facultad VARCHAR(50),
@fecha_ingreso DATE,
@titulo_academico VARCHAR(50),
@salario DECIMAL(10,2),

@id_curso INT,
@nombre_curso VARCHAR(100),
@codigo_curso VARCHAR(10),
@facultad_ofrecida VARCHAR(50),
@creditos INT,
@descripcion TEXT,

@id_asignacion INT,
@semestre VARCHAR(10)
AS
BEGIN
-- Modificar datos del docente
UPDATE docentes
SET
nombre_docente = @nombre_docente,
celular = @celular,
correo_electronico = @correo_electronico,
facultad = @facultad,
fecha_ingreso = @fecha_ingreso,
titulo_academico = @titulo_academico,
salario = @salario
WHERE id_docente = @id_docente;

-- Modificar datos del curso
UPDATE cursos
SET
nombre_curso = @nombre_curso,
codigo_curso = @codigo_curso,
facultad_ofrecida = @facultad_ofrecida,
```

```
creditos = @creditos,
descripcion = @descripcion
WHERE id_curso = @id_curso;

-- Modificar datos de asignación
UPDATE asignaciones
SET
semestre = @semestre
WHERE id_asignacion = @id_asignacion;

PRINT 'Datos modificados correctamente';
END;
GO
```

4) Implemente un Script para crear un **disparador** para verificar el control de datos (Ejemplo: que la nota ingresada este entre 0 y 20)

```sql
-- Crear la tabla de notas (si no está creada)
CREATE TABLE notas (
    id INT IDENTITY(1,1) PRIMARY KEY,
    id_estudiante INT NOT NULL,
    nombre_estudiante VARCHAR(100),
    nota DECIMAL(5,2)
);

GO  -- Separador de lote

-- Crear el disparador
CREATE TRIGGER trg_validar_nota
ON notas
FOR INSERT
AS
BEGIN
    DECLARE @nota DECIMAL(5,2);

    -- Obtener la nota del nuevo registro insertado
    SELECT @nota = nota FROM inserted;

    -- Verificar si la nota está fuera del rango permitido
    IF @nota < 0 OR @nota > 20
    BEGIN
        -- Si la nota está fuera del rango, se elimina la inserción con ROLLBACK
        ROLLBACK TRANSACTION;

        -- Devolver un error con un mensaje que será capturado por la aplicación
```

id	id_estudiante	nombre_estudiante	nota
1	1	Juan Pérez	18.00

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

```sql
-- Primero, asegurémonos de que estamos usando la base de datos correcta.
USE UniversidadDB;
GO

-- Crear una tabla para almacenar la información de los usuarios
CREATE TABLE Usuarios (
    id_usuario INT IDENTITY(1,1) PRIMARY KEY,
    nombre_usuario VARCHAR(100),
    correo_electronico VARCHAR(100),
    fecha_creacion DATETIME DEFAULT GETDATE()
);
GO

-- Insertar los usuarios (tus compañeros y tú)
INSERT INTO Usuarios (nombre_usuario, correo_electronico)
VALUES
('Usuario1', 'usuario1@dominio.com'),
('Usuario2', 'usuario2@dominio.com'),
('Usuario3', 'usuario3@dominio.com'),
('Geison', 'geison@dominio.com');
GO

-- Mostrar los usuarios insertados en la tabla
SELECT * FROM Usuarios;
GO
```

id_usuario	nombre_usuario	correo_electronico	fecha_creacion
1	Usuario1	usuario1@dominio.com	2024-12-21 17:12:17.073
2	Usuario2	usuario2@dominio.com	2024-12-21 17:12:17.073
3	Usuario3	usuario3@dominio.com	2024-12-21 17:12:17.073
4	Geison	geison@dominio.com	2024-12-21 17:12:17.073

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

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

```
BACKUP DATABASE [UniversidadDB]
TO DISK = 'C:\Backups\UniversidadDB_backup.bak'
WITH FORMAT,
    MEDIANAME = 'UniversidadDB_Media',
    NAME = 'Backup completo de la base de datos UniversidadDB';
```

```
-- Asegurarse de que los usuarios existen, en caso de no existir, crearlos
CREATE LOGIN Usuario1 WITH PASSWORD = 'password1';
CREATE LOGIN Usuario2 WITH PASSWORD = 'password2';
CREATE LOGIN Usuario3 WITH PASSWORD = 'password3';
USE UniversidadDB_Copy;

CREATE USER Usuario1 FOR LOGIN Usuario1;
CREATE USER Usuario2 FOR LOGIN Usuario2;
CREATE USER Usuario3 FOR LOGIN Usuario3;

ALTER ROLE db_datareader ADD MEMBER Usuario1;
ALTER ROLE db_datawriter ADD MEMBER Usuario1;

ALTER ROLE db_datareader ADD MEMBER Usuario2;
ALTER ROLE db_datawriter ADD MEMBER Usuario2;

ALTER ROLE db_datareader ADD MEMBER Usuario3;
ALTER ROLE db_datawriter ADD MEMBER Usuario3;

-- Paso 4: Verificar que los usuarios tengan acceso

-- Mostrar los usuarios en la base de datos copiada
SELECT name
FROM sys.database_principals
WHERE type IN ('S', 'U');  -- 'S' para usuarios de SQL, 'U' para usuarios de Windows
```

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

```
-- Cifrar los datos de la columna 'telefono' al insertarlos en la tabla
OPEN SYMMETRIC KEY MiClaveSimetrica
DECRYPTION BY CERTIFICATE MiCertificado;

-- Crear la tabla de ejemplo
CREATE TABLE Empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(100),
    telefono VARBINARY(MAX) -- Utilizamos VARBINARY para almacenar datos encriptados
);

-- Insertar datos en la tabla encriptados
INSERT INTO Empleados (id, nombre, telefono)
VALUES
(1, 'Carlos Mendoza', ENCRYPTBYKEY(KEY_GUID('MiClaveSimetrica'), '987654321')),
(2, 'Laura Ruiz', ENCRYPTBYKEY(KEY_GUID('MiClaveSimetrica'), '987654322'));
-- Consultar los datos desencriptados
SELECT id,
       nombre,
       CONVERT(VARCHAR, DECRYPTBYKEY(telefono)) AS telefono
FROM Empleados;

-- Cerrar la clave simétrica después de su uso
CLOSE SYMMETRIC KEY MiClaveSimetrica;
```

9) Utilizando un script, aplique la seguridad a nivel de columna, restringiendo el acceso a la columna DNI de la tabla empleado en el usuario con nombre de su compañero

```
CREATE TABLE Empleado (
  id INT PRIMARY KEY,
  nombre VARCHAR(100),
  DNI VARCHAR(15)
);
INSERT INTO Empleado (id, nombre, DNI)
VALUES
(1, 'Carlos Mendoza', '123456789'),
(2, 'Laura Ruiz', '987654321'),
(3, 'Juan Pérez', '543216789');
CREATE USER compañero_usuario FOR LOGIN compañero_usuario_login;
CREATE FUNCTION dbo.fn_CheckDNIAccess()
RETURNS INT
AS
BEGIN
  -- Asumimos que el compañero no tiene acceso a la columna DNI
  IF (USER_NAME() = 'compañero_usuario')
  BEGIN
    RETURN 0;  -- No acceso a la columna DNI
  END
  ELSE
  BEGIN
    RETURN 1;  -- Acceso permitido a la columna DNI
  END
END;

CREATE VIEW Empleado_Seguro AS
SELECT
  id,
  nombre,
  CASE
    WHEN dbo.fn_CheckDNIAccess() = 1 THEN DNI
    ELSE 'Acceso Restringido' -- Si el usuario no tiene permiso, mostrar mensaje
  END AS DNI
FROM Empleado;

REVOKE SELECT ON Empleado TO compañero_usuario;
GRANT SELECT ON Empleado_Seguro TO compañero_usuario;
SELECT * FROM Empleado_Seguro;

SELECT * FROM Empleado;
```

10) Utilizando un script, implementé seguridad a nivel de columna restringiendo el acceso a una de las columnas de una tabla.

```
CREATE TABLE Empleado (
    ID INT PRIMARY KEY,
    Nombre VARCHAR(100),
    DNI VARCHAR(20),
    Salario DECIMAL(10, 2)
);

INSERT INTO Empleado (ID, Nombre, DNI, Salario)
VALUES
(1, 'Juan Perez', '12345678', 3500.50),
(2, 'Maria Lopez', '87654321', 4200.75),
(3, 'Carlos Gomez', '11223344', 3100.60);

CREATE ROLE rolRestringido;

CREATE USER UsuarioRestringido FOR LOGIN UsuarioRestringido;
EXEC sp_addrolemember 'rolRestringido', 'UsuarioRestringido';

DENY SELECT ON COLUMN dbo.Empleado.DNI TO rolRestringido;
SELECT ID, Nombre, Salario
FROM Empleado;
```

11) Utilizando un script, realice el cifrado transparente de datos (TDE) para una las tablas.

```
CREATE DATABASE MiBaseDeDatos;
GO

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Sammy123!'
GO

CREATE CERTIFICATE MiCertificadoTDE
   WITH SUBJECT = 'Certificado para TDE';
GO

USE MiBaseDeDatos;
GO

CREATE DATABASE ENCRYPTION KEY;
GO

ALTER DATABASE MiBaseDeDatos
SET ENCRYPTION ON;
GO

SELECT name, is_encrypted
FROM sys.databases
WHERE name = 'MiBaseDeDatos';
GO
```

12) Utilizando un script, configure el usuario con el nombre de su compañero para otorgar permisos de SELECT, INSERT, UPDATE y DELETE en la base de datos.

```
USE MiBaseDeDatos;  --
GO


-- En caso de que el login no exista, primero lo creamos
CREATE LOGIN UsuarioDeMiCompanero WITH PASSWORD = 'sammy123';
CREATE USER Brayan FOR LOGIN Brayan;

-- Paso 3: Otorgar permisos SELECT, INSERT, UPDATE, DELETE
GRANT SELECT, INSERT, UPDATE, DELETE
ON SCHEMA :: dbo
TO brayan;
GO

```

13) Utilizando un script, configure la auditoría para el seguimiento y registro de acciones en la base de datos

```
-- Paso 1: Crear una auditoría (se guardará en un archivo de log)
CREATE SERVER AUDIT MiAuditoria
TO FILE (FILEPATH = 'C:\AuditLogs\');  -- Cambia la ruta de acuerdo a tu entorno
WITH (ON_FAILURE = CONTINUE);
GO

-- Paso 2: Crear una especificación de auditoría para capturar los eventos deseados
CREATE SERVER AUDIT SPECIFICATION MiEspecificacionAuditoria
FOR SERVER AUDIT MiAuditoria
ADD (FAILED_LOGIN_GROUP),  -- Auditar intentos de inicio de sesión fallidos
ADD (LOGIN_GROUP),         -- Auditar inicio de sesión exitoso
ADD (DATABASE_OBJECT_PERMISSION_CHANGE_GROUP),  -- Auditar cambios en permisos de objetos
ADD (SCHEMA_OBJECT_CHANGE_GROUP);  -- Auditar cambios en objetos del esquema
GO

-- Paso 3: Activar la auditoría
ALTER SERVER AUDIT MiAuditoria
WITH (STATE = ON);
GO
```

14) Utilizando un script, configure de la memoria y el disco duro

```
-- 1. Configurar el uso máximo de memoria para SQL Server (en MB)
EXEC sp_configure 'max server memory (MB)', 4096;  -- 4GB
RECONFIGURE;
GO

-- 2. Configurar el uso mínimo de memoria para SQL Server (en MB)
EXEC sp_configure 'min server memory (MB)', 1024;  -- 1GB
RECONFIGURE;
GO
```

```
-- 1. Aumentar el tamaño del archivo de base de datos (archivo de datos)
ALTER DATABASE examenbaseDatos
MODIFY FILE (NAME = 'examenbaseDatos_data', SIZE = 5GB);
GO

-- 2. Aumentar el tamaño del archivo de registro de la base de datos
ALTER DATABASE examenbaseDatos
MODIFY FILE (NAME = 'examenbaseDatos_log', SIZE = 2GB);
GO
```

15) Utilizando un script, genere una copia de seguridad de la base de datos

```
-- 1. Realizar una copia de seguridad completa de la base de datos
BACKUP DATABASE examenbaseDatos
TO DISK = 'C:\Backup\examenbaseDatos.bak'
WITH FORMAT, MEDIANAME = 'MiBackup', NAME = 'Copia de seguridad completa';
GO
```

16) Utilizando un script, genere la restauración de la base de datos

```
-- 1. Restaurar la base de datos desde el archivo de copia de seguridad
RESTORE DATABASE examenbaseDatos
FROM DISK = 'C:\Backup\examenbaseDatos.bak'
WITH REPLACE;
GO
```

17) Utilizando un script, cree un espejo de la base de datos

```
-- En el servidor primario
ALTER DATABASE examenbaseDatos SET PARTNER = 'TCP://ServidorSecundario:5022';
GO

-- En el servidor secundario
ALTER DATABASE examenbaseDatos SET PARTNER = 'TCP://ServidorPrimario:5022';
GO
-- En el servidor primario
ALTER DATABASE examenbaseDatos
SET PARTNER SAFETY FULL;
GO

-- En el servidor secundario
ALTER DATABASE examenbaseDatos
SET PARTNER SAFETY FULL;
GO
```

18) Utilizando un script, realice la replicación de bases de datos

```
-- 1. Configurar el servidor publicador
EXEC sp_replicationdboption
  @dbname = 'examenbaseDatos',
  @optname = 'publish',
  @value = 'true';
GO

-- 2. Configurar la publicación
EXEC sp_addpublication
  @publication = 'ExamenPublication',
  @description = 'Publicación de la base de datos examenbaseDatos',
  @pubtype = 'transational',
  @status = 'active';
GO
-- 3. Configurar el servidor suscriptor
EXEC sp_addsubscription
  @publication = 'ExamenPublication',
  @subscriber = 'ServidorSecundario',
  @destination_db = 'examenbaseDatos',
  @subscription_type = 'push',
  @sync_type = 'automatic';
GO
```


19) Explique que es Always On Availability Groups

Always On Availability Groups es una característica de alta disponibilidad y recuperación ante desastres de **SQL Server** que permite configurar grupos de bases de datos para mantener varias réplicas de las mismas en servidores diferentes. Proporciona conmutación por error automática y sincronización de datos en tiempo real, mejorando la disponibilidad y la resistencia a fallos.

Características principales de Always On Availability Groups:

1. **Grupo de bases de datos**:
   - Un *Availability Group* (AG) está compuesto por un grupo de bases de datos que se replican entre varios servidores.
   - Cada grupo puede tener hasta **8 réplicas** de bases de datos, de las cuales **uno** es el servidor principal (primario) y los demás son réplicas secundarias (secundarias).
2. **Conmutación por error automática**:
   - Siempre que se configure adecuadamente, **Always On** permite la conmutación por error automática entre servidores en caso de que el servidor principal falle. Esto minimiza el tiempo de inactividad.
3. **Sincronización de datos**:
   - Las bases de datos en el grupo se mantienen sincronizadas mediante un proceso de replicación de registros de transacciones.
   - Hay dos tipos de sincronización de datos:
     - **Sincrónica**: Las transacciones son confirmadas en la réplica primaria y secundaria al mismo tiempo. Esto garantiza la consistencia de los datos, pero puede tener una ligera sobrecarga en el rendimiento debido a la necesidad de esperar la confirmación en la réplica secundaria.
     - **Asíncrona**: Las transacciones son confirmadas en la réplica primaria sin esperar la confirmación de la réplica secundaria. Esto mejora el rendimiento pero puede causar una pequeña pérdida de datos en caso de falla del servidor primario.
4. **Réplica primaria y réplicas secundarias**:
   - **Réplica primaria**: Es la base de datos activa donde se realizan las operaciones de lectura y escritura.
   - **Réplica secundaria**: Es la base de datos pasiva que mantiene una copia actualizada de los datos de la réplica primaria. Las réplicas secundarias pueden estar configuradas para ser solo de lectura, lo que ayuda a distribuir la carga de lectura entre varios servidores.
5. **Alta disponibilidad y recuperación ante desastres**:
   - **Alta disponibilidad**: Las bases de datos se mantienen disponibles incluso si un servidor falla, gracias a la conmutación por error automática entre las réplicas.
   - **Recuperación ante desastres**: En caso de fallo catastrófico en el servidor primario, se puede conmutar el servicio a una réplica secundaria, minimizando el tiempo de inactividad.

6. **Acceso en solo lectura**:
    - Las réplicas secundarias pueden configurarse para permitir solo lecturas, lo que permite la distribución de la carga de consultas y mejora el rendimiento del sistema al aliviar la presión sobre la réplica primaria.

Componentes de Always On Availability Groups:

1. **Primaria (Primary Replica)**: Es donde se realizan las operaciones de escritura y lectura por defecto. Es la base de datos activa.
2. **Secundarias (Secondary Replicas)**: Son réplicas pasivas que reflejan los cambios hechos en la primaria. Pueden ser usadas para lectura si se configura adecuadamente.
3. **Listener**: Es un punto de acceso virtual que proporciona una dirección de red común para acceder al grupo de disponibilidad. Los clientes pueden conectarse al Listener, y Always On se encarga de dirigir el tráfico a la réplica primaria activa, incluso si se produce una conmutación por error.
4. **Cluster de conmutación por error (WSFC)**: **Windows Server Failover Clustering (WSFC)** es necesario para implementar Always On. Gestiona las conmutaciones por error y las condiciones de disponibilidad.

Beneficios de Always On Availability Groups:

- **Alta disponibilidad**: Asegura que las aplicaciones sigan funcionando sin interrupciones, incluso en caso de fallos de hardware o software.
- **Conmutación por error automática**: Garantiza una mínima interrupción en los servicios en caso de que se produzca una falla en el servidor.
- **Escalabilidad**: Permite la distribución de la carga de trabajo entre varias réplicas secundarias de solo lectura.
- **Protección de datos**: Los datos están protegidos y se mantienen sincronizados entre las réplicas.
- **Rendimiento optimizado**: Al distribuir las consultas de solo lectura a las réplicas secundarias, se mejora el rendimiento general del sistema.

Limitaciones:

- **Requiere SQL Server Enterprise Edition**: Always On Availability Groups solo está disponible en las ediciones Enterprise de SQL Server, lo que limita su disponibilidad en versiones más económicas.
- **Complejidad en la configuración**: La configuración de Always On es más compleja que otras soluciones como el Log Shipping y requiere una infraestructura más robusta, como un clúster de conmutación por error.
- **Dependencia de WSFC**: Requiere que el sistema operativo subyacente sea compatible con Windows Server Failover Clustering (WSFC).

Resumen: Always On Availability Groups es una solución avanzada de alta disponibilidad y recuperación ante desastres que permite a las organizaciones mantener bases de datos sincronizadas entre varias réplicas, proporcionando una conmutación por error automática, alta disponibilidad y escalabilidad para mejorar el rendimiento y la protección de los datos.

20) Explique que es Log Shipping

Log Shipping es una técnica de recuperación de base de datos en SQL Server, donde los archivos de registro de transacciones (log files) de una base de datos primaria se copian, restauran y aplican en una base de datos secundaria en un servidor diferente. Esta estrategia se utiliza principalmente para mantener una copia de la base de datos en un servidor de respaldo, lo cual ayuda a mejorar la disponibilidad y la recuperación ante desastres.

Cómo funciona:

1. **Base de datos primaria**: En el servidor primario, se realiza un respaldo completo de la base de datos y luego se hacen respaldos de los registros de transacciones (logs) a intervalos regulares.
2. **Copiado de los registros**: Los archivos de registro de transacciones generados en el servidor primario se copian automáticamente al servidor secundario en intervalos programados.
3. **Restauración en la base de datos secundaria**: Los registros copiados se restauran en la base de datos secundaria de forma secuencial, para mantenerla sincronizada con la base de datos primaria.
4. **Monitoreo y automatización**: SQL Server permite configurar el monitoreo automático para verificar el estado del log shipping, y también se pueden configurar alertas en caso de que no se logren copiar o restaurar los archivos de registro correctamente.

Componentes principales:

- **Servidor primario**: El servidor donde reside la base de datos activa y se generan los archivos de registro.
- **Servidor secundario**: Un servidor donde se restauran y aplican los archivos de registro, creando una copia actualizada de la base de datos primaria.
- **Servidor de monitoreo**: En ocasiones, se utiliza un tercer servidor para monitorear el estado de la operación de log shipping.

Beneficios:

- **Alta disponibilidad**: Permite tener una base de datos de respaldo que se mantiene casi sincronizada con la base de datos primaria, lo que ayuda a reducir el tiempo de inactividad.
- **Recuperación ante desastres**: En caso de que el servidor primario falle, se puede conmutar por error al servidor secundario para minimizar la pérdida de datos.
- **Escalabilidad de lectura**: En algunos casos, el servidor secundario se puede usar para consultas de solo lectura, aliviando la carga en el servidor primario.

Limitaciones:

- **Retraso de datos**: Dependiendo de la frecuencia de los respaldos y la restauración de logs, puede haber un pequeño retraso entre la base de datos primaria y la secundaria.
- **No es una solución de conmutación por error automática**: A diferencia de otras soluciones como AlwaysOn Availability Groups, Log Shipping no proporciona una conmutación automática. Si el servidor primario falla, se debe realizar la conmutación manualmente.

En resumen, Log Shipping es una solución efectiva para mantener una base de datos secundaria sincronizada con la primaria, pero tiene algunas limitaciones en cuanto a la automatización y la recuperación ante desastres**.**