Tema 3: Manejo de transacciones y transacciones anidadas:

El manejo de transacciones es un pilar fundamental en la arquitectura de los Sistemas de Gestión de Bases de Datos Relacionales (SGBDR), esencial para garantizar la integridad y consistencia de la información. Una transacción se define formalmente como una secuencia lógica de una o más operaciones de base de datos tratadas como una unidad atómica de trabajo. Esto significa que la transacción debe adherirse estrictamente al principio de Atomicidad: o todas las operaciones se completan exitosamente, o ninguna de ellas se ejecuta, revirtiendo el sistema a su estado original. Este comportamiento es indispensable para cumplir con las propiedades **ACID** (Atomicidad, Consistencia, Aislamiento y Durabilidad), que aseguran la fiabilidad del sistema. En entornos Transact-SQL (T-SQL), el manejo explícito se realiza mediante las instrucciones *BEGIN TRANSACTION* para delimitar el inicio de la unidad, y *COMMIT TRANSACTION* o *ROLLBACK TRANSACTION* para confirmar los cambios permanentemente o deshacerlos completamente, respectivamente.

El concepto de **transacciones anidadas** se refiere a la capacidad de iniciar una nueva transacción dentro del ámbito de una transacción ya existente. En **SQL Server**, este anidamiento se gestiona a través del contador de sistema *@@TRANCOUNT*, el cual se incrementa con cada *BEGIN TRANSACTION* y se decrementa con cada *COMMIT TRANSACTION* correspondiente. Es crucial destacar que, a pesar del anidamiento, el SGBDR opera bajo una única transacción real a nivel de base de datos: un COMMIT interno solo reduce el nivel de anidamiento, mientras que la confirmación de datos se produce únicamente cuando el COMMIT final de la transacción más externa reduce *@@TRANCOUNT* a cero. Sin embargo, un ***ROLLBACK TRANSACTION*** ejecutado en cualquier nivel anula la **transacción completa**, revirtiendo todos los cambios hasta el *BEGIN TRAN* inicial. Para un control de errores más granular, se emplea la instrucción ***SAVE TRANSACTION*** o **punto de guardado** (*Savepoint*), la cual permite al desarrollador anular selectivamente una porción de las operaciones internas sin necesidad de revertir toda la transacción principal.

La aplicación práctica de estas técnicas es imperativa en un sistema crítico como el **Proyecto Hospitalario** propuesto, donde la consistencia de los datos impacta directamente en la calidad asistencial. Por ejemplo, procesos como la **Internación** de un paciente, que requieren la inserción en la tabla Internacion y la simultánea actualización del estado de la Cama a "Ocupada", deben encapsularse en una única transacción para evitar inconsistencias que dejen al sistema en un estado ambiguo. El uso estratégico de **puntos de guardado** es especialmente valioso dentro de los procedimientos almacenados, permitiendo que una operación compleja, como la facturación de una Intervención con múltiples Procedimientos, pueda revertir un paso fallido individualmente sin comprometer los pasos anteriores exitosos, garantizando así la **integridad transaccional** sin sacrificar la eficiencia operativa del sistema.

Casos de aplicación:
El proceso de **Ingreso de un Paciente a Internación** requiere la actualización atómica de dos entidades principales:

Paso 1 (INSERT): Registro del ingreso en la tabla Internación.

Paso 2 (INSERT): Creación del primer evento clínico/administrativo en la tabla Intervención (Ej., toma inicial de signos vitales o consulta de ingreso).

Paso 3 (INSERT): Registro de la acción médica específica en la tabla Procedimiento, vinculándola a la Intervención generada.

Paso 4 (UPDATE): Actualización de la tabla Cama para cambiar su estado a "Ocupada".

Si los cuatro pasos se completan sin errores (ej., el ID de paciente existe, el tipo de procedimiento es válido y la cama está disponible), el COMMIT TRANSACTION garantiza que los datos se hagan permanentes de forma conjunta. Si alguna operación fallara, el ROLLBACK anularía todos los pasos, impidiendo inconsistencias.

Código:

```
BEGIN TRY
    -- Declaración de variable funcional para capturar la ID generada
    DECLARE @id_intervencion_reciente INT;

    -- 1. Iniciamos la transacción para asegurar atomicidad
    BEGIN TRANSACTION;

    -- PASO 1: INSERT 1 - Registramos el Ingreso en INTERNACION
    INSERT INTO internacion (fecha_inicio, fecha_fin, id_paciente, id_habitacion, id_cama)
    VALUES (GETDATE(), NULL, 1, 1, 2);

    -- PASO 2: INSERT 2 - Registramos la INTERVENCION inicial
    INSERT INTO intervencion (fecha, id_paciente)
    VALUES (GETDATE(), 1);

    -- Capturar el ID de Intervención recién creado
    SET @id_intervencion_reciente = SCOPE_IDENTITY();

    -- PASO 3: INSERT 3 - Insertar un PROCEDIMIENTO usando el ID capturado

    -- id_proc_real=1 (primer proc. en la intervención), id_intervencion=@id_intervencion_reciente
    INSERT INTO procedimiento (id_proc_real, id_profesion, id_tipo_procedimiento, id_profesional, id_intervencion)
    VALUES (1, 1, 1, 1, @id_intervencion_reciente);

    -- PASO 4: UPDATE - Actualizar el estado de la CAMA a Ocupada (ID 2)
    UPDATE cama
    SET estado = 2
    WHERE id_habitacion = 1 AND id_cama = 2;

    -- 5. Confirmar la transacción
    COMMIT TRANSACTION;
    PRINT 'TRANSACCIÓN COMPLETA CONFIRMADA. Los datos han sido actualizados con éxito.';

END TRY
BEGIN CATCH
    -- Bloque de CATCH
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
    PRINT 'ERROR: La transacción falló y fue revertida. Detalle: ' + ERROR_MESSAGE();--No deberia ejecutarse si todo sale bien
END CATCH
```

Resultado de ejecución:

```
(1 fila afectada)

(1 fila afectada)

(1 fila afectada)

(1 fila afectada)
TRANSACCIÓN COMPLETA CONFIRMADA. Los datos han sido actualizados con éxito.
```

Resultados | Mensajes

	fecha_inicio	fecha_fin	id_internacion	id_paciente	id_habitacion	id_cama
1	2025-11-17	NULL	3	1	1	2

	id_intervencion	fecha	id_paciente
1	4	2025-11-17	1

	id_proc_real	id_profesion	id_tipo_procedimiento	id_profesional	id_intervencion
1	1	1	1	1	4

	id_cama	id_habitacion	estado
1	2	1	2

Para demostrar la **Atomicidad** de la transacción, se forzó intencionalmente un fallo al intentar registrar la Intervención (Paso 2) con un id_paciente inválido (999), provocando una violación de Clave Foránea.

Resultado y Verificación

1. **Fallo en el Flujo:** El **Paso 1** (INSERT en Internacion) se ejecutó temporalmente con éxito. Sin embargo, el **Paso 2** (INSERT en Intervencion) falló.
2. **Activación del ROLLBACK:** Al detectar el error, el control del flujo pasó al bloque CATCH, donde se ejecutó la instrucción **ROLLBACK TRANSACTION**.
3. **Consistencia Final:** El ROLLBACK deshizo el INSERT del Paso 1, así como el intento fallido del Paso 2. Por lo tanto, **ningún registro se hizo permanente** en las tablas Internacion, Intervencion, ni se realizó el UPDATE en Cama.

Esta prueba verifica que la transacción cumple con el principio de **"Todo o Nada"**: la gestión completa del ingreso (registro en internación, documentación clínica y asignación de cama) debe ser exitosa para que la base de datos acepte los cambios, garantizando así la **consistencia** total del sistema hospitalario.

```sql
BEGIN TRY
    DECLARE @id_intervencion_reciente INT;
    -- 1. Iniciamos la transacción para asegurar atomicidad
    BEGIN TRANSACTION;

    -- PASO 1: INSERT 1 - Registramos el Ingreso en INTERNACION
    INSERT INTO internacion (fecha_inicio, fecha_fin, id_paciente, id_habitacion, id_cama)
    VALUES (GETDATE(), NULL, 1, 1, 2);
    PRINT '1. INSERT en Internacion EXITOSO (temporalmente).';

    -- PASO 2: INSERT 2 - Registramos la INTERVENCION inicial (FALLO PROVOCADO)
    -- ERROR INTENCIONAL: id_paciente 999 NO EXISTE en la tabla Paciente.
    INSERT INTO intervencion (fecha, id_paciente)
    VALUES (GETDATE(), 999);

    -- El fallo ocurre aquí, el control salta al CATCH, y las siguientes líneas no se ejecutan.
    SET @id_intervencion_reciente = SCOPE_IDENTITY();

    PRINT '2. INSERT en Intervencion FALLÓ (ESTA LÍNEA NO SE IMPRIME).';

    INSERT INTO procedimiento (id_proc_real, id_profesion, id_tipo_procedimiento, id_profesional, id_intervencion)
    VALUES (1, 1, 1, 1, @id_intervencion_reciente);

    UPDATE cama
    SET estado = 2
    WHERE id_habitacion = 1 AND id_cama = 2;

    COMMIT TRANSACTION;

END TRY
BEGIN CATCH
    -- 4. Manejo de errores y ROLLBACK
    IF @@TRANCOUNT > 0
    BEGIN
        ROLLBACK TRANSACTION;
    END
    PRINT 'TRANSACCIÓN REVERTIDA completamente. La operación falló en el Paso 2.'; --Debería mostrarse este mensaje
    PRINT 'Detalle del error: ' + ERROR_MESSAGE();
END CATCH
```

Resultado de ejecución:

```
(1 fila afectada)
1. INSERT en Internacion EXITOSO (temporalmente).

(0 filas afectadas)
TRANSACCIÓN REVERTIDA completamente. La operación falló en el Paso 2.
Detalle del error: The INSERT statement conflicted with the FOREIGN KEY constraint "FK_intervencion_paciente".
The conflict occurred in database "sistema_hospitalario", table "dbo.paciente", column 'id_paciente'.
```

Conclusión:

Las pruebas realizadas confirmaron que el sistema cumple con el principio de **Atomicidad** ("Todo o Nada"). Específicamente, al provocar un fallo intencional durante el flujo de ingreso del paciente (que abarca la inserción en Internacion, Intervencion, Procedimiento y la actualización de Cama), la instrucción **ROLLBACK TRANSACTION** revirtió todos los cambios. Esto asegura que si la documentación clínica (Procedimiento) no se puede registrar correctamente, la asignación de cama y el registro de la internación son anulados, previniendo incoherencias lógicas y manteniendo la base de datos en un estado fiable.