INFORME DE APE

I. PORTADA

Tema:	APE 2. Tratamiento de transacciones
Unidad de Organización Curricular:	PROFESIONAL
Nivel y Paralelo:	Quinto - A
Alumnos participantes:	Cholota Guamán Carlos Sebastián Mazabanda Pilamunga Diego Abraham Tixilema Puaquiza Kevin Alexander Tubon Chipantiza Danilo Alexander
Asignatura:	Sistemas de Bases de Datos Distribuidos
Docente:	Ing. José Rubén Caiza, Mg.

II. INFORME DE APE

2.1 Objetivos

General:
Determinar el comportamiento de un SGBD con transacciones
Específicos:

- Analizar el funcionamiento de las transacciones en SQL Server mediante la ejecución de operaciones controladas.
- Comprobar el uso de los comandos COMMIT y ROLLBACK para garantizar la atomicidad y consistencia de los datos.
- Implementar estructuras de manejo de errores como TRY...CATCH y SET XACT_ABORT para asegurar la integridad transaccional.
- Evaluar el comportamiento del sistema ante errores y sesiones concurrentes para comprender las propiedades ACID de las transacciones.

2.2 Modalidad

Presencial

2.3 Tiempo de duración

Presenciales: 6
No presenciales: 0

2.4 Instrucciones

Conéctese al motor de base de datos
· Cree una BD llamada Universidad
· Cree las tablas A(a char(1) PK, B(b char(1) referenciada a A, C(c char(1)
· Ingresemos algunos datos y verificamos la integridad referencial
· Creamos transacciones y probamos las características de atomicidad Commit y Rollback
· Habilitamos una nueva sesión para pruebas
· Manejamos errores On_error, set xact_abort y Try

2.5 Listado de equipos, materiales y recursos

Listado de equipos y materiales generales empleados en la guía práctica:

- **Computador**
- **SQL Server**

TAC (Tecnologías para el Aprendizaje y Conocimiento) empleados en la guía práctica:

☐Plataformas educativas
☐Simuladores y laboratorios virtuales
☐Aplicaciones educativas
☐Recursos audiovisuales
☐Gamificación
☒Inteligencia Artificial
Otros (Especifique): _____

2.6 Actividades por desarrollar

Con base en las instrucciones descritas en este apartado y las recomendaciones del docente, elabore la Guía APE en un único documento PDF con el formato establecido

2.7 Resultados obtenidos

Tratamiento de transacciones

El tratamiento de transacciones en SQL Server es un proceso fundamental dentro de los sistemas de gestión de bases de datos, ya que garantiza la integridad y consistencia de la información durante la ejecución de operaciones. Una transacción se compone de un conjunto de instrucciones SQL que deben cumplirse de forma completa o, en caso contrario, revertirse totalmente para evitar errores o datos inconsistentes.

Durante esta práctica se estudió el comportamiento de un sistema gestor de base de datos (SGBD) ante distintas situaciones transaccionales, aplicando los conceptos de atomicidad, consistencia, aislamiento y durabilidad (ACID). Además, se implementaron mecanismos de control mediante las sentencias COMMIT, ROLLBACK, y estructuras de manejo de errores como TRY...CATCH, SET XACT_ABORT y ON ERROR, con el fin de comprender cómo SQL Server responde ante fallos o interrupciones durante las operaciones.

Un sistema transaccional siempre finaliza con Commit o Rollback, caso contrario puede generar un estado inconsistente de la BD.
Para el manejo de errores existen varias alternativas que funciona de acuerdo al contexto

2.7.1 Conexión y Creación de Base de Datos

El primer paso consiste en establecer la conexión con SQL Server y crear la base de datos "Universidad" donde se realizarán todas las pruebas de transacciones.

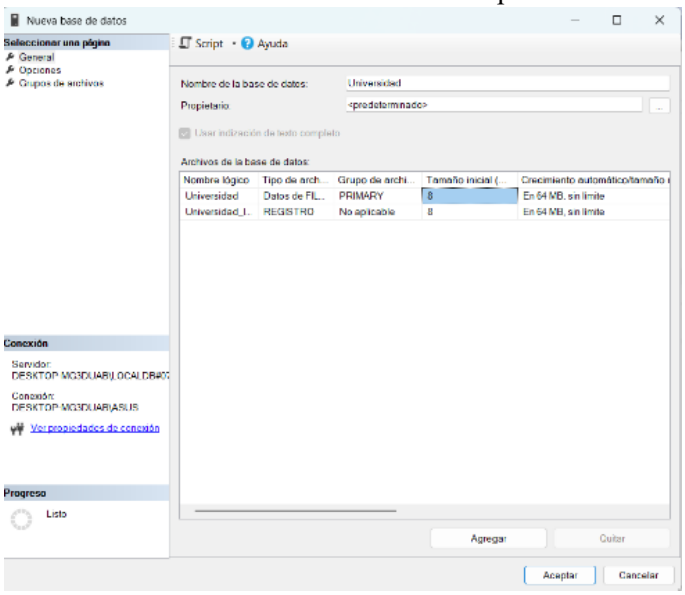


Ilustración 1. Creación de la Base de datos Universidad.

Este paso permite disponer de un entorno controlado donde se puedan realizar las pruebas sin afectar otras bases de datos.

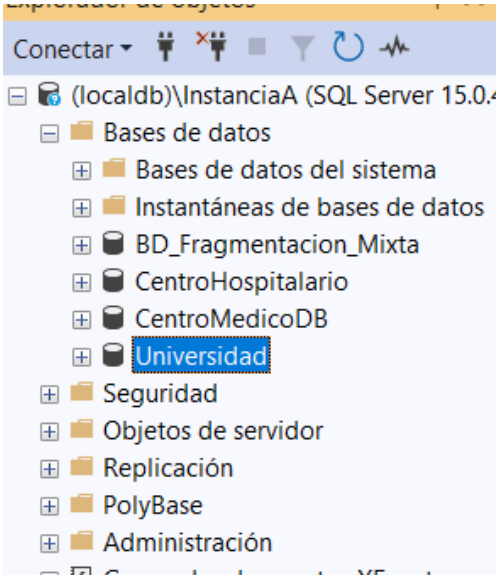


Ilustración 2. Verificación del BD creada

2.7.2 Creación de tablas con integridad referencial

Se crean tres tablas: Tabla A como tabla principal con clave primaria, Tabla B con una clave foránea que referencia a la Tabla A (implementando integridad referencial), y Tabla C como tabla independiente sin relaciones.

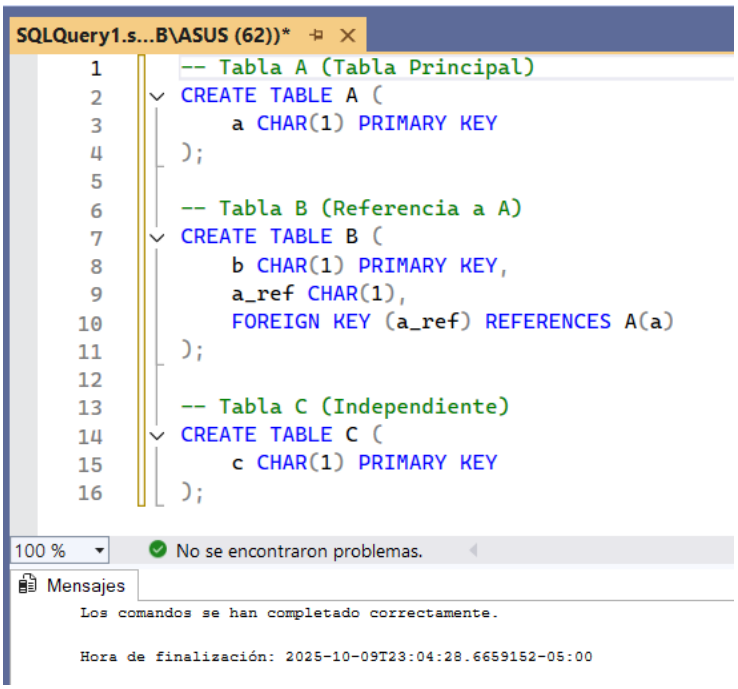


Ilustración 3. Creación de tablas A, B y C

Tablas creadas con sus relaciones

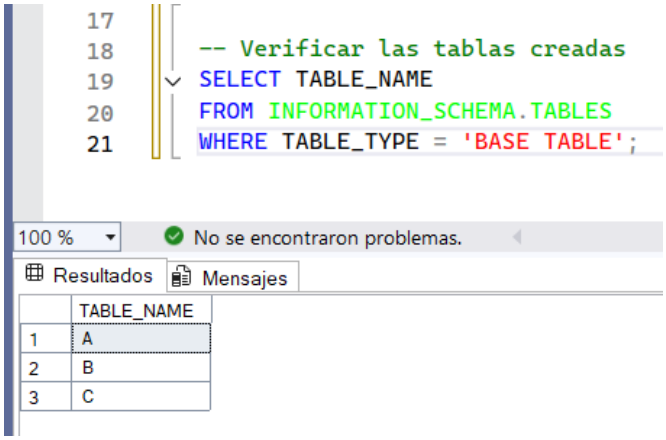


Ilustración 4. Verificación de tablas creadas

La integridad referencial asegura que los datos en las tablas relacionadas mantengan coherencia, siendo fundamental para comprobar el efecto de las transacciones.

2.7.3 Inserción de datos y verificación de integridad referencial

Se procede a insertar datos de prueba en las tres tablas, respetando las restricciones de integridad referencial.

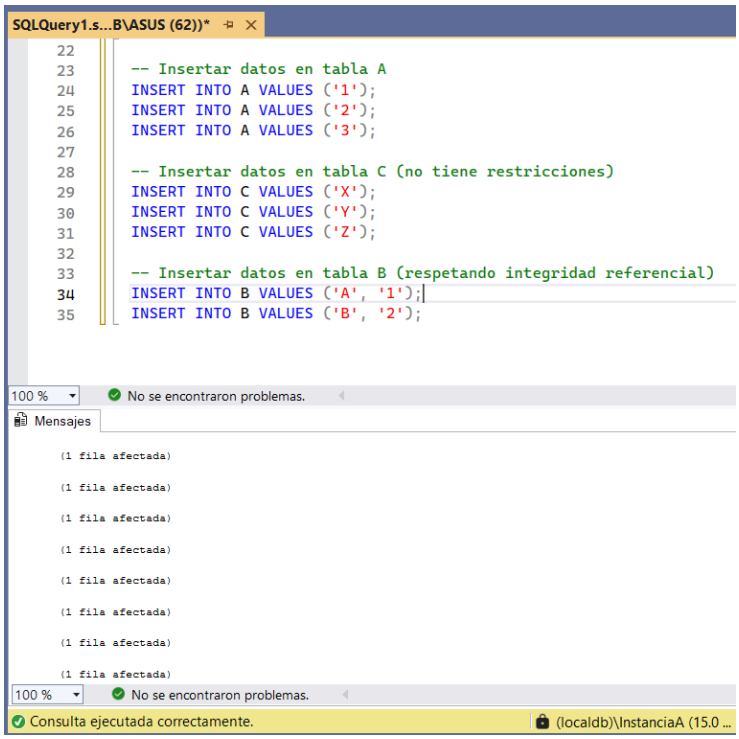


Ilustración 5. Inserción de datos en las tablas A, B y C

Los datos fueron insertados exitosamente en las tres tablas, respetando las restricciones de integridad referencial.

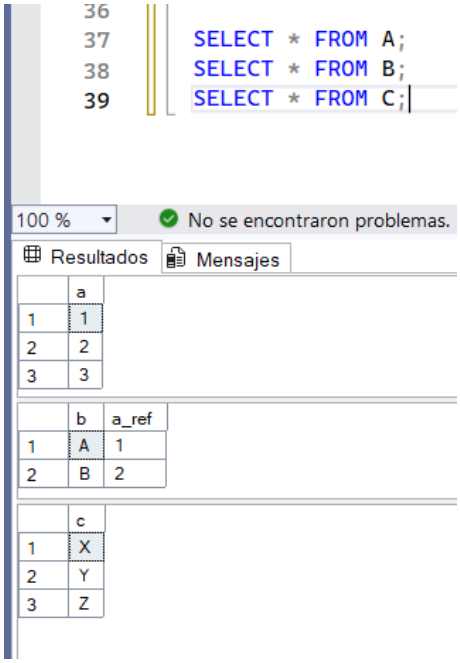


Ilustración 6. Consulta de datos insertados

2.7.4 Probar violación de integridad referencial

Para comprobar el funcionamiento de la integridad referencial, se intenta insertar un registro en la Tabla B con un valor que NO existe en la Tabla A. Esta operación debe ser rechazada por el sistema de gestión de base de datos.

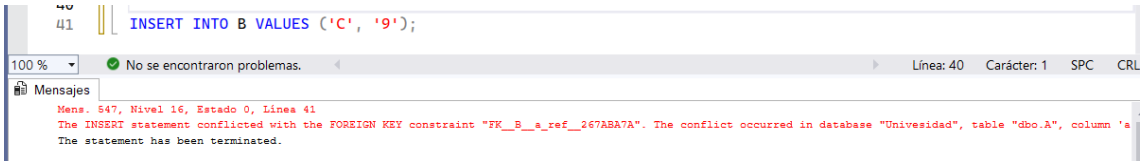


Ilustración 7.Error de violación de integridad referencial

2.7.5 Transacciones – COMMIT (Confirmar cambios)

En esta etapa se estudió el comportamiento de una transacción cuando todas las operaciones son exitosas y se confirma con el comando **COMMIT**.

Iniciar una transacción

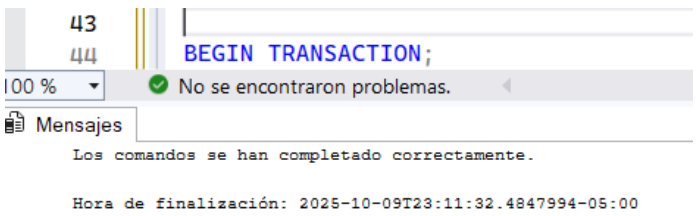


Ilustración 8. Inicio de transacción

Se realizaron inserciones en las tablas **A** y **B** con valores válidos.

```
45  INSERT INTO A VALUES ('4');
46  INSERT INTO B VALUES ('D', '4');
```

% No se encontraron problemas.

Mensajes

```
(1 fila afectada)

(1 fila afectada)

Hora de finalización: 2025-10-09T23:13:00.9718465-05:00
```

Ilustración 9. Inserción de datos dentro de la transacción

Antes de ejecutar el **COMMIT**, se verificó que los datos insertados solo eran visibles en la sesión actual, sin haberse guardado de forma definitiva.

```
48  SELECT 'Datos ANTES del COMMIT' AS Estado;
49  SELECT * FROM A;
50  SELECT * FROM B;
```

100 % No se encontraron problemas.

Resultados Mensajes

	Estado
1	Datos ANTES del COMMIT

	a
1	1
2	2
3	3
4	4

	b	a_ref
1	A	1
2	B	2
3	D	4

Ilustración 10. Verificación de datos antes del COMMIT

Finalmente, se aplicó el comando `COMMIT TRANSACTION` para confirmar los cambios.

```
52  COMMIT TRANSACTION;
```

% No se encontraron problemas.

Mensajes

```
Los comandos se han completado correctamente.

Hora de finalización: 2025-10-09T23:15:02.5502279-05:00
```

Ilustración 11. Confirmación del COMMIT

Verificar datos después del COMMIT

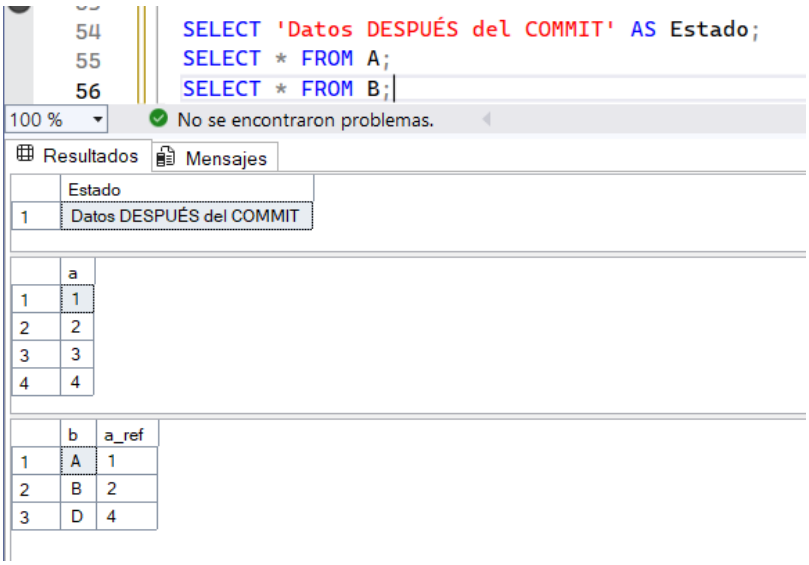


Ilustración 12. Verificación de datos después del COMMIT

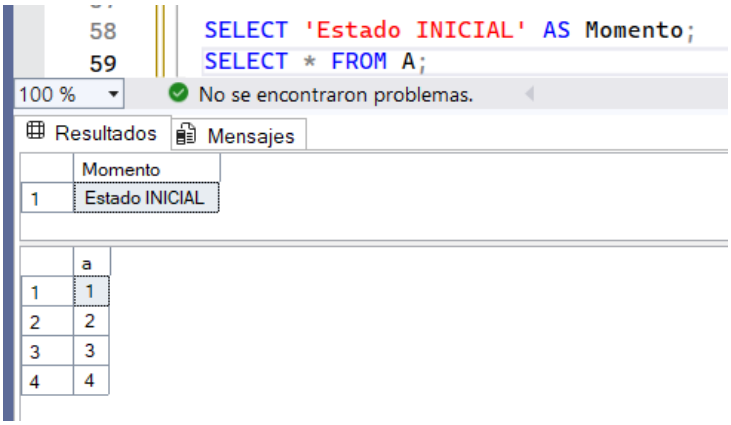
Dentro de la transacción se insertan nuevos valores ('4' en Tabla A y 'D' en Tabla B). Al consultar los datos antes del COMMIT, estos son visibles únicamente en la sesión actual.

Este comportamiento demuestra la propiedad de **atomicidad**, en la que todas las operaciones de la transacción se ejecutan completamente o no se ejecutan en absoluto.

2.7.6 Transacciones – ROLLBACK (Revertir cambios)

El comando ROLLBACK permite revertir todos los cambios realizados dentro de una transacción que aún no ha sido confirmada. Esto garantiza la atomicidad: si algo falla, todos los cambios se deshacen.

Mostrar estado inicial de las tablas.



Se inició una transacción

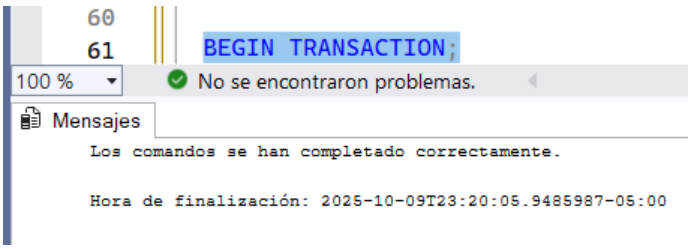


Ilustración 13.Inicio de una transacción

Insertar datos

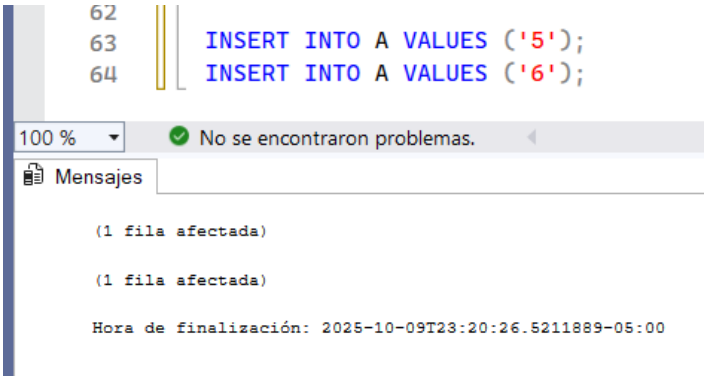


Ilustración 14.Inserción de datos

Verificar datos dentro de la transacción

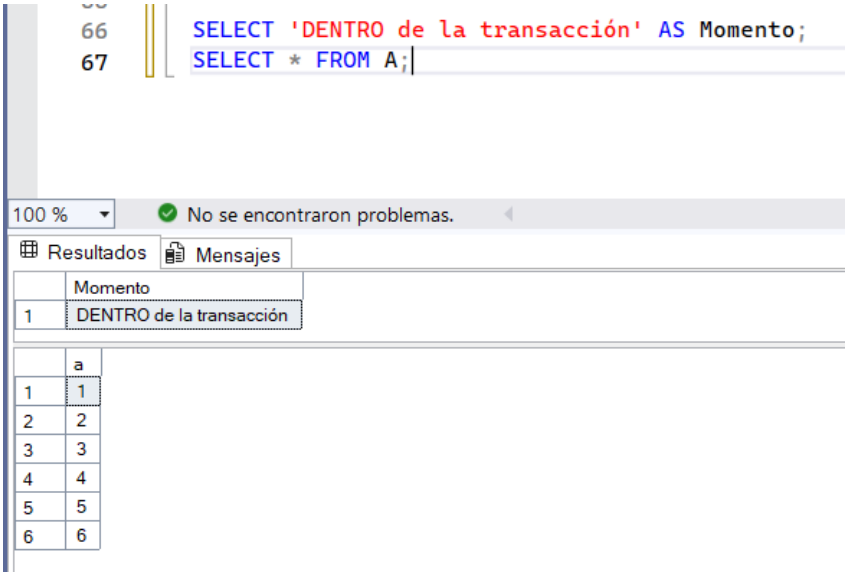


Ilustración 15.Verificación de datos de la transacción

Se ejecutó un ROLLBACK TRANSACTION para revertir los cambios antes del COMMIT

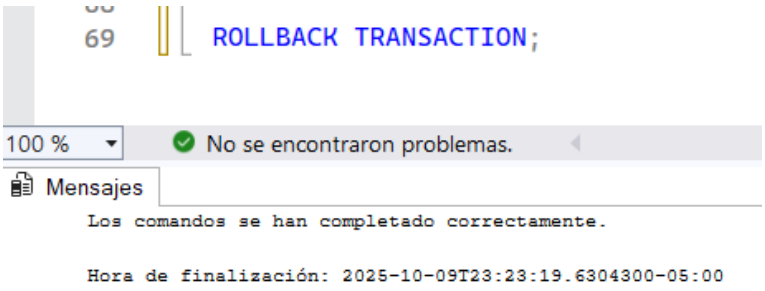


Ilustración 16.Ejecución del rollback

Ninguno de los registros insertados dentro de la transacción fue almacenado definitivamente. Al consultar las tablas, se confirmó que los datos regresaron a su estado original

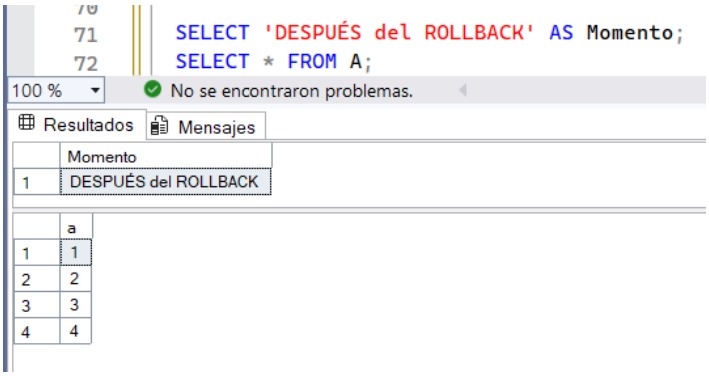


Ilustración 17.Verificación de datos no guardados

2.7.7 Comprobación de la propiedad de ATOMICIDAD — "Todo o nada"

Para validar la propiedad de atomicidad, se realizaron dos casos prácticos que demostraron cómo SQL Server garantiza que una transacción se ejecute en su totalidad o, en caso de error, se revierta completamente sin dejar rastros de operaciones parciales.

Caso 1: Transacción exitosa (todo se ejecuta)

En este caso, se ejecutó una transacción con instrucciones válidas que no generaron errores:

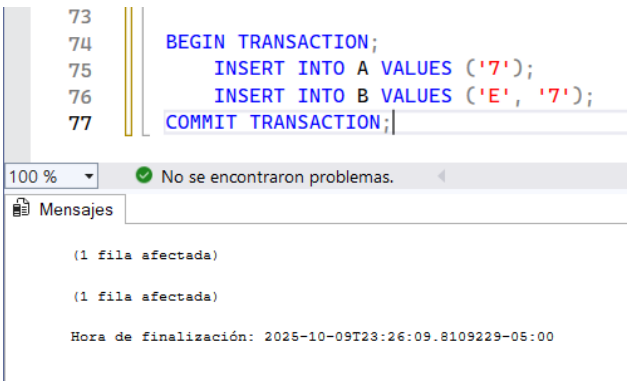


Ilustración 18.Inicio de transacción exitosa

Caso 2: Transacción fallida (nada se ejecuta), Este INSERT fallará por integridad referencial ya que el '99' NO existe en A

A continuación, se realizó una prueba intencional para provocar un error de integridad referencial dentro de la misma estructura transaccional:

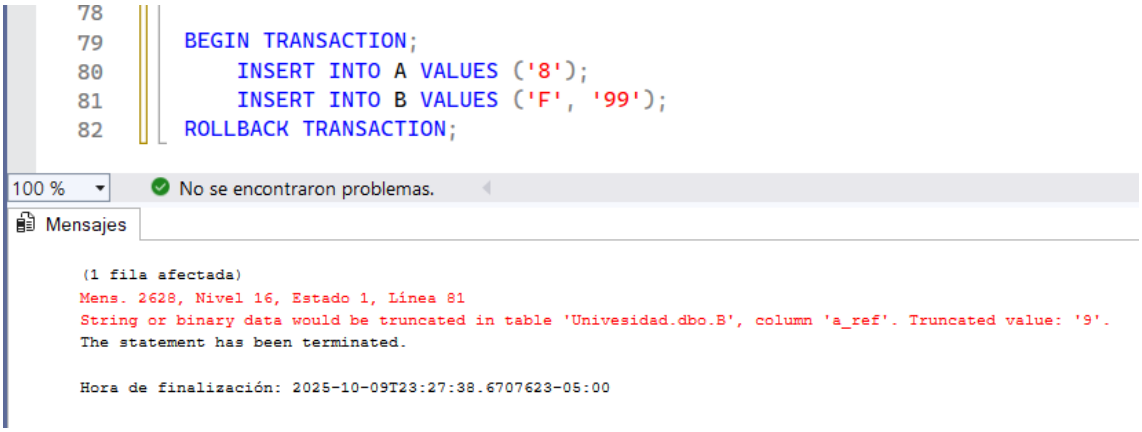


Ilustración 19.Fallo de transacción

se verificó que el valor '8' (insertado en una prueba anterior) permanecía, pero el nuevo valor `'9'` no fue agregado a la tabla **A**.

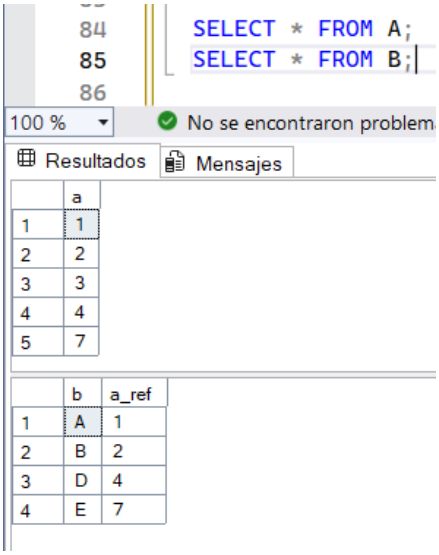


Ilustración 20.Anulación de transacción

2.7.8 Pruebas de sesiones concurrentes (Abrir NUEVA VENTANA de consulta)

Para demostrar el control de bloqueo y aislamiento de transacciones, se trabajó con dos sesiones simultáneas en SQL Server.

Ventana 1 (Sesión 1)

NO hacer COMMIT todavía, dejar la transacción abierta

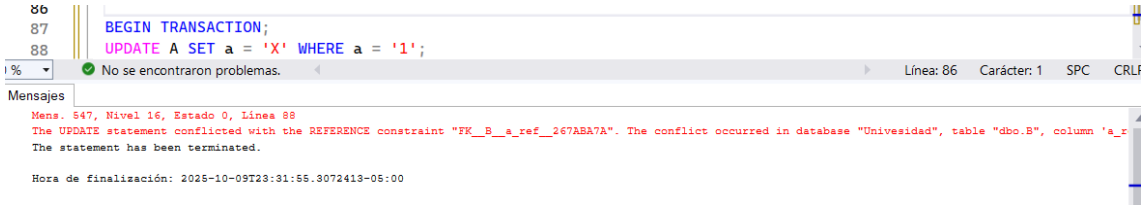


Ilustración 21.Inserción sin COMMIT

Ventana 2 (Sesión 2)

Esto se BLOQUEARÁ hasta que Sesión 1 haga COMMIT o ROLLBACK

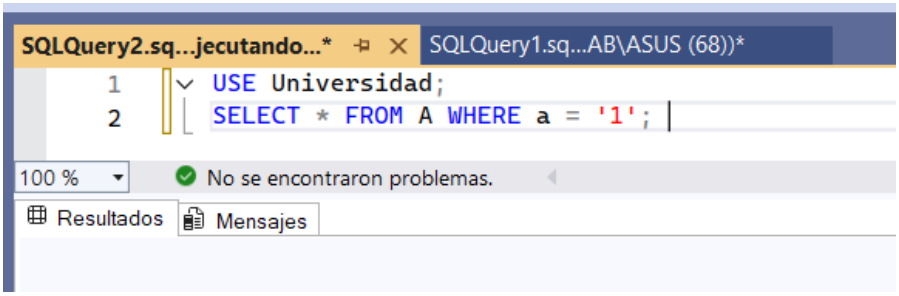


Ilustración 22.Bloque de consulta

Volver a Sesión 1 y hacer COMMIT

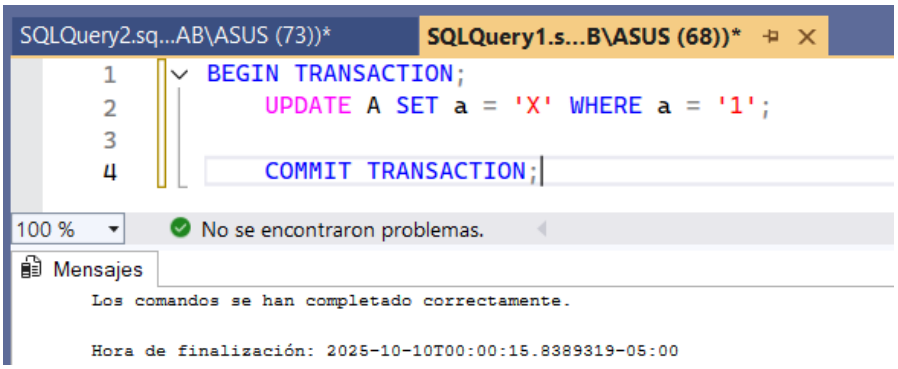


Ilustración 23.Realización de COMMIT en sesión 1

Sesión 2 desbloqueada después del COMMIT

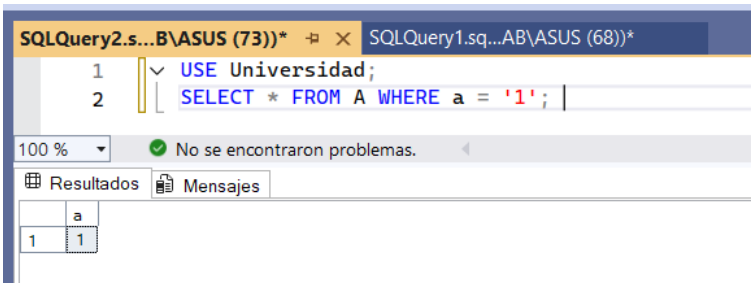


Ilustración 24.Desbloque de sesión 2

2.7.9 Manejo de errores – TRY...CATCH Limpiar tabla A para pruebas

Se implementó un bloque de control de errores mediante la estructura **TRY...CATCH**, para capturar excepciones durante la ejecución de transacciones.

Se limpiaron las tablas para realizar nuevas pruebas.

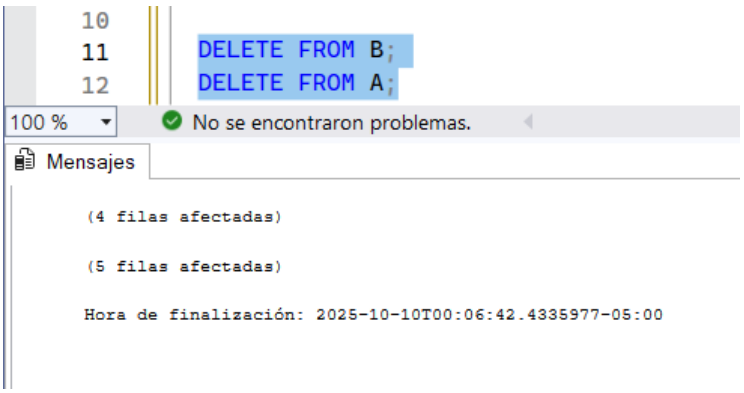


Ilustración 25.Limpieza de tablas

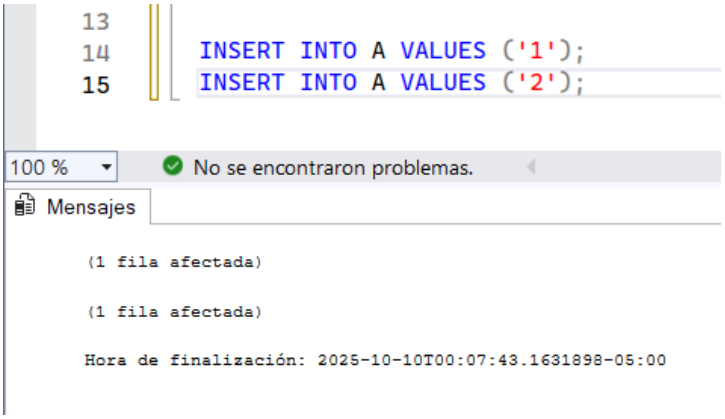


Ilustración 26.Inserción después de limpieza

Dentro del bloque TRY, se ejecutaron inserciones que provocaban una violación de integridad referencial.

TRY...CATCH básico, daría un ERROR: '999' ya que no existe en A

```
16
17  BEGIN TRY
18      BEGIN TRANSACTION;
19          INSERT INTO B VALUES ('A', '1');
20          INSERT INTO B VALUES ('B', '999'); -- ERROR: '999' no existe en A
21      COMMIT TRANSACTION;
22      PRINT 'Transacción exitosa';
23  END TRY
24  BEGIN CATCH
25      ROLLBACK TRANSACTION;
26      PRINT 'ERROR detectado - Transacción revertida';
27      PRINT 'Mensaje: ' + ERROR_MESSAGE();
28      PRINT 'Línea: ' + CAST(ERROR_LINE() AS VARCHAR);
29  END CATCH;
```

```
0 %   No se encontraron problemas.
Mensajes

(1 fila afectada)

(0 filas afectadas)
ERROR detectado - Transacción revertida
Mensaje: String or binary data would be truncated in table 'Universidad.dbo.B', column 'a_ref'. Truncated value: '9'.
Línea: 4

Hora de finalización: 2025-10-10T00:09:40.8399792-05:00
```

Ilustración 27.Ejecucion de error

Verificar que NO se insertó ningún dato. Manejo de errores con TRY...CATCH

```
30
31  SELECT * FROM B;
32
100 %   No se encontraron problemas.
Resultados   Mensajes
b | a_ref
```

Ilustración 28.Verificación de ningún dato insertado

2.7.10 Prueba con SET XACT_ABORT ON

Se analizó el comportamiento del sistema con la instrucción **SET XACT_ABORT ON**, que fuerza la cancelación automática de una transacción si ocurre un error.

Sin XACT_ABORT (comportamiento por defecto), esto dará un error, pero continua, se ejecutará aunque habrá un error.

```
33  DELETE FROM B;
34
35  BEGIN TRANSACTION;
36      INSERT INTO B VALUES ('A', '1'); -- OK
37      INSERT INTO B VALUES ('B', '999'); -- ERROR pero continúa
38      INSERT INTO B VALUES ('C', '2'); -- Se ejecuta aunque hubo error
39  COMMIT TRANSACTION;
40
```

No se encontraron problemas.

Mensajes

```
(2 filas afectadas)

(1 fila afectada)
Mens. 2628, Nivel 16, Estado 1, Línea 37
String or binary data would be truncated in table 'Universidad.dbo.B', column 'a_ref'. Truncated value: '9'.
The statement has been terminated.

(1 fila afectada)

Hora de finalización: 2025-10-10T00:17:53.8379434-05:00
```

Ilustración 29.Comportamiento por defecto del error

Sin XACT_ABORT - datos parciales insertados

```
40
41  SELECT 'Datos con XACT_ABORT OFF' AS Modo;
42  SELECT * FROM B;
```

No se encontraron problemas.

Resultados | Mensajes

	Modo
1	Datos con XACT_ABORT OFF

	b	a_ref
1	A	1
2	C	2

Ilustración 30.Datos insertados antes del error

Con XACT_ABORT ON (aborta automáticamente)

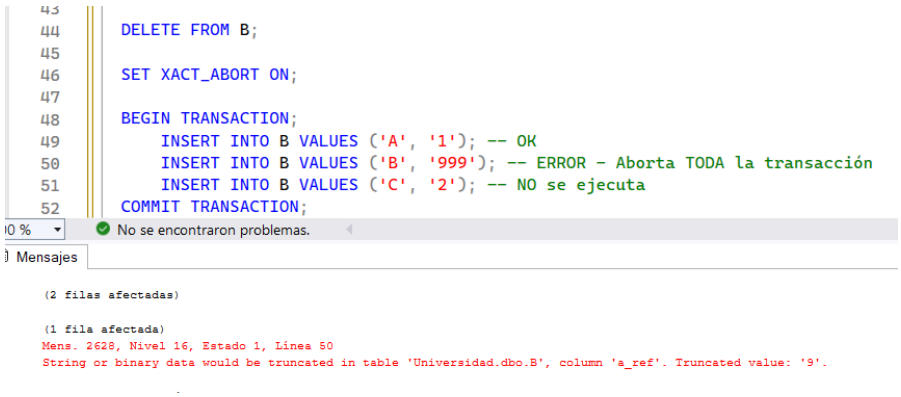


Ilustración 31.Error de aborto automático

Con XACT_ABORT - ningún dato insertado

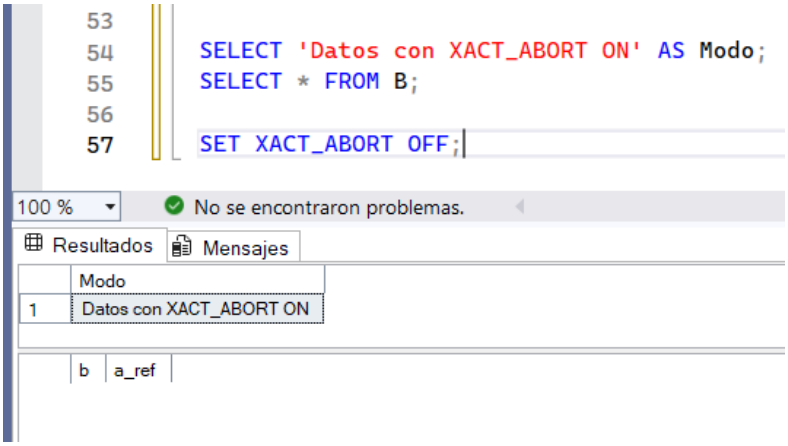


Ilustración 32.Ningun cambio producido tras error

2.7.11 Manejo completo de errores y estado inconsistente

Durante esta parte de la práctica se realizaron pruebas avanzadas para comprender cómo el sistema gestiona distintos tipos de errores dentro de una transacción y qué sucede cuando esta no se finaliza correctamente.

Manejo completo de diferentes tipos de errores

```
59  BEGIN TRY
60      BEGIN TRANSACTION;
61
62          -- Operaciones múltiples
63          INSERT INTO A VALUES ('5');
64          INSERT INTO B VALUES ('G', '5');
65          INSERT INTO C VALUES ('W');
66
67          -- Simular un error
68          -- RAISERROR('Error simulado', 16, 1);
69
70      COMMIT TRANSACTION;
71      PRINT '✓ Transacción completada exitosamente';
72
73  END TRY
74  BEGIN CATCH
75      IF @@TRANCOUNT > 0
76          ROLLBACK TRANSACTION;
77
78      -- Información detallada del error
79      SELECT
80          ERROR_NUMBER() AS ErrorNumber,
81          ERROR_SEVERITY() AS ErrorSeverity,
82          ERROR_STATE() AS ErrorState,
83          ERROR_PROCEDURE() AS ErrorProcedure,
84          ERROR_LINE() AS ErrorLine,
85          ERROR_MESSAGE() AS ErrorMessage;
86
87      PRINT '✗ Error - Transacción revertida';
88  END CATCH;
89
```

Ilustración 33.Manejo de diversos errores

Mensajes

```
(1 fila afectada)

(1 fila afectada)
? Transacción completada exitosamente

Hora de finalización: 2025-10-10T00:27:29.4005695-05:00
```

Ilustración 34.Mensaje de confirmaciones

ESTADO INCONSISTENTE (Demostración)

Ejemplo de cómo se genera un estado inconsistente

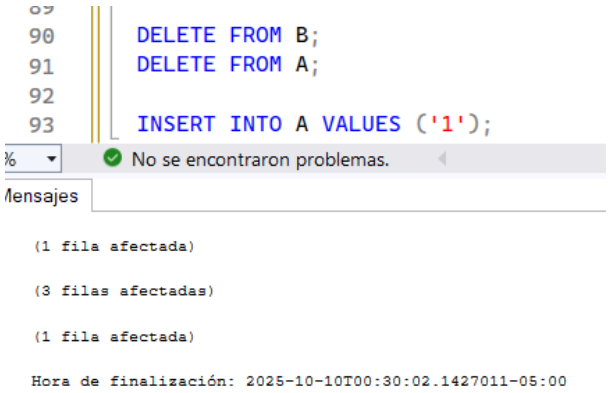


Ilustración 35.Inserción de datos en un estado inconsistente

Iniciar transacción y NO finalizarla “NO hacer COMMIT ni ROLLBACK”

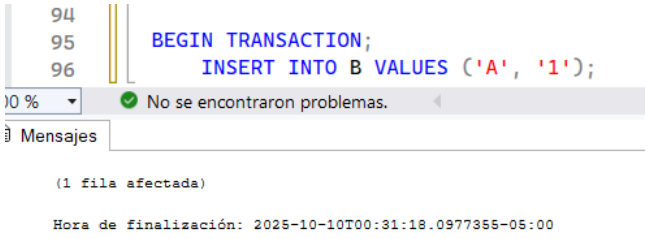


Ilustración 36.Inicio de transacción sin finalización

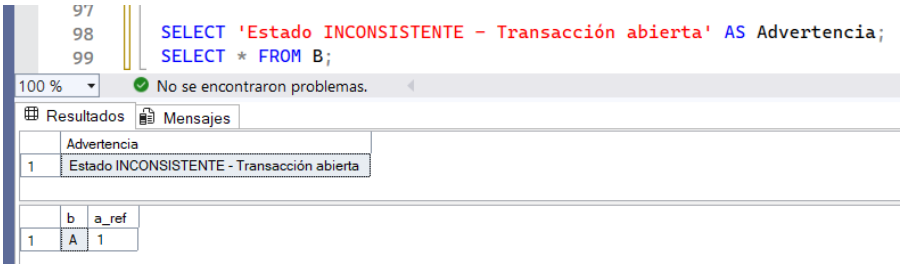


Ilustración 37.Datos de Transacción Abierta

En la Sesión 2, se verificó que los datos no eran visibles mientras la transacción permanecía abierta.

En sesión 2 no se verá el dato insertado pero en la sesión 1 sí se ve

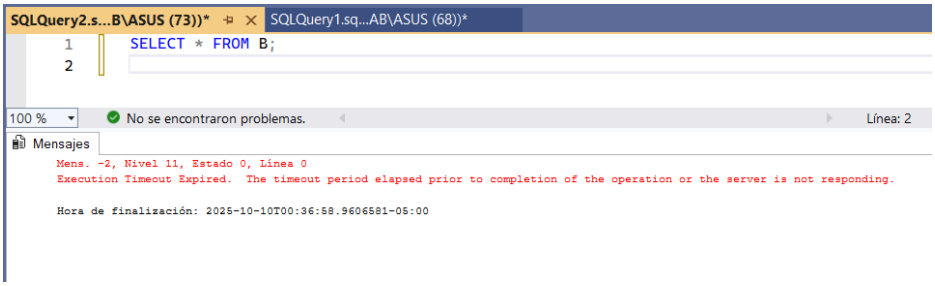


Ilustración 38.Sesión 2 sin datos

Para que nos aparezca el dato insertado en la sesión 2 debemos hacer COMMIT o ROLLBACK

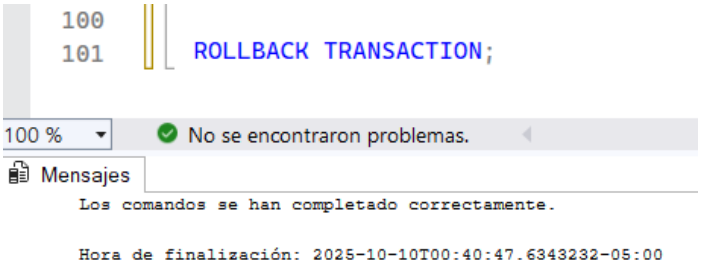


Ilustración 39.Ejecución de rollback en sesión 1

Demostración de estado inconsistente

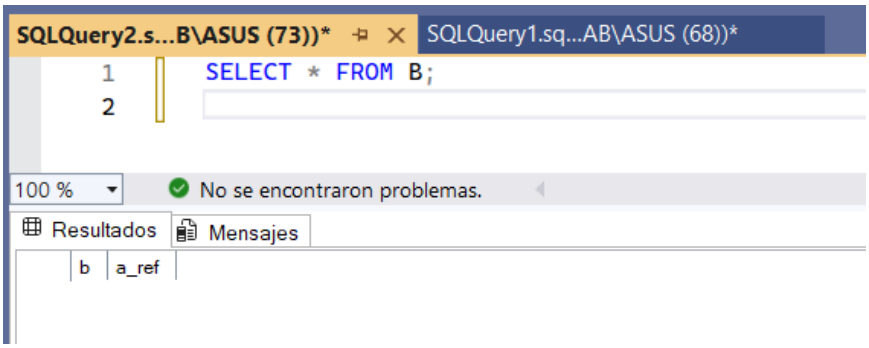


Ilustración 40.Datos sincronizados

Para finalizar la práctica, se realizó una verificación general del contenido de todas las tablas de la base de datos *Universidad*.
El objetivo fue comprobar cuántos registros permanecieron almacenados después de ejecutar las distintas transacciones y confirmar que el sistema mantuvo la consistencia de los datos.

Estado final de las tablas

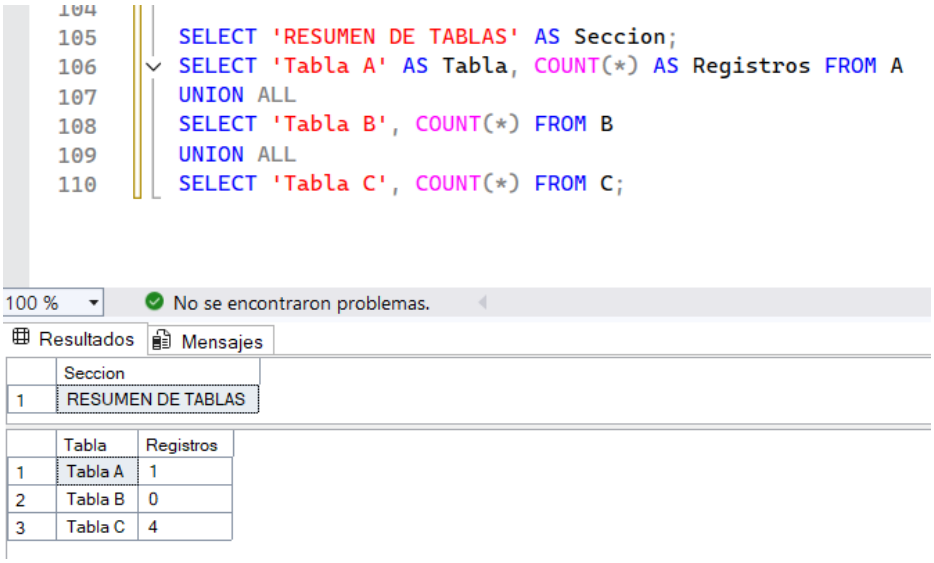


Ilustración 41.Reporte con conteo final de registros

Script

```
-- Tabla A (Tabla Principal)
CREATE TABLE A (
    a CHAR(1) PRIMARY KEY
);

-- Tabla B (Referencia a A)
CREATE TABLE B (
    b CHAR(1) PRIMARY KEY,
    a_ref CHAR(1),
    FOREIGN KEY (a_ref) REFERENCES A(a)
);

-- Tabla C (Independiente)
CREATE TABLE C (
    c CHAR(1) PRIMARY KEY
);

-- Verificar las tablas creadas
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE';

-- Insertar datos en tabla A
INSERT INTO A VALUES ('1');
INSERT INTO A VALUES ('2');
INSERT INTO A VALUES ('3');

-- Insertar datos en tabla C (no tiene restricciones)
INSERT INTO C VALUES ('X');
INSERT INTO C VALUES ('Y');
INSERT INTO C VALUES ('Z');

-- Insertar datos en tabla B (respetando integridad referencial)
INSERT INTO B VALUES ('A', '1');
INSERT INTO B VALUES ('B', '2');

SELECT * FROM A;
SELECT * FROM B;
SELECT * FROM C;

INSERT INTO B VALUES ('C', '9');

BEGIN TRANSACTION;

INSERT INTO A VALUES ('4');
INSERT INTO B VALUES ('D', '4');

 SELECT 'Datos ANTES del COMMIT' AS Estado;
 SELECT * FROM A;
 SELECT * FROM B;

 COMMIT TRANSACTION;

SELECT 'Datos DESPUÉS del COMMIT' AS Estado;
SELECT * FROM A;
SELECT * FROM B;

SELECT 'Estado INICIAL' AS Momento;
```

```
SELECT * FROM A;

BEGIN TRANSACTION;

INSERT INTO A VALUES ('5');
INSERT INTO A VALUES ('6');

SELECT 'DENTRO de la transacción' AS Momento;
SELECT * FROM A;

ROLLBACK TRANSACTION;

SELECT 'DESPUÉS del ROLLBACK' AS Momento;
SELECT * FROM A;

BEGIN TRANSACTION;
    INSERT INTO A VALUES ('7');
    INSERT INTO B VALUES ('E', '7');
COMMIT TRANSACTION;

BEGIN TRANSACTION;
    INSERT INTO A VALUES ('8');
    INSERT INTO B VALUES ('F', '99');
ROLLBACK TRANSACTION;

SELECT * FROM A;
SELECT * FROM B;

BEGIN TRANSACTION;
UPDATE A SET a = 'X' WHERE a = '1';

--Sesion 1
BEGIN TRANSACTION;
    UPDATE A SET a = 'X' WHERE a = '1';---sin el commit
--luego ejecutar el commit
    COMMIT TRANSACTION;

--Sesion 2
USE Universidad;
SELECT * FROM A WHERE a = '1';

DELETE FROM B;
DELETE FROM A;

INSERT INTO A VALUES ('1');
INSERT INTO A VALUES ('2');

BEGIN TRY
    BEGIN TRANSACTION;
        INSERT INTO B VALUES ('A', '1');
        INSERT INTO B VALUES ('B', '999'); -- ERROR: '999' no existe en A
    COMMIT TRANSACTION;
    PRINT 'Transacción exitosa';
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION;
    PRINT 'ERROR detectado - Transacción revertida';
    PRINT 'Mensaje: ' + ERROR_MESSAGE();
    PRINT 'Línea: ' + CAST(ERROR_LINE() AS VARCHAR);
END CATCH;
```

```
SELECT * FROM B;

DELETE FROM B;

BEGIN TRANSACTION;
    INSERT INTO B VALUES ('A', '1'); -- OK
    INSERT INTO B VALUES ('B', '999'); -- ERROR pero continúa
    INSERT INTO B VALUES ('C', '2'); -- Se ejecuta aunque hubo error
COMMIT TRANSACTION;

SELECT 'Datos con XACT_ABORT OFF' AS Modo;
SELECT * FROM B;

DELETE FROM B;

SET XACT_ABORT ON;

BEGIN TRANSACTION;
    INSERT INTO B VALUES ('A', '1'); -- OK
    INSERT INTO B VALUES ('B', '999'); -- ERROR - Aborta TODA la transacción
    INSERT INTO B VALUES ('C', '2'); -- NO se ejecuta
COMMIT TRANSACTION;

SELECT 'Datos con XACT_ABORT ON' AS Modo;
SELECT * FROM B;

SET XACT_ABORT OFF;

BEGIN TRY
    BEGIN TRANSACTION;

        -- Operaciones múltiples
        INSERT INTO A VALUES ('5');
        INSERT INTO B VALUES ('G', '5');
        INSERT INTO C VALUES ('W');

        -- Simular un error
        -- RAISERROR('Error simulado', 16, 1);

    COMMIT TRANSACTION;
    PRINT '✓ Transacción completada exitosamente';

END TRY
BEGIN CATCH
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;

    -- Información detallada del error
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;

    PRINT '✗ Error - Transacción revertida';
END CATCH;
```

```
DELETE FROM B;
DELETE FROM A;

INSERT INTO A VALUES ('1');

BEGIN TRANSACTION;
    INSERT INTO B VALUES ('A', '1');

SELECT 'Estado INCONSISTENTE - Transacción abierta' AS Advertencia;
SELECT * FROM B;

ROLLBACK TRANSACTION;

SELECT * FROM B;

SELECT 'RESUMEN DE TABLAS' AS Seccion;
SELECT 'Tabla A' AS Tabla, COUNT(*) AS Registros FROM A
UNION ALL
SELECT 'Tabla B', COUNT(*) FROM B
UNION ALL
SELECT 'Tabla C', COUNT(*) FROM C;
```

2.8 Habilidades blandas empleadas en la práctica

☐ Liderazgo
☒ Trabajo en equipo
☐ Comunicación asertiva
☐ La empatía
☐ Pensamiento crítico
☐ Flexibilidad
☐ La resolución de conflictos
☐ Adaptabilidad
☐ Responsabilidad

- El trabajo en equipo se lo organizo en un repositorio de GitHub https://github.com/KevinTixilema/APE-2.-Tratamiento-de-transacciones.git

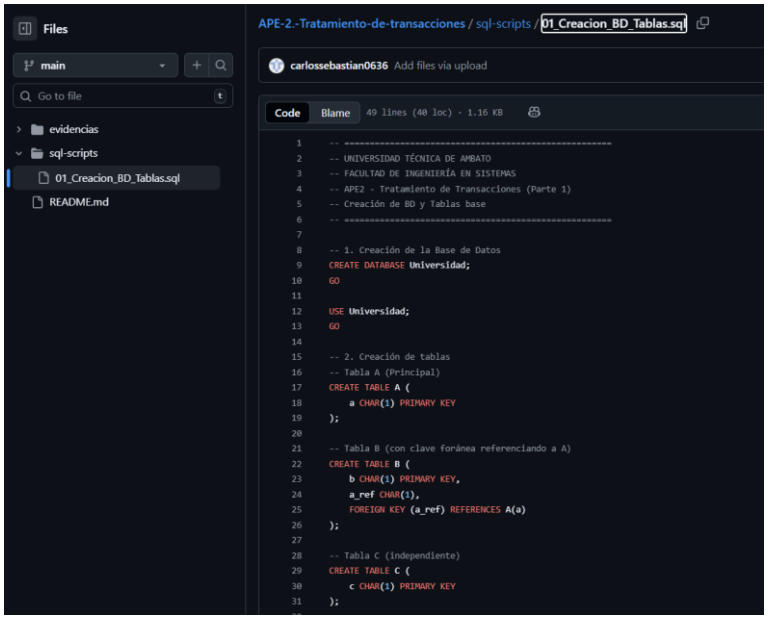


Ilustración 43. Aportación de Cholota Guamán Carlos Sebastián

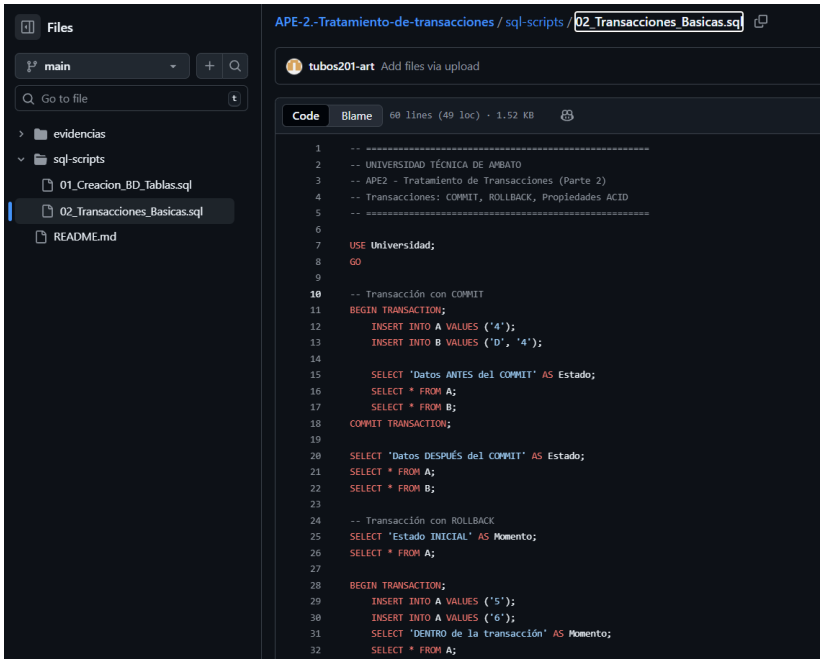


Ilustración 42. Aporte de Tubon Chipantiza Danilo Alexander

2.9 Conclusiones

- Al manejar transacciones se debe cuidar de no producir un estado inconsistente de la BD.
- La captura de errores va a depender del tipo de error y su severidad.
- Las transacciones permiten mantener la integridad y coherencia de los datos dentro de una base de datos.
- Los comandos COMMIT y ROLLBACK garantizan la propiedad de atomicidad al confirmar o revertir los cambios realizados.

- El uso de TRY...CATCH y SET XACT_ABORT mejora el control de errores y evita inconsistencias en la base de datos.

- SQL Server cumple con las propiedades ACID, asegurando que las operaciones sean confiables y seguras.

- Un manejo incorrecto de las transacciones puede generar bloqueos o estados inconsistentes, por lo que deben finalizarse siempre con COMMIT o ROLLBACK.

2.10 Recomendaciones

- Un sistema transaccional se complica muchísimo en ambientes distribuidos

- Finalizar siempre las transacciones con COMMIT o ROLLBACK para evitar estados inconsistentes.

- Utilizar TRY...CATCH o SET XACT_ABORT en los procedimientos para manejar errores de forma controlada.

- Verificar la integridad referencial antes de ejecutar transacciones que involucren varias tablas.

- Evitar mantener transacciones abiertas por mucho tiempo para prevenir bloqueos entre sesiones concurrentes.

- Documentar cada paso de las pruebas para facilitar la comprensión y el control del proceso transaccional.

2.11 Referencias bibliográficas

[1] "ProQuest Ebook Central - Reader," *Proquest.com*, 2025. https://ebookcentral.proquest.com/lib/uta-ebooks/reader.action?docID=821996&query=sql+server+replication&c=RVBVQg&pg=1 (accessed Oct. 10, 2025).
[2] "*Postgresql 10 administration cookbook : Over 165 effective recipes for database management and maintenance in postgresql 10*. (2018). Packt Publishing, Limited. (accessed Oct. 10, 2025).

2.12 Anexos