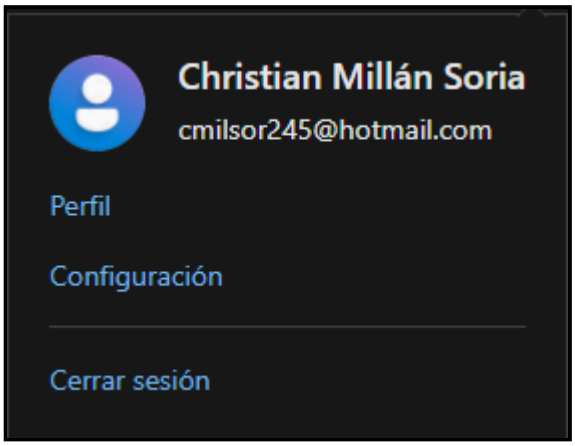
CUARTO CURSO

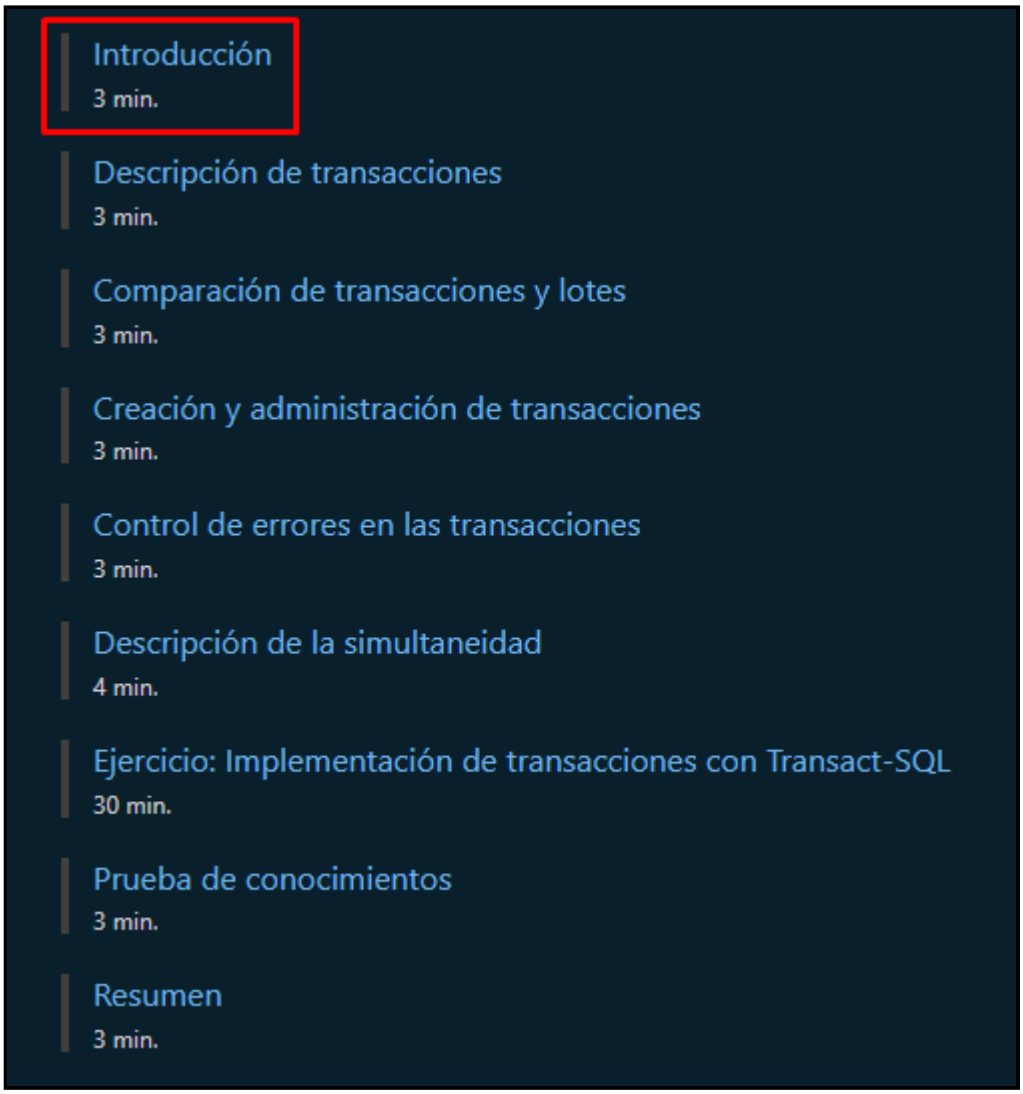
CHRISTIAN MILLÁN SORIA

Comenzamos entrando en este enlace.

Iniciamos sesión con una cuenta de Hotmail.



Una vez tenemos una cuenta, bajamos y encontramos una serie de apartados. Entramos en el primero.



INTRODUCCIÓN

En este módulo, aprenderá a construir transacciones para controlar el comportamiento de varias instrucciones Transact-SQL (T-SQL). Aprenderá a determinar si se ha producido algún error y cuándo revertir instrucciones.

Después de completar este módulo, podrá:

- Describir transacciones
- Comparar transacciones y lotes
- Crear y administrar transacciones
- Controlar errores en las transacciones
- Describir la simultaneidad

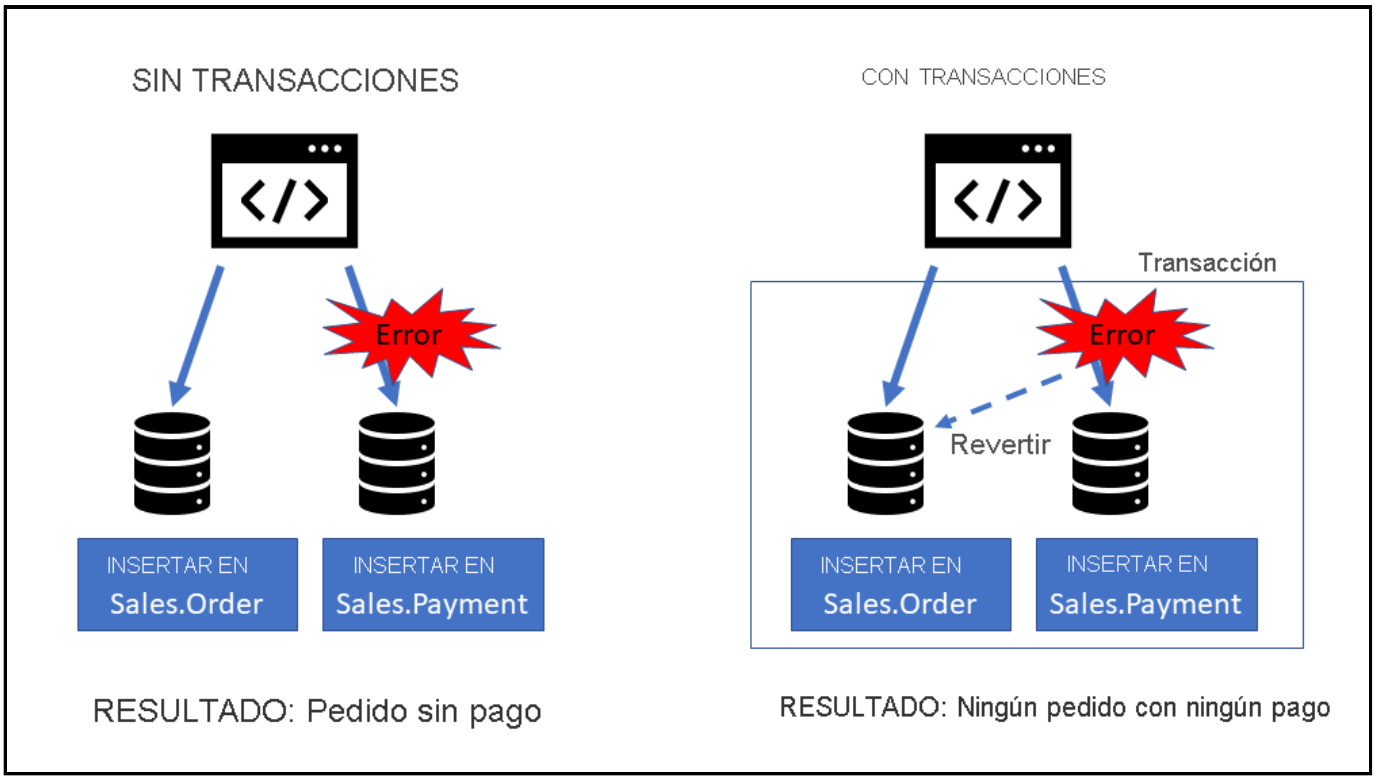
DESCRIPCIÓN DE TRANSACCIONES

Una transacción se refiere a una o varias instrucciones T-SQL que se tratan como una unidad. Si se produce un error en una sola transacción, se producirá un error en todas las instrucciones. Si una transacción se completa correctamente, sabrá que todas las instrucciones de modificación de datos de la transacción se realizaron de manera correcta y que se confirmaron en la base de datos.

Las transacciones garantizan que todas las instrucciones dentro de una transacción se completen correctamente o que se producirá un error en todas: no se permite ninguna finalización parcial. Las transacciones encapsulan las operaciones que deben producirse de forma lógica, como varias entradas en tablas relacionadas que forman parte de una sola operación.

Considere una empresa que almacena las compras en una tabla Sales.Order y los pagos en una tabla Sales.Payment. Cuando alguien compra algo, ambas tablas deben actualizarse. Si esto se implementa sin transacciones y se produce un error cuando el pago se escribe en la base de datos, se seguirá confirmando la inserción Sales.Order, dejando la tabla de pago sin una entrada.

Cuando se implementa con transacciones, se realizan ambas entradas o no se realiza ninguna entrada. Si se produce un error al escribir el pago en la tabla, también se revertirá la inserción del pedido. Esto significa que la base de datos siempre está en un estado coherente.



Se debe tener en cuenta que esto hace referencia a errores graves, como errores de hardware o de red. Los errores en instrucciones SQL solo harían que la transacción se revierta en circunstancias determinadas y es importante revisar las unidades posteriores de este módulo a fin de comprender completamente lo que implica el uso de transacciones.

Hay varios tipos de transacciones:

TRANSACCIONES EXPLÍCITAS

Las palabras clave BEGIN TRANSACTION y COMMIT o ROLLBACK inician y finalizan cada lote de instrucciones. Esto le permite especificar qué instrucciones se deben confirmar o revertir en conjunto.

TRANSACCIONES IMPLÍCITAS

Una transacción se inicia cuando se completa la transacción anterior. Cada transacción se completa explícitamente con una instrucción COMMIT o ROLLBACK.

CARACTERÍSTICAS ACID

Los sistemas de procesamiento de transacciones en línea (OLTP) requieren transacciones para cumplir las características "ACID":

· Atomicidad: cada transacción se trata como una unidad única, la cual se completa correctamente o produce un error general. Por ejemplo, una transacción que conlleve el adeudo de fondos de una cuenta y el abono de la misma cantidad en otra debe completar ambas acciones. Si alguna de las acciones no se puede completar, se debe producir un error en la otra.
· Coherencia: las transacciones solo pueden pasar los datos de la base de datos de un estado válido a otro. Para continuar con el ejemplo anterior del adeudo y el abono, el estado completado de la transacción debe reflejar la transferencia de fondos de una cuenta a la otra.
· Aislamiento: las transacciones simultáneas no pueden interferir entre sí y deben dar lugar a un estado coherente de la base de datos. Por ejemplo, mientras la transacción para transferir fondos de una cuenta a otra está en proceso, otra transacción que comprueba el saldo de las cuentas debe devolver resultados coherentes. Es decir, la transacción de comprobación del saldo no puede recuperar un valor para una cuenta que refleje el saldo antes de la transferencia y un valor para la otra cuenta que refleje el saldo después de la transferencia.
· Durabilidad: cuando se ha confirmado una transacción, permanece confirmada. Una vez que la transacción de transferencia de la cuenta se ha completado, los saldos revisados de las cuentas se conservan, de modo que, incluso si el sistema de base de datos se desactiva, la transacción confirmada se refleje cuando se vuelva a activar.

COMPARACIÓN DE TRANSACCIONES Y LOTES

Resulta útil comparar el comportamiento de los lotes de T-SQL, incluidos dentro de un bloque TRY/CATCH, con el comportamiento de las transacciones.

Tenga en cuenta el código siguiente que inserta dos pedidos de cliente y que requiere una fila en la tabla SalesLT.SalesOrderHeader y una o varias filas en la tabla SalesLT.SalesOrderDetail. Todas las instrucciones INSERT se incluyen dentro del bloque TRY.

· Si se produce un error en la primera inserción, la ejecución pasa al bloque CATCH y no se ejecuta más código.
· Si se produce un error en la segunda inserción, la ejecución pasa al bloque CATCH y no se ejecuta más código. Sin embargo, la primera inserción se realizó correctamente y no se revierte, lo que deja la base de datos en un estado incoherente. Se insertó una fila para el pedido, pero no para los detalles del pedido.

```
BEGIN TRY
    INSERT INTO dbo.Orders(custid, empid, orderdate)
        VALUES (68, 9, '2021-07-12');
    INSERT INTO dbo.Orders(custid, empid, orderdate)
        VALUES (88, 3, '2021-07-15');
    INSERT INTO dbo.OrderDetails(orderid,productid,unitprice,qty)
        VALUES (1, 2, 15.20, 20);
    INSERT INTO dbo.OrderDetails(orderid,productid,unitprice,qty)
```

```
        VALUES (999, 77, 26.20, 15);
END TRY
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrNum, ERROR_MESSAGE() AS ErrMsg;
END CATCH;
```

Compare esto con la implementación del código dentro de una transacción. El bloque TRY/CATCH se sigue utilizando para el control de errores; sin embargo, las instrucciones INSERT de las tablas Orders y OrderDetails se incluyen dentro de las palabras clave BEGIN TRANSACTION/COMMIT TRANSACTION. Esto garantiza que todas las instrucciones se tratan como una transacción única, que puede completarse correctamente o generar un error. Una fila se escribe tanto en la tabla Orders como en la tabla OrderDetails, o bien no se inserta ninguna fila. De esta manera, la base de datos nunca puede estar en un estado incoherente.

```
BEGIN TRY
 BEGIN TRANSACTION;
    INSERT INTO dbo.Orders(custid, empid, orderdate)
        VALUES (68,9,'2006-07-15');
    INSERT INTO dbo.OrderDetails(orderid,productid,unitprice,qty)
        VALUES (99, 2,15.20,20);
    COMMIT TRANSACTION;
END TRY
BEGIN CATCH
 SELECT ERROR_NUMBER() AS ErrNum, ERROR_MESSAGE() AS ErrMsg;
 ROLLBACK TRANSACTION;
END CATCH;
```

CREACIÓN Y ADMINISTRACIÓN DE TRANSACCIONES

Para iniciar explícitamente una transacción, use BEGIN TRANSACTION o la versión abreviada, BEGIN TRAN.

Una vez que se inicia una transacción, debe finalizar con:

- COMMIT TRANSACTION o
- ROLLBACK TRANSACTION.

Esto garantiza que todas las instrucciones de la transacción se confirman juntas o, si se produce un error, se revierten juntas.

Las transacciones duran hasta que se emite un comando COMMIT TRANSACTION o ROLLBACK TRANSACTION o se descarta la conexión. Si la conexión se descarta en el transcurso de una transacción, se revierte toda la transacción.

Las transacciones pueden estar anidadas, en cuyo caso las transacciones internas se revertirán si la transacción externa se revierte.

NO SE DETECTA NINGÚN ERROR

Cuando las instrucciones de la transacción se completen sin errores, use COMMIT TRANSACTION, que algunas veces se abrevia como COMMIT TRAN. De este modo, los datos se confirman en la base de datos. Esto también liberará recursos, como bloqueos, retenidos durante la transacción.

SÍ SE DETECTA UN ERROR

Si se produjo un error dentro de la transacción, use el comando ROLLBACK.

ROLLBACK deshace las modificaciones realizadas en los datos durante la transacción y los deja en el estado que tenían antes de que se iniciara la transacción. ROLLBACK también libera recursos, como bloqueos, retenidos para la transacción.

XACT_ABORT

Cuando el estado de SET XACT_ABORT es ON, si SQL Server genera un error, se revierte toda la transacción. Cuando el estado de SET XACT_ABORT es OFF, solo se revierte la instrucción que produjo el error si la gravedad del error es baja.

Por ejemplo, cuando el estado de SET XACT_ABORT es OFF, una transacción tiene tres instrucciones. Dos no tienen ningún error, pero la tercera interrumpe una restricción CHECK. En este ejemplo, aunque las tres instrucciones estén en una transacción, se confirman dos de ellas. En el mismo ejemplo, si el error se hubiera producido por un tipo de datos incorrecto, habría sido lo suficientemente grave como para emitir una reversión y no se habría confirmado ninguna de las instrucciones.

Como no siempre está claro si la transacción se confirma o se revierte, es esencial agregar el control de errores a las transacciones.

CONTROL DE ERRORES EN LAS TRANSACCIONES

El control estructurado de excepciones usa la construcción TRY/CATCH para comprobar si hay errores y, si los hay, controlarlos. Cuando usa el control de excepciones con transacciones, es importante colocar las palabras clave COMMIT o ROLLBACK en el lugar correcto en relación con los bloques TRY/CATCH.

CONFIRMACIÓN DE TRANSACCIONES

Cuando use transacciones con control estructurado de excepciones, coloque la palabra clave COMMIT de la transacción dentro del bloque TRY, como se muestra en el ejemplo de código siguiente:

```
BEGIN TRY
 BEGIN TRANSACTION
    INSERT INTO dbo.Orders(custid, empid, orderdate)
    VALUES (68,9,'2006-07-12');
    INSERT INTO dbo.OrderDetails(orderid,productid,unitprice,qty)
    VALUES (1, 2,15.20,20);
 COMMIT TRANSACTION
END TRY
```

REVERSIÓN DE TRANSACCIÓN

Cuando use transacciones con control estructurado de excepciones, coloque la palabra clave ROLLBACK de la transacción dentro del bloque CATCH, como se muestra en el ejemplo de código siguiente:

```
BEGIN TRY
 BEGIN TRANSACTION;
    INSERT INTO dbo.Orders(custid, empid, orderdate)
    VALUES (68,9,'2006-07-12');
    INSERT INTO dbo.OrderDetails(orderid,productid,unitprice,qty)
    VALUES (1, 2,15.20,20);
 COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrNum, ERROR_MESSAGE() AS ErrMsg;
    ROLLBACK TRANSACTION;
END CATCH;
```

XACT_STATE

Para evitar la reversión de una transacción activa, use la función XACT_STATE. XACT_STATE devuelve estos valores:

```
Valor devuelto                                  Significado
1                                               La solicitud actual tiene una
transacción de usuario activa y confirmable.
0                                               No hay ninguna transacción activa.
```

```
-1                                          La solicitud actual tiene una
transacción de usuario activa, pero se ha producido un error por el cual la
transacción se clasificó como no confirmable.

XACT_State se puede usar antes del comando ROLLBACK para comprobar si la
transacción está activa.

En el código siguiente se muestra la función XACT_STATE que se usa dentro del
bloque CATCH, de manera que la transacción solo se revierte si hay una transacción
de usuario activa.
```

```
BEGIN TRY
 BEGIN TRANSACTION;
    INSERT INTO dbo.SimpleOrders(custid, empid, orderdate)
    VALUES (68,9,'2006-07-12');
    INSERT INTO dbo.SimpleOrderDetails(orderid,productid,unitprice,qty)
    VALUES (1, 2,15.20,20);
 COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    SELECT ERROR_NUMBER() AS ErrNum, ERROR_MESSAGE() AS ErrMsg;
    IF (XACT_STATE()) <> 0
        BEGIN
        ROLLBACK TRANSACTION;
        END;
    ELSE .... -- provide for other outcomes of XACT_STATE()
END CATCH;
```

DESCRIPCIÓN DE LA SIMULTANEIDAD

```
Una característica principal de las bases de datos multiusuario es la
simultaneidad. La simultaneidad usa el bloqueo con el fin de que los datos sigan
siendo coherentes con muchos usuarios que actualizan y leen datos al mismo tiempo.
Por ejemplo, debido a los costos de envío, todos nuestros productos tienen un
aumento de precio de USD 5. Al mismo tiempo, debido al tipo de moneda, todos los
productos tienen una reducción del precio del 3 %. Si estas actualizaciones se
producen exactamente al mismo tiempo, el precio final será variable y es probable
que haya muchos errores. Mediante el bloqueo, puede asegurarse de que una
actualización se completará antes de que comience la otra.

La simultaneidad se produce en el nivel de transacción. Una transacción de
escritura puede impedir que otras transacciones actualicen e incluso lean los
mismos datos. Igualmente, una transacción de lectura puede bloquear a otros
lectores o incluso a algunos escritores. Por esta razón, es importante evitar
transacciones excesivamente largas o transacciones que abarquen cantidades
excesivas de datos.

Hay muchos niveles de aislamiento de transacción específicos que se pueden usar
```

para definir cómo un sistema de base de datos controla varios usuarios. Para los fines de este módulo, veremos categorías amplias de nivel de aislamiento, bloqueo optimista y bloqueo pesimista.

SIMULTANEIDAD OPTIMISTA

Con el bloqueo optimista, se supone que se ocurrirán pocos conflictos de actualizaciones. Al principio de la transacción, se registra el estado inicial de los datos. Antes de que se confirme la transacción, el estado actual se compara con el estado inicial. Si los estados son iguales, se completa la transacción. Si los estados son diferentes, se revierte la transacción.

Por ejemplo, tiene una tabla que contiene los pedidos de ventas de los últimos años. Estos datos se actualizan con poca frecuencia, pero los informes se ejecutan a menudo. Mediante el bloqueo optimista, las transacciones no se bloquean entre sí y el sistema se ejecuta de manera más eficaz. Desafortunadamente, se encontraron errores en los datos de los últimos años y es necesario realizar actualizaciones. Mientras una transacción actualiza cada fila, al mismo tiempo otra realiza una edición secundaria en una sola fila. Como el estado de los datos cambió mientras se ejecutaba la transacción inicial, se revierte toda la transacción.

SIMULTANEIDAD PESIMISTA

Con el bloqueo pesimista, se supone se producen muchas actualizaciones en los datos al mismo tiempo. Mediante el uso de bloqueos, solo puede ocurrir una actualización al mismo tiempo y las lecturas de los datos no se pueden realizar mientras ocurren actualizaciones. Esto puede evitar reversiones de gran tamaño, como las del ejemplo anterior, pero puede hacer que las consultas se bloqueen innecesariamente.

Es importante tener en cuenta la naturaleza de los datos y las consultas que se ejecutan en los datos a la hora de decidir si se debe usar la simultaneidad optimista o pesimista a fin de garantizar un rendimiento óptimo.

AISLAMIENTO DE INSTANTÁNEA

Hay cinco niveles de aislamiento distintos en SQL Server, pero para este módulo solo nos centraremos en READ_COMMITTED_SNAPSHOT_OFF y READ_COMMITTED_SNAPSHOT_ON. READ_COMMITTED_SNAPSHOT_OFF es el nivel de aislamiento predeterminado para SQL Server. READ_COMMITTED_SNAPSHOT_ON es el nivel de aislamiento predeterminado para Azure SQL Database.

READ_COMMITTED_SNAPSHOT_OFF bloqueará las filas afectadas hasta el final de la transacción si la consulta usa el nivel de aislamiento de transacción de lectura confirmada. Si bien es posible que se produzcan algunas actualizaciones, como la creación de una fila, esto evitará la mayoría de los conflictos en los datos que se leen o actualizan. Esta es la simultaneidad pesimista.

READ_COMMITTED_SNAPSHOT_ON toma una instantánea de los datos. A continuación, se realizan actualizaciones en esa instantánea, lo que permite que otras conexiones consulten los datos originales. Al final de la transacción, el estado actual de los datos se compara con la instantánea. Si los datos son iguales, se confirma la transacción. Si los datos son distintos, la transacción se revierte.

Para cambiar el nivel de aislamiento a READ_COMMITTED_SNAPSHOT_ON, emita el comando siguiente:

```
ALTER DATABASE *db_name* SET READ_COMMITTED_SNAPSHOT ON;
```

Para cambiar el nivel de aislamiento a READ_COMMITTED_SNAPSHOT_OFF, emita el comando siguiente:

```
ALTER DATABASE *db_name* SET READ_COMMITTED_SNAPSHOT OFF;
```

Si se modificó la base de datos para activar la instantánea de lectura confirmada, cualquier transacción que use el nivel de aislamiento de lectura confirmada predeterminada usará el bloqueo optimista.

EJERCICIO

INSERT DATA WITHOUT TRANSACTIONS

4.

```
insert into SalesLT.Customer(NameStyle, FirstName, LastName, EmailAddress,
PasswordHash, PasswordSalt,    rowguid, ModifiedDate)
values(0,  'Norman','Newcustomer','norman0@adventure-
works.com','U1/CrPqSzwLTtwgBehfpIl7f1LHSFpZw1qnG1sMzFjo=','QhHP+y8=',NEWID(),
GETDATE());

insert into SalesLT.Address(AddressLine1, City, StateProvince, CountryRegion,
PostalCode, rowguid,    ModifiedDate)
values('6388 Lake City Way', 'Burnaby','British Columbia','Canada','V5A
3A6',NEWID(), GETDATE());

insert into SalesLT.CustomerAddress(CustomerID, AddressID, AddressType, rowguid,
ModifiedDate)
```

```
values(IDENT_CURRENT('SalesLT.Customer'), IDENT_CURRENT('SalesLT.Address'),
'Home', NEWID(), '12-1-20212');
```

```
Started executing query at Line 1
(1 row affected)
(1 row affected)
Msg 241, Level 16, State 1, Line 7
Conversion failed when converting date and/or time from character string.
Total execution time: 00:00:01.992
```

8.

```
select * from SalesLT.Customer order by ModifiedDate desc;
```

	CustomerID	NameStyle	Title	FirstName	MiddleName	LastName	Suffix	CompanyName	SalesPerson	EmailAddress	Phone
1	30119	0	*NULL*	Norman	*NULL*	Newcustomer	*NULL*	*NULL*	*NULL*	norman0@adventure-works.com	*NULL*
2	544	0	Mr.	Jay	*NULL*	Adams	*NULL*	Valley Bicycle Specialists	adventure-works\jillian0	jay1@adventure-works.com	158-5
3	29492	0	Mr.	Jay	*NULL*	Adams	*NULL*	Valley Bicycle Specialists	adventure-works\jillian0	jay1@adventure-works.com	158-5
4	29490	0	Ms.	Margaret	J.	Smith	*NULL*	Bicycle Accessories and Kits	adventure-works\david8	margaret0@adventure-works.com	959-5
5	488	0	Ms.	Margaret	J.	Smith	*NULL*	Bicycle Accessories and Kits	adventure-works\david8	margaret0@adventure-works.com	959-5
6	491	0	Ms.	Frances	B.	Adams	*NULL*	Area Bike Accessories	adventure-works\shu0	frances0@adventure-works.com	991-5
7	29489	0	Ms.	Frances	B.	Adams	*NULL*	Area Bike Accessories	adventure-works\shu0	frances0@adventure-works.com	991-5
8	29485	0	Ms.	Catherine	R.	Abel	*NULL*	Professional Sales and Servi…	adventure-works\linda3	catherine0@adventure-works.c…	747-5
9	29486	0	Ms.	Kim	*NULL*	Abercrombie	*NULL*	Riders Company	adventure-works\jillian0	kim2@adventure-works.com	334-5
10	579	0	Ms.	Kim	*NULL*	Abercrombie	*NULL*	Riders Company	adventure-works\jillian0	kim2@adventure-works.com	334-5
11	582	0	Ms.	Catherine	R.	Abel	*NULL*	Professional Sales and Servi…	adventure-works\linda3	catherine0@adventure-works.c…	747-5
12	655	0	Ms.	Irene	J.	Hernandez	*NULL*	Operational Manufacturing	adventure-works\garrett1	irene0@adventure-works.com	790-5
13	244	0	Mr.	Bronson	R.	Jacobs	*NULL*	Retail Discount Store	adventure-works\josé1	bronson0@adventure-works.com	697-5
14	29831	0	Ms.	Irene	J.	Hernandez	*NULL*	Operational Manufacturing	adventure-works\garrett1	irene0@adventure-works.com	790-5
15	29870	0	Mr.	Bronson	R.	Jacobs	*NULL*	Retail Discount Store	adventure-works\josé1	bronson0@adventure-works.com	697-5
16	29978	0	Mr.	Ajay	*NULL*	Manchepalli	*NULL*	Shipping Specialists	adventure-works\jae0	ajay0@adventure-works.com	1 (11
17	659	0	Mr.	Ajay	*NULL*	Manchepalli	*NULL*	Shipping Specialists	adventure-works\jae0	ajay0@adventure-works.com	1 (11
18	630	0	Mr.	Andrew	R.	Hill	*NULL*	Eastside Parts Shop	adventure-works\josé1	andrew4@adventure-works.com	194-5
19	672	0	Mr.	Gerald	M.	Drury	*NULL*	Utilitarian Sporting Goods	adventure-works\shu0	gerald0@adventure-works.com	169-5

10.

```
delete SalesLT.Customer
where CustomerID=IDENT_CURRENT('SalesLT.Customer');

delete SalesLT.Address
where AddressID=IDENT_CURRENT('SalesLT.Address');
```

```
Started executing query at Line 1
(1 row affected)
(1 row affected)
Total execution time: 00:00:00.036
```

INSERT DATA AS USING A TRANSACTION

1.

```
begin transaction;

  insert into SalesLT.Customer (NameStyle, FirstName, LastName, EmailAddress,
PasswordHash, PasswordSalt,    rowguid, ModifiedDate)
  values(0,  'Norman','Newcustomer','norman0@adventure-
works.com','U1/CrPqSzwLTtwgBehfpIl7f1LHSFpZw1qnG1sMzFjo=','QhHP+y8=', NEWID(),
GETDATE());

  insert into SalesLT.Address (AddressLine1, City, StateProvince, CountryRegion,
PostalCode, rowguid,    ModifiedDate)
  values('6388 Lake City Way', 'Burnaby','British Columbia','Canada','V5A 3A6',
NEWID(), GETDATE());

  insert into SalesLT.CustomerAddress (CustomerID, AddressID, AddressType,
rowguid, ModifiedDate)
  values(IDENT_CURRENT('SalesLT.Customer'), IDENT_CURRENT('SalesLT.Address'),
'Home', NEWID(), '12-1-20212');

commit transaction;
```

```
Started executing query at Line 1
(1 row affected)
(1 row affected)
Msg 241, Level 16, State 1, Line 9
Conversion failed when converting date and/or time from character string.
Total execution time: 00:00:00.011
```

HANDLE ERRORS AND EXPLICITLY ROLLBACK TRANSACTIONS

1.

```
begin transaction;

  insert into SalesLT.Customer (NameStyle, FirstName, LastName, EmailAddress,
PasswordHash, PasswordSalt,    rowguid, ModifiedDate)
  values(0,  'Norman','Newcustomer','norman0@adventure-
works.com','U1/CrPqSzwLTtwgBehfpIl7f1LHSFpZw1qnG1sMzFjo=','QhHP+y8=', NEWID(),
GETDATE());

  insert into SalesLT.Address (AddressLine1, City, StateProvince, CountryRegion,
PostalCode, rowguid,    ModifiedDate)
  values('6388 Lake City Way', 'Burnaby','British Columbia','Canada','V5A 3A6',
NEWID(), GETDATE());

  insert into SalesLT.CustomerAddress (CustomerID, AddressID, AddressType,
rowguid, ModifiedDate)
  values(IDENT_CURRENT('SalesLT.Customer'), IDENT_CURRENT('SalesLT.Address'),
'Home', '16765338-dbe4-4421-b5e9-3836b9278e63', GETDATE());

commit transaction;
```

```
Started executing query at Line 1
(1 row affected)
(1 row affected)
Msg 2627, Level 14, State 1, Line 9
Violation of UNIQUE KEY constraint 'AK_CustomerAddress_rowguid'. Cannot insert duplicate key in object 'SalesLT.CustomerAddress'. The duplicate key value is (167653
38-dbe4-4421-b5e9-3836b9278e63).
The statement has been terminated.
Total execution time: 00:00:00.083
```

5.

```
begin try
begin transaction;

  insert into SalesLT.Customer (NameStyle, FirstName, LastName, EmailAddress,
PasswordHash, PasswordSalt,    rowguid, ModifiedDate)
  values(0,  'Norman','Newcustomer','norman0@adventure-
works.com','U1/CrPqSzwLTtwgBehfpIl7f1LHSFpZw1qnG1sMzFjo=','QhHP+y8=',NEWID(),
GETDATE());

  insert into SalesLT.Address (AddressLine1, City, StateProvince, CountryRegion,
PostalCode, rowguid,    ModifiedDate)
  values('6388 Lake City Way', 'Burnaby','British Columbia','Canada','V5A
3A6',NEWID(), GETDATE());

  insert into SalesLT.CustomerAddress (CustomerID, AddressID, AddressType,
rowguid, ModifiedDate)
  values(IDENT_CURRENT('SalesLT.Customer'), IDENT_CURRENT('SalesLT.Address'),
'Home', '16765338-dbe4-4421-b5e9-3836b9278e63', GETDATE());

commit transaction;
print 'Transaction committed.';

end try
begin catch
  rollback transaction;
  print 'Transaction rolled back.';
end catch;
```

```
Started executing query at Line 1
(1 row affected)
(1 row affected)
(0 rows affected)
Transaction rolled back.
Total execution time: 00:00:00.009
```

CHECK THE TRANSACTION STATE BEFORE ROLLING BACK

1.

```
begin try
begin transaction;
```

```
  insert into SalesLT.Customer (NameStyle, FirstName, LastName, EmailAddress,
PasswordHash, PasswordSalt, rowguid, ModifiedDate)
  values(0, 'Norman','Newcustomer','norman0@adventure-
works.com','U1/CrPqSzwLTtwgBehfpIl7f1LHSFpZw1qnG1sMzFjo=','QhHP+y8=',NEWID(),
GETDATE());

  insert into SalesLT.Address (AddressLine1, City, StateProvince, CountryRegion,
PostalCode, rowguid,  ModifiedDate)
  values('6388 Lake City Way', 'Burnaby','British Columbia','Canada','V5A
3A6',NEWID(), GETDATE());

  insert into SalesLT.CustomerAddress (CustomerID, AddressID, AddressType,
rowguid, ModifiedDate)
  values(IDENT_CURRENT('SalesLT.Customer'), IDENT_CURRENT('SalesLT.Address'),
'Home', '16765338-dbe4-4421-b5e9-3836b9278e63', GETDATE());

commit transaction;
print 'Transaction committed.';

end try
begin catch
  print 'An error occurred.'
  if (XACT_STATE()) <> 0
  begin
    print 'Transaction in process.'
    rollback transaction;
    print 'Transaction rolled back.';
  end;
end catch
```

```
Started executing query at Line 1
(1 row affected)
(1 row affected)
(0 rows affected)
An error occurred.
Transaction in process.
Transaction rolled back.
Total execution time: 00:00:00.013
```

3.

```
begin try
begin transaction;

  insert into SalesLT.Customer (NameStyle, FirstName, LastName, EmailAddress,
PasswordHash, PasswordSalt, rowguid, ModifiedDate)
  values(0, 'Norman','Newcustomer','norman0@adventure-
works.com','U1/CrPqSzwLTtwgBehfpIl7f1LHSFpZw1qnG1sMzFjo=','QhHP+y8=',NEWID(),
GETDATE());

  insert into SalesLT.Address (AddressLine1, City, StateProvince, CountryRegion,
```

```
PostalCode, rowguid,  ModifiedDate)
  values('6388 Lake City Way', 'Burnaby','British Columbia','Canada','V5A
3A6',NEWID(), GETDATE());

  insert into SalesLT.CustomerAddress (CustomerID, AddressID, AddressType,
ModifiedDate)
  values(IDENT_CURRENT('SalesLT.Customer'), IDENT_CURRENT('SalesLT.Address'),
'Home', GETDATE());

commit transaction;
print 'Transaction committed.';
end try
begin catch
  print 'An error occurred.'
  if (XACT_STATE()) <> 0
  begin
    print 'Transaction in process.'
    rollback transaction;
    print 'Transaction rolled back.';
  end;
end catch
```

```
Started executing query at Line 1
(1 row affected)
(1 row affected)
(1 row affected)
Transaction committed.
Total execution time: 00:00:00.010
```

6.

```
begin try
begin transaction;

  insert into SalesLT.Customer (NameStyle, FirstName, LastName, EmailAddress,
PasswordHash, PasswordSalt, rowguid, ModifiedDate)
  values(0, 'Ann','Othercustomr','ann0@adventure-
works.com','U1/CrPqSzwLTtwgBehfpIl7f1LHSFpZw1qnG1sMzFjo=','QhHP+y8=',NEWID(),
GETDATE());;

  insert into SalesLT.Address (AddressLine1, City, StateProvince, CountryRegion,
PostalCode, rowguid,  ModifiedDate)
  values('6388 Lake City Way', 'Burnaby','British Columbia','Canada','V5A
3A6',NEWID(), GETDATE());

  insert into SalesLT.CustomerAddress (CustomerID, AddressID, AddressType,
ModifiedDate)
  values(IDENT_CURRENT('SalesLT.Customer'), IDENT_CURRENT('SalesLT.Address'),
'Home', GETDATE());

commit transaction;
```

```
print 'Transaction committed.';

throw 51000, 'Some kind of error', 1;

end try
begin catch
  print 'An error occurred.'
  if (XACT_STATE()) <> 0
  begin
    print 'Transaction in process.'
    rollback transaction;
    print 'Transaction rolled back.';
  end;
end catch
```

```
Started executing query at Line 1
(1 row affected)
(1 row affected)
(1 row affected)
Transaction committed.
An error occurred.
Total execution time: 00:00:00.661
```

EXPLORE TRANSACTION CONCURRENCY

4.

```
begin transaction;

  insert into SalesLT.Customer (NameStyle, FirstName, LastName, EmailAddress,
PasswordHash, PasswordSalt,    rowguid, ModifiedDate)
  values(0,  'Yeta','Nothercustomer','yeta0@adventure-
works.com','U1/CrPqSzwLTtwgBehfpIl7f1LHSFpZw1qnG1sMzFjo=','QhHP+y8=', NEWID(),
GETDATE());

  insert into SalesLT.Address (AddressLine1, City, StateProvince, CountryRegion,
PostalCode, rowguid,    ModifiedDate)
  values('2067 Park Lane', 'Redmond','Washington','United States','98007',
NEWID(), GETDATE());

  insert into SalesLT.CustomerAddress (CustomerID, AddressID, AddressType,
rowguid, ModifiedDate)
  values(IDENT_CURRENT('SalesLT.Customer'), IDENT_CURRENT('SalesLT.Address'),
'Home', NEWID(), GETDATE());

commit transaction;
```

```
Started executing query at Line 1
(1 row affected)
(1 row affected)
(1 row affected)
Total execution time: 00:00:01.844
```

5.

```
select count(*) from SalesLT.Customer
```

	(No column name)
1	851

CHANGE HOW CONCURRENCY IS HANDLED ON A DATABASE

1.

```
alter database AdventureWorks set read_committed_snapshot on with rollback
immediate
go
```

```
Started executing query at Line 1
Nonqualified transactions are being rolled back. Estimated rollback completion: 0%.
Nonqualified transactions are being rolled back. Estimated rollback completion: 100%.
Total execution time: 00:00:03.266
```

COMPROBACIÓN DE CONOCIMIENTO

1. ¿Qué ocurre con las transacciones anidadas cuando se revierte la transacción externa?

- ○ Se ejecuta la transacción interna.
- ○ La transacción interna evita que se revierta la transacción externa. Se ejecutarán la transacción interna y la externa.
- ◉ La transacción interna también se revierte.

✔ Correcto. Todas las transacciones internas se revierten cuando se revierte la transacción externa.

2. ¿Cuáles de las palabras clave de T-SQL siguientes se usan para controlar las transacciones?

- ◉ BEGIN TRANSACTION

✔ Correcto. Las palabras clave BEGIN TRANSACTION se usan para iniciar una transacción explícita.

- ○ BEGIN TRY
- ○ BREAK

3. ¿Qué prueba XACT_STATE?

- ○ Si hay transacciones anidadas.
- ◉ Estado de la solicitud actual.

✔ Correcto. XACT_STATE comprueba el estado de la solicitud actual.

- ○ Devuelve el número de error que provocó un error en la transacción.

4. ¿Cuál es el nivel de aislamiento de transacción predeterminado para Azure SQL Database?

- ◉ READ_COMMITTED_SNAPSHOT_ON

✔ Correcto. READ_COMMITTED_SNAPSHOT_ON es el nivel de aislamiento de transacción predeterminado para Azure SQL Database.

- ○ READ_COMMITTED_SNAPSHOT_OFF
- ○ XACT_ABORT

RESUMEN

Ahora que completó este módulo, ya sabe qué es una transacción y cómo se compara con los lotes de SQL. Aprendió a construir una transacción y dónde colocar las palabras clave BEGIN TRANSACTION y COMMIT o ROLLBACK TRANSACTION al usar el control estructurado de errores. Ahora que ha completado este módulo, puede hacer lo siguiente:

- Describir transacciones
- Comparar transacciones y lotes
- Crear y administrar transacciones
- Controlar errores en las transacciones
- Describir la simultaneidad