TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN UTN-FRC

TUP

UTN

Facultad Regional Córdoba

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN

DISEÑO Y ADMINISTRACIÓN DE BASE DE DATOS

Unidad Temática IV: Transacciones y bloqueos

Material Teórico

2do Año – 4to Cuatrimestre

V.0.1

Índice

Transacciones y bloqueos

Una transacción es una unidad única de trabajo. Si una transacción tiene éxito, todas las modificaciones de los datos realizadas durante la transacción se confirman y se convierten en una parte permanente de la base de datos. Si una transacción encuentra errores y debe cancelarse o revertirse, se borran todas las modificaciones de los datos.

SQL Server funciona en los siguientes tres modos de transacción.

Transacciones de confirmación automática

Cada instrucción individual es una transacción.

Transacciones explícitas

Cada transacción se inicia explícitamente con la instrucción BEGIN TRANSACTION y se termina explícitamente con una instrucción COMMIT o ROLLBACK.

Transacciones implícitas

Se inicia implícitamente una nueva transacción cuando se ha completado la anterior, pero cada transacción se completa explícitamente con una instrucción COMMIT o ROLLBACK.

Transacciones de ámbito de lote

Una transacción implícita o explícita de Transact-SQL que se inicia en una sesión de MARS (conjuntos de resultados activos múltiples), que sólo es aplicable a MARS, se convierte en una transacción de ámbito de lote. Si no se confirma o revierte una transacción de ámbito de lote cuando se completa el lote, SQL Server la revierte automáticamente.

Resumiendo, Una transacción es una secuencia de operaciones realizadas como una sola unidad lógica de trabajo. Una unidad lógica de trabajo debe exhibir cuatro propiedades, conocidas como propiedades de atomicidad, coherencia, aislamiento y durabilidad (ACID), para ser calificada como transacción.

Atomicidad

Una transacción debe ser una unidad atómica de trabajo, tanto si se realizan todas sus modificaciones en los datos, como si no se realiza ninguna de ellas.

Coherencia

Cuando finaliza, una transacción debe dejar todos los datos en un estado coherente. En una base de datos relacional, se deben aplicar todas las reglas a las modificaciones de la transacción para mantener la integridad de todos los datos. Todas las estructuras internas de datos, como índices de árbol b o listas doblemente vinculadas, deben estar correctas al final de la transacción.

Aislamiento

Las modificaciones realizadas por transacciones simultáneas se deben aislar de las modificaciones llevadas a cabo por otras transacciones simultáneas. Una

transacción reconoce los datos en el estado en que estaban antes de que otra transacción simultánea los modificara o después de que la segunda transacción haya concluido, pero no reconoce un estado intermedio. Esto se conoce como seriabilidad, ya que deriva en la capacidad de volver a cargar los datos iniciales y reproducir una serie de transacciones para finalizar con los datos en el mismo estado en que estaban después de realizar las transacciones originales.

Durabilidad

Una vez concluida una transacción, sus efectos son permanentes en el sistema. Las modificaciones persisten aún en el caso de producirse un error del sistema.

Los programadores de SQL son los responsables de iniciar y finalizar las transacciones en puntos que exijan la coherencia lógica de los datos. El programador debe definir la secuencia de modificaciones de datos que los dejan en un estado coherente en relación con las reglas corporativas de la organización. El programador incluye estas instrucciones de modificación en una sola transacción de forma que SQL Server Database Engine (Motor de base de datos de SQL Server) puede exigir la integridad física de la misma.

Es responsabilidad de un sistema de base de datos corporativo, como una instancia de Database Engine (Motor de base de datos), proporcionar los mecanismos que aseguren la integridad física de cada transacción. Database Engine (Motor de base de datos) proporciona:

- Servicios de bloqueo que preservan el aislamiento de la transacción.
- Servicios de registro que aseguran la durabilidad de la transacción. Aunque se produzca un error en el hardware del servidor, el sistema operativo o la instancia de Database Engine (Motor de base de datos), la instancia utiliza registros de transacciones, al reiniciar, para revertir automáticamente las transacciones incompletas al punto en que se produjo el error del sistema.
- Características de administración de transacciones que exigen la atomicidad y coherencia de la transacción. Una vez iniciada una transacción, debe concluirse correctamente; en caso contrario, la instancia de Database Engine (Motor de base de datos) deshará todas las modificaciones de datos realizadas desde que se inició la transacción.

La falta de administración de las transacciones a menudo produce problemas de prioridades y de rendimiento en sistemas con muchos usuarios. A medida que aumenta el número de usuarios de un sistema, adquiere importancia el que las aplicaciones utilicen las transacciones eficazmente. SQL Server Database Engine (Motor de base de datos de SQL Server) también admite transacciones anidadas, puntos de almacenamiento de transacciones y transacciones enlazadas, que ofrecen a los programadores opciones adicionales para escribir transacciones eficaces.

Transacciones implícitas

Este tipo de transacciones, se conocen como transacciones "de Confirmación automática" y es el comportamiento predeterminado de SQL Server, donde ejecuta (o hace efectivo los cambios en los ficheros de datos) por separado cada sentencia Transact-SQL justo después de que se termine dicha sentencia.

Por ejemplo, pensemos que tenemos dos INSERT, el primero de los cuales nos da error, y el segundo se ejecuta de forma exitosa, si trabajamos con Transacciones Implícitas, veremos como el segundo cambio se mantiene porque cada INSERT se incluye automáticamente en su propia transacción.

Podemos encontrar escenarios donde la confirmación automática de las transacciones (Transacciones Implícitas) puede ser peligroso, por ejemplo, si borramos accidentalmente todas las filas (o un grupo de filas) de una tabla, no tendremos opción de deshacer la transacción.

Las Transacciones implícitas, crean una nueva transacción, cuando en una sesión de SQL Server se ejecuta algún ALTER TABLE, FETCH, REVOKE, CREATE,
GRANT, SELECT, DELETE, INSERT, TRUNCATE TABLE, DROP, OPEN, y UPDATE.

Una transacción no implícita, una vez creada, permanece abierta y no finaliza hasta que no se produce un ROLLBACK o se invoca al COMMIT. La Transacciones Implícitas finalizan cuando se inicia una nueva transacción en la misma sesión.

Para activar el modo de trabajo con las transacciones implícitas se ejecuta:

SET IMPLICIT_TRANSACTIONS ON

Para desactivar las transacciones implícitas:

SET IMPLICIT_TRANSACTIONS OFF

Trabajar con transacciones implícitas, puede ser problemático en un entorno de producción, ya que los desarrolladores o los usuarios finales se puede olvidar de cerrar las transacciones, y esto puede ser origen de bloqueos.

Transacciones explícitas

Una transacción explícita es aquella en la que se definen explícitamente el inicio y el final de la transacción. Se pueden especificar mediante instrucciones SQL o funciones API de la base de datos.

Con SQL Server Management Studio se pueden utilizar las siguientes instrucciones SQL para definir transacciones explícitas:

- **BEGIN TRANSACTION**

  Marca el punto de inicio de una transacción explícita para una conexión.

- **COMMIT TRANSACTION**
  Finaliza correctamente una transacción si no se han encontrado errores. Todos los datos modificados por la transacción se convierten en parte permanente de la base de datos. Se liberan los recursos ocupados por la transacción.
- **ROLLBACK TRANSACTION**
  Borra una transacción en la que se han encontrado errores. Todos los datos modificados por la transacción vuelven al estado en el que estaban al inicio de la transacción. Se liberan los recursos ocupados por la transacción.

También puede utilizar transacciones explícitas en ADO .NET y OLE DB.

En ADO .NET, use el método BeginTransaction en un objeto SqlCeConnection para iniciar una transacción explícita. Para finalizar la transacción, llame al método Commit o Rollback del objeto SqlCeTransaction.

En OLE DB, llame al método ITransactionLocal::StartTransaction para iniciar una transacción. Llame al método ITransaction::Commit o ITransaction::Abort con fRetaining establecido en FALSE para finalizar la transacción sin iniciar otra de forma automática.

Transacciones locales

Marca el punto de inicio de una transacción local explícita. La instrucción BEGIN TRANSACTION incrementa @@TRANCOUNT en 1.

```
BEGIN { TRAN | TRANSACTION }
   [ { transaction_name | @tran_name_variable }
     [ WITH MARK [ 'description' ] ]
   ]
[ ; ]
```

Argumentos

transaction_name

Es el nombre asignado a la transacción. El parámetro transaction_name debe cumplir las reglas de los identificadores, pero no se admiten identificadores de más de 32 caracteres. Utilice nombres de transacciones solamente en la pareja más externa de instrucciones BEGIN…COMMIT o BEGIN…ROLLBACK anidadas.

@tran_name_variable

Se trata del nombre de una variable definida por el usuario que contiene un nombre de transacción válido. La variable debe declararse con un tipo de datos char, varchar, nchar o nvarchar. Si se pasan más de 32 caracteres a la variable, sólo se utilizarán los primeros 32; el resto de caracteres se truncará.

WITH MARK ['description']

Especifica que la transacción está marcada en el registro. El parámetro description es una cadena que describe la marca. Si description es una cadena Unicode, los valores de más de 255 caracteres se truncan a 255 antes de almacenarse en la tabla msdb.dbo.logmarkhistory. Si description no es una cadena Unicode, los valores de más de 510 caracteres se truncan a 510 caracteres.

Si utiliza WITH MARK, debe especificar un nombre de transacción. WITH MARK permite restaurar un registro de transacciones hasta una marca con nombre.

Notas

BEGIN TRANSACTION representa un punto en el que los datos a los que hace referencia una conexión son lógica y físicamente coherentes. Si se producen errores, se pueden revertir todas las modificaciones realizadas en los datos después de BEGIN TRANSACTION para devolver los datos al estado conocido de coherencia. Cada transacción dura hasta que se completa sin errores y se emite COMMIT TRANSACTION para hacer que las modificaciones sean una parte permanente de la base de datos, o hasta que se produzcan errores y se borren todas las modificaciones con la instrucción ROLLBACK TRANSACTION.

BEGIN TRANSACTION inicia una transacción local para la conexión que emite la instrucción. Según la configuración del nivel de aislamiento de la transacción actual, la transacción bloquea muchos recursos adquiridos para aceptar las instrucciones Transact-SQL emitidas por la conexión hasta que la misma finaliza con una instrucción COMMIT TRANSACTION o ROLLBACK TRANSACTION. Las transacciones que quedan pendientes durante mucho tiempo pueden impedir que otros usuarios tengan acceso a estos recursos bloqueados y pueden impedir también el truncamiento del registro.

Aunque BEGIN TRANSACTION inicia una transacción local, ésta no se guardará en el registro de transacciones hasta que la aplicación realice posteriormente una acción que se deba almacenar en el registro, como la ejecución de una instrucción INSERT, UPDATE o DELETE. Una aplicación puede realizar acciones tales como adquirir bloqueos para proteger el nivel de aislamiento de transacciones de instrucciones SELECT, pero no se guarda ningún dato en el registro hasta que la aplicación realiza una acción de modificación.

Asignar un nombre a varias transacciones en un conjunto de transacciones anidadas afecta mínimamente a la transacción. Solamente el nombre de la primera transacción (la más externa) se registra en el sistema. Revertir a otro nombre (que no sea un nombre de punto de almacenamiento válido) genera un error. De hecho, no se revierte ninguna de las instrucciones ejecutadas antes de la operación de revertir en el momento en que se produce este error. Sólo se revierten las instrucciones cuando se revierte la transacción externa.

La transacción local iniciada por la instrucción BEGIN TRANSACTION aumenta al nivel de transacción distribuida si se realizan las siguientes acciones antes de confirmarla o revertirla:

- Se ejecuta una instrucción INSERT, DELETE o UPDATE que hace referencia a una tabla remota de un servidor vinculado. La instrucción INSERT, UPDATE o DELETE causa un error si el proveedor OLE DB utilizado para obtener acceso al servidor vinculado no es compatible con la interfaz **ITransactionJoin**.
- Se realiza una llamada a un procedimiento almacenado remoto cuando la opción REMOTE_PROC_TRANSACTIONS es ON.

La copia local de SQL Server se convierte en el controlador de la transacción y utiliza el Coordinador de transacciones distribuidas de Microsoft (MS DTC) para administrar la transacción distribuida.

Una transacción se puede ejecutar explícitamente como una transacción distribuida utilizando BEGIN DISTRIBUTED TRANSACTION.

Transacciones distribuidas

Las transacciones distribuidas abarcan dos o más servidores conocidos como administradores de recursos. La administración de la transacción debe ser coordinada entre los administradores de recursos mediante un componente de servidor llamado administrador de transacciones. Cada instancia de SQL Server Database Engine (Motor de base de datos de SQL Server) puede funcionar como administrador de recursos en las transacciones distribuidas que coordinan los administradores de transacciones, como el Coordinador de transacciones distribuidas de Microsoft (MS DTC) u otros administradores que admitan la especificación X/Open XA del procesamiento de transacciones distribuidas. Para obtener más información, consulte la documentación de MS DTC.

Una transacción de una sola instancia de Database Engine (Motor de base de datos) que abarque dos o más bases de datos es, de hecho, una transacción distribuida. La instancia administra la transacción distribuida internamente; para el usuario funciona como una transacción local.

En la aplicación, una transacción distribuida se administra de forma muy parecida a una transacción local. Al final de la transacción, la aplicación pide que se confirme o se revierta la transacción. El administrador de transacciones debe administrar una confirmación distribuida de forma diferente para reducir al mínimo el riesgo de que, si se produce un error en la red, algunos administradores de recursos realicen confirmaciones mientras los demás revierten la transacción. Esto se consigue mediante la administración del proceso de confirmación en dos fases (la fase de preparación y la fase de confirmación), que se conoce como confirmación en dos fases (2PC).

Fase de preparación

Cuando el administrador de transacciones recibe una solicitud de confirmación, envía un comando de preparación a todos los administradores de recursos implicados en la transacción. Cada administrador de recursos hace lo necesario para que la transacción sea duradera y todos los búferes que contienen imágenes del registro de la transacción se pasan a disco. A medida que cada administrador de recursos completa la fase de preparación, notifica si la preparación ha tenido éxito o no al administrador de transacciones.

Fase de confirmación

Si el administrador de transacciones recibe la notificación de que todas las preparaciones son correctas por parte de todos los administradores de recursos, envía comandos de confirmación a cada administrador de recursos. A continuación, los administradores de recursos pueden completar la confirmación. Si todos los administradores de recursos indican que la confirmación ha sido correcta, el administrador de transacciones envía una notificación de éxito a la aplicación. Si algún administrador de recursos informó de un error al realizar la preparación, el administrador de transacciones envía un comando para revertir la transacción a cada administrador de recursos e indica a la aplicación que se ha producido un error de confirmación.

Las aplicaciones de Database Engine (Motor de base de datos) pueden administrar transacciones distribuidas a través de Transact-SQL o de la API de base de datos.

Transacciones anidadas

Las transacciones explícitas se pueden anidar. El objetivo principal de esto es aceptar transacciones en procedimientos almacenados a los que se puede llamar desde un proceso que ya esté en una transacción o desde procesos que no tengan transacciones activas.

En el ejemplo siguiente se muestra el uso para el que están diseñadas las transacciones anidadas. El procedimiento TransProc exige su transacción, sin tener en cuenta el modo de transacción del proceso que lo ejecute. Si se llama a TransProc cuando una transacción está activa, la transacción anidada de TransProc se pasará por alto y se confirmarán o revertirán sus instrucciones INSERT basándose en la acción final adoptada para la transacción externa. Si un proceso que no tiene ninguna transacción pendiente ejecuta TransProc, la instrucción COMMIT TRANSACTION al final del procedimiento confirmará de manera efectiva las instrucciones INSERT.

```
SET QUOTED_IDENTIFIER OFF;
GO
```

```
SET NOCOUNT OFF;
GO
USE AdventureWorks;
GO
CREATE TABLE TestTrans(Cola INT PRIMARY KEY,
            Colb CHAR(3) NOT NULL);
GO
CREATE PROCEDURE TransProc @PriKey INT, @CharCol CHAR(3) AS
BEGIN TRANSACTION InProc
INSERT INTO TestTrans VALUES (@PriKey, @CharCol)
INSERT INTO TestTrans VALUES (@PriKey + 1, @CharCol)
COMMIT TRANSACTION InProc;
GO
/* Start a transaction and execute TransProc. */
BEGIN TRANSACTION OutOfProc;
GO
EXEC TransProc 1, 'aaa';
GO
/* Roll back the outer transaction, this will
   roll back TransProc's nested transaction. */
ROLLBACK TRANSACTION OutOfProc;
GO
EXECUTE TransProc 3,'bbb';
GO
/* The following SELECT statement shows only rows 3 and 4 are
   still in the table. This indicates that the commit
   of the inner transaction from the first EXECUTE statement of
   TransProc was overridden by the subsequent rollback. */
SELECT * FROM TestTrans;
GO
```

SQL Server Database Engine (Motor de base de datos de SQL Server) omite la confirmación de las transacciones internas. La transacción se confirma o se revierte basándose en la acción realizada al final de la transacción más externa. Si se confirma la transacción externa, también se confirmarán las transacciones anidadas internas. Si se revierte la transacción externa, también se revertirán todas las transacciones internas, independientemente de si se confirmaron individualmente o no.

Cada llamada a COMMIT TRANSACTION o COMMIT WORK se aplica a la última instrucción BEGIN TRANSACTION ejecutada. Si las instrucciones BEGIN TRANSACTION están anidadas, la instrucción COMMIT sólo se aplica a la última

transacción anidada, que es la más interna. Aunque una instrucción COMMIT TRANSACTION transaction_name de una transacción anidada haga referencia al nombre de la transacción externa, la confirmación sólo se aplicará a la transacción más interna.

No es válido que el parámetro transaction_name de una instrucción ROLLBACK TRANSACTION haga referencia a las transacciones internas de un conjunto de transacciones anidadas con nombre. transaction_name sólo puede hacer referencia al nombre de la transacción más externa. Si se ejecuta una instrucción ROLLBACK TRANSACTION transaction_name con el nombre de la transacción externa en cualquier nivel de un conjunto de transacciones anidadas, se revertirán todas las transacciones anidadas. Si se ejecuta una instrucción ROLLBACK WORK o ROLLBACK TRANSACTION sin el parámetro transaction_name en cualquier nivel de un conjunto de transacciones anidadas, se revertirán todas las transacciones anidadas, incluida la más externa.

La función @@TRANCOUNT registra el nivel de anidamiento de la transacción actual. Cada instrucción BEGIN TRANSACTION incrementa @@TRANCOUNT en uno. Cada instrucción COMMIT TRANSACTION o COMMIT WORK reduce @@TRANCOUNT en uno. Una instrucción ROLLBACK WORK o ROLLBACK TRANSACTION que no tenga nombre de transacción revertirá todas las transacciones anidadas y establecerá @@TRANCOUNT en 0. Una instrucción ROLLBACK TRANSACTION que utilice el nombre de la transacción más externa de un conjunto de transacciones anidadas revertirá todas las transacciones anidadas y establecerá @@TRANCOUNT en 0. Cuando no esté seguro de si ya está en una transacción, use SELECT @@TRANCOUNT para determinar si es 1 o más. Si @@TRANCOUNT es 0, no está en una transacción.

Bloqueos

Para poder comprender los bloqueos en Microsoft SQL Server Compact Edition (SQL Server Compact Edition), debe estar familiarizado con los recursos que se pueden bloquear y los diferentes modos disponibles para bloquear los recursos.

La granularidad de los bloqueos se refiere al nivel en el que los bloqueos tienen lugar:

- Fila
- Tabla
- Página
- Base de datos

Los bloqueos realizados en una granularidad baja, por ejemplo en una fila, aumentan la simultaneidad, pero se deben mantener más bloqueos si se bloquea un gran número de filas. Los bloqueos realizados en una granularidad alta, por

ejemplo en una tabla, reducen la simultaneidad porque el bloqueo de toda una tabla restringe el acceso de otras transacciones a cualquier parte de la tabla. Sin embargo, en los bloqueos de tablas no es necesario mantener tantos bloqueos.

Modos de bloqueo

Los modos de bloqueo determinan el modo en que las transacciones simultáneas pueden obtener acceso a los datos. SQL Server Compact Edition determina qué modo de bloqueo se va a utilizar en función de los recursos que deben bloquearse y las operaciones que han de realizarse.
En la siguiente tabla se describen los modos de bloqueo compatibles con SQL Server Compact Edition.

Modo de bloqueo	Descripción
Compartido (S)	Protege un recurso del acceso de lectura. Ninguna otra transacción podrá modificar los datos mientras el bloqueo compartido (S) exista en el recurso.
Exclusivo (X)	Indica una modificación de datos, como pueda ser una inserción, una actualización o una eliminación. Garantiza que no pueden aplicarse varias actualizaciones simultáneamente en el mismo recurso.
Actualizar (U)	Impide que se produzca un tipo común de interbloqueo. Solamente una transacción a la vez puede obtener un bloqueo U en un recurso. Si la transacción modifica el recurso, el bloqueo U se convierte en un bloqueo X.
Esquema	Se utiliza cuando se está ejecutando una operación que depende del esquema de una tabla. Los tipos de bloqueo de esquema son la modificación de esquema (Sch-M) y la estabilidad de esquema (Sch-S).
Intención	Establece una jerarquía de bloqueos. Los tipos más comunes de bloqueos de intención son IS, IU e IX. Estos bloqueos indican que una transacción opera sobre algunos recursos inferiores de la jerarquía, pero no en todos. Los recursos de nivel inferior tendrán un bloqueo S, U o X.

Tabla 1: Elaboración propia

Comportamiento del bloqueo

SET TRANSACTION ISOLATION LEVEL controla el comportamiento del bloqueo y de las versiones de fila de las instrucciones Transact-SQL emitidas por una conexión a SQL Server.
SET TRANSACTION ISOLATION LEVEL

{ READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SNAPSHOT
| SERIALIZABLE
}
[;]

READ UNCOMMITTED

Especifica que las instrucciones pueden leer filas que han sido modificadas por otras transacciones pero todavía no se han confirmado.

Las transacciones que se ejecutan en el nivel READ UNCOMMITTED no emiten bloqueos compartidos para impedir que otras transacciones modifiquen los datos leídos por la transacción actual. Las transacciones READ UNCOMMITTED tampoco se bloquean mediante bloqueos exclusivos que impedirían que la transacción actual leyese las filas modificadas pero no confirmadas por otras transacciones. Cuando se establece esta opción, es posible leer las modificaciones no confirmadas, denominadas lecturas de datos sucios. Los valores de los datos se pueden cambiar, y las filas pueden aparecer o desaparecer en el conjunto de datos antes de que finalice la transacción. Esta opción tiene el mismo efecto que establecer NOLOCK en todas las tablas y en todas las instrucciones SELECT de una transacción. Se trata del nivel de aislamiento menos restrictivo.

En SQL Server, también se puede reducir al mínimo la contención de bloqueos y, al mismo tiempo, proteger las transacciones de las lecturas de datos sucios de modificaciones de datos no confirmadas mediante una de estas dos alternativas:

- El nivel de aislamiento READ COMMITTED con la opción de base de datos READ_COMMITTED_SNAPSHOT establecida en ON.
- El nivel de aislamiento SNAPSHOT.

READ COMMITTED

Especifica que las instrucciones no pueden leer datos que hayan sido modificados, pero no confirmados, por otras transacciones. Esto evita las lecturas de datos sucios. Otras transacciones pueden cambiar datos entre cada una de las instrucciones de la transacción actual, dando como resultado lecturas no repetibles o datos ficticios. Esta opción es la predeterminada para SQL Server.

El comportamiento de READ COMMITTED depende del valor de la opción de base de datos READ_COMMITTED_SNAPSHOT:

- Si READ_COMMITTED_SNAPSHOT se establece en OFF (valor predeterminado), el Database Engine (Motor de base de datos) utiliza bloqueos compartidos para impedir que otras transacciones modifiquen las filas mientras la transacción actual esté ejecutando una

operación de lectura. Los bloqueos compartidos impiden también que la instrucción lea las filas modificadas por otras transacciones hasta que la otra transacción haya finalizado. Los bloqueos compartidos se liberan cuando la instrucción termina.

- Si READ_COMMITTED_SNAPSHOT se establece en ON, el Database Engine (Motor de base de datos) utiliza versiones de fila para presentar a cada instrucción una instantánea coherente, desde el punto de vista transaccional, de los datos tal como se encontraban al comenzar la instrucción. No se utilizan bloqueos para impedir que otras transacciones actualicen los datos.

Cuando la opción de base de datos READ_COMMITTED_SNAPSHOT es ON, se puede utilizar la sugerencia de tabla READCOMMITTEDLOCK para solicitar el uso del bloqueo compartido en lugar de versiones de fila para las instrucciones individuales de las transacciones que se ejecutan en el nivel de aislamiento READ_COMMITTED.

REPEATABLE READ

Especifica que las instrucciones no pueden leer datos que han sido modificados pero aún no confirmados por otras transacciones y que ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que ésta finalice.

Se aplican bloqueos compartidos a todos los datos leídos por cada instrucción de la transacción, y se mantienen hasta que la transacción finaliza. De esta forma, se evita que otras transacciones modifiquen las filas que han sido leídas por la transacción actual. Otras transacciones pueden insertar filas nuevas que coincidan con las condiciones de búsqueda de las instrucciones emitidas por la transacción actual. Si la transacción actual vuelve a ejecutar la instrucción, recuperará las filas nuevas, dando como resultado lecturas ficticias. Debido a que los bloqueos compartidos se mantienen hasta el final de la transacción en lugar de liberarse al final de cada instrucción, la simultaneidad es inferior que en el nivel de aislamiento predeterminado READ COMMITTED. Utilice esta opción solamente cuando sea necesario.

SNAPSHOT

Especifica que los datos leídos por cualquier instrucción de una transacción sean la versión coherente, desde el punto de vista transaccional, de los datos existentes al comienzo de la transacción. La transacción únicamente puede reconocer las modificaciones de datos confirmadas antes del comienzo de la misma. Las instrucciones que se ejecuten en la transacción actual no verán las modificaciones de datos efectuadas por otras transacciones después del inicio de la transacción actual. El efecto es el mismo que se obtendría si las instrucciones de

una transacción obtuviesen una instantánea de los datos confirmados tal como se encontraban al comienzo de la transacción.

Las transacciones SNAPSHOT no solicitan bloqueos al leer los datos, excepto cuando se recupera una base de datos. Las transacciones SNAPSHOT que leen datos no bloquean la escritura de datos de otras transacciones. Las transacciones que escriben datos no bloquean la lectura de datos de las transacciones SNAPSHOT.

Durante la fase de reversión de la recuperación de una base de datos, las transacciones SNAPSHOT solicitan un bloqueo si se intenta leer datos bloqueados por otra transacción que está en proceso de reversión. La transacción SNAPSHOT se bloquea hasta que finalice la reversión de esa transacción. El bloqueo se libera justo después de haberse concedido.

La opción de base de datos ALLOW_SNAPSHOT_ISOLATION debe establecerse en ON para poder iniciar una transacción que utilice el nivel de aislamiento SNAPSHOT. Si una transacción que utiliza el nivel de aislamiento SNAPSHOT obtiene acceso a datos de varias bases de datos, será necesario establecer ALLOW_SNAPSHOT_ISOLATION en ON en cada una de ellas.

No es posible establecer en el nivel de aislamiento SNAPSHOT una transacción que se inició con otro nivel de aislamiento; si lo hace, la cancelará. Si una transacción comienza en el nivel de aislamiento SNAPSHOT, puede cambiarla a otro nivel de aislamiento y, después, de nuevo a SNAPSHOT. Una transacción se inicia la primera vez que obtiene acceso a los datos.

Una transacción que se ejecuta en el nivel de aislamiento SNAPSHOT puede ver los cambios realizados por esa transacción. Por ejemplo, si la transacción realiza una operación UPDATE en una tabla y después emite una instrucción SELECT para la misma tabla, los datos modificados se incluirán en el conjunto de resultados.

SERIALIZABLE

Especifica lo siguiente:

- Las instrucciones no pueden leer datos que hayan sido modificados, pero aún no confirmados, por otras transacciones.
- Ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que la transacción actual finalice.
- Otras transacciones no pueden insertar filas nuevas con valores de clave que pudieran estar incluidos en el intervalo de claves leído por las instrucciones de la transacción actual hasta que ésta finalice.

Se colocan bloqueos de intervalo en el intervalo de valores de clave que coincidan con las condiciones de búsqueda de cada instrucción ejecutada en una transacción. De esta manera, se impide que otras transacciones actualicen o inserten filas que satisfagan los requisitos de alguna de las instrucciones

ejecutadas por la transacción actual. Esto significa que, si alguna de las instrucciones de una transacción se ejecuta por segunda vez, leerá el mismo conjunto de filas. Los bloqueos de intervalo se mantienen hasta que la transacción finaliza. Éste es el nivel de aislamiento más restrictivo, porque bloquea intervalos de claves completos y mantiene esos bloqueos hasta que la transacción finaliza. Al ser menor la simultaneidad, sólo se debe utilizar esta opción cuando sea necesario. Esta opción tiene el mismo efecto que establecer HOLDLOCK en todas las tablas de todas las instrucciones SELECT de la transacción.

Notas

Sólo es posible establecer una de las opciones de nivel de aislamiento cada vez, y permanecerá activa para la conexión hasta que se cambie explícitamente. Todas las operaciones de lectura realizadas dentro de la transacción se rigen por las reglas del nivel de aislamiento especificado, a menos que se utilice una sugerencia de tabla en la cláusula FROM de una instrucción para especificar un comportamiento de bloqueo o versiones diferente para una tabla.

Los niveles de aislamiento de transacciones definen el tipo de bloqueo que se adquiere en las operaciones de lectura. Los bloqueos compartidos que se adquieren para READ COMMITTED o REPEATABLE READ suelen ser bloqueos de fila, aunque éstos se pueden escalar a bloqueos de página o tabla si la operación de lectura hace referencia a un número significativo de filas de una página o tabla. Si la transacción modifica una fila después de haberse leído, la transacción adquiere un bloqueo exclusivo para proteger esa fila, y ese bloqueo exclusivo se mantiene hasta que la transacción finaliza. Por ejemplo, si una transacción REPEATABLE READ tiene un bloqueo compartido en una fila y, después, la transacción modifica esa fila, el bloqueo compartido de fila se convierte en un bloqueo exclusivo de fila.

Con una excepción, se puede cambiar de un nivel de aislamiento a otro en cualquier momento de una transacción. La excepción se produce cuando se cambia de cualquier nivel de aislamiento al aislamiento SNAPSHOT. Esta acción generará un error en la transacción y hará que se revierta. Sin embargo, puede cambiar una transacción iniciada en aislamiento SNAPSHOT a cualquier otro nivel de aislamiento.

Cuando se cambia el nivel de aislamiento de una transacción por otro, los recursos leídos después del cambio se protegen de acuerdo con las reglas del nuevo nivel. Los recursos leídos antes del cambio siguen estando protegidos en función de las reglas del nivel anterior. Por ejemplo, si una transacción ha cambiado de READ COMMITTED a SERIALIZABLE, los bloqueos compartidos adquiridos después del cambio se mantienen hasta el final de la transacción.

Si se ejecuta SET TRANSACTION ISOLATION LEVEL en un procedimiento almacenado o un desencadenador, cuando el objeto devuelve el control, el nivel de

aislamiento se restablece en el nivel en efecto cuando se invocó el objeto. Por ejemplo, si se establece REPEATABLE READ en un lote y, después, este lote llama a un procedimiento almacenado que establece el nivel de aislamiento en SERIALIZABLE, el valor del nivel de aislamiento vuelve a REPEATABLE READ cuando el procedimiento almacenado devuelve el control al lote.

Cuando se utiliza sp_bindsession para enlazar dos sesiones, cada sesión mantiene su nivel de aislamiento. Si se utiliza SET TRANSACTION ISOLATION LEVEL para cambiar el valor del nivel de aislamiento de una sesión, no se verán afectados los valores de las sesiones enlazadas a ella.

SET TRANSACTION ISOLATION LEVEL se aplica en tiempo de ejecución, no en tiempo de análisis.

Las operaciones de carga masiva optimizadas que se realizan en montones bloquean las consultas que se ejecutan con los siguientes niveles de aislamiento:

- SNAPSHOT
- READ UNCOMMITTED
- READ COMMITTED con versiones de fila

A la inversa, las consultas que se ejecutan con estos niveles de aislamiento bloquean las operaciones de carga masiva optimizadas que se realizan en montones.

Escalamiento de bloqueo

Para poder evitar un escenario donde el bloqueo utiliza demasiados recursos, SQL Server introdujo la función de concentración de bloqueos.

SQL Server utiliza la concentración de bloqueos. Lo que esto implica es que en una situación en la que se logran más de 5.000 bloqueos en un solo nivel, SQL Server concentrará todos esos bloqueos a un único bloqueo a nivel tabla. Por defecto, SQL Server siempre concentrará directamente al nivel de la tabla, lo que implica que nunca se llegará a producir una concentración a nivel de la página. En vez de obtener numerosas filas y páginas bloqueadas, SQL Server concentrará al bloqueo exclusivo (X) a nivel de tabla.

Mientras tanto esto reducirá la necesidad de los recursos, los bloqueos exclusivos (X) en una tabla significan que ninguna otra operación podrá acceder a la tabla bloqueada y todas las consultas que traten de acceder a esa tabla serán bloqueadas. Por lo cual se reducirán los excesos de carga en el sistema, pero se aumentará la probabilidad de contención de concurrencia.

Para brindar control sobre la concentración, empezando con SQL Server, la opción LOCK_EXCALATION se introduce como parte de la instrucción ALTER TABLE.

```
USE AdventureWorks2014
GO
```

```
ALTER TABLE Table_name
SET (LOCK_ESCALATION = < TABLE | AUTO | DISABLE > –One of those options)
GO
```

Cada una de estas opciones se define para permitir un control específico sobre el proceso de escalamiento de bloqueo:

Tabla: esta es la opción por defecto para cualquier tabla recién creada, ya que de forma predeterminada SQL Server siempre llegará a ejecutar la concentración de bloqueo a nivel de la tabla, que también incluye tablas con particiones.

Auto: la opción permite concentrar el bloqueo a nivel de partición cuando una tabla ha sido particionada. Al obtener 5.000 bloqueos en una sola partición, la concentración de bloqueo obtendrá un bloqueo exclusivo (X) en esa partición mientras que la tabla obtendrá el bloqueo exclusivo intencionado (IX). Si esa tabla no está particionada, la concentración de bloqueo logrará el bloqueo a nivel de la tabla (igual a la opción Tabla).

Si bien parece una opción muy útil, se la debe utilizar con cuidado, debido a que puede causar un bloqueo de manera permanente. En caso de que se tengan dos transacciones en dos particiones donde se adquiere el bloqueo exclusivo (X) y las transacciones traten de acceder a la fecha desde la partición utilizada por otra transacción, se encontrará un bloqueo permanente.

Deshabilitar: la opción deshabilitará por completo la concentración del bloqueo de una tabla. De nuevo, la opción debe ser utilizada cautelosamente para poder así evitar que el administrador de bloqueo de SQL Server esté forzado a usar una en exceso la memoria.

Como se puede comprobar, la concentración de bloqueo puede llegar a ser un gran desafío para los administradores de bases de datos. Si el diseño de una aplicación requiere de la eliminación o actualización de más de 5.000 filas a la vez, una solución para evitar la concentración del bloqueo y los efectos resultantes es dividir la transacción individual en dos o más transacciones donde cada una manejará menos de 5.000 filas, de esta manera es que la concentración de bloqueo podría evitarse.

Compatibilidad de bloqueos

Una vez interpretados los conceptos de bloqueos, no es posible aplicar todos los modos de bloqueo en todos los niveles.

A nivel fila, los siguientes modos de bloqueo pueden ser aplicados:

- Exclusivo (X)
- Compartido (S)
- Actualización (U)

Para llegar a comprender la compatibilidad de esos modos, consulte a la siguiente tabla:

	Exclusivos (X)	Compartido (S)	Actualizados (U)
Exclusivos (X)	✗	✗	✗
Compartidos (S)	✗	✓	✓
Actualizados (U)	✗	✓	✗

✗ – Compatible ✓ – Incompatible

- Exclusivo (X)
- Compartido (S)
- Intención exclusiva (IX)
- Intención compartida (IS)
- Compartido con intención exclusiva (SIX)

La compatibilidad de estos modos se ven en la tabla a continuación

	(X)	(S)	(IX)	(IS)	(SIX)
(X)	✗	✗	✗	✗	✗
(S)	✗	✓	✗	✓	✗
(IX)	✗	✗	✓	✓	✗
(IS)	✗	✓	✓	✓	✓
(SIX)	✗	✗	✗	✓	✗

✓ – Compatible ✗ – Incompatible

Un bloqueo de esquema (Sch) también llega a ser un bloqueo a nivel tabla, pero no resulta ser un bloqueo relacionado con datos

Interbloqueos (DeadLock)

Un interbloqueo se produce cuando dos o más tareas se bloquean entre sí permanentemente teniendo cada tarea un bloqueo en un recurso que las otras tareas intentan bloquear. Por ejemplo:

- La transacción A tiene un bloqueo compartido de la fila 1.
- La transacción B tiene un bloqueo compartido de la fila 2.
- La transacción A ahora solicita un bloqueo exclusivo de la fila 2 y se bloquea hasta que la transacción B finalice y libere el bloqueo compartido que tiene de la fila 2.
- La transacción B ahora solicita un bloqueo exclusivo de la fila 1 y se bloquea hasta que la transacción A finalice y libere el bloqueo compartido que tiene de la fila 1.

La transacción A no puede completarse hasta que se complete la transacción B, pero la transacción B está bloqueada por la transacción A. Esta condición también se llama dependencia cíclica: la transacción A tiene una dependencia de la transacción B y la transacción B cierra el círculo teniendo una dependencia de la transacción A.

Ambas transacciones con un interbloqueo esperarán para siempre, a no ser que un proceso externo rompa el interbloqueo. La supervisión de interbloqueos del SQL Server Database Engine (Motor de base de datos de SQL Server) de Microsoft comprueba periódicamente si hay tareas con un interbloqueo. Si el monitor detecta una dependencia cíclica, selecciona una de las tareas como el sujeto y finaliza su transacción con un error. Esto permite a la otra tarea completar su transacción. La aplicación con la transacción que terminó con un error puede reintentar la transacción, que suele completarse después de que la otra transacción interbloqueada haya finalizado.

El uso de ciertas convenciones de código en las aplicaciones reduce la probabilidad de que las aplicaciones causen interbloqueos.

A menudo se confunden los interbloqueos con los bloqueos normales. Cuando una transacción solicita un bloqueo en un recurso bloqueado por otra transacción, la transacción solicitante espera hasta que se libere el bloqueo. De forma predeterminada, las transacciones de SQL Server no tienen tiempo de espera, a menos que se establezca LOCK_TIMEOUT. La transacción solicitante está bloqueada, no interbloqueada, porque la transacción solicitante no ha hecho nada para bloquear la transacción a la que pertenece el bloqueo. Finalmente, la transacción a la que pertenece el bloqueo se completará y liberará el bloqueo, y a la transacción solicitante se le concederá el bloqueo y continuará.

A veces, los interbloqueos se denominan "abrazo mortal".

Un interbloqueo es una condición que se puede dar en cualquier sistema con varios subprocesos, no sólo en un sistema de administración de bases de datos relacionales, y puede producirse para recursos distintos a los bloqueos en objetos de base de datos. Por ejemplo, un subproceso en un sistema operativo con varios subprocesos puede adquirir uno o más recursos, como bloqueos de memoria. Si el recurso que se desea adquirir pertenece actualmente a otro subproceso, puede que

el primer subproceso deba esperar a que el otro libere el recurso de destino. En consecuencia, se dice que el subproceso que está en espera depende del subproceso que posee ese recurso concreto. En una instancia del Database Engine (Motor de base de datos), las sesiones pueden interbloquearse cuando adquieren recursos ajenos a la base de datos, como memoria o subprocesos.

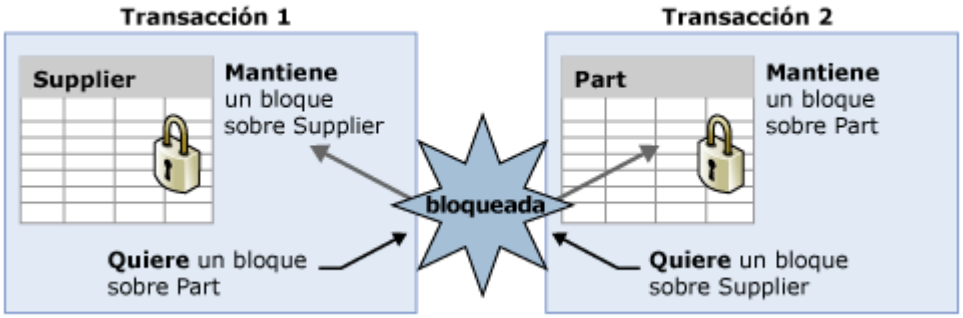

Imagen 1: Extraída de SQL

En la ilustración, la transacción T1 tiene una dependencia de la transacción T2 para el recurso de bloqueo de la tabla Part. Paralelamente, la transacción T2 tiene una dependencia de la transacción T1 para el recurso de bloqueo de la tabla Supplier. Puesto que estas dependencias forman un ciclo, hay un interbloqueo entre las transacciones T1 y T2.

Obtener información sobre bloqueos existentes

SQL Server brinda las vistas denominadas Dynamics Management View (DMV, Vistas de administración dinámica del sistema) sys.dm_tran_locks que muestran la información sobre los recursos del administrador de bloqueos que se encuentran actualmente en uso, lo que implica que mostrarán todos los bloqueos "activos" que se obtienen en las operaciones. Se pueden encontrar mayor información sobre este DMV en el artículo sys.dm_tran_locks (Transact-SQL).

La columna más importante que se utiliza para la identificación del bloqueo es **resource_type, request_mode y resource_description**. Si es necesario, se pueden incluir más columnas como recursos adicionales para obtener información, al solucionar problemas se puede incluir información extra.

Aquí se muestra un ejemplo de la consulta:

```
SELECT resource_type, request_mode, resource_description
FROM sys.dm_tran_locks
WHERE resource_type <> 'DATABASE'
```

La cláusula WHERE de esta consulta se utiliza como filtro en resource_type to eliminate.from de los resultados, los bloqueos obtenidos generalmente compartidos en la base de datos que siempre están presentes a nivel de base de datos.

	resource_type	request_mode	resource_description
1	OBJECT	IX	
2	KEY	X	(cca921fdc1dd)
3	KEY	X	(c0a381fd6cd4)
4	KEY	X	(9960aa6af927)
5	KEY	X	(1bd877bceb51)
6	KEY	X	(d69748a73252)
7	KEY	X	(e9069d930a93)
8	PAGE	IX	1:20432
9	KEY	X	(d8b6f3f4a521)
10	KEY	X	(b9b173bbe8d5)
11	KEY	X	(810420b6dcd1)
12	KEY	X	(5c09059d109d)
13	KEY	X	(23f7ff42b8f2)
14	KEY	X	(2ea3cb25ed41)
15	PAGE	IX	1:14352
16	KEY	X	(59855d342c69)
17	KEY	X	(ada3897e4a05)
18	KEY	X	(29cf3326f583)
19	PAGE	UIX	1:7865
20	PAGE	UIX	1:7893
21	PAGE	UIX	1:7895
22	PAGE	UIX	1:8108
23	PAGE	UIX	1:8107

Presentamos una breve explicación de las tres columnas a continuación:

resource_type – muestra un recurso de base de datos donde se realizan los bloqueos. La columna puede mostrar cualquiera de siguientes valores: ALLOCATION_UNIT, APPLICATION, DATABASE, EXTENT, FILE, HOBT, METADATA, OBJECT, PAGE, KEY, RID

request_mode – muestra el tipo de bloqueo que se adquiere en el recurso

resource_description – muestra una descripción breve del recurso y no se rellena para todos los modos de bloqueo. Muy frecuentemente, la columna contiene la identificación de la fila, página, objeto, archivo, etc.

Monitor de actividad

El monitor de SQL Server provee información necesaria para diagnosticar y resolver problemas de desempeño de del motor de base de datos (como por ejemplo bloqueos), al igual que optimizar SQL Server. Un rendimiento óptimo no es fácil de definir y configurar, dado que usualmente existe un intercambio entre múltiples factores de software y hardware, También depende de su ambiente, requerimientos de negocios y políticas de la compañía etc.

Mientras que reportes lentos pueden ser aceptables en una pequeña fábrica, no es en grandes empresas donde latencia, intermitencias y cuellos de botella afectan a una gran cantidad de usuarios y pueden afectar significativamente al negocio la producción y credibilidad de nuestros sistemas. Los problemas listados son usualmente inaceptables y deben ser resueltos tan pronto como sea posible.

Una vez que el rendimiento de SQL server está optimizado, tiene que ser monitoreado constantemente, dado que cada cambio en datos, esquemas y configuración usualmente llevan a una situación donde una optimización manual adicional es necesaria para seguir garantizando la calidad.

Bibliografía

Libros en pantalla SQL Server

Fundamentos de Bases de Datos - Quinta Edición, Silberschatz – Korth - Sudarshan, Mc Graw Hill.

W3Schools - SQL https://www.w3schools.com/sql/default.asp