# Transacciones y bloqueos

La definición de transacciones y el uso de los bloqueos son mecanismos que permiten garantizar la consistencia de una base de datos durante la ejecución de las operaciones con sus datos.

*Esta página se ha dejado en blanco intencionalmente.*

# *Capítulo 13*
# *Transacciones y bloqueos*

## Contenido


- *Transacciones*
  - ✓ *Definición de transacción*
  - ✓ *Transacción implícita*
  - ✓ *Transacción de autoconfirmación (autocommit)*
  - ✓ *Transaccion explícita*
  - ✓ *Las sentencias BEGIN TRANSACTION y COMMIT TRANSACTION*
  - ✓ *Cancelación de una transacción – La sentencia ROLLBACK TRANSACTION*
  - ✓ ***Ejercicio 123:** Error de sintáxis*
  - ✓ ***Ejercicio 124:** Error de tiempo de ejecución*
  - ✓ ***Ejercicio 125:** Transacción implícita*
  - ✓ ***Ejercicio 126:** Transacción explícita*
  - ✓ *El checkpoint y la recuperación de transacciones*
  - ✓ *Consideraciones al definir transacciones*
- *Bloqueos*
  - ✓ *Definición de bloqueo*
  - ✓ *Manejo de sesiones concurrentes*
    - *Problemas que pueden presentarse*
  - ✓ *Elementos que se pueden bloquear*
  - ✓ *Tipos de bloqueos*
    - *Bloqueos básicos*
    - *Bloqueos en situaciones especiales*
  - ✓ *Compatibilidad de los bloqueos*

*Esta página se ha dejado en blanco intencionalmente.*

# *Transacciones y bloqueos*

## Transacciones

Durante la ejecución de las operaciones con los datos, el programador debe definir el conjunto de operaciones a ejecutarse como una unidad lógica de proceso. Adicionalmente el motor de base de datos utiliza los bloqueos como un mecanismo para garantizar las transacciones y mantener la consistencia de la base de datos cuando muchos usuarios acceden simultáneamente a ella.

## Definición de transacción

Una **transacción** es un conjunto de modificaciones de datos (operaciones) que debe ser procesado como una unidad.

Por ejemplo, se tiene las tablas **factura**, **detalle_factura** y **producto** de una base de datos de **ventas**. La tabla **producto** registra el nivel de inventario (stock) de cada producto.

Supongamos que se tiene que registrar la venta de 10 unidades del producto ABC. Para ello, la aplicación procede de la siguiente manera:

**Operación 1** Registra en la tabla **factura**, los datos de la cabecera de la factura.

**Operación 2** Registra en la tabla **detalle_factura**, los detalles de la venta.

**Operación 3** Actualiza el nivel de inventario del producto ABC en la tabla **producto**.

### Pregunta:

¿Qué ocurriría si durante la ejecución de la aplicación se completan las operaciones 1 y 2, y por una falla del sistema, la operación 3 no se lleva a cabo?

### Respuesta:

Si no se diseña un mecanismo para corregir el error, la base de datos perdería consistencia. Según las tablas **factura** y **detalle_factura** se han vendido 10 unidades del producto ABC, por lo que en la tabla **producto** deberíamos tener 10 unidades menos. Esto último no es cierto al no haberse completado la operación 3.

En otras palabras, las operaciones 1, 2 y 3 forman una transacción, y si ésta no se completa, el sistema debe deshacer todas las operaciones.

Una transacción asegura que las operaciones se llevarán a cabo completas, o en caso contrario, la transacción será anulada para garantizar la consistencia de los datos.

Las transacciones pueden ser implícitas, de autoconfirmación, ó explícitas.

## Transacción implícita

Es aquella en la que cada una de las sentencias INSERT, UPDATE o DELETE se ejecuta como una transacción. Es decir, que si ejecutamos una de estas sentencias, al finalizar la misma, la siguiente sentencia inicia otra transacción.

El modo transacciones implícitas se configura ejecutando la instrucción SET IMPLICIT_TRANSACTIONS ON.

## Transacción de autoconfirmación (autocommit)

Se presenta cuando cada sentencia es confirmada automáticamente cuando se completa. No es necesario especificar ninguna instrucción de control de transacciones.

En este caso, las modificaciones hechas por cada una de las sentencias INSERT, UPDATE o DELETE no podrán ser deshechas o anuladas. Este es el modo predeterminado del motor de base de datos.

## Transacción explícita

Es aquella definida explícitamente por el programador utilizando la sentencia BEGIN TRANSACTION.

Consiste en un conjunto de sentencias agrupadas entre las sentencias BEGIN TRANSACTION y COMMIT TRANSACTION.

## Las sentencias BEGIN TRANSACTION y COMMIT TRANSACTION

### Sintáxis

```
BEGIN TRANSACTION [ nombre_transacción ]
   sentencias_SQL_transaccionales
COMMIT TRANSACTION [ nombre_transacción ]
```

- La sentencia BEGIN TRANSACTION define que cada una de las sentencias SQL que la siguen forman parte de una transacción.
- Todas las sentencias SQL después de BEGIN TRANSACTION deben ejecutarse para que se considere que la transacción se ha completado, salvo que se decida anular la transacción de manera explícita ejecutando ROLLBACK TRANSACTION.
- La sentencia COMMIT TRANSACTION confirma todas las modificaciones llevadas a cabo por las sentencias SQL anteriores y finaliza la transacción.

Cada transacción es registrada en el log de transacciones de la base de datos para mantener la consistencia de la base de datos y ayudar en la recuperación ante la eventualidad de una falla del sistema.

## Cancelación de una transacción – La sentencia ROLLBACK TRANSACTION

Como una parte del manejo de errores, puede incluir dentro de la transacción, sentencias de control, de manera tal que al producirse un error la transacción pueda ser deshecha utilizando la sentencia ROLLBACK TRANSACCIÓN.

### Sintáxis

```
ROLLBACK TRANSACTION [ nombre_transacción ]
```

## Ejercicio 123: Error de sintáxis

1. Digite y ejecute las siguientes instrucciones en el **Code Editor**:

```
CREATE DATABASE Pruebas
go

USE Pruebas
go

CREATE TABLE PruebaBatch(
   ColA int PRIMARY KEY,
   ColB char(3) not null )
go
```

2. Ahora digite y ejecute el siguiente batch. La instrucción GO al final del conjunto de sentencias define un batch ó lote de instrucciones a enviar y ejecutar juntas en el servidor. Observe que se ha establecido premeditadamente un error de sintáxis.

```
INSERT INTO PruebaBatch VALUES(1, 'aaa')
INSERT INTO PruebaBatch VALUES(2, 'bbb')
INSERT INTO PruebaBatch VALEUS(3, 'ccc')
-- Error de sintáxis en el último INSERT
go
```

3. Verifique el contenido de la tabla **PruebaBatch**.

```
SELECT * FROM PruebaBatch
go
```

No se ha ejecutado ninguna de las sentencias INSERT.

## Ejercicio 124: Error de tiempo de ejecución

Digite y ejecute las siguientes instrucciones en el **Code Editor**:

```
USE Pruebas
go

INSERT INTO PruebaBatch VALUES(1, 'aaa')
INSERT INTO PruebaBatch VALUES(2, 'bbb')
INSERT INTO PruebaBathc VALUES(3, 'ccc')
-- Error en nombre de tabla
go

SELECT * FROM PruebaBatch
go
```

Observe que el tercer INSERT tiene un error en el nombre de archivo. Pese a ello, los dos primeros INSERT si se ejecutaron sin problemas. El motor de base de datos utiliza resolución de nombres diferida; es decir, que no verifica los nombres hasta el momento que va a ejecutar la instrucción.

Adicionalmente, el motor de base de datos se encuentra en el modo transacciones de confirmación automática (autocommit).

## Ejercicio 125: Transacción implícita

1. Digite y ejecute las siguientes instrucciones en el **Code Editor**:

```
USE Pruebas
go

CREATE TABLE PruebaBatch2(
   ColA int PRIMARY KEY,
   ColB char(3) not null )
go
```

2. Ahora establezca el modo transacciones implícitas.

```
SET IMPLICIT_TRANSACTIONS ON
go
```

3. Ahora, inserte las siguientes filas:

```
INSERT INTO PruebaBatch2 VALUES(1, 'aaa')
INSERT INTO PruebaBatch2 VALUES(2, 'bbb')
INSERT INTO PruebaBatch2 VALUES(3, 'ccc')
go

SELECT * FROM Pruebabatch2
go
```

   Observe que las filas se han insertado en la tabla.

4. Ahora vamos a cancelar la transacción.

```
ROLLBACK TRANSACTION
go

SELECT * FROM PruebaBatch2
go

COMMIT TRANSACTION
go

SET IMPLICIT_TRANSACTIONS OFF
go
```

   Note que las filas ya no se encuentran en la tabla. Cuando el modo es transacciones implícitas, una transacción finaliza cuando se la confirma (COMMIT) ó cuando se la cancela (ROLLBACK).

   El último COMMIT TRANSACTION es necesario, ya que la instrucción SELECT ejecutada después del ROLLBACK TRANSACTION inicia una segunda tranascción.

## Ejercicio 126: Transacción explícita

1. Digite y ejecute las siguientes instrucciones en el **Code Editor**:

```
USE Pruebas
go

CREATE TABLE Padre(
   codPadre int PRIMARY KEY,
   nomPadre varchar(15) not null )

CREATE TABLE Hijo(
   codHijo int PRIMARY KEY,
   nomHijo varchar(15) not null,
   codPadre int not null FOREIGN KEY
   REFERENCES Padre )
go
```

2. Defina y ejecute la siguiente transacción explícita. En ella se ha establecido intencionalmente un error de integridad referencial.

```
BEGIN TRANSACTION
   INSERT INTO Padre VALUES(100, 'Juan')
      IF (@@error <> 0) GOTO hayError
   INSERT INTO Hijo VALUES(101, 'Pedro', 99)
   -- padre 99 no existe
      IF (@@error <> 0) GOTO hayError
COMMIT TRANSACTION
RETURN
hayError:
ROLLBACK TRANSACTION
go
```

3. Ahora consulte las tablas. Note que ninguno de los INSERT se ha confirmado.

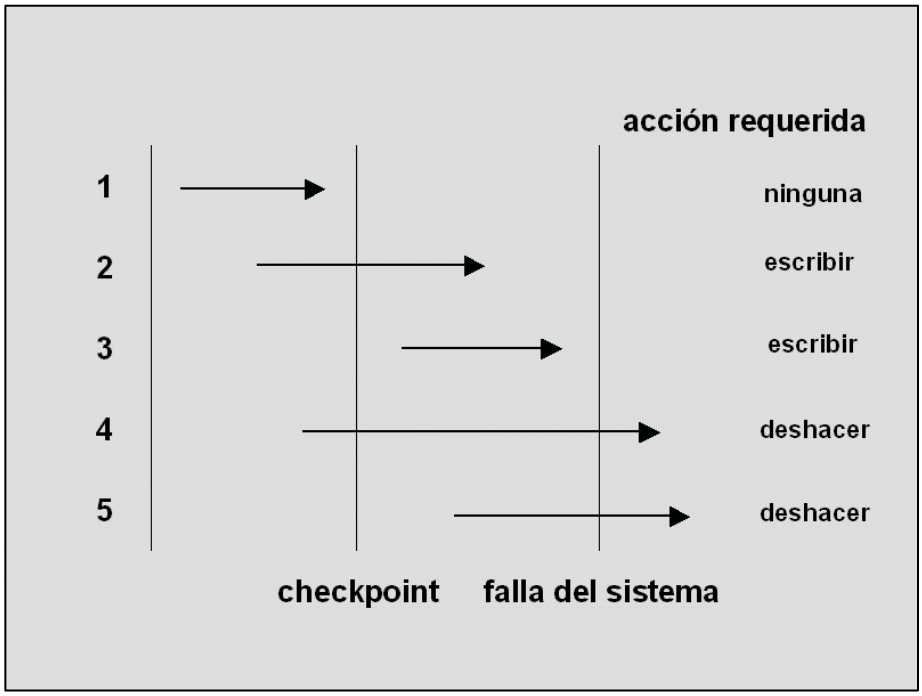
```
SELECT * FROM Padre
SELECT * FROM Hijo
go
```

## El checkpoint y la recuperación de transacciones

SQL Server garantiza que todas las transacciones confirmadas se reflejarán en la base de datos en la eventualidad de una falla del sistema. Para ello, utiliza el log de transacciones para escribir en la base de datos las transacciones confirmadas (commited transactions), y deshace las transacciones no confirmadas (uncommited transactions).

SQL Server ejecuta periódicamente el **checkpoint**, que consiste en la verificación del log de transacciones para determinar las transacciones confirmadas.

El siguiente diagrama resume algunas situaciones que pueden presentarse cuando se produce una falla del sistema.



- Para la transacción 1 se verificó COMMIT antes del checkpoint, por lo tanto ya está confirmada plenamente en la base de datos.
- Para las transacciones 2 y 3 el COMMIT se verificó despues del checkpoint, por lo tanto deben ser reconstruidas desde el log de transacciones.
- Para las transacciones 4 y 5 no se verificó el COMMIT, por lo tanto deben ser deshechas.

## Consideraciones al definir transacciones

Al momento de definir las transacciones se recomienda:

- Las transacciones deben ser lo mas cortas posible. Cuando toman demasiado tiempo se incrementa la probabilidad que los usuarios no puedan acceder a la data bloqueada.
- Evite que la transacción interactúe con el usuario. Cuando la transacción se inicia, ésta debe finalizar sin la intervención del usuario. Toda la interacción con el usuario debe ejecutarse antes que la transacción se inicie.
- Las principales sentencias en una transacción deben ser INSERT, UPDATE y DELETE.
- Evite anidar transacciones. Puede utilizar la variable global **@@trancount** para determinar cuántas transacciones están abiertas y cuan profundo es el anidamiento. La sentencia BEGIN TRANSACTION incrementa el valor de la variable en uno, y la sentencia ROLLBACK TRANSACTION la configura a cero.
- No se permite la creación de tablas temporales dentro de una transacción. Algunos procedimientos almacenados del sistema no pueden utilizarse porque generan tablas temporales.
- Algunas de las sentencias que tienen que ver con la creación, modificación o eliminación de objetos, no pueden ser utilizadas dentro de una transacción.

# Bloqueos

El motor de base de datos utiliza los bloqueos para garantizar la integridad de las transacciones y mantener la consistencia de la base de datos cuando muchos usuarios acceden a la data simultáneamente.

## Definición de bloqueo

Las transacciones usan bloqueos para impedir que otros usuarios puedan cambiar o leer los datos en una transacción que no se ha completado. Los bloqueos son necesarios para las transacciones en línea de los sistemas multiusuario. SQL Server usa el registro de transacciones (transaction log) para garantizar que las actualizaciones se completen y sean recuperables.

Los bloqueos previenen los conflictos de actualización. Los usuarios no pueden leer o modificar los datos que otros usuarios están modificando. Por ejemplo, si desea calcular una función agregada (como SUM()) sobre una columna, y asegurar que otra transacción no modifique los datos mientras se está calculando la función, puede solicitarle al sistema que aplique un bloqueo sobre los datos. Tenga en cuenta lo siguiente acerca de los bloqueos:

- Los bloqueos hacen posible la serialización de transacciones para que solo una persona en un momento dado pueda cambiar un dato.
- SQL Server asigna automáticamente el nivel apropiado de bloqueo para cada transacción. También es posible, por parte del desarrollador, controlar cómo algunos de los bloqueos se establecen.
- Los bloqueos son necesarios para permitir a los usuarios de las transacciones concurrentes acceder y actualizar los datos al mismo tiempo.

# Manejo de sesiones concurrentes

## Problemas que pueden presentarse

Durante la utilización de una base de datos SQL Server se presentan las sesiones concurrentes, es decir, mas de una sesión utilizando el mismo elemento de la base de datos. Durante las sesiones concurrentes se pueden presentar situaciones que comprometen la integridad de una transacción:

### Actualización perdida (Lost update)

Se presenta cuando una transacción B sobreescribe los cambios hechos por otra transacción A que todavía no ha terminado. Si la transacción A, que todavía no ha terminado, trata de leer los datos que ha cambiado, recibe un mensaje de datos no encontrados.

### Lectura sucia (Dirty read)

Se presenta cuando una transacción B lee datos que han sido modificados por otra transacción A, pero que todavía no han sido confirmados por ésta (committed). La transacción B podría estar utilizando data imprecisa o no existente, si es que la transacción A no confirma los cambios.

### Lectura no repetible (Nonrepeatable read)

Se presenta cuando una transacción A lee la misma fila mas de una vez, y obtiene valores diferentes debido a que una transacción B modificó la fila entre las lecturas de la transacción A.

### Valor fantasma (Phantom value)

Se presenta cuando una transacción A lee un conjunto de datos, y al volver a leer el conjunto obtiene datos adicionales inesperados. Esto puede ocurrir por un inadecuado aislamiento de la transacción A, que permitió que una transacción B insertara filas mientras la transacción A se estaba ejecutando.

Todos estos problemas se pueden evitar configurando adecuadamente los bloqueos (locks). Para ello, tenga presente lo siguiente:

- Una transacción no puede modificar los datos que están siendo modificados por otra transacción.
- Una transacción no puede leer los datos que están siendo modificados por otra transacción.
- Una transacción puede modificar las páginas que están siendo leídas por otra transacción.

## Elementos que se pueden bloquear

SQL Server bloquea automáticamente la cantidad de recursos adecuada al nivel de la tarea que se está ejecutando. Se pueden bloquear los siguientes elementos:

| Elemento | Descripción |
|---|---|
| **RID (Row Id)** | Un identificador de fila. Utilizado para bloquear una sola fila dentro de una tabla. |
| **Key (Clave)** | Un bloqueo de fila dentro de un índice. Usado para proteger rangos de claves en transacciones consecutivas. |
| **Page (Página)** | Una página de datos de 8 KB, o una página de indice. |
| **Extent (Extensión)** | Un grupo de páginas de datos, o páginas de índice contiguas. Utilizado durante la localización de espacio para la tabla. |
| **Table (Tabla)** | Toda la tabla incluyendo sus índices. |
| **Database (Base de datos)** | Toda la base de datos. Utilizado durante la restauración de la base de datos. |

# Tipos de bloqueos

## Bloqueos básicos

- Shared lock (Bloqueo compartido)
- Exclusive lock (Bloqueo exclusivo)

### Shared lock (Bloqueo compartido)

Se establece durante las operaciones de lectura. Cuando SQL Server aplica un bloqueo compartido a un elemento que está siendo leído por una transacción, una segunda transacción puede adquirir un bloqueo compartido sobre el elemento aun cuando la primera transacción no ha sido completada.

### Exclusive lock (Bloqueo exclusivo)

Se establece durante las operaciones que modifican datos (INSERT, UPDATE y DELETE).

Solo una transacción puede tener bloqueo exclusivo sobre un elemento. Una segunda transacción no puede adquirir bloqueo compartido sobre un elemento que tiene bloqueo exclusivo.

Una transacción no puede adquirir bloqueo exclusivo sobre un elemento hasta que todos sus bloqueos compartidos han sido liberados.

## Bloqueos en situaciones especiales

- Intent lock (Intento de bloqueo)
- Update lock (Bloqueo de actualización)
- Schema lock (Bloqueo de esquema)
- Bulk update lock (Bloqueo de actualización por volumen)

### Intent lock (Intento de bloqueo)

Se utiliza internamente para que una transacción no pueda adquirir un bloqueo de nivel superior cuando otra transacción ya tiene un bloqueo de nivel inferior. Por ejemplo, si una transacción tiene un bloqueo exclusivo a nivel de fila, otra transacción no podrá adquirir un bloqueo exclusivo a nivel de la tabla.

### Update lock (Bloqueo de actualización)

Se establece cuando una transacción lee una página para luego modificarla. Antes que la página sea modificada, el bloqueo de actualización es promovido a bloqueo exclusivo.

El bloqueo de actualización es compatible con el bloqueo compartido, porque indica que la página sera modificada posteriormente.

### Schema lock (Bloqueo de esquema)

Garantiza que una tabla o índice no seran eliminados (Schema stability, Sch-S), o su esquema modificado (Schema modification, Sch-M), cuando están siendo utilizados por otra sesión.

### Bulk update lock (Bloque de actualización por volumen)

Impide que otros procesos accedan a una tabla que está siendo actualizada mediante un proceso de copia por volumen (bulkcopy).

## Compatibilidad de los bloqueos

La siguiente tabla muestra la compatibilidad de los bloqueos existentes sobre un mismo elemento.

| | **Bloqueo existente sobre el elemento** | | | | | |
|---|---|---|---|---|---|---|
| **Bloqueo requerido para el elemento** | **IS** | **S** | **U** | **IX** | **SIX** | **X** |
| Intent Shared (IS) | Si | Si | Si | Si | Si | No |
| Shared (S) | Si | Si | Si | No | No | No |
| Update (U) | Si | Si | No | No | No | No |
| Intent Exclusive (IX) | Si | No | No | Si | No | No |
| Shared with Intent Exclusive (SIX) | Si | No | No | No | No | No |
| Exclusive (X) | No | No | No | No | No | No |

El bloqueo de modificación de esquema (Sch-M) es incompatible con todos los bloqueos.

El bloqueo de estabilidad de esquema (Sch-S) es compatible con todos los bloqueos excepto con el de modificación de esquema (Sch-M).

*Esta página se ha dejado en blanco intencionalmente.*