Niveles de Aislamiento en SQL Server

Es posible indicar el nivel de aislamiento de una determinada Transacción y de esta forma podemos definir el grado en que se debe aislar una transacción de las modificaciones de datos realizadas por otras transacciones.

Algunos niveles de aislamiento pueden permitir ciertos efectos secundarios de la simultaneidad, como las lecturas de datos sucios o las lecturas fantasmas.

En SQL Server, se pueden definir los siguientes niveles de aislamiento:

Isolation Level	Lectura sucia	Lectura no repetible	Lectura fantasma
Read uncommitted	Si	Si	Si
Read committed	No	Si	Si
Repeatable read	No	No	Si
Serializable	No	No	No

De forma predeterminada, SQL Server funciona con el nivel de aislamiento READ COMMITTED.

Lectura sucia: Se produce una lectura sucia cuando una transacción lee un dato que ha sido modificado por otra transacción pero aún ese cambio no fue confirmado. Esto puede generar inconsistencias, ya que la transacción que hace el cambio puede estar incompleta o bien puede terminar abortando el cambio y en ese caso la primer transacción habrá leído un valor que nunca debió ver.

Lectura no repetible: Las lecturas no repetibles se producen cuando una transacción tiene acceso a las mismas filas varias veces y lee datos distintos en cada ocasión. Eso se produce porque la información es modificada por otra transacción.

Lectura fantasma: Las lecturas fantasmas es cuando aparecen nuevas filas que cumplen una determinada condición de búsqueda y que inicialmente no fueron vistas cuando se consultaron las filas que cumplen ese criterio. Por ejemplo, si una transacción 1 lee un conjunto de filas que cumplen una determinada condición y luego una transacción 2 inserta una nueva fila que cumple con la condición consultada por la transacción 1 (o modifica alguna fila existente que no cumplía la condición de forma tal de que luego si la cumple), si la transacción 1 vuelve a ejecutar la consulta, obtendrá un resultado diferente, ya que aparecerán nuevas filas.

READ COMMITTED - La transacción no puede leer datos modificados por otras transacciones. Permite a otras transacciones pueden modificar los datos que se han leído. Esta opción es la predeterminada para SQL Server (incluido 2005).

READ UNCOMMITTED - La transacción es capaz de leer los datos modificados por otras transacciones pero que aún no han sido confirmadas (pendientes de COMMIT).

REPEATABLE READ - La transacción no puede leer datos modificados por otras transacciones y otras transacciones no pueden modificar los datos que se han leído.

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

Tipos de Bloqueos:
La selección de un nivel de aislamiento determina el tipo y el alcance de los bloqueos que va a adquirir en cada sentencia para proteger las modificaciones de datos. Estos bloqueos se mantienen hasta que se completa la transacción.

<table>
<tr><th></th><th>Read uncommitted</th><th>Read committed</th><th>Repeatable read</th><th>Serializable</th></tr>
<tr><td>SELECT</td><td>No genera ningún bloqueo sobre los datos consultados e ignora los bloqueos de otras transacciones.</td><td>No genera ningún bloqueo sobre los datos consultados.</td><td>Alcance: solo las filas del resultado.

Tipo de Bloqueo: Compartido (de lectura)

Se bloquean solo las filas consultadas.
Otras transacciones pueden consultar esas filas pero no podrán modificarlas.
Otras transacciones pueden insertar filas nuevas en la tabla.</td><td>Alcance: toda la tabla.

Tipo de Bloqueo: Compartido (de lectura)

Se bloquea toda la tabla consultada.
Otras transacciones pueden consultar los datos pero no podrán modificarlos.
No le permite a otras transacciones insertar filas nuevas en la tabla.</td></tr>
<tr><td>UPDATE</td><td colspan="3">Alcance: toda la tabla.

Tipo de Bloqueo: Exclusivo (de escritura)

Se bloquea toda la tabla modificada.
Otras transacciones no pueden consultar ni modificar ninguna fila, pero <u>si pueden insertar filas nuevas.</u></td><td>Alcance: toda la tabla.

Tipo de Bloqueo: Exclusivo (de escritura)

Se bloquea toda la tabla modificada.
Otras transacciones no pueden consultar ni modificar ninguna fila y <u>no pueden insertar filas nuevas.</u></td></tr>
<tr><td>INSERT</td><td colspan="4">Alcance: toda la tabla.

Tipo de Bloqueo: Exclusivo (de escritura)

Se bloquea toda la tabla modificada.
Otras transacciones no pueden consultar ni modificar ninguna fila, <u>pero si pueden insertar filas nuevas.</u></td></tr>
<tr><td>DELETE</td><td colspan="3">Alcance: toda la tabla.

Tipo de Bloqueo: Exclusivo (de escritura)

Se bloquea toda la tabla modificada.
Otras transacciones no pueden consultar ni modificar ninguna fila, pero <u>si pueden insertar filas nuevas.</u></td><td>Alcance: toda la tabla.

Tipo de Bloqueo: Exclusivo (de escritura)

Se bloquea toda la tabla modificada.
Otras transacciones no pueden consultar ni modificar ninguna fila y <u>no pueden insertar filas nuevas.</u></td></tr>
</table>

Ejemplos de los bloqueos que genera un SELECT:

Supongamos la siguiente tabla:

EMP

LEGAJO	NOMBRE	COD_DEPTO
1	Dani	1
2	Guille	2
3	Ale	2

Caso 1 - READ UNCOMMITTED

Transacción 1

```
SET TRANSACTION ISOLATION
LEVEL READ UNCOMMITTED;

BEGIN TRANSACTION;

SELECT  count(*)
FROM    Emp
WHERE   cod_depto=2;

SELECT  count(*)
FROM    Emp
WHERE   cod_depto=2;

COMMIT TRANSACTION;
```

Resultado = 2

Resultado = 1
(Ve los cambios hechos por T2, antes de que los mismos sean confirmados)

Transacción 2

```
SET TRANSACTION ISOLATION
LEVEL READ COMMITTED;

BEGIN TRANSACTION;

UPDATE Emp
SET    cod_depto=1
WHERE  nombre= 'Ale';

COMMIT TRANSACTION;
```

En este caso se produce una **Lectura sucia** en la segunda consulta de T1, ya que ve los cambios hechos por otras transacciones antes de que los mismos sean confirmados.

Caso 2 - READ COMMITTED (por defecto)

Transacción 1

```
SET TRANSACTION ISOLATION
LEVEL READ COMMITTED;

BEGIN TRANSACTION;

SELECT  count(*)
FROM    Emp
WHERE   cod_depto=2;

SELECT  count(*)
FROM    Emp
WHERE   cod_depto=2;
```

Resultado =2

Queda en espera de que T2 confirme sus cambios…

Luego del COMMIT de T2, la consulta devuelve 1.

```
COMMIT TRANSACTION;
```

Transacción 2

```
SET TRANSACTION ISOLATION
LEVEL READ COMMITTED;

BEGIN TRANSACTION;

UPDATE Emp
SET    cod_depto=1
WHERE  nombre= 'Ale';

COMMIT TRANSACTION;
```

En este ejemplo, se puede ver el problema de **Lectura no repetible**, ya que dentro de una misma Transacción (T1) se hace dos veces la misma lectura y arroja valores distintos.

Caso 3 - REPEATABLE READ

Transacción 1

```
SET TRANSACTION ISOLATION
LEVEL REPEATABLE READ;

BEGIN TRANSACTION;

SELECT  count(*)
FROM    Emp
WHERE   cod_depto=2;        } Resultado = 2

SELECT  count(*)
FROM    Emp
WHERE   cod_depto=2;        } Resultado = 2

COMMIT TRANSACTION;
```

Transacción 2

```
SET TRANSACTION ISOLATION
LEVEL READ COMMITTED;

BEGIN TRANSACTION;

UPDATE Emp
SET    cod_depto=1
WHERE  nombre='Ale';        } La primer lectura de T1, dejó bloqueadas las filas de su resultado y este UPDATE queda en espera de que finalice T1.

Luego del COMMIT de T1, el UPDATE finaliza.

COMMIT TRANSACTION;
```

T1 bloquea las filas leídas en la primer consulta y así evita el problema de **Lectura no repetible**.

Sin embargo, este nivel de aislamiento permite que otra transacción inserte filas nuevas.

Caso 4 - REPEATABLE READ

Transacción 1

```
SET TRANSACTION ISOLATION
LEVEL REPEATABLE READ;

BEGIN TRANSACTION;

SELECT  count(*)
FROM    Emp
WHERE   cod_depto=2;        } Resultado = 2

SELECT  count(*)
FROM    Emp
WHERE   cod_depto=2;        } Queda en espera de que T2 confirme sus cambios…

Luego del COMMIT de T2, la consulta devuelve 3.

COMMIT TRANSACTION;
```

Transacción 2

```
SET TRANSACTION ISOLATION
LEVEL READ COMMITTED;

BEGIN TRANSACTION;

INSERT INTO EMP
VALUES (4, 'Ceci',2);

COMMIT TRANSACTION;
```

En este ejemplo, se produce el problema de **Lectura fantasma**, ya que las dos consultas de T1 devuelven distintos valores debido a la aparición de nuevas filas.

Lo mismo sucedería si T2 en lugar de insertar una fila, modifica una fila que no fue incluida en la primer consulta de T1, de forma tal de que luego de la modificación si cumpla la condición de las consultas de T1. Por ejemplo: UPDATE Emp SET cod_depto=2 WHERE nombre='Dani';

Caso 5 - SERIALIZABLE

Transacción 1

```
SET TRANSACTION ISOLATION
LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

SELECT  count(*)
FROM    Emp
WHERE   cod_depto=2;
```

Resultado = 2

```
SELECT  count(*)
FROM    Emp
WHERE   cod_depto=2;
```

Resultado = 2

```
COMMIT TRANSACTION;
```

Transacción 2

```
SET TRANSACTION ISOLATION
LEVEL READ COMMITTED;

BEGIN TRANSACTION;

INSERT INTO EMP
VALUES (4, 'Ceci',2);
```

La primer lectura de T1, dejó bloqueada toda la tabla EMP y este INSERT queda en espera de que finalice T1.

Luego del COMMIT de T1, el INSERT finaliza.

```
COMMIT TRANSACTION;
```

En este ejemplo, no produce ningún problema. La ejecución es **Serializable**.