MCP MCTS MPN ADW SQL Server
Isaias.islas@live.com.mx

Niveles de aislamiento (Isolation Levels) en SQL Server

Los niveles de aislamiento (isolation levels) de SQL Server se utilizan para definir el grado en que una transacción debe estar aislada de las modificaciones de los datos o los datos realizados por otras transacciones simultáneas. Los diferentes niveles de aislamiento son:

1. Read Uncommitted
2. Read Committed
3. Repeatable Read
4. Serializable
5. Snapshot

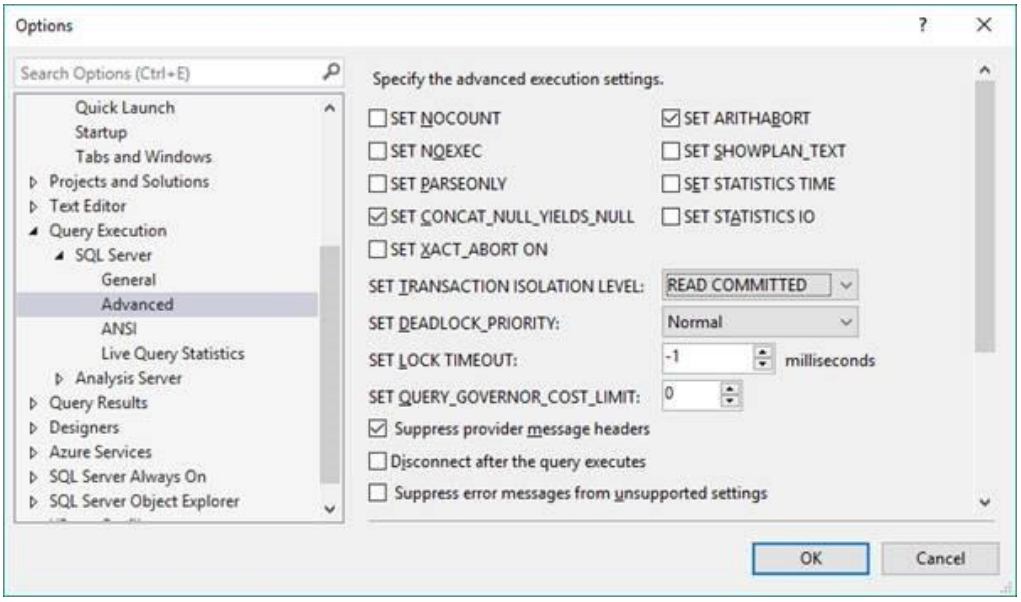
Read Committed es el nivel de aislamiento predeterminado. Sin embargo, se puede cambiar desde la ventana de consulta, así como desde las herramientas de Management Studio.

La sintaxis es:

```
SET TRANSACTION ISOLATION LEVEL
    {READ UNCOMMITTED
    | READ COMMITTED
    | REPEATABLE READ
    | SNAPSHOT
    | SERIALIZABLE
    }
```

En el menú de herramientas de SSMS, las opciones deben estar seleccionadas. En Query Execution -> Advanced, se puede modificar el menú desplegable para establecer el nivel de aislamiento de transacción.

MCP MCTS MPN ADW SQL Server
Isaias.islas@live.com.mx



Prerrequisitos

Los scripts para la creación de tablas de muestra y la población de datos se detallan a continuación. Estas tablas serán necesarias para demostrar el funcionamiento de diferentes niveles de aislamiento.

Dept_Exam_StudentMarks.sql:

Este script tiene la creación de tablas y los ejemplos de inserción de datos para las tablas de Dept y Exam. Esta secuencia de comandos también tiene declaraciones de creación de tablas y de inserción de datos de muestra para la tabla StudentMarks. Puede agregar tantos datos de muestra como desee. Para mi configuración, la tabla de StudentMarks se rellena inicialmente con 2, 397,616 registros.

```
INSERT INTO Exam
(examId,examName,examDesc)
VALUES
(201,'Data Structure','Theory Paper and Lab in Data Structure'),
(202,'Database Management System','Theory Paper and Lab in Database Management
System')

--create the table
CREATE TABLE StudentMarks
(
	studentId INT IDENTITY(1,1) PRIMARY KEY,
	deptId INT,
	examId INT,
	marksObtained INT
```

```
)

SELECT COUNT(1) FROM StudentMarks
--2397616

--Insert record in StudentMarks repeatedly
INSERT INTO StudentMarks (deptId,examId,marksObtained)
VALUES
(101,201,95)

--duplicate the records to increase the count
INSERT INTO StudentMarks (deptId,examId,marksObtained)
SELECT deptId,examId,marksObtained FROM StudentMarks
```

Discusión sobre Isolation Levels

En esta sección, analizaremos las características y limitaciones de los diferentes niveles de aislamiento. El comportamiento de dos transacciones simultáneas se simulará ejecutando dos scripts diferentes desde dos sesiones de usuario diferentes que accedan a los mismos recursos. La salida de ejecución será diferente para estas transacciones concurrentes en diferentes niveles de aislamiento.

El código de cada transacción se encuentra en los archivos al final del artículo. Los comentarios dentro de los archivos indican qué código de consulta se está ejecutando.

Cada uno de los diferentes niveles de aislamiento se muestra a continuación.

Read Uncommitted

Las transacciones que se ejecutan en este nivel no emiten bloqueos compartidos para evitar que otras transacciones modifiquen los datos leídos por la transacción actual. Además, las transacciones no están bloqueadas por bloqueos exclusivos en el momento de la modificación de los datos, lo que permite que otras transacciones lean los datos modificados que aún no se han confirmado.

Para este ejemplo, el script Query2.sql se ejecuta justo después del inicio de Query1.sql. En Consulta1, el valor de la columna marksObtained se establece en 90 en la primera declaración de actualización. Luego, este valor de columna se establece en 80 en la tercera declaración de actualización. Al final de la transacción 1, se confirma el valor 80.

La transacción 2 comienza después de que la transacción 1 ejecuta la primera instrucción de actualización. La transacción 2 lee el valor 90. Pero estos no son los datos comprometidos. La lectura de modificaciones no confirmadas se conoce como lectura sucia. Una vez que se completó la ejecución del script Query1.sql, Query2.sql se ejecutó nuevamente. Esta vez, da una salida como 80, que son los últimos datos confirmados por la Transacción 1.

La ejecución de las consultas se puede organizar de la siguiente manera:

Transaction 1 se inicia (Query1.sql).

```
BEGIN TRANSACTION

UPDATE StudentMarks
SET marksObtained = 90
WHERE deptId = 101 AND examId = 201
```

La ejecución de la transacción 1 continúa mientras la transacción 2 (Query2.sql) se inicia y confirma.

```
BEGIN TRANSACTION

SELECT marksObtained
FROM StudentMarks
WHERE deptId = 101 AND examId = 201 AND studentId = 1

COMMIT TRANSACTION
```

La transacción 1 (Query1.sql) ahora está completa y confirmada.

…

```
Update Exam
SET examDesc = 'Theory Paper and Lab Assignmnet in Data Structure'
WHERE examId = 201

UPDATE StudentMarks
SET marksObtained = 80
WHERE deptId = 101 AND examId = 201

COMMIT TRANSACTION
```

Read Committed

Con Read Committed, las transacciones emiten bloqueos exclusivos en el momento de la modificación de los datos, lo que no permite que otras transacciones lean los datos modificados que aún no se han confirmado. El nivel de aislamiento de lectura confirmada evita el problema de lectura sucia. Sin embargo, los datos pueden modificarse mediante otras transacciones entre extractos individuales dentro de la transacción actual, lo que da como resultado una lectura no repetible o una fila fantasma.

El comportamiento de READ COMMITTED depende de la configuración de la opción de base de datos READ_COMMITTED_SNAPSHOT.

Si READ_COMMITTED_SNAPSHOT se establece en OFF, los bloqueos compartidos evitan que otras transacciones modifiquen filas mientras la transacción actual está ejecutando una operación de lectura. Los bloqueos compartidos también impiden que la declaración lea filas modificadas por otras transacciones hasta que se complete la otra transacción. Si READ_COMMITTED_SNAPSHOT está establecido en ON (el valor predeterminado en SQL Azure Database), el control de versiones de fila se usa para presentar cada declaración con una instantánea coherente transaccional de los datos tal como existían al inicio de la declaración. No se emiten bloqueos en los datos.

Ejemplo 1

En este primer ejemplo, READ_COMMITTED_SNAPSHOT se establece en OFF. El script Query2.sql se ejecuta justo después del inicio de Query1.sql. En Consulta1, el valor de la columna marksObtained se establece en 90 en la primera declaración de actualización. Luego, este valor de columna se establece en 70 en la tercera declaración de actualización. Al final de la transacción 1, se compromete el valor 70.

Hasta que se complete la ejecución de Query1.sql, el lote de Query2.sql no devuelve ningún resultado. Está en estado de espera. Una vez que se completa la transacción 1, la transacción 2 devuelve la salida como 70, es decir, los últimos datos confirmados.

La ejecución de las consultas se completó como se muestra a continuación.

La transacción 1 se inicia (Query1.sql).

```
BEGIN TRANSACTION

UPDATE StudentMarks
SET marksObtained = 90
WHERE deptId = 101 AND examId = 201

…
```

La ejecución de la transacción 1 continúa y la transacción 2 (Query2.sql) también se inicia y confirma.

```
BEGIN TRANSACTION

SELECT marksObtained
FROM StudentMarks
WHERE deptId = 101 AND examId = 201 AND studentId = 1

COMMIT TRANSACTION
```

La transacción 1 se confirma (Query1.sql).

…

```
Update Exam
SET examDesc = 'Theory Paper and Lab Assignmnet in Data Structure'
WHERE examId = 201

UPDATE StudentMarks
SET marksObtained = 70 --80
WHERE deptId = 101 AND examId = 201

COMMIT TRANSACTION
```

Ejemplo 2

Para este ejemplo, READ_COMMITTED_SNAPSHOT se establece en ON. La instrucción ALTER debe ejecutarse para establecer la propiedad de instantánea con el nivel de aislamiento de lectura confirmada.

```
ALTER DATABASE <DB Name>
SET READ_COMMITTED_SNAPSHOT ON
```

El script Query2.sql se ejecuta justo después de iniciar Query1.sql. En Consulta1, el valor de la columna marksObtained se establece en 90 en la primera instrucción UPDATE. Luego, este valor de columna se establece en 80 en la tercera instrucción UPDATE. Al final de la transacción 1, se confirma el valor 80.

La instrucción SELECT en la transacción 2 no espera a que se confirme la transacción 1. Devuelve los últimos datos confirmados instantáneamente mientras Query1.sql todavía se está ejecutando. El resultado fue 70 en mi prueba. Si Query2.sql se ejecuta nuevamente después de completar Query1.sql, recuperará los últimos datos confirmados.

La ejecución de las consultas se muestra a continuación.

La transacción 1 se inicia (Query1.sql).

```
BEGIN TRANSACTION

UPDATE StudentMarks
SET marksObtained = 90
WHERE deptId = 101 AND examId = 201
```

La ejecución de la transacción 1 continúa y la transacción 2 (Query2.sql) se inicia y confirma.

```
BEGIN TRANSACTION

SELECT marksObtained
FROM StudentMarks
```

```
WHERE deptId = 101 AND examId = 201 AND studentId = 1

COMMIT TRANSACTION
```

La transaction 1 se confirma (Query1.sql).

…

```
Update Exam
SET examDesc = 'Theory Paper and Lab Assignmnet in Data Structure'
WHERE examId = 201

UPDATE StudentMarks
SET marksObtained = 70 --80
WHERE deptId = 101 AND examId = 201

COMMIT TRANSACTION
```

Ejemplo 3

En Query3, la transacción 1 primero recupera los valores de columna para el examId 201 de la tabla Exam. Después de esta declaración, ejecuta dos declaraciones UPDATE para la tabla StudentMarks y finalmente la misma instrucción SELECT para recuperar valores de columna para el examId 201 de la tabla Exam se ejecuta nuevamente.

La primera ejecución de la instrucción SELECT da un valor de examDesc donde:

```
examDesc = 'Theory Paper and Lab Assignmnet in Data Structure'.
```

El script Query4.sql se ejecuta justo después del inicio de Query3.sql. Cuando la transacción 1 está ejecutando las declaraciones de actualización de StudentMarks, la transacción 2 inició y confirmó el valor examDesc modificado para examId 201 donde:

```
examDesc = 'Corrected: Theory Paper and Lab Assignment in Data
Structure'.
```

La segunda ejecución de la instrucción SELECT en la transacción 1 da el valor examDesc modificado. Aquí, la ejecución múltiple de la misma instrucción de selección dentro de la misma transacción proporciona datos de salida diferentes. Este problema se conoce como lectura no repetible (Non-Repeatable Read).

La ejecución de las consultas es la siguiente. Primero, se inicia la transacción 1 (Query3.sql).

```
BEGIN TRANSACTION

SELECT examId,examName,examDesc
```

```
FROM Exam
WHERE examId = 201

…
```

La ejecución de la transacción 1 continúa. La transacción 2 (Query4.sql) se inicia y se confirma.

```
BEGIN TRANSACTION

Update Exam
SET examDesc = 'Corrected: Theory Paper and Lab Assignment in Data
Structure'
WHERE examId = 201

COMMIT TRANSACTION
```

La transacción 1es confirmada (Query3.sql).

```
UPDATE StudentMarks
SET marksObtained = 61
WHERE deptId = 101 AND examId = 201

UPDATE StudentMarks
SET marksObtained = 62
WHERE deptId = 101 AND examId = 201

SELECT examId,examName,examDesc
FROM Exam
WHERE examId = 201

COMMIT TRANSACTION
```

Repeatable Read

En Repeatable Read, las declaraciones no pueden leer datos que han sido modificados pero que otras transacciones aún no han confirmado. Ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que se complete la transacción actual.

Los bloqueos compartidos se colocan en todos los datos leídos por cada extracto en la transacción y se mantienen hasta que se completa la transacción. Esto evita que otras transacciones modifiquen las filas leídas por la transacción actual. Este nivel de aislamiento evita el problema de lectura no repetible.

Otras transacciones pueden insertar nuevas filas que coincidan con las condiciones de búsqueda de los extractos emitidos por la transacción actual. Si la transacción actual vuelve

a intentar la declaración, recuperará las nuevas filas, lo que da como resultado lecturas fantasmas.

Ejemplo 4

En Query3, la transacción 1 recupera los valores de columna para examId 201 de la tabla Examen. Después de esta instrucción, el lote ejecuta dos instrucciones UPDATE para la tabla StudentMarks. Por último, la misma instrucción SELECT se ejecuta de nuevo.

El script Query4.sql se ejecuta justo después del inicio de Query3.sql. Mientras se ejecuta la transacción 1, la transacción 2 intenta modificar el valor de examDesc para examId 201. Como el bloqueo compartido está activo hasta que se confirme la transacción 1, la transacción 2 debe esperar hasta que se confirme.

La primera y la segunda ejecución de la instrucción SELECT proporcionan el mismo valor examDesc que la salida y se resuelve el problema de lectura no repetible.

La ejecución de las consultas es la siguiente.

Se inicia la transacción 1 (Query3.sql).

```
BEGIN TRANSACTION

SELECT examId,examName,examDesc
FROM Exam
WHERE examId = 201
```

La ejecución de la transacción 1 continúa y se inicia la transacción 2 (Query4.sql) e intenta modificar el valor de examDesc para examId n.o 201. Sin embargo, la transacción 1 utiliza este registro con fines de lectura y los datos se bloquean para su actualización hasta que se confirma la transacción 1. Por lo tanto, la transacción 2 está en estado wating. Puede confirmar el cambio solo cuando la transacción 1 se confirma o se revierte.

```
BEGIN TRANSACTION

Update Exam
SET examDesc = 'Corrected: Theory Paper and Lab Assignment in Data
Structure'
WHERE examId = 201

COMMIT TRANSACTION
```

Se ha completado la transacción 1 (Query3.sql).

```
UPDATE StudentMarks
SET marksObtained = 61
WHERE deptId = 101 AND examId = 201

UPDATE StudentMarks
SET marksObtained = 62
WHERE deptId = 101 AND examId = 201

SELECT examId,examName,examDesc
FROM Exam
WHERE examId = 201

COMMIT TRANSACTION
```

Ejemplo 5

En Query5, la transacción 1 recupera los valores de columna Examen para examName - 'Estructura de datos'. Después de esta instrucción, ejecuta dos instrucciones UPDATE para la tabla StudentMarks y, finalmente, se vuelve a ejecutar la misma instrucción SELECT para recuperar los valores de columna Exam para examName.

Una vez completada la primera ejecución de la instrucción SELECT, se inicia la transacción 2 (Query6.sql). Esta transacción inserta un nuevo registro en la tabla Exam para examName á 'Data Structure' y confirma el cambio. Aquí, el bloqueo compartido está activo en los registros seleccionados en la instrucción SELECT de Query5.sql hasta que se confirme la transacción 1, pero no se impide la nueva inserción de registros con la misma condición de búsqueda. Como resultado, la transacción 2 agrega un nuevo registro en la tabla Examen.

La segunda ocurrencia de la instrucción SELECT en la transacción 1 ahora recuperará un registro adicional. La misma instrucción SELECT dentro de la misma transacción está dando algunas filas nuevas para la segunda ejecución en comparación con la primera. Estas filas adicionales se conocen como filas fantasmas.

La ejecución de las consultas se puede organizar según su hora de inicio y hora de finalización, como se indica a continuación:

Se inicia la transacción 1 (Query5.sql).

```
BEGIN TRANSACTION

SELECT examId,examName,examDesc
FROM Exam
WHERE examName = 'Data Structure'
```

La ejecución de la transacción 1 continúa y se inicia y confirma la transacción 2 (Query6.sql).

```
BEGIN TRANSACTION

INSERT INTO Exam
(examId,examName,examDesc)
VALUES
(203,'Data Structure','Duplicate: Corrected:Theory Paper and Lab
Assignment in Data Structure')

COMMIT TRANSACTION
```

Se ha completado la transacción 1 (Query5.sql).

```
UPDATE StudentMarks
SET marksObtained = 61
WHERE deptId = 101 AND examId = 201


UPDATE StudentMarks
SET marksObtained = 62
WHERE deptId = 101 AND examId = 201


SELECT examId,examName,examDesc
FROM Exam
WHERE examName = 'Data Structure'

COMMIT TRANSACTION
```

Serializable

En el nivel de aislamiento serializable, las instrucciones no pueden leer datos que se han modificado pero que otras transacciones aún no han confirmado. Ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que se complete la transacción actual. Otras transacciones no pueden insertar nuevas filas con valores de clave que estarían en el intervalo de claves leídas por cualquier instrucción de la transacción actual hasta que se complete la transacción actual.

Ejemplo 6

En Query5, la transacción 1 recupera los valores de columna Examen para examName = 'Data Structure. Después de esta instrucción, ejecuta dos instrucciones UPDATE para la tabla StudentMarks y, finalmente, se vuelve a ejecutar la misma instrucción SELECT para recuperar los valores de columna Exam para examName.

Una vez completada la primera ejecución de la instrucción SELECT, se inicia la transacción 2 (Query8.sql). Esta transacción intenta insertar un nuevo registro en la tabla Exam para examName á 'Data Structure' y confirmar el cambio. Sin embargo, se impide que la transacción 2 inserte un nuevo registro en la tabla Exam con el mismo valor de clave que se utiliza en los criterios de busqueda de la instrucción SELECT utilizada en la transacción 1 que aún no se ha confirmado. Por lo tanto, la transacción 2 tiene que esperar para confirmar los cambios hasta que se complete la transacción.

La segunda ocurrencia de la instrucción SELECT en la transacción 1 ahora recuperará el mismo número de registros recuperados por la primera ocurrencia de la instrucción SELECT. Por lo tanto, se resuelve el problema de las filas fantasma.

La ejecución de las consultas se puede organizar según su hora de inicio y hora de finalización, como se indica a continuación:

Se inicia la transacción 1 (Query5.sql).

```
BEGIN TRANSACTION

SELECT examId,examName,examDesc
FROM Exam
WHERE examName = 'Data Structure'
```

La ejecución de la transacción 1 continúa y se inicia la transacción 2 (Query8.sql). La transacción 2 tiene que esperar para confirmar sus cambios hasta que se complete la transacción 1.

```
BEGIN TRANSACTION

INSERT INTO Exam
(examId,examName,examDesc)
VALUES
(204,'Data Structure','Duplicate: Corrected: Theory Paper and Lab
Assignment in Data Structure')

COMMIT TRANSACTION
```

Se ha completado la transacción 1 (Query5.sql).

```
UPDATE StudentMarks
SET marksObtained = 61
WHERE deptId = 101 AND examId = 201

UPDATE StudentMarks
SET marksObtained = 62
```

```
WHERE deptId = 101 AND examId = 201

SELECT examId,examName,examDesc
FROM Exam
WHERE examName = 'Data Structure'

COMMIT TRANSACTION
```

Snapshot Isolation

En el aislamiento de SNAPSHOT, los datos leídos por cualquier instrucción de una transacción serán la versión transaccional coherente de los datos que existían al inicio de la transacción. Las modificaciones de datos realizadas por otras transacciones después del inicio de la transacción actual no son visibles para las instrucciones que se ejecutan en la transacción actual. Las transacciones SNAPSHOT no solicitan bloqueos al leer datos. Las transacciones SNAPSHOT que leen datos no impiden que otras transacciones escriban datos. Las transacciones que escriben datos no bloquean las transacciones SNAPSHOT para que no lean datos.

La opción ALLOW_SNAPSHOT_ISOLATION de base de datos debe establecerse en ON antes de iniciar una transacción con el nivel de aislamiento SNAPSHOT.

```
ALTER DATABASE <DB Name>
  SET ALLOW_SNAPSHOT_ISOLATION ON
```

La opción READ_COMMITTED_SNAPSHOT base de datos determina el comportamiento del nivel de aislamiento READ COMMITTED predeterminado cuando el aislamiento de instantáneas está habilitado en una base de datos.

Si READ_COMMITTED_SNAPSHOT opción de base de datos está establecida en ON, el motor de base de datos usa el control de versiones de fila y el aislamiento de instantáneas como valor predeterminado, en lugar de usar bloqueos para proteger los datos.

```
ALTER DATABASE <DB Name>
  SET READ_COMMITTED_SNAPSHOT ON
```

La importancia de los diferentes niveles de aislamiento

Solo se puede establecer una de las opciones de nivel de aislamiento a la vez y permanece establecida para esa conexión hasta que se cambia explícitamente. Un nivel de aislamiento más bajo aumenta la capacidad de muchos usuarios para acceder a los datos al mismo tiempo, pero aumenta el número de efectos de simultaneidad, como lecturas sucias o actualizaciones perdidas, etc. Por el contrario, un nivel de aislamiento más alto reduce los

tipos de efectos de simultaneidad que los usuarios pueden encontrar, pero requiere más recursos del sistema y aumenta las posibilidades de que una transacción bloquee otra.

El nivel de aislamiento más bajo, leído sin confirmar, puede recuperar datos modificados, pero no confirmados por otras transacciones. Todos los efectos secundarios de simultaneidad pueden ocurrir en lectura no confirmada, pero no hay bloqueo de lectura ni control de versiones, por lo que se minimiza la sobrecarga.

READ COMMITTED es el nivel de aislamiento predeterminado para SQL Server. Evita las lecturas sucias especificando que las instrucciones no pueden leer valores de datos que se han modificado pero que otras transacciones aún no han confirmado. Si la opción READ_COMMITTED_SNAPSHOT se establece como ON, las transacciones de lectura no tienen que esperar y pueden tener acceso a los últimos registros confirmados. Otras transacciones pueden modificar, insertar o eliminar datos entre ejecuciones de instrucciones SELECT individuales dentro de la transacción actual, lo que da como resultado lecturas no repetibles o filas fantasma.

REPEATABLE READ es un nivel de aislamiento más restrictivo que READ COMMITTED. Abarca READ COMMITTED y, además, especifica que ninguna otra transacción puede modificar o eliminar datos leídos por la transacción actual hasta que se confirme la transacción actual. La simultaneidad es menor que para READ COMMITTED porque los bloqueos compartidos en los datos de lectura se mantienen durante la transacción en lugar de liberarse al final de cada instrucción. Pero otras transacciones pueden insertar datos entre ejecuciones de instrucciones SELECT individuales dentro de la transacción actual, lo que da como resultado filas fantasma.

El nivel de aislamiento más alto, serializable, garantiza que una transacción recuperará exactamente los mismos datos cada vez que repita una operación de lectura, pero lo hace realizando un nivel de bloqueo que probablemente afecte a otros usuarios en sistemas multiusuario.

El aislamiento SNAPSHOT especifica que los datos leídos dentro de una transacción nunca reflejarán los cambios realizados por otras transacciones simultáneas. La transacción utiliza las versiones de fila de datos que existen cuando comienza la transacción. No se colocan bloqueos en los datos cuando se leen, por lo que las transacciones SNAPSHOT no impiden que otras transacciones escriban datos. Las transacciones que escriben datos no bloquean las transacciones de instantáneas para que no lean datos. Si no se puede esperar para la operación SELECT, pero los últimos datos confirmados son suficientes para mostrarse, este nivel de aislamiento puede ser adecuado.