(T24)討論Concurrent(同時進行的)Transactions  
CourseGUID: e48417fc-9db5-4e99-822c-706c5ccef6cc  
=======================================================================  
(T24)討論Concurrent(同時進行的)Transactions  
=======================================================================  
0. Summary

0.1. Concurrent Transactions

0.2. Summary

-----------

1. Create Sample Data

-----------

2. Transaction and Error Handling

2.1. Transaction and Error Handling

2.2. Transaction and Error Handling

-----------

3. Concurrent Transactions DirtyReadProblem(髒讀取問題)

3.1. ReadUncommitted(包括讀未提交) has DirtyReadProblem (髒讀取問題)

3.2. ReadCommitted(包括讀提交) fix DirtyReadProblem (髒讀取問題)

-----------

4. Concurrent Transactions LostUpdateProblem(更新遺失問題)

4.1. ReadCommitted(包括讀提交) has LostUpdateProblem(更新遺失問題)

4.2. "RepeatableRead"(可重複讀取) can fix LostUpdateProblem(更新遺失問題)

-----------

5. Concurrent Transactions NonRepeatableReadProblem(不可重複讀取問題)

5.1. ReadCommitted(包括讀未提交) has NonRepeatableReadProblem(不可重複讀取問題)

5.2. RepeatableRead(可重複讀取) can fix NonRepeatableReadProblem(不可重複讀取問題)

-----------

6. Concurrent Transactions PhantomReadProblem(幻讀問題)

6.1. "RepeatableRead"(可重複讀取) has PhantomReadProblem (幻讀問題)

6.2. "SerializableRead"(可串行化的讀取) can fix PhantomReadProblem (幻讀問題)

-----------

7. SerializableRead(可串行化的讀取) V.S. ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) V.S. READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

7.1. SerializableRead(可串行化的讀取)

7.2. ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

7.3. ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

7.4. ReadCommitted(包括讀提交)

7.5. READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

7.6. READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

7.7. READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

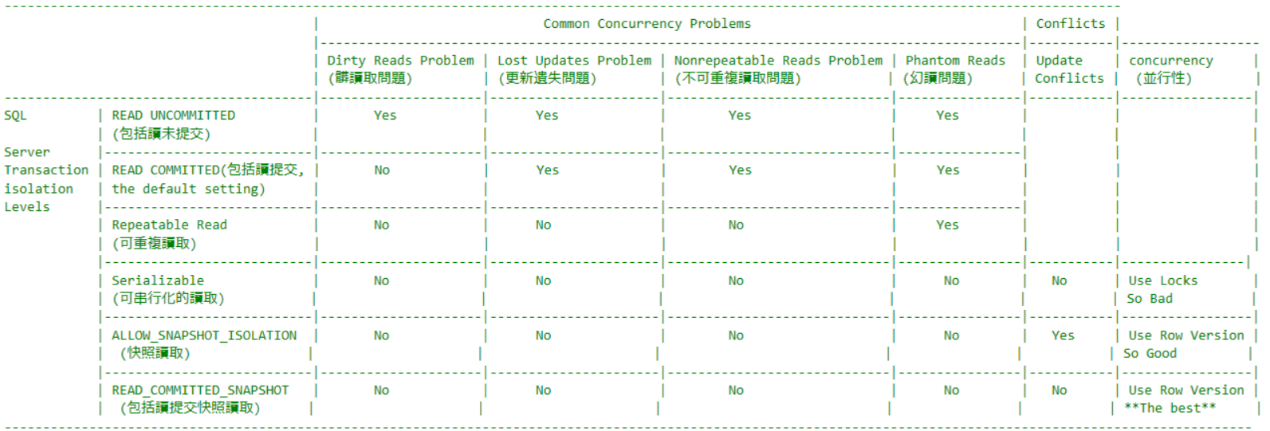
7.8. ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

-----------

8. Clean up  
=======================================================================

0. Summary

0.1. Concurrent Transactions



0.2. Summary

/\*

Transaction Revise :

1.

ErrorHandling\_Transaction\_@@Error\_TryCatch

1.1.

We have to ensure a group of sql statement

can perform successfully together or unsuccessfully together.

Thus, we need SQL Transaction.

--BEGIN TRANSACTION;

BEGIN TRAN

...

--ROLLBACK TRANSACTION;

COMMIT TRAN;

1.2.

Prohibit to ROLLBACK any inner Transaction

No matter inner Transaction has name or not.

If you really want to roll back inner Transaction,

don't use inner Transaction, Use Savepoint with SavepointName

--BEGIN TRAN Tranl;

--PRINT @@TRANCOUNT;       --1st TRANCOUNT, 1

--SAVE TRAN SavePoint;

--PRINT @@TRANCOUNT;       --2nd TRANCOUNT, 1

--...

--ROLLBACK TRAN SavePoint;

--PRINT @@TRANCOUNT;       --3rd TRANCOUNT, 1

----ROLLBACK TRAN Tranl

--COMMIT TRAN Tranl;

1.3.

When ROLLBACK Outter Transaction

No matter you have commit inner Transaction or not,

the inner Transaction will be forced to rollback too.

1.4.

--SELECT  ERROR\_NUMBER() AS [ERROR\_NUMBER()] , --245

--    ERROR\_MESSAGE() AS [ERROR\_MESSAGE()] ,         --Conversion failed when converting the varchar value 'Account1' to data type int.

--    ERROR\_PROCEDURE() AS [ERROR\_PROCEDURE()] ,     --NULL

--    ERROR\_STATE() AS [ERROR\_STATE()] ,                    --1

--    ERROR\_SEVERITY() AS [ERROR\_SEVERITY()] , --16

--    ERROR\_LINE() AS [ERROR\_LINE()]                        --9

Reference:

<https://docs.microsoft.com/en-us/sql/t-sql/functions/error-procedure-transact-sql>

<https://docs.microsoft.com/en-us/sql/t-sql/functions/error-state-transact-sql>

1.4.1.

Each kind of Error has ONE Error number just like and id, and ONE ERROR\_MESSAGE

In this case, ERROR\_NUMBER is 245.

ERROR\_MESSAGE is 'Conversion failed when converting the varchar value 'Account1' to data type int.'

1.4.2.

ERROR\_PROCEDURE() returns the name of the stored procedure or trigger

where an error occurred that caused the CATCH block of a TRY…CATCH.

In this case, ERROR\_PROCEDURE is NULL, because this is not stored procedure or trigger.

1.4.3.

ERROR\_STATE is kind of flat for debugging.

Each specific condition that raises the error assigns a unique state code.

A SQL Server support engineer can also use the state code from an error to find the location

in the source code where that error is being raised,

which may provide additional ideas on how to diagnose the problem.

1.4.4.

ERROR\_SEVERITY 16 means a general error.

This is kind of the category of error message.

1.4.5.

ERROR\_LINE returns the lind number where an error occurred.

1.5.

We have to ensure a group of sql statement

can perform successfully together or unsuccessfully together.

Thus, we need SQL Transaction and try catch

--BEGIN TRY

--    --BEGIN TRANSACTION;

--       BEGIN TRAN

--       ...

--       --ROLLBACK TRANSACTION;

--       COMMIT TRAN;

--END TRY

--BEGIN CATCH

--     ...

--END CATCH

1.5.1.

--INSERT  INTO [dbo].[BankTransaction]

--     (      [FromBankAccountID] ,

--           [ToBankAccountID] ,

--           [Amount]

--     )

--VALUES  ('Account1' ,    -- datatype Error

--           'Account2' , --datatype Error

--           @TransferAmount

--     );

FromBankAccountID and ToBankAccountID need int type parameter,

but the input is character string.

This will raise an error and automaticly "ROLLBACK" to beginning of transaction.

and then jump to   BEGIN CATCH clause.

\*/

--==============================================================================================================

/\*

-------------------------------------------------------------------------------------------------------------------------------------------------

                                        |                             Common Concurrency Problems                                   | Conflicts |

                                        |-------------------------------------------------------------------------------------------|-----------|------------------

                                        | Dirty Reads Problem | Lost Updates Problem | Nonrepeatable Reads Problem | Phantom Reads  | Update    | concurrency     |

                                        | (髒讀取問題)         | (更新遺失問題)        | (不可重複讀取問題)           | (幻讀問題)      | Conflicts |  (並行性)        |

----------------------------------------|---------------------|----------------------|-----------------------------|----------------|-----------|-----------------|

SQL         | READ UNCOMMITTED          |       Yes           |      Yes             |        Yes                  |     Yes        |           |                 |

            | (包括讀未提交)             |                     |                      |                             |                |           |                 |

Server      |---------------------------|---------------------|----------------------|-----------------------------|----------------|           |                 |

Transaction | READ COMMITTED(包括讀提交, |       No            |      Yes             |        Yes                  |     Yes        |           |                 |

isolation   | the default setting)      |                     |                      |                             |                |           |                 |

Levels      |---------------------------|---------------------|----------------------|-----------------------------|----------------|           |                 |

            | Repeatable Read           |       No            |      No              |        No                   |     Yes        |           |                 |

            | (可重複讀取)               |                     |                      |                             |                |           |                 |

            |---------------------------|---------------------|----------------------|-----------------------------|----------------|-----------|----------------|

            | Serializable              |       No            |      No              |        No                   |      No        |   No      | Use Locks       |

            | (可串行化的讀取)           |                     |                      |                             |                |           | So Bad          |

            |---------------------------|---------------------|----------------------|-----------------------------|----------------|-----------|-----------------|

            | ALLOW\_SNAPSHOT\_ISOLATION  |       No            |      No              |        No                   |      No        |   Yes     | Use Row Version |

            |  (快照讀取)               |                     |                      |                             |                |           | So Good         |

            |---------------------------|---------------------|----------------------|-----------------------------|----------------|-----------|-----------------|

            | READ\_COMMITTED\_SNAPSHOT   |       No            |      No              |        No                   |      No        |   No      | Use Row Version |

            |  (包括讀提交快照讀取)      |                     |                      |                             |                |           | \*\*The best\*\*     |

------------------------------------------------------------------------------------------------------------------------------------------------------------------

\*/

1.

Transaction Concurrency Problems : Dirty Reads Problem (髒讀取問題)

--------------------------------------------------------

1.1.

A Dirty Reads Problem (髒讀取問題) happens

when Transaction01 has been permitted to read the data

that has been modified by Transaction02 that has not yet been committed.

If the Transaction01 is rolled back after the second reads the data,

the Transaction02 has dirty data that does not exist anymore.

--------------------------------------------------------

1.2.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

Transaction1 read the ColumnA which stored value1 from database.

Transaction1 updated the ColumnA from value1 to value2

but has not committed yet.

In the mean time,

Transaction2 TRANSACTION ISOLATION LEVEL is "READ UNCOMMITTED"(包括讀未提交),

hence, Transaction2 read the ColumnA uncommitted value2.

Afterwards, Transaction1 met the Error and rollback.

Thus, the ColumnA will ROLLBACK from value2 back to value1.

However, Transaction2 was still using the

uncommitted value2 of ColumnA and doing its tasks.

The value2 is dirty data which does not exist anymore.

This is Dirty Reads Problem (髒讀取問題).

--------------------------------------------------------

1.3.

E.g.2

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

READ COMMITTED can fix Dirty Reads Problem (髒讀取問題)

Transaction1 read the ColumnA which stored value1 from database.

Transaction1 updated the ColumnA from value1 to value2

but has not committed yet.

In the mean time,

Transaction2 also try to read ColumnA.

Transaction2 TRANSACTION ISOLATION LEVEL is "READ COMMITTED"(包括讀提交),

hence, Transaction1 blocks the Transaction2 to read ColumnA

until Transaction1 has committed.

Afterwards, Transaction1 met the Error and rollback.

Thus, the ColumnA will ROLLBACK from value2 back to value1 and committed.

Therefore, Transaction2 finally can read the ColumnA committed value1.

--------------------------------------------------------

1.4.

Transaction isolation Level :

--SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

V.S.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

----------------------------------------

1.4.1.

--SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

ReadUncommitted(包括讀未提交) has DirtyReadProblem (髒讀取問題)

----------------------------------------

1.4.2.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

1.4.2.1.

ReadCommitted(包括讀提交the default setting),

can fix DirtyReadProblem(髒讀取問題),

but ReadCommitted has LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

-----------------------

1.4.2.1.

READ COMMITTED is the default isolation level for SQL Server.

It prevents dirty reads(髒讀取問題) by locking the uncommitted data.

---------------------------------------------------------------------

2.

Transaction Concurrency Problems : Lost Updates Problem (更新遺失問題).

--------------------------------------------------------

2.1.

Lost update problem happens when 2 transactions

read and update the same data.

--------------------------------------------------------

2.2.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

By default,

Transaction1 and Transaction2 Transaction isolation Levels

are both Read Committed (包括讀提交).

Transaction1 read the ColumnA which stored value1 from database.

In the mean time,

Transaction2 also read the ColumnA which stored value1 from database.

After 1 seconds, Transaction2 update the ColumnA from value1 to value2 and commit.

After Transaction2 COMMIT TRANSACTION,

then Transaction1 finally updated ColumnA from value1 to value3 and commit.

Thus, the ColumnA will actually store value3.

value2 does not exist any more, it is Lost Updates Problem (更新遺失問題).

--------------------------------------------------------

2.3.

E.g.2

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

REPEATABLE READ can fix Lost Updates Problem (更新遺失問題).

Transaction1 read the ColumnA which stored value1 from database.

In the mean time,

Transaction2 also read the ColumnA which stored value1 from database.

Transaction1 REPEATABLE READ level use locks on ColumnA to prevent

Transaction2 from "updating" the ColumnA from value1 to value2.

Then it makes Transaction2 return errors and discards the value2.

Transaction1 finally updated ColumnA from value1 to value3 and commit.

Thus, the ColumnA will actually store value3.

--------------------------------------------------------

2.4.

Transaction isolation Level :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

V.S.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

----------------------------------------

2.4.1.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

2.4.1.1.

ReadCommitted(包括讀提交the default setting),

can fix DirtyReadProblem(髒讀取問題),

but ReadCommitted has LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

-----------------------

2.4.1.2.

READ COMMITTED is the default isolation level for SQL Server.

It prevents dirty reads(髒讀取問題) by locking the uncommitted data.

----------------------------------------

2.4.2.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-----------------------

2.4.2.1.

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題),

but ReadCommitted has PhantomReadsProblem(幻讀).

-----------------------

2.4.2.2.

"Repeatable Read" (可重複讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

However, "Repeatable Read" (可重複讀取) isolation level

does not prevent new rows from being "inserted" by other transactions.

Thus, "Repeatable Read" (可重複讀取) CAN NOT fix PhantomReadProblem (幻讀問題).

---------------------------------------------------------------------

3.

Transaction Concurrency Problems : Non-Repeatable Read Problem (不可重複讀取問題)

--------------------------------------------------------

3.1.

Non-Repeatable Read Problem (不可重複讀取問題)

when Transaction1 reads the same data twice,

and Transaction2 updates that data

in between the first and second read of Transaction1.

Thus, Transaction1 first read and

Transaction1 second read became differenct value.

This is Non-Repeatable Read Problem (不可重複讀取問題)

--------------------------------------------------------

3.2.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

By default,

Transaction1 and Transaction2 Transaction isolation Levels

are both Read Committed (包括讀提交).

Transaction1 do the "First Read" for the ColumnA

which stored value1 from database.

During Transaction1 spends 5 seconds to do some tasks,

Transaction2 updates the ColumnA from value1 to value2 and commit.

After Transaction1 finished that some tasks,

and do the "Second Read" for the ColumnA

which stored value2 from database now.

Thus, Transaction1 first read(value1) and

Transaction1 second read(value2) became differenct value.

This is Non-Repeatable Read Problem (不可重複讀取問題).

--------------------------------------------------------

3.3.

E.g.2

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

REPEATABLE READ can fix Non-Repeatable Read Problem (不可重複讀取問題)

Transaction1 do the "First Read" for the ColumnA

which stored value1 from database.

Transaction1 REPEATABLE READ level use locks to prevent

Transaction2 from "updating" the ColumnA from value1 to value2.

Therefore, when Transaction1 finally read the ColumnA again

which will store value1 from database again.

Afterwards, Transaction1 REPEATABLE READ level release its locks on ColumnA.

Therefore, Transaction2 can finally update the ColumnA from value1 to value2.

--------------------------------------------------------

3.4.

Transaction isolation Level :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

V.S.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

----------------------------------------

3.4.1.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

3.4.1.1.

ReadCommitted(包括讀提交the default setting),

can fix DirtyReadProblem(髒讀取問題),

but ReadCommitted has LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

-----------------------

3.4.1.2.

READ COMMITTED is the default isolation level for SQL Server.

It prevents dirty reads(髒讀取問題) by locking the uncommitted data.

----------------------------------------

3.4.2.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-----------------------

3.4.2.1.

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題),

but ReadCommitted has PhantomReadsProblem(幻讀).

-----------------------

3.4.2.2.

"Repeatable Read" (可重複讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

However, "Repeatable Read" (可重複讀取) isolation level

does not prevent new rows from being "inserted" by other transactions.

Thus, "Repeatable Read" (可重複讀取) CAN NOT fix PhantomReadProblem (幻讀問題).

---------------------------------------------------------------------

4.

Transaction Concurrency Problems : Phantom Read Problem (幻讀問題)

--------------------------------------------------------

4.1.

when Transaction1 executes the same select query twice,

and Transaction2 insert a new data row

in between the first and second execution of Transaction1.

The new data row, which was added by Transaction2, matches

the WHERE clause of the query executed by the Transaction1.

Thus, Transaction1 gets a different number of rows

in the result set each time.

--------------------------------------------------------

4.2.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Transaction1 do the "1st Read"

and get "N1" rows in return.

During Transaction1 spends 5 seconds to do some tasks,

Transaction2 "inserted" 1 row and committed.

Thus, when Transaction1 do the "2nd Read"

and get "N1+1" rows in return.

"RepeatableRead"(可重複讀取)) has PhantomReadsProblem(幻讀).

--------------------------------------------------------

4.3.

E.g.2

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction1 do the "1st Read"

and get "N1" rows in return.

During Transaction1 spends 5 seconds to do some tasks,

Transaction2 "inserted" 1 row and committed.

Transaction1 SERIALIZABLE use locks to lock on table

and prevent Transaction2 from

"updating", "deleting", or "inserting" to the table.

When Transaction1 finished "2nd Read"

and still get "N1" rows in return,

Transaction2 finally can "insert" a new row

and make the table become "N1+1" rows.

"serializableRead" (可串行化的讀取) can fix PhantomReadsProblem(幻讀).

--------------------------------------------------------

4.4.

Transaction isolation Level :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

V.S.

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

----------------------------------------

4.4.1.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-----------------------

4.4.1.1.

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題),

but ReadCommitted has PhantomReadsProblem(幻讀).

-----------------------

4.4.1.2.

"Repeatable Read" (可重複讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

However, "Repeatable Read" (可重複讀取) isolation level

does not prevent new rows from being "inserted" by other transactions.

Thus, "Repeatable Read" (可重複讀取) CAN NOT fix PhantomReadProblem (幻讀問題).

----------------------------------------

4.4.2.

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

-----------------------

4.4.2.1.

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

, NonrepeatableReadsProblem(不可重複讀取問題),

and ReadCommitted has PhantomReadsProblem(幻讀).

-----------------------

4.4.2.2.

"serializable Read" (可串行化的讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

SerializableRead(可串行化的讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

In addition, SerializableRead(可串行化的讀取) isolation level

prevent new rows from being "inserted" by other transactions.

Thus, "serializable Read" (可串行化的讀取) CAN fix Phantom Read Problem (幻讀問題).

---------------------------------------------------------------------

5.

SerializableRead(可串行化的讀取) V.S.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) V.S.

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

--------------------------------------------------------

5.0.

----------------------------------------

5.0.1.

All these 3 isolation levels can fix

DirtyReadsProblem (髒讀取問題)

LostUpdateProblem(更新遺失問題),

NonrepeatableReadsProblem(不可重複讀取問題),

and PhantomReadsProblem(幻讀問題)

----------------------------------------

5.0.2.

SerializableRead(可串行化的讀取) uses locks to

block all other transactions.

Therefore, its concurrency(並行性) of transaction is bad.

----------------------------------------

5.0.3.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) and

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level copy the resource and maintains row versioning in Tempdb.

Row version is a unique transaction sequence number identifies each transaction,

and it determine the sequence of executing transactions.

Because it does not use locks.

Thus, other transactions still can use the resource.

Therefore, concurrency(並行性) of transaction is good.

----------------------------------------

5.0.4.

-----------------------

5.0.4.1.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

is vulnerable to update conflicts.

When both Transaction1 and Transaction2 are

using ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取),

and both Transaction1 and Transaction2 are updating the same column value,

One of Transaction will return Error as  update conflicts.

-----------------------

5.0.4.2.

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

has no update conflicts problems, because

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

uses row version to perform whatever ReadCommitted(包括讀提交) can do, plus

whatever ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) can do.

whatever ReadCommitted(包括讀提交) can do is to prevent update conflicts.

When both Transaction1 and Transaction2 are

using READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取),

and both Transaction1 and Transaction2 are updating the same column value,

one Transaction will wait another Transaction to finish,

then it will start to execute.

One of Transaction will NOT return Error as  update conflicts.

----------------------------------------

5.0.5.

Using READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

is so much easier than using ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取).

When using ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

might need to change some existing code.

However, using ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) is much easier.

All you need is to add this line.

--ALTER DATABASE [Sample] SET READ\_COMMITTED\_SNAPSHOT ON;

In addition, by default,

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Thus, you don't need to do anything else.

----------------------------------------

5.0.6.

Consider the following example,

More details will be discussed later.

-----------------------

5.0.6.1.

E.g.

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use SERIALIZABLE(可串行化的讀取) level : Select data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

-----------------------

5.0.6.2.

E.g.

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Select data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

-----------------------

5.0.6.3.

E.g.

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Update same data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

-----------------------

5.0.6.4.

E.g.

Transaction1 use ReadCommitted(包括讀提交) level : Update Data

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use ReadCommitted(包括讀提交) level : Select data

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

5.0.6.5.

E.g.

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Select data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

5.0.6.6.

E.g.

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update same data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

5.0.6.7.

E.g.

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level :

1st select before update, 2nd select after update

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

5.0.6.8.

E.g.

Transaction1 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Update Data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level :

1st select before update, 2nd select after update

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

This is the logic error and hard to debug.

--------------------------------------------------------

5.1.

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

----------------------------------------

5.1.1.

SerializableRead(可串行化的讀取) isolation level use locks

to block all other transactions,

so all other transactions can not insert, update, delete any thing.

Therefore,

SerializableRead(可串行化的讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

In addition, SerializableRead(可串行化的讀取) isolation level

prevent new rows from being "inserted" by other transactions.

Thus, "serializable Read" (可串行化的讀取) CAN fix Phantom Read Problem (幻讀問題).

Because of using locks, concurrency(並行性) of transaction is bad.

--------------------------------------------------------

5.2.

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

...

--SET TRANSACTION ISOLATION LEVEL OFF;

----------------------------------------

5.2.1.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) and

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level copy the resource and maintains row versioning in Tempdb.

Row version is a unique transaction sequence number identifies each transaction,

and it determine the sequence of executing transactions.

Because it does not use locks.

Thus, other transactions still can use the resource.

Therefore, concurrency(並行性) of transaction is good.

----------------------------------------

5.2.2.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

is vulnerable to update conflicts.

----------------------------------------

5.2.3.

Reference:

<https://stackoverflow.com/questions/232333/how-long-should-set-read-committed-snapshot-on-take>

Firstly,

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION ON;

Secondly, you may

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

If it take too long to enable SNAPSHOT,

close all ssms session, and re-open ssms, re-open the query.

execute the fillowing.

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION ON WITH ROLLBACK IMMEDIATE

it will immediately rollback any open transactions before starting the ALTER DATABASE statement.

Remember to disable it when you finished

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION OFF;

--------------------------------------------------------

5.3.

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

...

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT OFF;

----------------------------------------

5.3.1.

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

...

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT OFF;

V.S.

ReadCommitted(包括讀提交the default setting).

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level is very similar to

ReadCommitted(包括讀提交the default setting).

The differenct is the following.

----------------------------------------

5.3.2.

ReadCommitted(包括讀提交the default setting) use locks,

and it can fix DirtyReadsProblem(髒讀取問題).

It has LostUpdatesProblem(更新遺失問題),

NonrepeatableReadsProblem(不可重複讀取問題),

and PhantomReadsProblem(幻讀問題).

Because of locks, concurrency(並行性) of transaction is bad.

----------------------------------------

5.3.3.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) and

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level copy the resource and maintains row versioning in Tempdb.

Row version is a unique transaction sequence number identifies each transaction,

and it determine the sequence of executing transactions.

Because it does not use locks.

Thus, other transactions still can use the resource.

Therefore, concurrency(並行性) of transaction is good.

----------------------------------------

5.3.4.

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

has no update conflicts problems, because

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

uses row version to perform whatever ReadCommitted(包括讀提交) can do, plus

whatever ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) can do.

whatever ReadCommitted(包括讀提交) can do is to prevent update conflicts.

----------------------------------------

5.3.5.

Reference:

<https://stackoverflow.com/questions/232333/how-long-should-set-read-committed-snapshot-on-take>

Firstly,

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

Secondly, you may

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

If it take too long to enable SNAPSHOT,

close all ssms session, and re-open ssms, re-open the query.

execute the fillowing.

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON WITH ROLLBACK IMMEDIATE

it will immediately rollback any open transactions before starting the ALTER DATABASE statement.

Remember to disable it when you finished

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT OFF;

=====================================================

1. Create Sample Data

--===========================================================================

--T024\_01\_Create Sample Data

--===========================================================================

--\*\*\*\* Changeable variable

--Ctrl + F --> Search '@Product1AvailableQuantity',

--then update its stock if you want.

DECLARE @Product1AvailableQuantity INT = 20;

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'Product2' ) )

    BEGIN

        DROP TABLE Product2;

    END;

CREATE TABLE Product2

    (

      ProductID INT IDENTITY(1, 1)

                    PRIMARY KEY

                    NOT NULL ,

      ProductName NVARCHAR(100) ,

      AvailableQuantity INT

    );

INSERT  INTO Product2

VALUES  ( 'Product1', @Product1AvailableQuantity );

GO -- Run the previous command and begins new batch

SELECT  \*

FROM    dbo.Product2;

GO -- Run the previous command and begins new batch



=====================================================

2. Transaction and Error Handling

--===========================================================================

--T024\_02\_Transaction and Error Handling

--===========================================================================

2.1. Transaction and Error Handling

--===========================================================================

--T024\_02\_01

--Transaction and Error Handling

DECLARE @ProductID INT= 1;

--\*\*\*\* Changeable variable

DECLARE @OrderedQuantity INT= 100;

--DECLARE @OrderedQuantity INT= 2;

/\*

if ( @Product1AvailableQuantity < @OrderedQuantity ) then return Error.

In this case, IF(100 < 20), return Error, because 'Not enough stock available.'

Please try

--DECLARE @OrderedQuantity INT= 100;

Then Try

--DECLARE @OrderedQuantity INT= 2;

\*/

BEGIN TRY

    BEGIN TRAN;

       --Find the [AvailableQuantity]

    DECLARE @AvailableQuantity INT;

    SELECT  @AvailableQuantity = [AvailableQuantity]

    FROM    [dbo].[Product2]

    WHERE   [ProductID] = @ProductID;

       -- 1. Throw an error if Not enough stock available.

    IF ( @AvailableQuantity < @OrderedQuantity )

        BEGIN

            RAISERROR('Not enough stock available.',16,1);

        END;

       ----2. Adding new records to [Order] table

       --INSERT  INTO [dbo].[Order]

    --VALUES  ( ...);

       -- 3. Updating existing records

    UPDATE  [dbo].[Product2]

    SET     [AvailableQuantity] -= @OrderedQuantity

    WHERE   [ProductID] = @ProductID;

       -- 4. COMMIT TRANSACTION;

    COMMIT TRAN;

    PRINT 'Transaction Committed';

END TRY

BEGIN CATCH

    SELECT  ERROR\_NUMBER() AS [ERROR\_NUMBER()] ,

            ERROR\_MESSAGE() AS [ERROR\_MESSAGE()] ,

            ERROR\_PROCEDURE() AS [ERROR\_PROCEDURE()] ,

            ERROR\_STATE() AS [ERROR\_STATE()] ,

            ERROR\_SEVERITY() AS [ERROR\_SEVERITY()] ,

            ERROR\_LINE() AS [ERROR\_LINE()];

END CATCH;

--See the stock.

SELECT  \*

FROM    Product2

WHERE   [ProductID] = @ProductID;

/\*

1.

--SELECT  ERROR\_NUMBER() AS [ERROR\_NUMBER()] , --50000

--    ERROR\_MESSAGE() AS [ERROR\_MESSAGE()] ,         --Not enough stock available.

--    ERROR\_PROCEDURE() AS [ERROR\_PROCEDURE()] ,     --NULL

--    ERROR\_STATE() AS [ERROR\_STATE()] ,                    --1

--    ERROR\_SEVERITY() AS [ERROR\_SEVERITY()] , --16

--    ERROR\_LINE() AS [ERROR\_LINE()]                        --23

Reference:

<https://docs.microsoft.com/en-us/sql/t-sql/functions/error-procedure-transact-sql>

<https://docs.microsoft.com/en-us/sql/t-sql/functions/error-state-transact-sql>

1.1.

Each kind of Error has ONE Error number just like and id, and ONE ERROR\_MESSAGE

In this case, ERROR\_NUMBER is 50000.

ERROR\_MESSAGE is 'Not enough stock available.'

1.2.

ERROR\_PROCEDURE() returns the name of the stored procedure or trigger

where an error occurred that caused the CATCH block of a TRY…CATCH.

In this case, ERROR\_PROCEDURE is NULL, because this is not stored procedure or trigger.

1.3.

ERROR\_STATE is kind of flat for debugging.

Each specific condition that raises the error assigns a unique state code.

A SQL Server support engineer can also use the state code from an error to find the location

in the source code where that error is being raised,

which may provide additional ideas on how to diagnose the problem.

1.4.

ERROR\_SEVERITY 16 means a general error.

This is kind of the category of error message.

1.5.

ERROR\_LINE returns the lind number where an error occurred.

2.

We have to ensure a group of sql statement

can perform successfully together or unsuccessfully together.

Thus, we need SQL Transaction and try catch

When transaction failed, it rollback that transaction and raise Error,

then Perform Catch cluase.

Otherwise, commit the transaction.

--BEGIN TRY

--    --BEGIN TRANSACTION;

--       BEGIN TRAN

--       ...

--       --ROLLBACK TRANSACTION;

--       COMMIT TRAN;

--END TRY

--BEGIN CATCH

--     ...

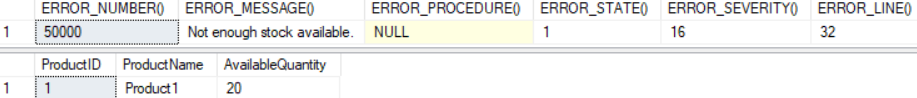
--END CATCH

\*/

When

DECLARE @OrderedQuantity INT= 100;

Returns ...



--------------------------------------------------------------------------

When

DECLARE @OrderedQuantity INT= 2;

Returns ...



2.2. Transaction and Error Handling

--===========================================================================

--T024\_02\_02

--Clean up

--\*\*\*\* Changeable variable

DECLARE @Product1AvailableQuantity INT= 20;

DECLARE @ProductID INT= 1;

UPDATE  [dbo].[Product2]

SET     [AvailableQuantity] = @Product1AvailableQuantity

WHERE   [ProductID] = @ProductID;

--See the stock.

SELECT  \*

FROM    Product2

WHERE   [ProductID] = @ProductID;

GO -- Run the previous command and begins new batch



=====================================================

3. Concurrent Transactions DirtyReadProblem(髒讀取問題)

Graphical user interface, text, application

Description automatically generated

--===========================================================================

--T024\_03\_Concurrent Transactions DirtyReadProblem(髒讀取問題)

--===========================================================================

/\*

1.

Transaction Concurrency Problems : Dirty Reads Problem (髒讀取問題)

--------------------------------------------------------

1.1.

A Dirty Reads Problem (髒讀取問題) happens

when Transaction01 has been permitted to read the data

that has been modified by Transaction02 that has not yet been committed.

If the Transaction01 is rolled back after the second reads the data,

the Transaction02 has dirty data that does not exist anymore.

--------------------------------------------------------

1.2.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

Transaction1 read the ColumnA which stored value1 from database.

Transaction1 updated the ColumnA from value1 to value2

but has not committed yet.

In the mean time,

Transaction2 TRANSACTION ISOLATION LEVEL is "READ UNCOMMITTED"(包括讀未提交),

hence, Transaction2 read the ColumnA uncommitted value2.

Afterwards, Transaction1 met the Error and rollback.

Thus, the ColumnA will ROLLBACK from value2 back to value1.

However, Transaction2 was still using the

uncommitted value2 of ColumnA and doing its tasks.

The value2 is dirty data which does not exist anymore.

This is Dirty Reads Problem (髒讀取問題).

--------------------------------------------------------

1.3.

E.g.2

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

READ COMMITTED can fix Dirty Reads Problem (髒讀取問題)

Transaction1 read the ColumnA which stored value1 from database.

Transaction1 updated the ColumnA from value1 to value2

but has not committed yet.

In the mean time,

Transaction2 also try to read ColumnA.

Transaction2 TRANSACTION ISOLATION LEVEL is "READ COMMITTED"(包括讀提交),

hence, Transaction1 blocks the Transaction2 to read ColumnA

until Transaction1 has committed.

Afterwards, Transaction1 met the Error and rollback.

Thus, the ColumnA will ROLLBACK from value2 back to value1 and committed.

Therefore, Transaction2 finally can read the ColumnA committed value1.

--------------------------------------------------------

1.4.

Transaction isolation Level :

--SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

V.S.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

----------------------------------------

1.4.1.

--SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

ReadUncommitted(包括讀未提交) has DirtyReadProblem (髒讀取問題)

----------------------------------------

1.4.2.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

1.4.2.1.

ReadCommitted(包括讀提交the default setting),

can fix DirtyReadProblem(髒讀取問題),

but ReadCommitted has LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

-----------------------

1.4.2.1.

READ COMMITTED is the default isolation level for SQL Server.

It prevents dirty reads(髒讀取問題) by locking the uncommitted data.

\*/

--------------------------------------------------------------------------------------

3.1. ReadUncommitted(包括讀未提交) has DirtyReadProblem (髒讀取問題)

--===========================================================================

--T024\_03\_01

--ReadUncommitted(包括讀未提交) has DirtyReadProblem (髒讀取問題)

/\*

1.

--SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

ReadUncommitted(包括讀未提交) has DirtyReadProblem (髒讀取問題)

2.

E.g.

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

\*/

----------------------------------------------------------------------------

--T024\_03\_01\_01

--Transaction1 : READ COMMITTED

--\*\*

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

DECLARE @AvailableQuantity INT;

BEGIN TRAN;

--Find the [AvailableQuantity]

SELECT  @AvailableQuantity = [AvailableQuantity]

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--1.

--Check if [AvailableQuantity] > orderedQuantity

--Other wise raise Error and roll back.

--2. Update AvailableQuantity

SET @AvailableQuantity -= 1;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--3. check if customer has enough money to pay.

--...

WAITFOR DELAY '00:00:10';

--4. Insufficient Funds. Rollback transaction

ROLLBACK TRANSACTION;

SELECT  @AvailableQuantity = [AvailableQuantity]

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

GO -- Run the previous command and begins new batch

----------------------------------------------------------------------------

--T024\_03\_01\_02

--Transaction2 : READ UNCOMMITTED

DECLARE @AvailableQuantity INT;

--\*\*

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

--Find the [AvailableQuantity]

SELECT  @AvailableQuantity = [AvailableQuantity]

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Transaction2 is still using dirty data that does not exist anymore.

--1.

--Check if [AvailableQuantity] > orderedQuantity

--Other wise raise Error and roll back.

--2. Update AvailableQuantity

--3. check if customer has enough money to pay.

--...

GO -- Run the previous command and begins new batch

----------------------------------------------------------------------------

--T024\_03\_01\_03

--Transaction3 :   FROM dbo.Product2 (NOLOCK)

--Transaction3 can replaced Transaction2

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = [AvailableQuantity]

FROM    dbo.Product2 (NOLOCK)

WHERE   ProductID = 1;

--Transaction2 is still using dirty data that does not exist anymore.

--1.

--Check if [AvailableQuantity] > orderedQuantity

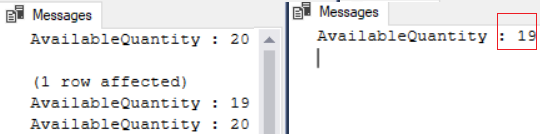
--Other wise raise Error and roll back.

--2. Update AvailableQuantity

--3. check if customer has enough money to pay.

--...

GO -- Run the previous command and begins new batch



/\*

1.

1.1.

If Transaction1 using

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

If Transaction2 using

--SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

--AvailableQuantity : 20

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 19

The AvailableQuantity will be 20,

and 19 does not exist any more.

ReadUncommitted(包括讀未提交) has DirtyReadProblem (髒讀取問題)

\*/

----------------------------------------------------------------------------

--T024\_03\_01\_04

--Clean up.

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 20

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

--AvailableQuantity : 20

--------------------------------------------------------------------------------------

3.2. ReadCommitted(包括讀提交) fix DirtyReadProblem (髒讀取問題)

--===========================================================================

--T024\_03\_02

--ReadCommitted(包括讀提交) fix DirtyReadProblem (髒讀取問題)

/\*

1.

--SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

ReadUncommitted(包括讀未提交) has DirtyReadProblem (髒讀取問題)

2.

READ COMMITTED is the default isolation level for SQL Server.

READ COMMITTED can fix Dirty Reads Problem (髒讀取問題)

It prevents dirty reads(髒讀取問題) by locking the uncommitted data.

3.

E.g.

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

\*/

----------------------------------------------------------------------------

--T024\_03\_02\_01

--Transaction1 : READ COMMITTED

--\*\*

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

DECLARE @AvailableQuantity INT;

BEGIN TRAN;

--Find the [AvailableQuantity]

SELECT  @AvailableQuantity = [AvailableQuantity]

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--1.

--Check if [AvailableQuantity] > orderedQuantity

--Other wise raise Error and roll back.

--2. Update AvailableQuantity

SET @AvailableQuantity -= 1;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--3. check if customer has enough money to pay.

--...

WAITFOR DELAY '00:00:10';

--4. Insufficient Funds. Rollback transaction

ROLLBACK TRANSACTION;

SELECT  @AvailableQuantity = [AvailableQuantity]

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

GO -- Run the previous command and begins new batch

----------------------------------------------------------------------------

--T024\_03\_02\_02

--Transaction2 : READ COMMITTED

DECLARE @AvailableQuantity INT;

--\*\*

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

--SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

--Find the [AvailableQuantity]

SELECT  @AvailableQuantity = [AvailableQuantity]

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Transaction2 is still using dirty data that does not exist anymore.

--1.

--Check if [AvailableQuantity] > orderedQuantity

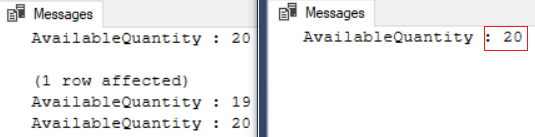
--Other wise raise Error and roll back.

--2. Update AvailableQuantity

--3. check if customer has enough money to pay.

--...

GO -- Run the previous command and begins new batch



/\*

1.

1.1.

If Transaction1 using

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

If Transaction2 using

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

--AvailableQuantity : 20

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 20

READ COMMITTED is the default isolation level for SQL Server.

It prevents dirty reads(髒讀取問題) by locking the uncommitted data.

\*/

----------------------------------------------------------------------------

--T024\_03\_02\_03

--Clean up.

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 20

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

--AvailableQuantity : 20

=====================================================

4. Concurrent Transactions LostUpdateProblem(更新遺失問題)

Text

Description automatically generated with medium confidence

/\*

2.

Transaction Concurrency Problems : Lost Updates Problem (更新遺失問題).

--------------------------------------------------------

2.1.

Lost update problem happens when 2 transactions

read and update the same data.

--------------------------------------------------------

2.2.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

By default,

Transaction1 and Transaction2 Transaction isolation Levels

are both Read Committed (包括讀提交).

Transaction1 read the ColumnA which stored value1 from database.

In the mean time,

Transaction2 also read the ColumnA which stored value1 from database.

After 1 seconds, Transaction2 update the ColumnA from value1 to value2 and commit.

After Transaction2 COMMIT TRANSACTION,

then Transaction1 finally updated ColumnA from value1 to value3 and commit.

Thus, the ColumnA will actually store value3.

value2 does not exist any more, it is Lost Updates Problem (更新遺失問題).

--------------------------------------------------------

2.3.

E.g.2

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

REPEATABLE READ can fix Lost Updates Problem (更新遺失問題).

Transaction1 read the ColumnA which stored value1 from database.

In the mean time,

Transaction2 also read the ColumnA which stored value1 from database.

Transaction1 REPEATABLE READ level use locks on ColumnA to prevent

Transaction2 from "updating" the ColumnA from value1 to value2.

Then it makes Transaction2 return errors and discards the value2.

Transaction1 finally updated ColumnA from value1 to value3 and commit.

Thus, the ColumnA will actually store value3.

--------------------------------------------------------

2.4.

Transaction isolation Level :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

V.S.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

----------------------------------------

2.4.1.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

2.4.1.1.

ReadCommitted(包括讀提交the default setting),

can fix DirtyReadProblem(髒讀取問題),

but ReadCommitted has LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

-----------------------

2.4.1.2.

READ COMMITTED is the default isolation level for SQL Server.

It prevents dirty reads(髒讀取問題) by locking the uncommitted data.

----------------------------------------

2.4.2.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-----------------------

2.4.2.1.

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題),

but ReadCommitted has PhantomReadsProblem(幻讀).

-----------------------

2.4.2.2.

"Repeatable Read" (可重複讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

However, "Repeatable Read" (可重複讀取) isolation level

does not prevent new rows from being "inserted" by other transactions.

Thus, "Repeatable Read" (可重複讀取) CAN NOT fix PhantomReadProblem (幻讀問題).

\*/

--------------------------------------------------------------------------------------

4.1. ReadCommitted(包括讀提交) has LostUpdateProblem(更新遺失問題)

--===========================================================================

--T024\_04\_01

--ReadCommitted(包括讀提交) has LostUpdateProblem(更新遺失問題)

/\*

1.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

ReadCommitted(包括讀提交the default setting),

can fix DirtyReadProblem(髒讀取問題),

but ReadCommitted has LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

2.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

\*/

----------------------------------------------------------------------------

--T024\_04\_01\_01

--Transaction1 : READ COMMITTED

--\*\*

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRAN;

--Find the [AvailableQuantity]

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = [AvailableQuantity]

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

-- Transaction takes 5 seconds

WAITFOR DELAY '00:00:5';

--1.

--Check if [AvailableQuantity] > orderedQuantity

--Other wise raise Error and roll back.

--2. check if customer has enough money to pay.

--3. Add new record to Order Table.

--...

--4. Update AvailableQuantity

SET @AvailableQuantity = @AvailableQuantity - 1;

UPDATE  dbo.Product2

SET     [AvailableQuantity] = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------------------------------------------

--T024\_04\_01\_02

--Transaction2 : READ COMMITTED

--\*\*

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRAN;

--Find the [AvailableQuantity]

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = [AvailableQuantity]

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

-- Transaction takes 1 seconds

WAITFOR DELAY '00:00:1';

--1.

--Check if [AvailableQuantity] > orderedQuantity

--Other wise raise Error and roll back.

--2. check if customer has enough money to pay.

--3. Add new record to Order Table.

--...

--4. Update AvailableQuantity

SET @AvailableQuantity = @AvailableQuantity - 3;

UPDATE  dbo.Product2

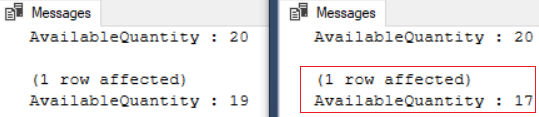
SET     [AvailableQuantity] = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch



/\*

1.

1.1.

If Transaction1 using

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

If Transaction2 using

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 17

The AvailableQuantity will be finally 19,

and 17 does not exist any more.

ReadCommitted has LostUpdateProblem(更新遺失問題)

\*/

----------------------------------------------------------------------------

--T024\_04\_01\_03

--Clean up :

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 19

--Clean up the changes. Rollback the the AvailableQuantity

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

--AvailableQuantity : 20

--------------------------------------------------------------------------------------

4.2. "RepeatableRead"(可重複讀取) can fix LostUpdateProblem(更新遺失問題)

--===========================================================================

--T024\_04\_02

--"RepeatableRead"(可重複讀取) can fix LostUpdateProblem(更新遺失問題)

/\*

1.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

ReadCommitted(包括讀提交the default setting),

can fix DirtyReadProblem(髒讀取問題),

but ReadCommitted has LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

2.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

"Repeatable Read" (可重複讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

However, "Repeatable Read" (可重複讀取) isolation level

does not prevent new rows from being "inserted" by other transactions.

Thus, "Repeatable Read" (可重複讀取) CAN NOT fix PhantomReadProblem (幻讀問題).

3.

E.g.2

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

REPEATABLE READ can fix Lost Updates Problem (更新遺失問題).

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

\*/

----------------------------------------------------------------------------

--T024\_04\_02\_01

--Transaction1 : READ COMMITTED

--\*\*

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRAN;

--Find the [AvailableQuantity]

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = [AvailableQuantity]

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

-- Transaction takes 5 seconds

WAITFOR DELAY '00:00:5';

--1.

--Check if [AvailableQuantity] > orderedQuantity

--Other wise raise Error and roll back.

--2. check if customer has enough money to pay.

--3. Add new record to Order Table.

--...

--4. Update AvailableQuantity

SET @AvailableQuantity = @AvailableQuantity - 1;

UPDATE  dbo.Product2

SET     [AvailableQuantity] = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------------------------------------------

--T024\_04\_02\_02

--Transaction2 : READ COMMITTED

--\*\*

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRAN;

--Find the [AvailableQuantity]

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = [AvailableQuantity]

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

-- Transaction takes 1 seconds

WAITFOR DELAY '00:00:1';

--1.

--Check if [AvailableQuantity] > orderedQuantity

--Other wise raise Error and roll back.

--2. check if customer has enough money to pay.

--3. Add new record to Order Table.

--...

--4. Update AvailableQuantity

SET @AvailableQuantity = @AvailableQuantity - 3;

UPDATE  dbo.Product2

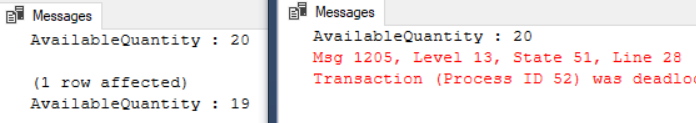
SET     [AvailableQuantity] = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch



/\*

1.

1.1.

If Transaction1 using

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

If Transaction2 using

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 20

When update AvailableQuantity, it will return error.

The AvailableQuantity will be finally 19.

REPEATABLE READ can fix LostUpdateProblem(更新遺失問題)

\*/

----------------------------------------------------------------------------

--T024\_04\_02\_03

--Clean up :

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 19

--Clean up the changes. Rollback the the AvailableQuantity

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

SELECT  \*

FROM    dbo.Product2

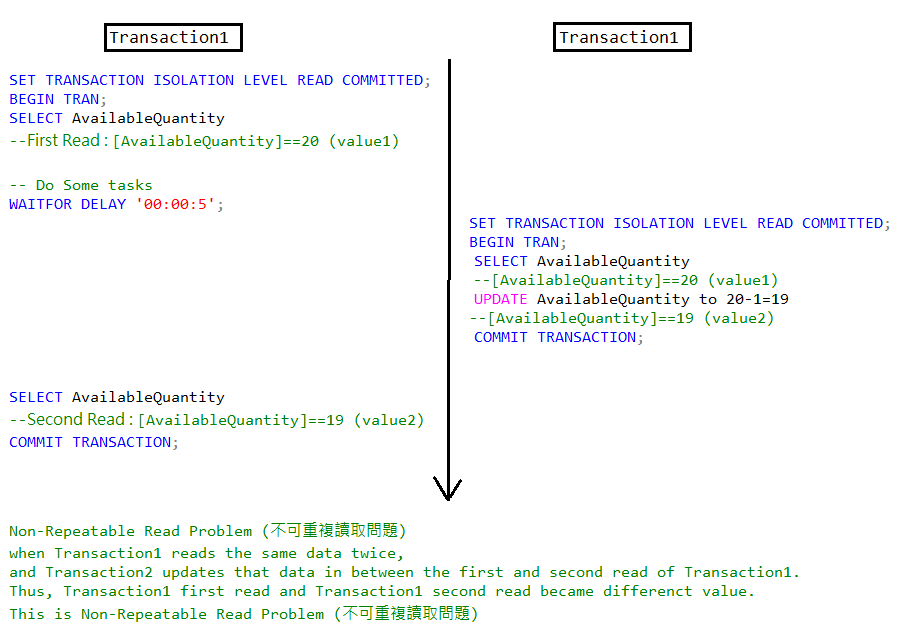
WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

--AvailableQuantity : 20

=====================================================

5. Concurrent Transactions NonRepeatableReadProblem(不可重複讀取問題)



--===========================================================================

--T024\_05\_Concurrent Transactions NonRepeatableReadProblem(不可重複讀取問題)

--===========================================================================

/\*

3.

Transaction Concurrency Problems : Non-Repeatable Read Problem (不可重複讀取問題)

--------------------------------------------------------

3.1.

Non-Repeatable Read Problem (不可重複讀取問題)

when Transaction1 reads the same data twice,

and Transaction2 updates that data

in between the first and second read of Transaction1.

Thus, Transaction1 first read and

Transaction1 second read became differenct value.

This is Non-Repeatable Read Problem (不可重複讀取問題)

--------------------------------------------------------

3.2.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

By default,

Transaction1 and Transaction2 Transaction isolation Levels

are both Read Committed (包括讀提交).

Transaction1 do the "First Read" for the ColumnA

which stored value1 from database.

During Transaction1 spends 5 seconds to do some tasks,

Transaction2 updates the ColumnA from value1 to value2 and commit.

After Transaction1 finished that some tasks,

and do the "Second Read" for the ColumnA

which stored value2 from database now.

Thus, Transaction1 first read(value1) and

Transaction1 second read(value2) became differenct value.

This is Non-Repeatable Read Problem (不可重複讀取問題).

--------------------------------------------------------

3.3.

E.g.2

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

REPEATABLE READ can fix Non-Repeatable Read Problem (不可重複讀取問題)

Transaction1 do the "First Read" for the ColumnA

which stored value1 from database.

Transaction1 REPEATABLE READ level use locks to prevent

Transaction2 from "updating" the ColumnA from value1 to value2.

Therefore, when Transaction1 finally read the ColumnA again

which will store value1 from database again.

Afterwards, Transaction1 REPEATABLE READ level release its locks on ColumnA.

Therefore, Transaction2 can finally update the ColumnA from value1 to value2.

--------------------------------------------------------

3.4.

Transaction isolation Level :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

V.S.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

----------------------------------------

3.4.1.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

3.4.1.1.

ReadCommitted(包括讀提交the default setting),

can fix DirtyReadProblem(髒讀取問題),

but ReadCommitted has LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

-----------------------

3.4.1.2.

READ COMMITTED is the default isolation level for SQL Server.

It prevents dirty reads(髒讀取問題) by locking the uncommitted data.

----------------------------------------

3.4.2.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-----------------------

3.4.2.1.

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題),

but ReadCommitted has PhantomReadsProblem(幻讀).

-----------------------

3.4.2.2.

"Repeatable Read" (可重複讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

However, "Repeatable Read" (可重複讀取) isolation level

does not prevent new rows from being "inserted" by other transactions.

Thus, "Repeatable Read" (可重複讀取) CAN NOT fix PhantomReadProblem (幻讀問題).

\*/

--------------------------------------------------------------------------------------

5.1. ReadCommitted(包括讀未提交) has NonRepeatableReadProblem(不可重複讀取問題)

--===========================================================================

--T024\_05\_01

--ReadCommitted(包括讀未提交) has NonRepeatableReadProblem(不可重複讀取問題)

/\*

1.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

ReadCommitted(包括讀提交the default setting),

can fix DirtyReadProblem(髒讀取問題),

but ReadCommitted has LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

2.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

\*/

----------------------------------------------------------------------------

--T024\_05\_01\_01

--Transaction1 : READ COMMITTED

--\*\*

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

DECLARE @AvailableQuantity INT;

--First read.

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

-- Do Some tasks

WAITFOR DELAY '00:00:5';

--Second read.

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------------------------------------------

--T024\_05\_01\_02

--Transaction2 : READ COMMITTED

--\*\*

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

DECLARE @AvailableQuantity INT;

--First read.

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Update

SET @AvailableQuantity -= 1;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

Graphical user interface, application

Description automatically generated

/\*

1.

1.1.

If Transaction1 using

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

If Transaction2 using

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

Transaction1 didn't update the AvailableQuantity,

but the 1st read and 2nd read value are different.

ReadUncommitted(包括讀未提交) has NonrepeatableReadsProblem(不可重複讀取問題)

\*/

----------------------------------------------------------------------------

--T024\_05\_01\_03

--Clean up :

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 19

--Clean up the changes. Rollback the the AvailableQuantity

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

--AvailableQuantity : 20

--------------------------------------------------------------------------------------

5.2. RepeatableRead(可重複讀取) can fix NonRepeatableReadProblem(不可重複讀取問題)

--===========================================================================

--T024\_05\_02

--RepeatableRead(可重複讀取) can fix NonRepeatableReadProblem(不可重複讀取問題)

/\*

1.

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

ReadCommitted(包括讀提交the default setting),

can fix DirtyReadProblem(髒讀取問題),

but ReadCommitted has LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

2.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

"Repeatable Read" (可重複讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

However, "Repeatable Read" (可重複讀取) isolation level

does not prevent new rows from being "inserted" by other transactions.

Thus, "Repeatable Read" (可重複讀取) CAN NOT fix PhantomReadProblem (幻讀問題).

3.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

\*/

----------------------------------------------------------------------------

--T024\_05\_02\_01

--Transaction1 : READ COMMITTED

--\*\*

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

DECLARE @AvailableQuantity INT;

--First read.

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

-- Do Some tasks

WAITFOR DELAY '00:00:5';

--Second read.

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------------------------------------------

--T024\_05\_02\_02

--Transaction2 : READ COMMITTED

--\*\*

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

DECLARE @AvailableQuantity INT;

--First read.

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Update

SET @AvailableQuantity -= 1;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

Graphical user interface, application, Word

Description automatically generated

/\*

1.

1.1.

If Transaction1 using

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

If Transaction2 using

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 20

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

Transaction1 REPEATABLE READ use locks on AvailableQuantity to prevent

Transaction2 from "updating" from 20 to 19.

After Transaction1 finished and release the locks, then

Transaction2 can finally update AvailableQuantity from 20 to 19.

Repeatable Read(可重複讀取) can fix NonrepeatableReadsProblem(不可重複讀取問題)

\*/

----------------------------------------------------------------------------

--T024\_05\_02\_03

--Clean up :

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 19

--Clean up the changes. Rollback the the AvailableQuantity

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

SELECT  \*

FROM    dbo.Product2

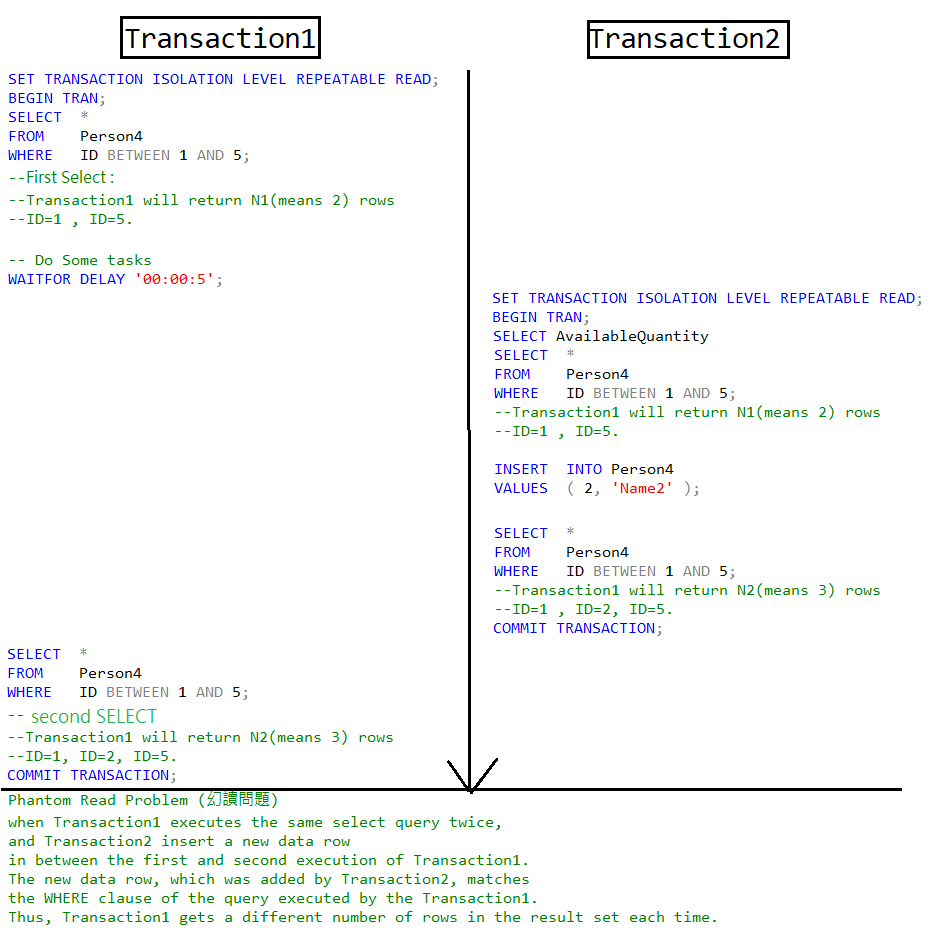
WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

--AvailableQuantity : 20

=====================================================

6. Concurrent Transactions PhantomReadProblem(幻讀問題)



--===========================================================================

--T024\_06\_Concurrent Transactions PhantomReadProblem(幻讀問題)

--===========================================================================

/\*

4.

Transaction Concurrency Problems : Phantom Read Problem (幻讀問題)

--------------------------------------------------------

4.1.

when Transaction1 executes the same select query twice,

and Transaction2 insert a new data row

in between the first and second execution of Transaction1.

The new data row, which was added by Transaction2, matches

the WHERE clause of the query executed by the Transaction1.

Thus, Transaction1 gets a different number of rows

in the result set each time.

--------------------------------------------------------

4.2.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Transaction1 do the "1st Read"

and get "N1" rows in return.

During Transaction1 spends 5 seconds to do some tasks,

Transaction2 "inserted" 1 row and committed.

Thus, when Transaction1 do the "2nd Read"

and get "N1+1" rows in return.

"RepeatableRead"(可重複讀取)) has PhantomReadsProblem(幻讀).

--------------------------------------------------------

4.3.

E.g.2

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction1 do the "1st Read"

and get "N1" rows in return.

During Transaction1 spends 5 seconds to do some tasks,

Transaction2 "inserted" 1 row and committed.

Transaction1 SERIALIZABLE use locks to lock on table

and prevent Transaction2 from

"updating", "deleting", or "inserting" to the table.

When Transaction1 finished "2nd Read"

and still get "N1" rows in return,

Transaction2 finally can "insert" a new row

and make the table become "N1+1" rows.

"serializableRead" (可串行化的讀取) can fix PhantomReadsProblem(幻讀).

--------------------------------------------------------

4.4.

Transaction isolation Level :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

V.S.

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

----------------------------------------

4.4.1.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-----------------------

4.4.1.1.

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題),

but ReadCommitted has PhantomReadsProblem(幻讀).

-----------------------

4.4.1.2.

"Repeatable Read" (可重複讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

However, "Repeatable Read" (可重複讀取) isolation level

does not prevent new rows from being "inserted" by other transactions.

Thus, "Repeatable Read" (可重複讀取) has PhantomReadProblem (幻讀問題).

----------------------------------------

4.4.2.

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

-----------------------

4.4.2.1.

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

, NonrepeatableReadsProblem(不可重複讀取問題),

and ReadCommitted has PhantomReadsProblem(幻讀).

-----------------------

4.4.2.2.

"serializable Read" (可串行化的讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

SerializableRead(可串行化的讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

In addition, SerializableRead(可串行化的讀取) isolation level

prevent new rows from being "inserted" by other transactions.

Thus, "serializable Read" (可串行化的讀取) CAN fix Phantom Read Problem (幻讀問題).

\*/

--------------------------------------------------------------------------------------

6.1. "RepeatableRead"(可重複讀取) has PhantomReadProblem (幻讀問題)

--===========================================================================

--T024\_06\_01

--"RepeatableRead" (可重複讀取) has PhantomReadProblem (幻讀問題)

/\*

1.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

"Repeatable Read" (可重複讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

However, "Repeatable Read" (可重複讀取) isolation level

does not prevent new rows from being "inserted" by other transactions.

Thus, "Repeatable Read" (可重複讀取) CAN NOT fix PhantomReadProblem (幻讀問題).

2.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

\*/

----------------------------------------------------------------------------

--T024\_06\_01\_01

--Create Sample Data

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'Person4' ) )

    BEGIN

        TRUNCATE TABLE dbo.Person4;

        DROP TABLE Person4;

    END;

GO -- Run the previous command and begins new batch

CREATE TABLE Person4

    (

      ID INT PRIMARY KEY

             NOT NULL ,

      [Name] NVARCHAR(50)

    );

GO -- Run the previous command and begins new batch

INSERT  INTO Person4

VALUES  ( 1, 'Name1' );

INSERT  INTO Person4

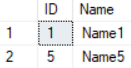
VALUES  ( 5, 'Name5' );

GO -- Run the previous command and begins new batch

SELECT  \*

FROM    Person4;

GO -- Run the previous command and begins new batch



----------------------------------------------------------------------------

--T024\_06\_01\_02

--Transaction1 : REPEATABLE READ

--"Repeatable Read" (可重複讀取) has PhantomReadProblem (幻讀問題).

--"serializable Read" (可串行化的讀取) CAN fix Phantom Read Problem (幻讀問題).

--\*\*

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

SELECT  \*

FROM    Person4

WHERE   ID BETWEEN 1 AND 5;

-- Do Some work

WAITFOR DELAY '00:00:5';

SELECT  \*

FROM    Person4

WHERE   ID BETWEEN 1 AND 5;

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------------------------------------------

--T024\_06\_01\_03

--Transaction2 : REPEATABLE READ

--"Repeatable Read" (可重複讀取) has PhantomReadProblem (幻讀問題).

--"serializable Read" (可串行化的讀取) CAN fix Phantom Read Problem (幻讀問題).

--\*\*

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

SELECT  \*

FROM    Person4

WHERE   ID BETWEEN 1 AND 5;

INSERT  INTO Person4

VALUES  ( 2, 'Name2' );

SELECT  \*

FROM    Person4

WHERE   ID BETWEEN 1 AND 5;

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

Graphical user interface, application

Description automatically generated

/\*

1.

1.1.

If Transaction1 using

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

If Transaction2 using

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 do the "1st Read"

and get "2" rows in return.

During Transaction1 spends 5 seconds to do some tasks,

Transaction2 "inserted" 1 row and committed.

Thus, when Transaction1 do the "2nd Read"

and get "2+1" rows in return.

"RepeatableRead"(可重複讀取)) has PhantomReadsProblem(幻讀).

\*/

----------------------------------------------------------------------------

--T024\_06\_01\_04

--Clean up :

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'Person4' ) )

    BEGIN

        TRUNCATE TABLE dbo.Person4;

        DROP TABLE Person4;

    END;

GO -- Run the previous command and begins new batch

CREATE TABLE Person4

    (

      ID INT PRIMARY KEY

             NOT NULL ,

      [Name] NVARCHAR(50)

    );

GO -- Run the previous command and begins new batch

INSERT  INTO Person4

VALUES  ( 1, 'Name1' );

INSERT  INTO Person4

VALUES  ( 5, 'Name5' );

GO -- Run the previous command and begins new batch

SELECT  \*

FROM    Person4

GO -- Run the previous command and begins new batch

--------------------------------------------------------------------------------------

6.2. "SerializableRead"(可串行化的讀取) can fix PhantomReadProblem (幻讀問題)

--===========================================================================

--T024\_06\_02

--"SerializableRead"(可串行化的讀取) can fix PhantomReadProblem (幻讀問題)

/\*

1.

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

"Repeatable Read" (可重複讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

However, "Repeatable Read" (可重複讀取) isolation level

does not prevent new rows from being "inserted" by other transactions.

Thus, "Repeatable Read" (可重複讀取) has PhantomReadProblem (幻讀問題).

2.

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

"serializable Read" (可串行化的讀取) isolation level ensures

that the data that one transaction has read,

will be prevented from being "updated" or "deleted" by any other transaction.

Therefore,

Repeatable Read(可重複讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

In addition, "serializable Read" (可串行化的讀取) isolation level

prevent new rows from being "inserted" by other transactions.

Thus, "serializable Read" (可串行化的讀取) CAN fix Phantom Read Problem (幻讀問題).

3.

E.g.1

Transaction1 :

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 :

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

\*/

----------------------------------------------------------------------------

--T024\_06\_02\_01

--Transaction1 : REPEATABLE READ

--"Repeatable Read" (可重複讀取) has PhantomReadProblem (幻讀問題).

--"serializable Read" (可串行化的讀取) CAN fix Phantom Read Problem (幻讀問題).

--\*\*

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

SELECT  \*

FROM    Person4

WHERE   ID BETWEEN 1 AND 5;

-- Do Some work

WAITFOR DELAY '00:00:5';

SELECT  \*

FROM    Person4

WHERE   ID BETWEEN 1 AND 5;

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------------------------------------------

--T024\_06\_02\_02

--Transaction2 : REPEATABLE READ

--"Repeatable Read" (可重複讀取) has PhantomReadProblem (幻讀問題).

--"serializable Read" (可串行化的讀取) CAN fix Phantom Read Problem (幻讀問題).

--\*\*

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

SELECT  \*

FROM    Person4

WHERE   ID BETWEEN 1 AND 5;

INSERT  INTO Person4

VALUES  ( 2, 'Name2' );

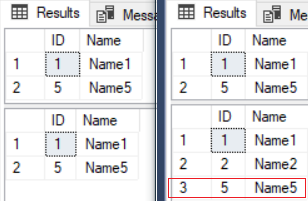
SELECT  \*

FROM    Person4

WHERE   ID BETWEEN 1 AND 5;

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch



/\*

1.

1.1.

If Transaction1 using

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

If Transaction2 using

--SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 do the "1st Read"

and get "2" rows in return.

During Transaction1 spends 5 seconds to do some tasks,

Transaction2 "inserted" 1 row and committed.

Transaction1 SERIALIZABLE use locks to lock on table

and prevent Transaction2 from

"updating", "deleting", or "inserting" to the table.

When Transaction1 finished "2nd Read"

and still get "2" rows in return,

Transaction2 finally can "insert" a new row

and make the table become "2+1" rows.

"serializableRead" (可串行化的讀取) can fix PhantomReadsProblem(幻讀).

\*/

----------------------------------------------------------------------------

--T024\_06\_02\_03

--Clean up :

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'Person4' ) )

    BEGIN

        TRUNCATE TABLE dbo.Person4;

        DROP TABLE Person4;

    END;

GO -- Run the previous command and begins new batch

CREATE TABLE Person4

    (

      ID INT PRIMARY KEY

             NOT NULL ,

      [Name] NVARCHAR(50)

    );

GO -- Run the previous command and begins new batch

INSERT  INTO Person4

VALUES  ( 1, 'Name1' );

INSERT  INTO Person4

VALUES  ( 5, 'Name5' );

GO -- Run the previous command and begins new batch

SELECT  \*

FROM    Person4

GO -- Run the previous command and begins new batch

7. SerializableRead(可串行化的讀取) V.S. ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) V.S. READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

--===========================================================================

--T024\_07\_SerializableRead(可串行化的讀取)

--V.S. ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

--V.S. READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

--===========================================================================

/\*

5.

SerializableRead(可串行化的讀取) V.S.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) V.S.

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

--------------------------------------------------------

5.0.

----------------------------------------

5.0.1.

All these 3 isolation levels can fix

DirtyReadsProblem (髒讀取問題)

LostUpdateProblem(更新遺失問題),

NonrepeatableReadsProblem(不可重複讀取問題),

and PhantomReadsProblem(幻讀問題)

----------------------------------------

5.0.2.

SerializableRead(可串行化的讀取) uses locks to

block all other transactions.

Therefore, its concurrency(並行性) of transaction is bad.

----------------------------------------

5.0.3.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) and

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level copy the resource and maintains row versioning in Tempdb.

Row version is a unique transaction sequence number identifies each transaction,

and it determine the sequence of executing transactions.

Because it does not use locks.

Thus, other transactions still can use the resource.

Therefore, concurrency(並行性) of transaction is good.

----------------------------------------

5.0.4.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

is vulnerable to update conflicts.

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

has no update conflicts problems, because

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

uses row version to perform whatever ReadCommitted(包括讀提交) can do, plus

whatever ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) can do.

whatever ReadCommitted(包括讀提交) can do is to prevent update conflicts.

----------------------------------------

5.0.5.

Using READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

is so much easier than using ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取).

When using ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

might need to change some existing code.

However, using ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) is much easier.

All you need is to add this line.

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION ON;

In addition, by default,

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Thus, you don't need to do anything else.

----------------------------------------

5.0.6.

Consider the following example,

More details will be discussed later.

-----------------------

5.0.6.1.

E.g.

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use SERIALIZABLE(可串行化的讀取) level : Select data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

-----------------------

5.0.6.2.

E.g.

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Select data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

-----------------------

5.0.6.3.

E.g.

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Update same data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

-----------------------

5.0.6.4.

E.g.

Transaction1 use ReadCommitted(包括讀提交) level : Update Data

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use ReadCommitted(包括讀提交) level : Select data

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

5.0.6.5.

E.g.

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Select data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

5.0.6.6.

E.g.

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update same data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

5.0.6.7.

E.g.

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level :

1st select before update, 2nd select after update

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

-----------------------

5.0.6.8.

E.g.

Transaction1 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Update Data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level :

1st select before update, 2nd select after update

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

This is the logic error and hard to debug.

--------------------------------------------------------

5.1.

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

----------------------------------------

5.1.1.

SerializableRead(可串行化的讀取) isolation level use locks

to block all other transactions,

so all other transactions can not insert, update, delete any thing.

Therefore,

SerializableRead(可串行化的讀取)

can fix LostUpdateProblem(更新遺失問題)

and NonrepeatableReadsProblem(不可重複讀取問題).

In addition, SerializableRead(可串行化的讀取) isolation level

prevent new rows from being "inserted" by other transactions.

Thus, "serializable Read" (可串行化的讀取) CAN fix Phantom Read Problem (幻讀問題).

Because of using locks, concurrency(並行性) of transaction is bad.

--------------------------------------------------------

5.2.

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

...

--SET TRANSACTION ISOLATION LEVEL OFF;

----------------------------------------

5.2.1.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) and

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level copy the resource and maintains row versioning in Tempdb.

Row version is a unique transaction sequence number identifies each transaction,

and it determine the sequence of executing transactions.

Because it does not use locks.

Thus, other transactions still can use the resource.

Therefore, concurrency(並行性) of transaction is good.

----------------------------------------

5.2.2.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

is vulnerable to update conflicts.

----------------------------------------

5.2.3.

Reference:

<https://stackoverflow.com/questions/232333/how-long-should-set-read-committed-snapshot-on-take>

Firstly,

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION ON;

Secondly, you may

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

If it take too long to enable SNAPSHOT,

close all ssms session, and re-open ssms, re-open the query.

execute the fillowing.

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION ON WITH ROLLBACK IMMEDIATE

it will immediately rollback any open transactions before starting the ALTER DATABASE statement.

Remember to disable it when you finished

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION OFF;

--------------------------------------------------------

5.3.

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

...

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT OFF;

----------------------------------------

5.3.1.

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

...

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT OFF;

V.S.

ReadCommitted(包括讀提交the default setting).

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level is very similar to

ReadCommitted(包括讀提交the default setting).

The differenct is the following.

----------------------------------------

5.3.2.

ReadCommitted(包括讀提交the default setting) use locks,

and it can fix DirtyReadsProblem(髒讀取問題).

It has LostUpdatesProblem(更新遺失問題),

NonrepeatableReadsProblem(不可重複讀取問題),

and PhantomReadsProblem(幻讀問題).

Because of locks, concurrency(並行性) of transaction is bad.

----------------------------------------

5.3.3.

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) and

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level copy the resource and maintains row versioning in Tempdb.

Row version is a unique transaction sequence number identifies each transaction,

and it determine the sequence of executing transactions.

Because it does not use locks.

Thus, other transactions still can use the resource.

Therefore, concurrency(並行性) of transaction is good.

----------------------------------------

5.3.4.

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

has no update conflicts problems, because

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

uses row version to perform whatever ReadCommitted(包括讀提交) can do, plus

whatever ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) can do.

whatever ReadCommitted(包括讀提交) can do is to prevent update conflicts.

----------------------------------------

5.3.5.

Reference:

<https://stackoverflow.com/questions/232333/how-long-should-set-read-committed-snapshot-on-take>

Firstly,

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

Secondly, you may

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

If it take too long to enable SNAPSHOT,

close all ssms session, and re-open ssms, re-open the query.

execute the fillowing.

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON WITH ROLLBACK IMMEDIATE

it will immediately rollback any open transactions before starting the ALTER DATABASE statement.

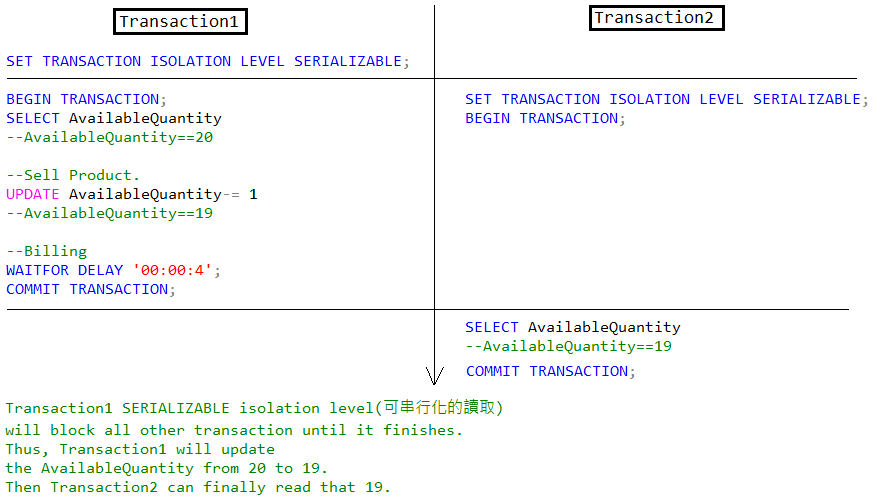
Remember to disable it when you finished

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT OFF;

\*/

--------------------------------------------------------------------------------------

7.1. SerializableRead(可串行化的讀取)



--===========================================================================

--T024\_07\_01

--SerializableRead(可串行化的讀取)

/\*

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use SERIALIZABLE(可串行化的讀取) level : Select data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

\*/

-----------------------------------------------------------------------------

--T024\_07\_01\_01

--Transaction1 use SERIALIZABLE level : Update Data

/\*

SERIALIZABLE transaction isolation level will

block all other transaction until it finishes.

\*/

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Sell One Product.

SET @AvailableQuantity -= 1;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

WAITFOR DELAY '00:00:4';

--Billing to customer...

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

-----------------------------------------------------------------------------

--T024\_07\_01\_02

--Transaction2 use SERIALIZABLE level : Select data

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

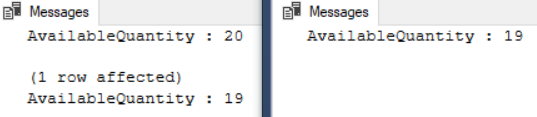
FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch



/\*

1.

1.1.

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use SERIALIZABLE(可串行化的讀取) level : Select data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 19

Transaction1 SERIALIZABLE isolation level(可串行化的讀取)

will block all other transaction until it finishes.

Thus, Transaction1 will update

the AvailableQuantity from 20 to 19.

Then Transaction2 can finally read that 19.

\*/

-----------------------------------------------------------------------------

--T024\_07\_01\_03

--Clean up:

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 19

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

--AvailableQuantity : 20

--------------------------------------------------------------------------------------

7.2. ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

Timeline

Description automatically generated

--===========================================================================

--T024\_07\_02

--ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

/\*

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Select data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

\*/

-------------------------------------------------------

--T024\_07\_02\_01

ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

/\*

Reference:

<https://stackoverflow.com/questions/232333/how-long-should-set-read-committed-snapshot-on-take>

Firstly,

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION ON;

Secondly, you may

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

If it take too long to enable SNAPSHOT,

close all ssms session, and re-open ssms, re-open the query.

execute the fillowing.

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION ON WITH ROLLBACK IMMEDIATE

it will immediately rollback any open transactions before starting the ALTER DATABASE statement.

Remember to disable it when you finished

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION OFF;

\*/

-------------------------------------------------------

--T024\_07\_02\_02

--Transaction1 use SERIALIZABLE level : Update Data

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Sell One Product.

SET @AvailableQuantity -= 1;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

WAITFOR DELAY '00:00:4';

--Billing to customer...

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------------------

--T024\_07\_02\_03

--Transaction2 use SNAPSHOT level : Select data

SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

BEGIN TRANSACTION;

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

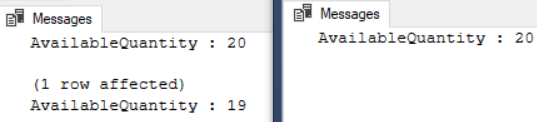
FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch



/\*

1.

1.1.

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Select data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 20

Transaction1 SERIALIZABLE isolation level(可串行化的讀取)

will block all other transactions

to insert/update/delete until it finishes.

Thus, Transaction1 will update

the AvailableQuantity from 20 to 19.

However, Transaction2 ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

will take the copy version data to read the data.

When Transaction2 read the AvailableQuantity,

it was still 20.  Thus, return 20.

\*/

----------------------------------------------------

--T024\_07\_02\_04

--Clean up.

ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION OFF;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 19

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

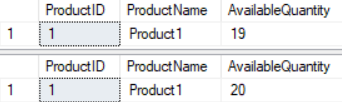
SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

--AvailableQuantity : 20



--------------------------------------------------------------------------------------

7.3. ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)



--===========================================================================

--T024\_07\_03

--ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

/\*

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Update same data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

\*/

-------------------------------------------------------

--T024\_07\_03\_01

ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

/\*

Reference:

<https://stackoverflow.com/questions/232333/how-long-should-set-read-committed-snapshot-on-take>

Firstly,

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION ON;

Secondly, you may

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

If it take too long to enable SNAPSHOT,

close all ssms session, and re-open ssms, re-open the query.

execute the fillowing.

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION ON WITH ROLLBACK IMMEDIATE

it will immediately rollback any open transactions before starting the ALTER DATABASE statement.

Remember to disable it when you finished

--ALTER DATABASE Sample3 SET ALLOW\_SNAPSHOT\_ISOLATION OFF;

\*/

------------------------------------------------------------------------------

--T024\_07\_03\_02

--Transaction1 use SERIALIZABLE level : Update Data

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

--Get the stock quanty

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Deduct the quantity.

SET @AvailableQuantity -= 3;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Do some tasks.

--E.g. Billing to customers

WAITFOR DELAY '00:00:4';

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

------------------------------------------------------------------------------

--T024\_07\_03\_03

--Transaction2 use SNAPSHOT level : Update same data

SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

BEGIN TRANSACTION;

--Get the stock quanty

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Deduct the quantity.

SET @AvailableQuantity -= 1;

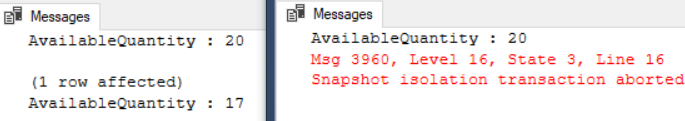
UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;



/\*

1.

1.1.

Transaction1 use SERIALIZABLE(可串行化的讀取) level : Update Data

--SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Update same data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 17

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--Then return Error when update same data.

Transaction1 SERIALIZABLE isolation level(可串行化的讀取)

will block all other transactions

to insert/update/delete until it finishes.

Thus, Transaction1 will update

the AvailableQuantity from 20 to 17.

However, Transaction2 ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

will take the copy version data to read the data.

When Transaction2 read the AvailableQuantity,

it was still 20.  Thus, return 20.

When Transaction2 tried to update the AvailableQuantity,

it was blocked by Transaction1 SERIALIZABLE isolation level(可串行化的讀取).

Thus, return Error.

\*/

-------------------------------------------------------

--T024\_07\_03\_04

--Clean up

ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION OFF;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 17

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

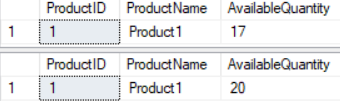
SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

--AvailableQuantity : 20



7.4. ReadCommitted(包括讀提交)

Graphical user interface, text

Description automatically generated

--===========================================================================

--T024\_07\_04

--ReadCommitted(包括讀提交)

/\*

Transaction1 use ReadCommitted(包括讀提交) level : Update Data

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use ReadCommitted(包括讀提交) level : Select data

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

\*/

----------------------------------------

--T024\_07\_04\_01

--Transaction1 use ReadCommitted(包括讀提交) level : Update Data

--Alter database [Sample] SET READ\_COMMITTED\_SNAPSHOT OFF

/\*

Remember to disable READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

We use normal ReadCommitted(包括讀提交) in this example.

\*/

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

--Get the stock quantity

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Sell the product

SET @AvailableQuantity -= 1;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Billing to Customer

WAITFOR DELAY '00:00:4';

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------

--T024\_07\_04\_02

--Transaction2 use ReadCommitted(包括讀提交) level : Select data

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

--Get the stock quantity

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

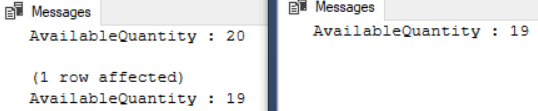
FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch



/\*

1.

1.1.

Transaction1 use ReadCommitted(包括讀提交) level : Update Data

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use ReadCommitted(包括讀提交) level : Select data

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 19

Transaction1 ReadCommitted isolation level(包括讀提交)

will block all other transactions

to insert/update/delete until it finishes.

Thus, Transaction1 will update

the AvailableQuantity from 20 to 19.

Transaction2 ReadCommitted isolation level(包括讀提交)

will read the committed value which is 19.

\*/

----------------------------------------------------

--T024\_07\_04\_03

--Clean up the changes. Rollback the the AvailableQuantity

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

Graphical user interface, application, table

Description automatically generated with medium confidence

--------------------------------------------------------------------------------------

7.5. READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

Graphical user interface, text, timeline

Description automatically generated with medium confidence

--===========================================================================

--T024\_07\_05

--READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

/\*

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Select data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

\*/

----------------------------------------

--T024\_07\_05\_01

ALTER DATABASE [Sample] SET READ\_COMMITTED\_SNAPSHOT ON;

/\*

Reference:

<https://stackoverflow.com/questions/232333/how-long-should-set-read-committed-snapshot-on-take>

Firstly,

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

Secondly, you may

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

If it take too long to enable SNAPSHOT,

close all ssms session, and re-open ssms, re-open the query.

execute the fillowing.

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON WITH ROLLBACK IMMEDIATE

it will immediately rollback any open transactions before starting the ALTER DATABASE statement.

Remember to disable it when you finished

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT OFF;

\*/

----------------------------------------

--T024\_07\_05\_02

--Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

--Get the stock quantity

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Sell the product

SET @AvailableQuantity -= 1;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Billing to Customer

WAITFOR DELAY '00:00:4';

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------

--T024\_07\_05\_03

--Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Select data

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

--Get the stock quantity

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

Graphical user interface, application, Word

Description automatically generated

/\*

1.

1.1.

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Select data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

1.2.

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

has no update conflicts problems, because

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

uses row version to perform whatever ReadCommitted(包括讀提交) can do, plus

whatever ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) can do.

whatever ReadCommitted(包括讀提交) can do is to prevent update conflicts.

1.3.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 20

Transaction1 READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level will use row version to block all other transactions

to insert/update/delete until it finishes.

Thus, Transaction1 will update

the AvailableQuantity from 20 to 19.

However, Transaction2 READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level will take the copy version data to read the data.

When Transaction2 read the AvailableQuantity,

it was still 20.  Thus, return 20.

\*/

----------------------------------------------------

--T024\_07\_05\_04

--Clean up.

ALTER DATABASE [Sample] SET READ\_COMMITTED\_SNAPSHOT OFF;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 19

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

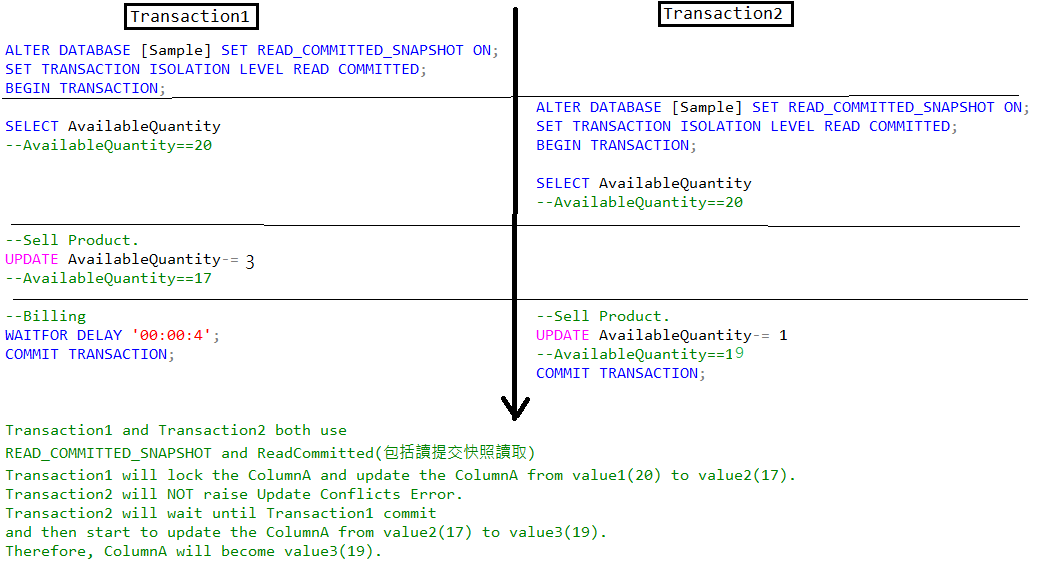
--AvailableQuantity : 20

Graphical user interface, application

Description automatically generated

--------------------------------------------------------------------------------------

7.6. READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)



--===========================================================================

--T024\_07\_06

--READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

/\*

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update same data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

\*/

----------------------------------------

--T024\_07\_06\_01

ALTER DATABASE [Sample] SET READ\_COMMITTED\_SNAPSHOT ON;

/\*

Reference:

<https://stackoverflow.com/questions/232333/how-long-should-set-read-committed-snapshot-on-take>

Firstly,

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

Secondly, you may

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

If it take too long to enable SNAPSHOT,

close all ssms session, and re-open ssms, re-open the query.

execute the fillowing.

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON WITH ROLLBACK IMMEDIATE

it will immediately rollback any open transactions before starting the ALTER DATABASE statement.

Remember to disable it when you finished

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT OFF;

\*/

----------------------------------------

--T024\_07\_06\_02

--Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE [Sample] SET READ\_COMMITTED\_SNAPSHOT ON;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

--Get the stock quantity

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Sell the product

SET @AvailableQuantity -= 3;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Billing to Customer

WAITFOR DELAY '00:00:4';

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------------------

--T024\_07\_06\_03

--Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update same data

--ALTER DATABASE [Sample] SET READ\_COMMITTED\_SNAPSHOT ON;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Sell One Product.

SET @AvailableQuantity -= 1;

UPDATE  dbo.Product2

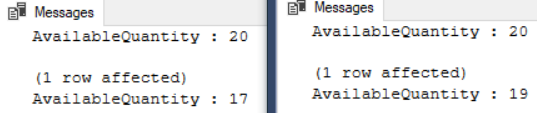
SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch



/\*

1.

1.1.

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update same data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

1.2.

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

has no update conflicts problems, because

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

uses row version to perform whatever ReadCommitted(包括讀提交) can do, plus

whatever ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) can do.

whatever ReadCommitted(包括讀提交) can do is to prevent update conflicts.

1.3.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 17

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 19

Transaction1 and Transaction2 both use

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

Transaction1 will lock the ColumnA and update the ColumnA from value1(20) to value2(17).

Transaction2 will NOT raise Update Conflicts Error.

Transaction2 will wait until Transaction1 commit

and then start to update the ColumnA from value2(17) to value3(19).

Therefore, ColumnA will become value3(19).

\*/

----------------------------------------------------

--T024\_07\_06\_04

--Clean up

ALTER DATABASE [Sample] SET READ\_COMMITTED\_SNAPSHOT OFF;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

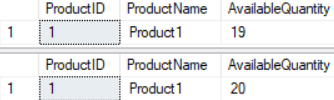
WHERE   ProductID = 1;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch



--------------------------------------------------------------------------------------

7.7. READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)



--===========================================================================

--T024\_07\_07

--READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

/\*

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level :

1st select before update, 2nd select after update

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

\*/

----------------------------------------

--T024\_07\_07\_01

ALTER DATABASE [Sample] SET READ\_COMMITTED\_SNAPSHOT ON;

/\*

Reference:

<https://stackoverflow.com/questions/232333/how-long-should-set-read-committed-snapshot-on-take>

Firstly,

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

Secondly, you may

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

If it take too long to enable SNAPSHOT,

close all ssms session, and re-open ssms, re-open the query.

execute the fillowing.

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON WITH ROLLBACK IMMEDIATE

it will immediately rollback any open transactions before starting the ALTER DATABASE statement.

Remember to disable it when you finished

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT OFF;

\*/

----------------------------------------

--T024\_07\_07\_02

--Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

--Get the stock quantity

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Sell the product

SET @AvailableQuantity -= 3;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Billing to Customer

WAITFOR DELAY '00:00:4';

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------

--T024\_07\_07\_03

--Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level :

--1st select before update, 2nd select after update

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

--Check stock

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Do something

WAITFOR DELAY '00:00:8';

--Check stock again

SELECT  @AvailableQuantity = AvailableQuantity

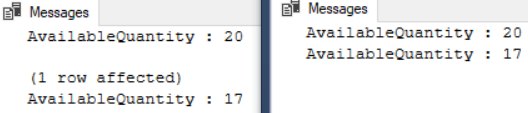
FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch



/\*

1.

1.1.

Transaction1 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update Data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction2 use READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取) level : Update same data

--ALTER DATABASE Sample3 SET READ\_COMMITTED\_SNAPSHOT ON;

--SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

1.2.

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

has no update conflicts problems, because

READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

uses row version to perform whatever ReadCommitted(包括讀提交) can do, plus

whatever ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) can do.

whatever ReadCommitted(包括讀提交) can do is to prevent update conflicts.

1.3.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 17

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 17

Transaction1 READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level will use row version to block all other transactions

to insert/update/delete until it finishes.

Thus, Transaction1 will update

the AvailableQuantity from 20 to 17.

However, Transaction2 READ\_COMMITTED\_SNAPSHOT and ReadCommitted(包括讀提交快照讀取)

isolation level will take the copy version data to read the data.

When Transaction2 read the AvailableQuantity,

Transaction2 only read the committed value.

it was still 20.

Thus, 1st read of Transaction2 returns 20.

After a few seconds, it became 17.

Thus, 2nd read of Transaction2 returns 17.

\*/

----------------------------------------

--T024\_07\_07\_04

--Clean up

ALTER DATABASE [Sample] SET READ\_COMMITTED\_SNAPSHOT OFF;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 17

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

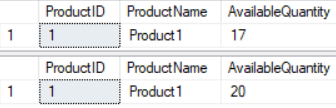
SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

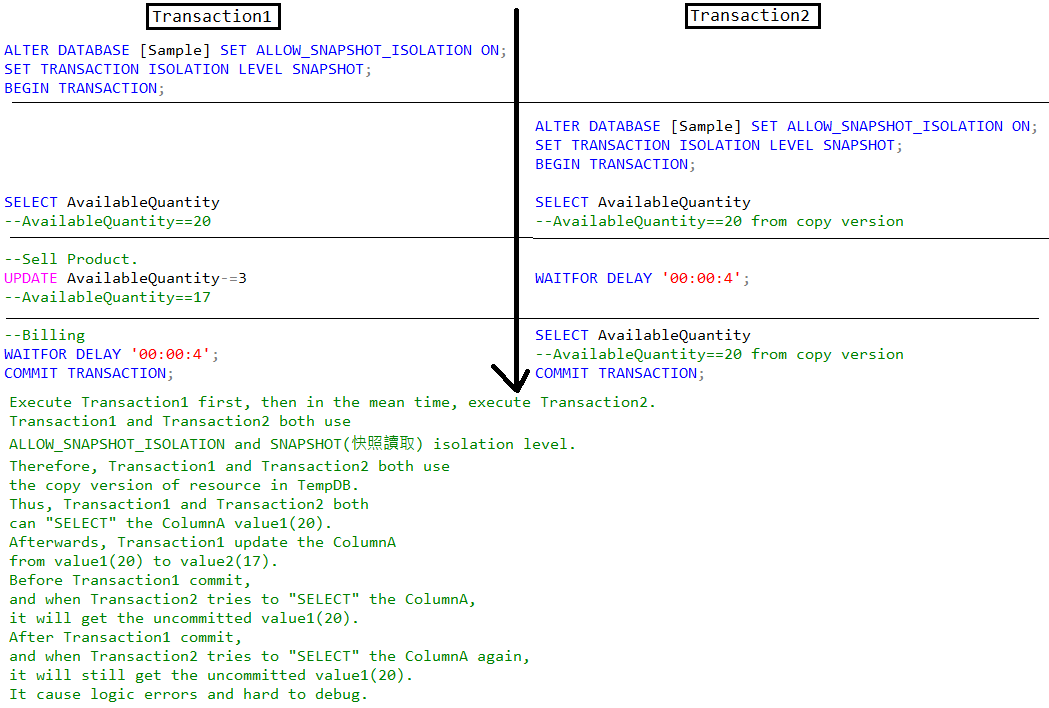
GO -- Run the previous command and begins new batch

--AvailableQuantity : 20



--------------------------------------------------------------------------------------

7.8. ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)



--===========================================================================

--T024\_07\_08

--ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取)

/\*

Transaction1 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Update Data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level :

1st select before update, 2nd select after update

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

This is the logic error and hard to debug.

\*/

----------------------------------------

--T024\_07\_08\_01

ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

/\*

Reference:

<https://stackoverflow.com/questions/232333/how-long-should-set-read-committed-snapshot-on-take>

Firstly,

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

Secondly, you may

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

If it take too long to enable SNAPSHOT,

close all ssms session, and re-open ssms, re-open the query.

execute the fillowing.

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON WITH ROLLBACK IMMEDIATE

it will immediately rollback any open transactions before starting the ALTER DATABASE statement.

Remember to disable it when you finished

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION OFF;

\*/

----------------------------------------

--T024\_07\_08\_02

--Transaction1 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Update Data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

BEGIN TRANSACTION;

--Get the stock quantity

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Sell the product

SET @AvailableQuantity -= 3;

UPDATE  dbo.Product2

SET     AvailableQuantity = @AvailableQuantity

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Billing to Customer

WAITFOR DELAY '00:00:4';

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch

----------------------------------------------------

--T024\_07\_08\_03

--Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level :

--1st select before update, 2nd select after update

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

BEGIN TRANSACTION;

--Check stock

DECLARE @AvailableQuantity INT;

SELECT  @AvailableQuantity = AvailableQuantity

FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

--Do something

WAITFOR DELAY '00:00:8';

--Check stock again

SELECT  @AvailableQuantity = AvailableQuantity

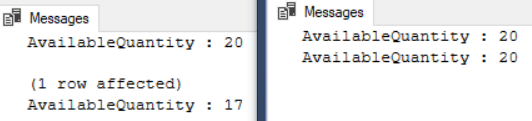
FROM    dbo.Product2

WHERE   ProductID = 1;

PRINT 'AvailableQuantity : ' + CONVERT(NVARCHAR, @AvailableQuantity);

COMMIT TRANSACTION;

GO -- Run the previous command and begins new batch



/\*

1.

1.1.

Transaction1 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level : Update Data

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

Transaction2 use ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) level :

1st select before update, 2nd select after update

--ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION ON;

--SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

1.2.

Execute Transaction1,

then during Transaction1 is still running, execute Transaction2

Transaction1 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 17

Transaction2 [AvailableQuantity] will return ...

--AvailableQuantity : 20

--AvailableQuantity : 20

Execute Transaction1 first, then in the mean time, execute Transaction2.

Transaction1 and Transaction2 both use

ALLOW\_SNAPSHOT\_ISOLATION and SNAPSHOT(快照讀取) isolation level.

Therefore, Transaction1 and Transaction2 both use

the copy version of resource in TempDB.

Thus, Transaction1 and Transaction2 both

can "SELECT" the ColumnA value1(20).

Afterwards, Transaction1 update the ColumnA

from value1(20) to value2(17).

Before Transaction1 commit,

and when Transaction2 tries to "SELECT" the ColumnA,

it will get the uncommitted value1(20).

After Transaction1 commit,

and when Transaction2 tries to "SELECT" the ColumnA again,

it will still get the uncommitted value1(20).

It cause logic errors and hard to debug.

\*/

----------------------------------------------------

--T024\_07\_08\_04

--Clean up

ALTER DATABASE [Sample] SET ALLOW\_SNAPSHOT\_ISOLATION OFF;

SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

--AvailableQuantity : 17

UPDATE  dbo.Product2

SET     [AvailableQuantity] = 20

WHERE   ProductID = 1;

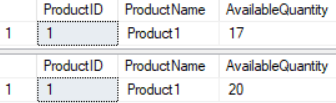
SELECT  \*

FROM    dbo.Product2

WHERE   ProductID = 1;

GO -- Run the previous command and begins new batch

--AvailableQuantity : 20



=====================================================

8. Clean up

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'Product2' ) )

    BEGIN

        DROP TABLE Product2;

    END;

GO -- Run the previous command and begins new batch

IF ( EXISTS ( SELECT    \*

              FROM      INFORMATION\_SCHEMA.TABLES

              WHERE     TABLE\_NAME = 'Person4' ) )

    BEGIN

        TRUNCATE TABLE dbo.Person4;

        DROP TABLE Person4;

    END;

GO -- Run the previous command and begins new batch