Isolation Levels in SQL Server

Sucharita Das, 2019-03-12

SQL Server isolation levels are used to define the degree to which one transaction must be isolated from resource or data modifications made by other concurrent transactions. The different Isolation Levels are:

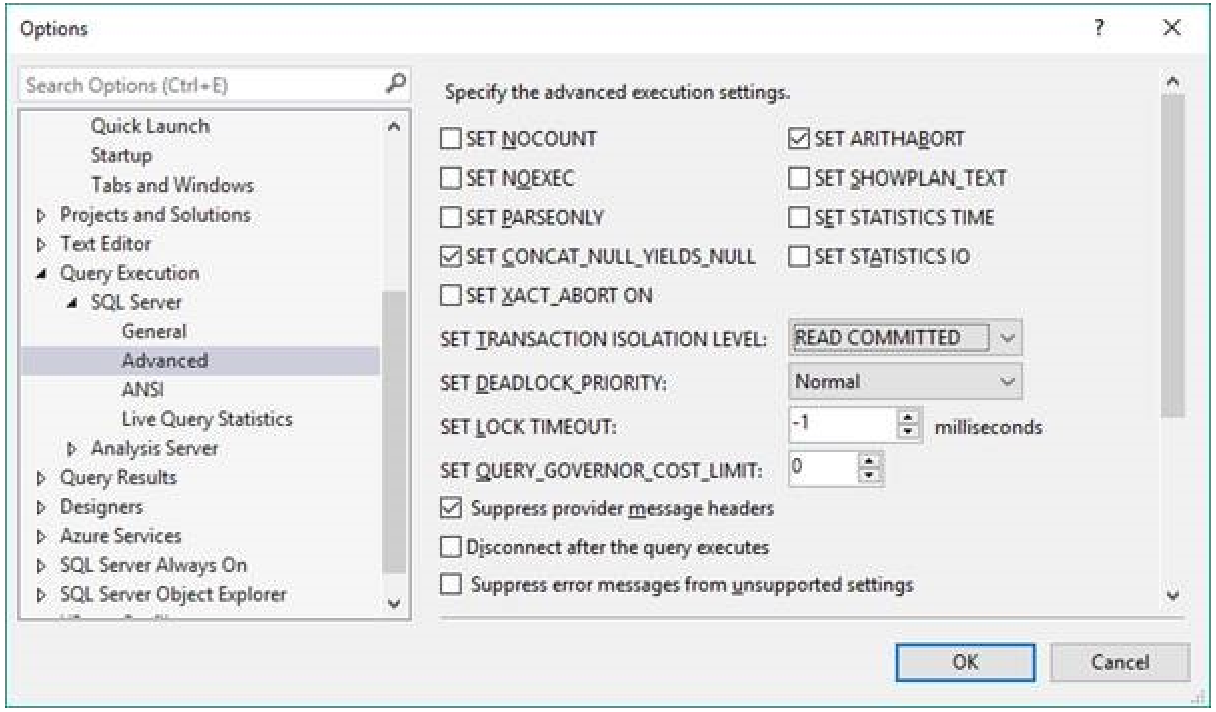
1. Read Uncommitted
2. Read Committed
3. Repeatable Read
4. Serializable
5. Snapshot

Read Committed is the default isolation level. However, it can be changed from Query Window as well as Management Studio Tools.

The syntax is:

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

In SSMS tools menu, Options should be selected. Under Query Execution -> Advanced, then the drop down for Set Transaciton Isolation Level can be modified.



Specify the advanced execution settings.

- [ ] SET NOCOUNT
- [ ] SET NOEXEC
- [ ] SET PARSEONLY
- [x] SET CONCAT_NULL_YIELDS_NULL
- [ ] SET XACT_ABORT ON
- [x] SET ARITHABORT
- [ ] SET SHOWPLAN_TEXT
- [ ] SET STATISTICS TIME
- [ ] SET STATISTICS IO

SET TRANSACTION ISOLATION LEVEL: READ COMMITTED
SET DEADLOCK_PRIORITY: Normal
SET LOCK TIMEOUT: -1 milliseconds
SET QUERY_GOVERNOR_COST_LIMIT: 0

- [x] Suppress provider message headers
- [ ] Disconnect after the query executes
- [ ] Suppress error messages from unsupported settings

Prerequisites

Scripts for sample table creation and data population are detailed below. These tables will be required to demonstrate the working of different isolation levels.

Dept_Exam_StudentMarks.sql:

This script has the table creation and sample data insertion statements for the Dept and Exam tables. This script also has table creation and sample data insertion statements for StudentMarks table. You can add as much sample data you would like. For my setup, the StudentMarks table is initially populated with 2,397,616 records.

```
CREATE TABLE Dept
(
       deptId INT PRIMARY KEY,
       deptName VARCHAR(100),
       deptDesc VARCHAR(300)
)
INSERT INTO Dept
(deptId,deptName,deptDesc)
VALUES
(101,'Computer Science And Engineering','UG and PG Courses in Computer Science And Engineering'),
(102,'Electronics And TeleCommunications','UG and PG Courses in Electronics And TeleCommunications')
CREATE TABLE Exam
(
       examId INT PRIMARY KEY,
       examName VARCHAR(100),
       examDesc VARCHAR(300)
)
INSERT INTO Exam
(examId,examName,examDesc)
VALUES
(201,'Data Structure','Theory Paper and Lab in Data Structure'),
(202,'Database Management System','Theory Paper and Lab in Database Management System')
--create the table
CREATE TABLE StudentMarks
(
       studentId INT IDENTITY(1,1) PRIMARY KEY,
       deptId INT,
       examId INT,
        marksObtained INT
)
SELECT COUNT(1) FROM StudentMarks
--2397616
--Insert record in StudentMarks repeatedly
INSERT INTO StudentMarks (deptId,examId,marksObtained)
VALUES
(101,201,95)
--duplicate the records to increase the count
INSERT INTO StudentMarks (deptId,examId,marksObtained)
SELECT deptId,examId,marksObtained FROM StudentMarks
```

Discussion on Isolation Levels

In this section, we will discuss about the features and limitations of the different isolation levels. The behaviour of two concurrent transactions will be simulated by executing two different scripts from two different user sessions accessing the same resources. The execution output will be different for these concurrent transactions in different isolation levels.

The code for each transaction is located in the files at the end of the article. Comments inside the files denote which query code is being run.

Each of the different isolation levels is shown below.

Read Uncommitted

Transactions running at this level do not issue shared locks to prevent other transactions from modifying data read by the current transaction. Also, transactions are not blocked by exclusive locks at the time of data modification, thus allowing other transactions to read the modified data which is not yet committed.

For this example, the Query2.sql script is executed just after the starting of Query1.sql. In Query1, marksObtained column value is set to 90 in the first update statement. Then, this column value is set to 80 in the third Update statement. At the end of the transaction 1, the value 80 is committed.

Transaction 2 starts after the first Update statement is executed by Transaction1. Transaction 2 reads the value 90. But this is not the committed data. Reading uncommitted modifications is known as a Dirty Read. Once the execution of Query1.sql script is completed, Query2.sql was executed again. This time, it gives output as 80, which is the last committed data by Transaction 1.

The execution of the queries can be arranged as as follows:

Transaction 1 (Query1.sql) is started.

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

Transaction 1 execution continues while transaction 2 (Query2.sql) is started and committed.

```
BEGIN TRANSACTION
SELECT marksObtained
FROM StudentMarks
WHERE deptId = 101 AND examId = 201 AND studentId = 1
COMMIT TRANSACTION
```

Transaction 1 (Query1.sql) is now completed and committed.

```
…
Update Exam
SET examDesc = 'Theory Paper and Lab Assignmnet in Data Structure'
WHERE examId = 201
UPDATE StudentMarks
SET marksObtained = 80
WHERE deptId = 101 AND examId = 201
COMMIT TRANSACTION
```

Read Committed

With Read Committed, transactions issue exclusive locks at the time of data modification, thus not allowing other transactions to read the modified data that is not yet committed. The Read Committed isolation level prevents the Dirty Read issue. However, data can be changed by other transactions between individual statements within the current transaction, resulting in a Non-repeatable Read or a Phantom Row.

The behaviour of READ COMMITTED depends on the setting of the READ_COMMITTED_SNAPSHOT database option. If READ_COMMITTED_SNAPSHOT is set to OFF, shared locks prevent other transactions from modifying rows while the current transaction is running a read operation. The shared locks also block the statement from reading rows modified by other transactions until the other transaction is completed. If READ_COMMITTED_SNAPSHOT is set to ON (the default on SQL Azure Database), row versioning is used to present each statement with a transactional consistent snapshot of the data as it existed at the start of the statement. No locks are issued on the data.

Example 1

In this first example, READ_COMMITTED_SNAPSHOT is set to OFF. The Query2.sql script is executed just after the starting of Query1.sql. In Query1, marksObtained column value is set to 90 in the first update statement. Then, this column value is set to 70 in the third update statement. At the end of the transaction 1, the value 70 is committed.

Until Query1.sql execution is completed, the Query2.sql batch does not return any output. It is in a waiting state. Once transaction 1 is completed, transaction 2 returns the output as 70, i.e. the last committed data.

The execution of the queries was completed as shown below.

Transaction 1 (Query1.sql) is started.

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

Transaction 1 execution continues and transaction 2 (Query2.sql) is started and committed as well.

```
BEGIN TRANSACTION
SELECT marksObtained
FROM StudentMarks
WHERE deptId = 101 AND examId = 201 AND studentId = 1
COMMIT TRANSACTION
```

Transaction 1 (Query1.sql) is completed.

```
…
Update Exam
SET examDesc = 'Theory Paper and Lab Assignmnet in Data Structure'
```

```sql
WHERE examId = 201
UPDATE StudentMarks
SET marksObtained = 70 --80
WHERE deptId = 101 AND examId = 201
COMMIT TRANSACTION
```

Example 2

For this example, READ_COMMITTED_SNAPSHOT is set to ON. The ALTER statement needs to be executed to set the snapshot property with Read Committed Isolation level.

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

Query2.sql script is executed just after starting Query1.sql. In Query1, marksObtained column value is set to 90 in the first UPDATE statement. Then, this column value is set to 80 in the third UPDATE statement. At the end of the transaction 1, the value 80 is committed

The SELECT statement in transaction 2 does not wait for transaction 1 to be committed. It returns the last committed data instantly while Query1.sql is still running. The result was 70 in my test. If Query2.sql is again executed after the completion of Query1.sql, it will retrieve the last committed data.

The execution of the queries is shown below.

Transaction 1 (Query1.sql) is started.

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

Transaction 1 execution continues and transaction 2 (Query2.sql) is started and committed.

```
BEGIN TRANSACTION
SELECT marksObtained
FROM StudentMarks
WHERE deptId = 101 AND examId = 201 AND studentId = 1
COMMIT TRANSACTION
```

Transaction 1 (Query1.sql) is completed.

```
…
Update Exam
SET examDesc = 'Theory Paper and Lab Assignmnet in Data Structure'
WHERE examId = 201
UPDATE StudentMarks
SET marksObtained = 70 --80
WHERE deptId = 101 AND examId = 201
COMMIT TRANSACTION
```

Example 3

In Query3, transaction 1 first retrieves column values for examId 201 from Exam table. After this statement, it executes two UPDATE statements for the StudentMarks table and finally the same SELECT statement to retrieve column values for examId 201 from Exam table is executed again.

The first execution of the SELECT statement gives examDesc value where

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

The Query4.sql script is executed just after the starting of Query3.sql. When the StudentMarks update statments are being executed by transaction 1, transaction 2 started and committed the modified examDesc value for examId 201 where

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

The second execution of the SELECT statement in transaction 1 gives the modified examDesc value. Here, multiple execution of the same select statement within the same transaction give different output data. This issue is known as a Non-Repeatable Read.

The execution of the queries is as follows. First, transaction 1 (Query3.sql) is started.

```sql
BEGIN TRANSACTION
SELECT examId,examName,examDesc
FROM Exam
WHERE examId = 201
…
```

Transaction 1 execution continues. Transaction 2 (Query4.sql) is started and committed.

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

Transaction 1 (Query3.sql) is completed.

```
UPDATE StudentMarks
SET marksObtained = 61
WHERE deptId = 101 AND examId = 201
UPDATE StudentMarks
SET marksObtained = 62
WHERE deptId = 101 AND examId = 201
SELECT examId,examName,examDesc
FROM Exam
WHERE examId = 201
COMMIT TRANSACTION
```

Repeatable Read

In Repeatable Read, statements cannot read data that has been modified but not yet committed by other transactions. No other transaction can modify data that has been read by the current transaction until the current transaction completes.

Shared locks are placed on all data read by each statement in the transaction and are held until the transaction completes. This prevents other transactions from modifying any rows that have been read by the current transaction. This isolation level prevents the Non-Repeatable Read issue.

Other transactions can insert new rows that match the search conditions of statements issued by the current transaction. If the current transaction then retries the statement it will retrieve the new rows, which results in phantom reads.

Example 4

In Query3, transaction 1 retrieves column values for examId 201 from the Exam table. After this statement, the batch executes two UPDATE statements for the StudentMarks table. Finally, the same SELECT statement is executed again.

Query4.sql script is executed just after the start of Query3.sql. While transaction 1 is running, transaction 2 tries to modify examDesc value for examId 201. As shared lock is active untill transaction 1 is committed, transaction 2 needs to wait untill commit.

The first and second executions of the SELECT statement give the same examDesc value as the output and the Non-Repeatable Read issue is resolved.

The execution of the queries is as follows.

Transaction 1 (Query3.sql) is started.

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

Transaction 1 execution continues and transaction 2 (Query4.sql) is started and tries to modify the examDesc value for examId = 201. But, transaction 1 is using this record for reading purpose and the data is locked for updating until transaction 1 commits. So, transaction 2 is in wating state. It can commit the change only when transaction 1 commits or rollbacks.

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

Transaction 1 (Query3.sql) is completed.

```
UPDATE StudentMarks
SET marksObtained = 61
WHERE deptId = 101 AND examId = 201
UPDATE StudentMarks
SET marksObtained = 62
WHERE deptId = 101 AND examId = 201
SELECT examId,examName,examDesc
FROM Exam
WHERE examId = 201
COMMIT TRANSACTION
```

Example 5

In Query5, transaction 1 retrieves Exam column values for examName = ‘Data Structure’. After this statement, it executes two UPDATE statements for StudentMarks table and finally the same SELECT statement to retrieve Exam column values for examName = ‘Data Structure’ is executed again.

Once the first SELECT statement execution is completed, transaction 2 starts (Query6.sql). This transaction inserts a new record into Exam table for examName = 'Data Structure' and commits the change. Here, shared lock is active on the records selected in the SELECT statement of Query5.sql untill transaction 1 is committed, but the new record insertion with the same search condition is not prevented. As a result, Transaction 2 adds a new record in Exam table.

The second occurance of the SELECT statement in transaction 1 will now retrieve an extra record. The same SELECT statement within the same transaction is giving some new rows for the second execution as compared to the first one. These extra rows are known as Phantom Rows.

The execution of the queries can be arranged as per their start time and end time, as follows:

Transaction 1 (Query5.sql) is started.

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

Transaction 1 Execution continues and Transaction 2 (Query6.sql) is started and committed.

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

Transaction 1 (Query5.sql) is completed.

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

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

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

Serializable

In the serializable isolation level, statements cannot read data that has been modified but not yet committed by other transactions. No other transactions can modify data that has been read by the current transaction until the current transaction completes. Other transactions cannot insert new rows with key values that would fall in the range of keys read by any statements in the current transaction until the current transaction completes.

Example 6

In Query5, transaction 1 retrieves Exam column values for examName = 'Data Structure'. After this statement, it executes two UPDATE statements for StudentMarks table and finally the same SELECT statement to retrieve Exam column values for examName = 'Data Structure' is executed again.

Once the first SELECT statement execution is completed, transaction 2 starts (Query8.sql). This transaction tries to insert a new record into Exam table for examName = 'Data Structure' and commit the change. But, transaction 2 is prevented to insert a new record in the Exam table with the same key value as used in the serach criteria of the SELECT statement used in transaction 1 which is not yet committed. Thus, transaction 2 has to wait to commit the changes until the transaction is completed.

The second occurance of the SELECT statement in transaction 1 will now retrieve the same number of records as retrieved by the first occurance of the SELECT statement. Thus, Phantom Rows issue is resolved.

The execution of the queries can be arranged as per their start time and end time, as follows:

Transaction 1 (Query5.sql) is started.

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

Transaction 1 execution continues and transaction 2 (Query8.sql) is started. Transaction 2 has to wait to commit its changes until transaction 1 completes.

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

Transaction 1 (Query5.sql) is completed.

```
UPDATE StudentMarks
SET marksObtained = 61
WHERE deptId = 101 AND examId = 201
UPDATE StudentMarks
SET marksObtained = 62
WHERE deptId = 101 AND examId = 201
SELECT examId,examName,examDesc
FROM Exam
WHERE examName = 'Data Structure'
COMMIT TRANSACTION
```

Snapshot Isolation

In snapshot isolation, data read by any statement in a transaction will be the transactional consistent version of the data that existed at the start of the transaction. Data modifications made by other transactions after the start of the current transaction are not visible to statements executing in the current transaction. SNAPSHOT transactions do not request locks when reading data. SNAPSHOT transactions reading data do not block other transactions from writing data. Transactions writing data do not block SNAPSHOT transactions from reading data.

The ALLOW_SNAPSHOT_ISOLATION database option must be set to ON before starting a transaction with SNAPSHOT isolation level.

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

The READ_COMMITTED_SNAPSHOT database option determines the behaviour of the default READ COMMITTED isolation level when snapshot isolation is enabled in a database.

If READ_COMMITTED_SNAPSHOT database option is set to ON, the database engine uses row versioning and snapshot isolation as the default, instead of using locks to protect the data.

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

The Significance of Different Isolation Levels

Only one of the isolation level options can be set at a time, and it remains set for that connection until it is explicitly changed. A lower isolation level increases the ability of many users to access data at the same time, but increases the number of concurrency effects, such as dirty reads or lost updates etc. Conversely, a higher isolation level reduces the types of concurrency effects that users might encounter, but requires more system resources and increases the chances that one transaction will block another.

The lowest isolation level, read uncommitted, can retrieve data that has been modified but not committed by other transactions. All concurrency side effects can happen in read uncommitted, but there's no read locking or versioning, so overhead is minimized.

READ COMMITTED is the default isolation level for SQL Server. It prevents dirty reads by specifying that statements cannot read data values that have been modified but not yet committed by other transactions. If the READ_COMMITTED_SNAPSHOT option is set as ON, the Read transactions need not wait and can access the last committed records. Other transactions can modify, insert, or delete data between executions of individual SELECT statements within the current transaction, resulting in non-repeatable reads or phantom rows.

REPEATABLE READ is a more restrictive isolation level than READ COMMITTED. It encompasses READ COMMITTED and additionally specifies that no other transactions can modify or delete data that has been read by the current transaction until the current transaction commits. Concurrency is lower than for READ COMMITTED because shared locks on read data are held for the duration of the transaction instead of being released at the end of each statement. But Other transactions can insert data between executions of individual SELECT statements within the current transaction, resulting in phantom rows.

The highest isolation level, serializable, guarantees that a transaction will retrieve exactly the same data every time it repeats a read operation, but it does this by performing a level of locking that is likely to impact other users in multi-user systems.

SNAPSHOT isolation specifies that data read within a transaction will never reflect changes made by other simultaneous transactions. The transaction uses the data row versions that exist when the transaction begins. No locks are placed on the data when it is read, so SNAPSHOT transactions do not block other transactions from writing data. Transactions that write data do not block snapshot transactions from reading data. If no waiting is acceptable for the SELECT operation but the last committed data is enough to be displayed, this isolation level may be appropriate.

The scripts for tables are attached below.

Resources

- Dept_Exam_StudentMarks.sql
- Transaction1_Queries.sql
- Transaction2_Queries.sql