Acquire Locks by Using the Read Committed Isolation Level

In this exercise, you execute the same type of transactions as in the previous exercise, but use the read committed snapshot transaction isolation level.

1. Open Microsoft SQL Server Management Studio, and connect to an instance of SQL Server.
2. In a new query window, which will be referred to as Connection 1, type and execute the following SQL statements to create the *TestDB* database, the Test schema, and the table that you will use in this exercise:

```
-- Connection 1 – Session ID: <put @@SPID result here>
/* Leave the above line to easily see that this query window
belongs to Connection 1. */
SELECT @@SPID;
GO
CREATE DATABASE TestDB;
GO
USE TestDB;
GO
CREATE SCHEMA Test;
GO
CREATE TABLE Test.TestTable (
Col1 INT NOT NULL
,Col2 INT NOT NULL
);
INSERT Test.TestTable (Col1, Col2) VALUES (1,10);
INSERT Test.TestTable (Col1, Col2) VALUES (2,20);
INSERT Test.TestTable (Col1, Col2) VALUES (3,30);
INSERT Test.TestTable (Col1, Col2) VALUES (4,40);
INSERT Test.TestTable (Col1, Col2) VALUES (5,50);
INSERT Test.TestTable (Col1, Col2) VALUES (6,60);
```

3. Open another query window, which will be referred to as Connection 2, and type and execute the following SQL statement to prepare the connection:

```
-- Connection 2 – Session ID: <put @@SPID result here>
/* Leave the above line to easily see that this query window
belongs to Connection 2. */
SELECT @@SPID;
GO
USE TestDB;
```

4. Open a third query window, which will be referred to as Connection 3, and type and execute the following SQL statement to prepare the connection:

```
-- Connection 3
/* Leave the above line to easily see that this query window
belongs to Connection 3. */
USE TestDB;
```

5. In Connection 1, execute the following SQL statements to start a transaction in the read committed transaction isolation level, and read a row from the test table (but do not commit the transaction!).

```
-- Connection 1
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN TRAN;
SELECT * FROM Test.TestTable
WHERE Col1 = 1;
```

6. To see which locks have been acquired by the transaction in Connection 1, open Connection 3, and execute the following SELECT statement. In the line of code that contains *@@SPID of Connection 1*>, be sure to replace this with the ID value returned by the code executed in step 2 of this exercise.

```
SELECT
resource_type
,request_mode
,request_status
FROM sys.dm_tran_locks
WHERE resource_database_id = DB_ID('TestDB')
AND request_session_id = <@@SPID of Connection 1>
AND request_mode IN ('S', 'X')
AND resource_type <> 'DATABASE';
```

7. Why doesn't Connection 1 have a shared lock on the row that it read using the SELECT statement?
   In Connection 1, execute the following SQL statement to end the started transaction:

```
-- Connection 1
COMMIT TRAN;
```

8. In Connection 2, execute the following SQL statements to start a transaction, and acquire an exclusive lock on one row in the test table.

```
-- Connection 2
BEGIN TRAN;
UPDATE Test.TestTable SET Col2 = Col2 + 1
WHERE Col1 = 1;
```

9. In Connection 1, execute the following transaction to try to read the row that has been updated (but not committed) by Connection 2. After you execute the code in this step, move on to the next step, as this connection will now be blocked.

```
-- Connection 1
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN TRAN;
SELECT * FROM Test.TestTable
WHERE Col1 = 1;
```

```
-- This SELECT statement will be blocked!
```

10. To see which locks have been acquired by the transaction in Connection 1, open Connection 3, and execute the following SELECT statement. In the line of code that contains *@@SPID of Connection 1*, be sure to replace this with the ID value returned by the code executed in step 2 of this exercise.

```
SELECT resource_type, request_mode, request_status
FROM sys.dm_tran_locks
WHERE resource_database_id = DB_ID('TestDB')
AND request_session_id = <@@SPID of Connection 1>
AND request_mode IN ('S', 'X')
AND resource_type <> 'DATABASE';
```

11. Here you can see that Connection 1 tries to acquire a shared lock on the row.
    In Connection 2, execute the following SQL statements to end the transaction started earlier.

```
-- Connection 2
COMMIT TRAN;
```

12. Now, first have a look in Connection 1 and note that the SELECT statement has been completed. Switch to Connection 3, and execute its SELECT statement again to see which locks are now acquired by the transaction in Connection 1. In the line of code that contains *@@SPID of Connection 1*>, be sure to replace this with the ID value returned by the code executed in step 2 of this exercise.

```
SELECT resource_type, request_mode, request_status
FROM sys.dm_tran_locks
WHERE resource_database_id = DB_ID('TestDB')
AND request_session_id = <@@SPID of Connection 1>
AND request_mode IN ('S', 'X')
AND resource_type <> 'DATABASE';
```

13. You should now see that no locks are acquired by Connection 1. This is because, after acquiring the lock on the row, Connection 1 released the lock.
    Close the three query windows for Connections 1, 2, and 3. Open a new query window, and execute the following SQL statement to clean up after this exercise:

```
USE master;
DROP DATABASE TestDB;
```