**Gateway Technical College**

DATABASES

152-080

Unit 13 Assignment

# Introduction

In this lab you will create transactions and locks. In order to do this assignment, you will need to run SQL statements against the **LocksDB** database.

This assignment has two parts:

Exercise 1: Acquire locks by using the Read Committed Isolation Level

Exercise 2: Acquire locks by using the Read Committed Snapshot Isolation Level

MAKE SURE document your work and your commands work before you past them into the document.

Once completed, attach this completed word document to this assignment for grading.

Use the **Discussion Forum** if you have any questions regarding the how to approach this assignment. You can also email your instructor directly for assistance if you have any questions.

Save your submission as ***lastnameFirstname\_assign13.docx*** and submit it in the unit *Apply* section of the course.

# Instructions

You are to perform the following two exercises. Make sure to place print screens of your work into this document as well as answer any question in the exercise. For each question below – paste in print screens of your progress in each step.

# Exercise 1: Acquire locks by using the Read Committed Isolation Level

In this exercise, you execute some transactions using the read committed transaction isolation level. Open Microsoft SQL Server Management Studio and complete the following instructions.

1. In a new query window, which will be referred to as **Connection 1**, type and execute the following SQL statements to create the ***LocksDB*** database and the table that you will use in this exercise:

\*\* Make sure to record **@@SPID** which you will use later in another step\*\*\*\*\*

-- 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 LocksDB;

GO

USE LocksDB;

GO

CREATE TABLE LocksTable (

Col1 INT NOT NULL,

Col2 INT NOT NULL

);

INSERT LocksTable (Col1, Col2) VALUES (1,10);

INSERT LocksTable (Col1, Col2) VALUES (2,20);

INSERT LocksTable (Col1, Col2) VALUES (3,30);

INSERT LocksTable (Col1, Col2) VALUES (4,40);

INSERT LocksTable (Col1, Col2) VALUES (5,50);

INSERT LocksTable (Col1, Col2) VALUES (6,60);

1. Open another query window, which will be referred to as **Connection 2**, and type and execute the following SQL statement to prepare the connection:   
     
   \*\* Make sure to record **@@SPID** which you will use later in another step\*\*\*\*\*

-- 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 LocksDB;

1. 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 LocksDB;

1. 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 LocksTable

WHERE Col1 = 1;

1. 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('LocksDB')

AND request\_session\_id = <@@SPID of Connection 1> --replace id number--

AND request\_mode IN ('S', 'X')

AND resource\_type <> 'DATABASE';

Why doesn’t **Connection 1** have a shared lock on the row that it read using the SELECT statement?

|  |
| --- |
| Since they are in different connections |

1. In **Connection 1**, execute the following SQL statement to end the started transaction:

Connection 1

COMMIT TRAN;

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

-- Connection 2

BEGIN TRAN;

UPDATE LocksTable SET Col2 = Col2 + 1

WHERE Col1 = 1;

1. 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 LocksTable

WHERE Col1 = 1;

-- This SELECT statement will be blocked!

1. 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('LocksDB')

AND request\_session\_id = <@@SPID of Connection 1> --replace id number--

AND request\_mode IN ('S', 'X')

AND resource\_type <> 'DATABASE';

Here you can see that **Connection 1** tries to acquire a shared lock on the row.

1. In **Connection 2**, execute the following SQL statements to end the transaction started earlier.

Connection 2

COMMIT TRAN;

1. 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('LocksDB')

AND request\_session\_id = <@@SPID of Connection 1> --replace id number--

AND request\_mode IN ('S', 'X')

AND resource\_type <> 'DATABASE';

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.

1. 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 LocksDB;

# Exercise 2: Acquire locks by using the Read Committed Snapshot 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. In a new query window, which will be referred to as **Connection 1**, type and execute the following SQL statements to create the ***LocksDB*** database and the table that will be used in this exercise:

-- Connection 1

/\* Leave the above line to easily see that this query window

belongs to Connection 1. \*/

CREATE DATABASE LocksDB;

GO

ALTER DATABASE LocksDB SET READ\_COMMITTED\_SNAPSHOT ON;

GO

USE LocksDB;

GO

CREATE TABLE LocksTable (

Col1 INT NOT NULL,

Col2 INT NOT NULL

);

INSERT LocksTable (Col1, Col2) VALUES (1,10);

INSERT LocksTable (Col1, Col2) VALUES (2,20);

INSERT LocksTable (Col1, Col2) VALUES (3,30);

INSERT LocksTable (Col1, Col2) VALUES (4,40);

INSERT LocksTable (Col1, Col2) VALUES (5,50);

INSERT LocksTable (Col1, Col2) VALUES (6,60);

1. 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

/\* Leave the above line to easily see that this query window

belongs to Connection 2. \*/

USE LocksDB;

1. 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 LocksDB;

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

-- Connection 2

BEGIN TRAN;

UPDATE LocksTable SET Col2 = Col2 + 1

WHERE Col1 = 1;

1. In **Connection 1**, execute the following transaction to try to read the row that has been updated (but not committed) by **Connection 2**.

-- Connection 1

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRAN;

SELECT \* FROM LocksTable

WHERE Col1 = 1;

Why wasn’t the SELECT statement blocked by Connection 2?

|  |
| --- |
| Since they are both using a unified connection |

Which values were returned by the query, the values that existed before or after the update?

|  |
| --- |
| 1 and 10 |

1. To see which locks have been acquired by the transaction in Connections 1 and 2, open Connection 3, and execute the following SELECT statement:

SELECT

resource\_type,

request\_mode,

request\_status

FROM sys.dm\_tran\_locks

WHERE resource\_database\_id = DB\_ID('LocksDB')

AND request\_mode IN ('S', 'X')

AND resource\_type <> 'DATABASE';

1. To see if any row versions are available for the ***LocksDB*** database, execute the following query in Connection 3:

SELECT \* FROM sys.dm\_tran\_version\_store

WHERE database\_id = DB\_ID('LocksDB');

1. In Connection 2, execute the following SQL statements to end the transaction started earlier.

Connection 2

COMMIT TRAN;

1. In the open transaction in **Connection 1**, execute the SELECT statement again.

Connection 1

SELECT \* FROM LocksTable

WHERE Col1 = 1;

Which values are now returned, the values that existed before or after the update?

|  |
| --- |
| 1 and 10 |

Did this SELECT statement return dirty reads?

|  |
| --- |
| no |

Did the first SELECT statement in Connection 1 return dirty reads?

|  |
| --- |
| No |

1. Close the three query windows for Connection 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 LocksDB;

What were the main points you saw demonstrated in these two exercises?

|  |
| --- |
| How you can use limited connections in order to restrict access to manipulate and view certain data |