## Murach Chapter 16 Introduction to Concurrency

For all the following exercises make sure you are connected to the db2 database by running the following code.

use db2;

go

Exercise 1: Blocking

1.1: Open three connections, (call them Connection 1, Connection 2 and Connection 3). Run the following code in Connection 1 to update rows in Sales.OrderDetails:

--connection 1

use db2;

go

BEGIN TRAN;

UPDATE Sales.OrderDetails

SET discount = 0.05

WHERE orderid = 10249;

Query the database, in connection 1:

--connection 1

use db2;

go

SELECT orderid, productid, unitprice, qty, discount FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste results of connection 1 here:

1-2: Run the following code in Connection 2 to query Sales.OrderDetails:

--connection 2

use db2;

go

SELECT orderid, productid, unitprice, qty, discount FROM Sales.OrderDetails

WHERE orderid = 10249;

Connection 2 will be blocked:

Paste results of connection 2 here:

1-3: Run the following code in Connection 3 and identify the locks and process ids

involved in the blocking chain:

--connection 3

use db2;

go

SELECT

request\_session\_id AS sid,

resource\_type AS restype,

resource\_database\_id AS dbid,

resource\_description AS res,

resource\_associated\_entity\_id AS resid,

request\_mode AS mode,

request\_status AS status

FROM sys.dm\_tran\_locks;

Paste results of connection 3 here:

1-4: In the following exercises replace the session ids 52, 58 with the ones you found to be involved in the blocking chain in the previous exercise. Run the following code in connection 3 to obtain connection/session/blocking information about the processes involved in the blocking chain:

use db2;

go

-- Connection info

SELECT

session\_id AS sid,

connect\_time,

last\_read,

last\_write,

most\_recent\_sql\_handle

FROM sys.dm\_exec\_connections

WHERE session\_id IN(52, 58);

-- Session info

SELECT

session\_id AS sid,

login\_time,

host\_name,

program\_name,

login\_name,

nt\_user\_name,

last\_request\_start\_time,

last\_request\_end\_time

FROM sys.dm\_exec\_sessions

WHERE session\_id IN(52, 58);

-- Blocking

SELECT

session\_id AS sid,

blocking\_session\_id,

command,

sql\_handle,

database\_id,

wait\_type,

wait\_time,

wait\_resource

FROM sys.dm\_exec\_requests

WHERE blocking\_session\_id > 0;

Paste results of connection 3 here:

1-5: Run the following code to obtain the SQL text of the connections involved in the blocking chain, again replace the session ids 52, 58 with the ones you used in the previous exercises:

use db2;

go

SELECT session\_id, text

FROM sys.dm\_exec\_connections

CROSS APPLY sys.dm\_exec\_sql\_text(most\_recent\_sql\_handle) AS ST

WHERE session\_id IN(52, 58);

Paste results of connection 3 here:

1-6: Run the following code in Connection 1 to rollback the transaction, before executing observe in Connection 2 that the SELECT query still trying to execute.

--connection 1

use db2;

go

ROLLBACK TRAN;

Observe in Connection 2 that the SELECT query returned the two order detail rows, and that those rows were not modified.

Paste results of connection 2 here:

Close all connections.

Exercise 2: Isolation Levels

***2-1: The READ UNCOMMITTED Isolation Level***

2-1a: Open two new connections (call them Connection 1 and Connection 2). Run the following code in Connection 1 to query the table Sales.OrderDetails:

--Connection 1

use db2;

go

SELECT orderid, productid, unitprice, qty, discount FROM Sales.OrderDetails WHERE orderid = 10249;

Paste results of connection 1 here:

2-1b: Run the following code in Connection 1 to update rows in Sales.OrderDetails and query it:

--Connection 1

use db2;

go

BEGIN TRAN;

UPDATE Sales.OrderDetails SET discount += 0.05 WHERE orderid = 10249;

Paste messages of connection 1 here:

Run the following code in Connection 1 to query Sales.OrderDetails:

--Connection 1

use db2;

go

SELECT orderid, productid, unitprice, qty, discount FROM Sales.OrderDetails WHERE orderid = 10249;

Paste results of connection 1 here:

2-1c: Run the following code in Connection 2 to set the isolation level to READ UNCOMMITTED and query Sales.OrderDetails:

--connection 2

use db2;

go

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste results of connection 2 here:

Notice that you get the modified, uncommitted version of the rows.

2-1d: Run the following code in Connection 1 to rollback the transaction:

--connection 1

use db2;

go

ROLLBACK TRAN;

***2-2: The READ COMMITTED Isolation Level***

2-2a: Run the following code in Connection 1 to update rows in Sales.OrderDetails and query it:

--connection 1

use db2;

go

BEGIN TRAN;

UPDATE Sales.OrderDetails SET discount += 0.05 WHERE orderid = 10249;

Paste messages of connection 1 here:

Run the following code in Connection 1 to query Sales.OrderDetails:

SELECT orderid, productid, unitprice, qty, discount FROM Sales.OrderDetails WHERE orderid = 10249;

Paste results of connection 1 here:

2-2b: Run the following code in Connection 2 to set the isolation level to READ COMMITTED and query Sales.OrderDetails:

--connection 2

use db2;

go

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste results of connection 2 here:

What are you looking at:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2-2c: Run the following code in Connection 1 to commit the transaction

--connection 1

use db2;

go

COMMIT TRAN;

2-2d: Go to connection 2 and noticed that you got the modified, committed version of the rows:

Paste results of connection 2 here:

2-2e: Run the following code for cleanup:

--connection 1

use db2;

go

UPDATE Sales.OrderDetails SET discount = 0.00 WHERE orderid = 10249;

***2-3 The REPEATABLE READ Isolation Level***

2-3a: Run the following code in Connection 1 to set the isolation to REPEATABLE READ, open a transaction and read data from Sales.OrderDetails:

--connection 1

use db2;

go

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

BEGIN TRAN;

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste results of connection 1 here:

You get two rows with discount values 0.00

2-3b: Run the following code in Connection 2 and notice you are blocked:

--connection 2

use db2;

go

UPDATE Sales.OrderDetails

SET discount += 0.05

WHERE orderid = 10249;

Paste results of connection 2 here:

2-3c: Run the following code in Connection 1 to read the data again and commit the transaction:

--connection 1

use db2;

go

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

COMMIT TRAN;

Paste results of connection 1 here:

You will get the two rows with discount values 0.00 again, giving you repeatable reads. Note that if your code was running under a lower isolation level (READ UNCOMMITTED or READ COMMITTED), the UPDATE statement wouldn’t have been blocked, and you would have gotten non repeatable reads.

2-3d: Go to Connection 2 and notice that the update finished.

Paste results of connection 2 here:

2-3e: Run the following code for cleanup:

--connection 1

use db2;

go

UPDATE Sales.OrderDetails

SET discount = 0.00

WHERE orderid = 10249;

***2-4 The SERIALIZABLE Isolation Level***

2-4a: Run the following code in Connection 1 to set the isolation to SERIALIZABLE and query Sales.OrderDetails:

--connection 1

use db2;

go

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRAN;

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste results of connection 1 here:

2-4b: Run the following code in Connection 2 to attempt to insert a row to Sales.OrderDetails with the same order ID that is filtered by the previous query and notice that you are blocked:

--connection 2

use db2;

go

INSERT INTO Sales.OrderDetails

(orderid, productid, unitprice, qty, discount)

VALUES(10249, 2, 19.00, 10, 0.00);

Paste messages of connection 2 here:

Note that in lower isolations (READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ), this INSERT statement wouldn’t have been blocked.

2-4c: Run the following code in Connection 1 to query Sales.OrderDetails again, and commit the transaction:

--connection 1

use db2;

go

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

COMMIT TRAN;

Paste results of connection 1 here:

You get the same result set as you got from the previous query in the same transaction, and since the INSERT statement was blocked, you got no phantom reads.

2-4d: Go back to Connection 2 and notice that the INSERT statement finished:

Paste messages of connection 2 here:

2-4e: Run the following code for cleanup in connection 1:

--connection 1

use db2;

go

DELETE FROM Sales.OrderDetails

WHERE orderid = 10249 AND productid = 2;

2-4f: Run the following code in both Connection 1 and Connection 2

to set the isolation level to the default:

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

***2-5 The SNAPSHOT Isolation Level***

2-5a: Run the following code to allow SNAPSHOT isolation (in Azure SQL Database ON by default) in the db2 database:

--connection 1

use db2;

go

ALTER DATABASE db2 SET ALLOW\_SNAPSHOT\_ISOLATION ON;

2-5b: Run the following code in Connection 1 to open a transaction, update rows in Sales.OrderDetails and query it:

--connection 1

use db2;

go

BEGIN TRAN;

UPDATE Sales.OrderDetails

SET discount += 0.05

WHERE orderid = 10249;

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste results of connection 1 here:

2-5c: Run the following code in Connection 2 to set the isolation to SNAPSHOT and query Sales.OrderDetails Notice that you're not blocked, rather you get an earlier consistent version of the data that was available when the transaction started (discount values 0.00):

--connection 2

use db2;

go

SET TRANSACTION ISOLATION LEVEL SNAPSHOT;

BEGIN TRAN;

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste messages of connection 2 here:

Paste results of connection 2 here:

2-5d: Go to Connection 1 and commit the transaction:

--connection 1

use db2;

go

COMMIT TRAN;

2-5e: Go to Connection 2 and query the data again; notice that you get discount values 0.50:

--connection 2

use db2;

go

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste results of connection 2 here:

2-5f: In Connection 2 commit the transaction and query the data again; notice that now you get discount values 0.05:

--connection 2

use db2;

go

COMMIT TRAN;

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste results of connection 2 here:

2-5g: Run the following code for cleanup:

--connection 1

use db2;

go

UPDATE Sales.OrderDetails

SET discount = 0.00

WHERE orderid = 10249;

Close all connections

***2-6 The READ COMMITTED SNAPSHOT Isolation Level***

2-6a: Open two new connections (call them Connection 1 and Connection 2).

2-6b: Turn on READ\_COMMITTED\_SNAPSHOT in the db2 database: (in Azure SQL Database ON by default):

--connection 1

use db2;

go

ALTER DATABASE db2 SET READ\_COMMITTED\_SNAPSHOT ON;

2-6c: Run the following code in Connection 1 to open a transaction and update rows in Sales.OrderDetails:

--connection 1

use db2;

go

BEGIN TRAN;

UPDATE Sales.OrderDetails

SET discount += 0.05

WHERE orderid = 10249;

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste results of connection 1 here:

2-6d: Run the following code in Connection 2 which is now running under the isolation READ COMMITTED SNAPSHOT since the database flag READ\_COMMITTED\_SNAPSHOT is turned on Notice that you're not blocked, rather you get an earlier consistent version of the data that was available when the statement started (discount values 0.00):

--connection 2

use db2;

go

BEGIN TRAN;

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste messages of connection 2 here:

Paste results of connection 2 here:

2-6e: Go to Connection 1 and commit the transaction:

--connection 1

use db2;

go

COMMIT TRAN;

2-6f: Go to Connection 2 and query the data again and commit the transaction; notice that you get the new discount values 0.05:

--connection 2

use db2;

go

SELECT orderid, productid, unitprice, qty, discount

FROM Sales.OrderDetails

WHERE orderid = 10249;

Paste results of connection 2 here:

--connection 2

use db2;

go

COMMIT TRAN;

Paste messages of connection 2 here:

2-6g:

Run the following code for cleanup in connection 1:

--connection 1

use db2;

go

UPDATE Sales.OrderDetails

SET discount = 0.00

WHERE orderid = 10249;

Close all connections

2-6h: Open a new connections, change the database flags back to the defaults, disabling snapshot isolations:

use db2;

go

ALTER DATABASE db2 SET ALLOW\_SNAPSHOT\_ISOLATION OFF;

ALTER DATABASE db2 SET READ\_COMMITTED\_SNAPSHOT OFF;