# Name: Mi Gao

# Overview of the Assignment:

In this exercise we will simulate a deadlock.

# Required Software

This assignment is designed to be completed in modern versions of Oracle and Microsoft SQL Server as is. If you would like to use a different RDBMS, please check with your facilitator for approval. You may need to modify the SQL for successful execution, though the SQL should execute as is if your RDBMS is ANSI compliant. To create a deadlock we will need two or more sessions connected to the same database. You will also need a transactional database, so MySQL with the default MyISAM back end or MS Access won’t work. Most other RDBMS are transactional.

Note that if you are using Oracle, you are not required to use Oracle SQL Developer, however it is recommended for question 6. There are many capable SQL clients that connect to Oracle. You will have to find a compatible way to monitor sessions.

**To complete the assignment, please provide screenshots of the following:**

# Directions:

1. Create two sessions connected as the same user to a transactional DBMS.

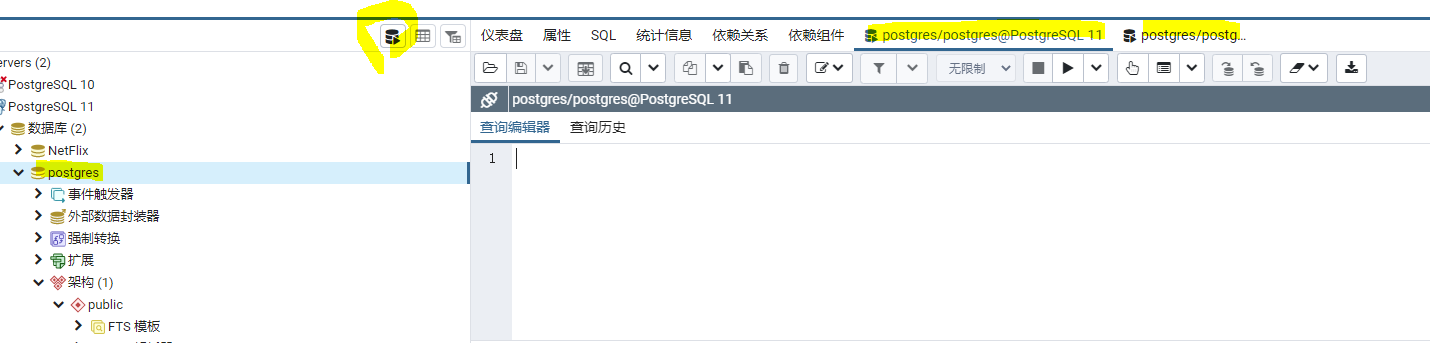
* **SQL Server Note**: you will need to open two separate query windows instead of two separate sessions.

**For this assignment, I used two different tools because pgAdmin4 do not have the clear GUI feature to show details about the deadlock. Thus, I downloaded SQL Server 2019 and SSMS 18 for it.**

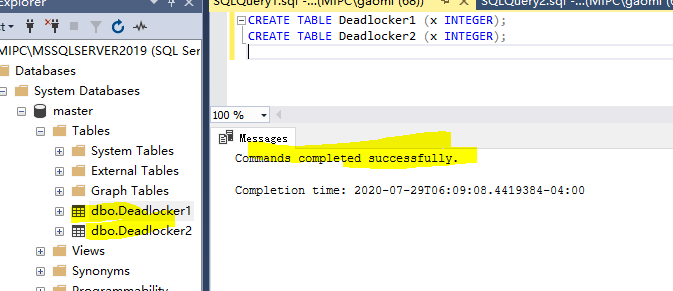
In pgAdmin4:



Then open two query tools.



In SSMS 18:

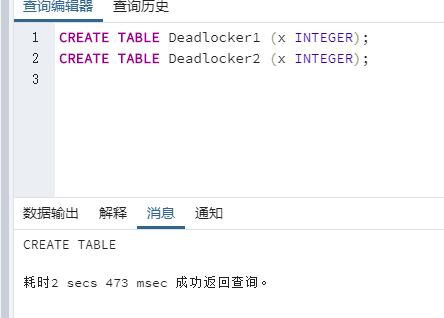


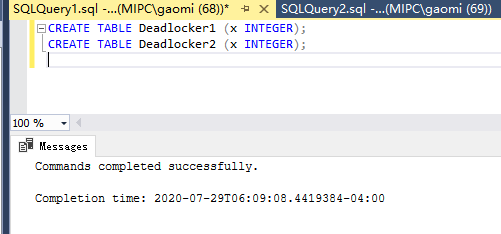
1. Create two tables :

CREATE TABLE Deadlocker1 (x INTEGER);

CREATE TABLE Deadlocker2 (x INTEGER);

[Take a screenshot to show your results and paste it here.]





1. Insert two rows in the tables with different data values.

INSERT INTO Deadlocker1 VALUES(1);

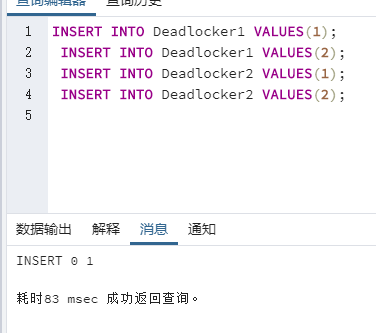
INSERT INTO Deadlocker1 VALUES(2);

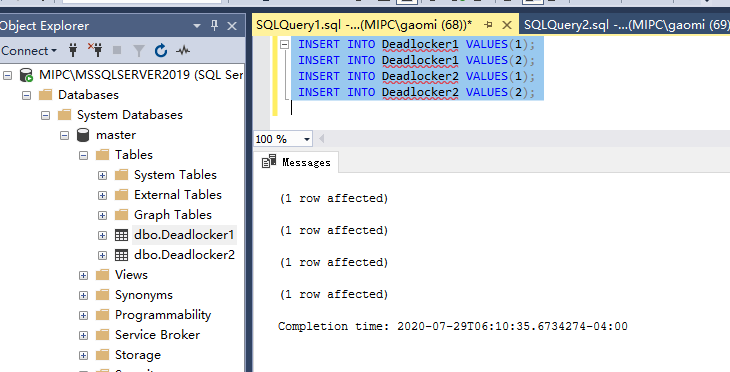
INSERT INTO Deadlocker2 VALUES(1);

INSERT INTO Deadlocker2 VALUES(2);

* **Note for Oracle and other RDBMS with autocommit off:** Commit your transaction.

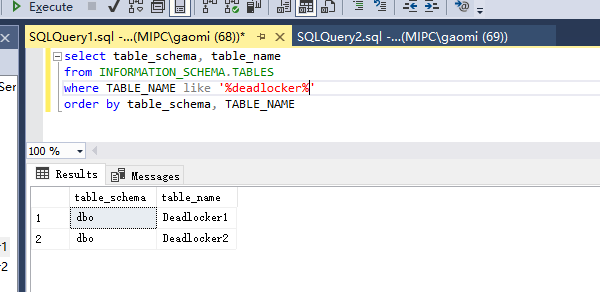
pgAdmin4 with autocommit:





Optionally check the data in the two tables by doing a simple select.



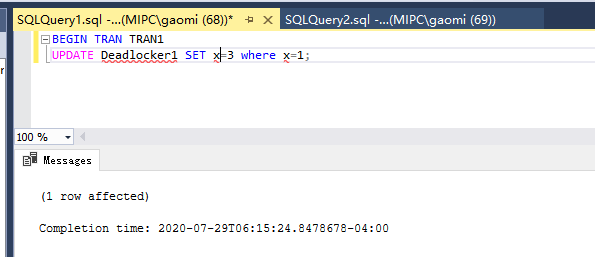


1. In the first session start a transaction by updating the first Deadlocker1 table. This procures an exclusive lock on the Dedlocker1 table, on the row with x=1.

UPDATE Deadlocker1 SET x=3 where x=1;

* **SQL Server Note**: you will need to use transactions to create a deadlock. To do this before the update add line **BEGIN TRAN TRAN1**





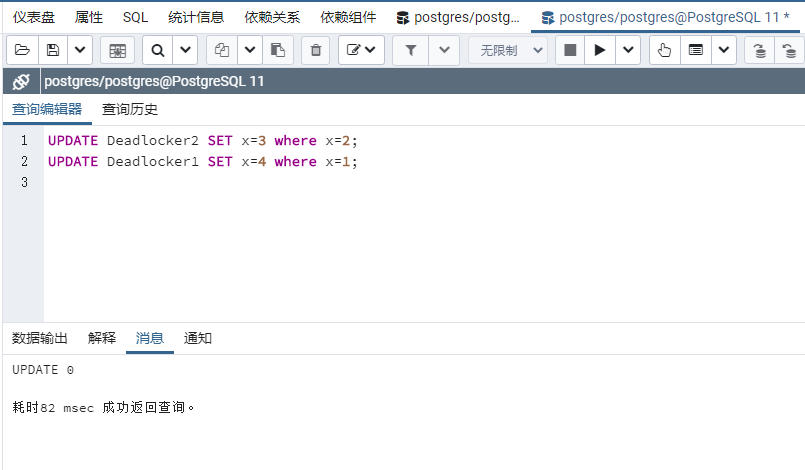
1. In the second session start a new transaction by updating the Deadlocker2 table, updating the second row (where x=2), placing a lock on the second row of the Deadlocker2 table. Also update the Deadlocker1 table, trying to set the first column (which is now locked by the other transaction)

UPDATE Deadlocker2 SET x=3 where x=2;

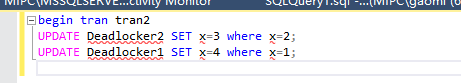
UPDATE Deadlocker1 SET x=4 where x=1;

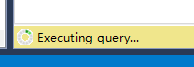
* **Oracle Note:** In Oracle Developer you will see ScriptRunner Task window running once you run the above commands.
* **SQL Server Note**: you will need to use transactions to create a deadlock. To do this before the update add line **BEGIN TRAN TRAN2**

In PostgreSQL, 0 row updated:



In SQL server, locked:





1. At this point, the second session should be trying to execute the query. Let us take a look at the locks.

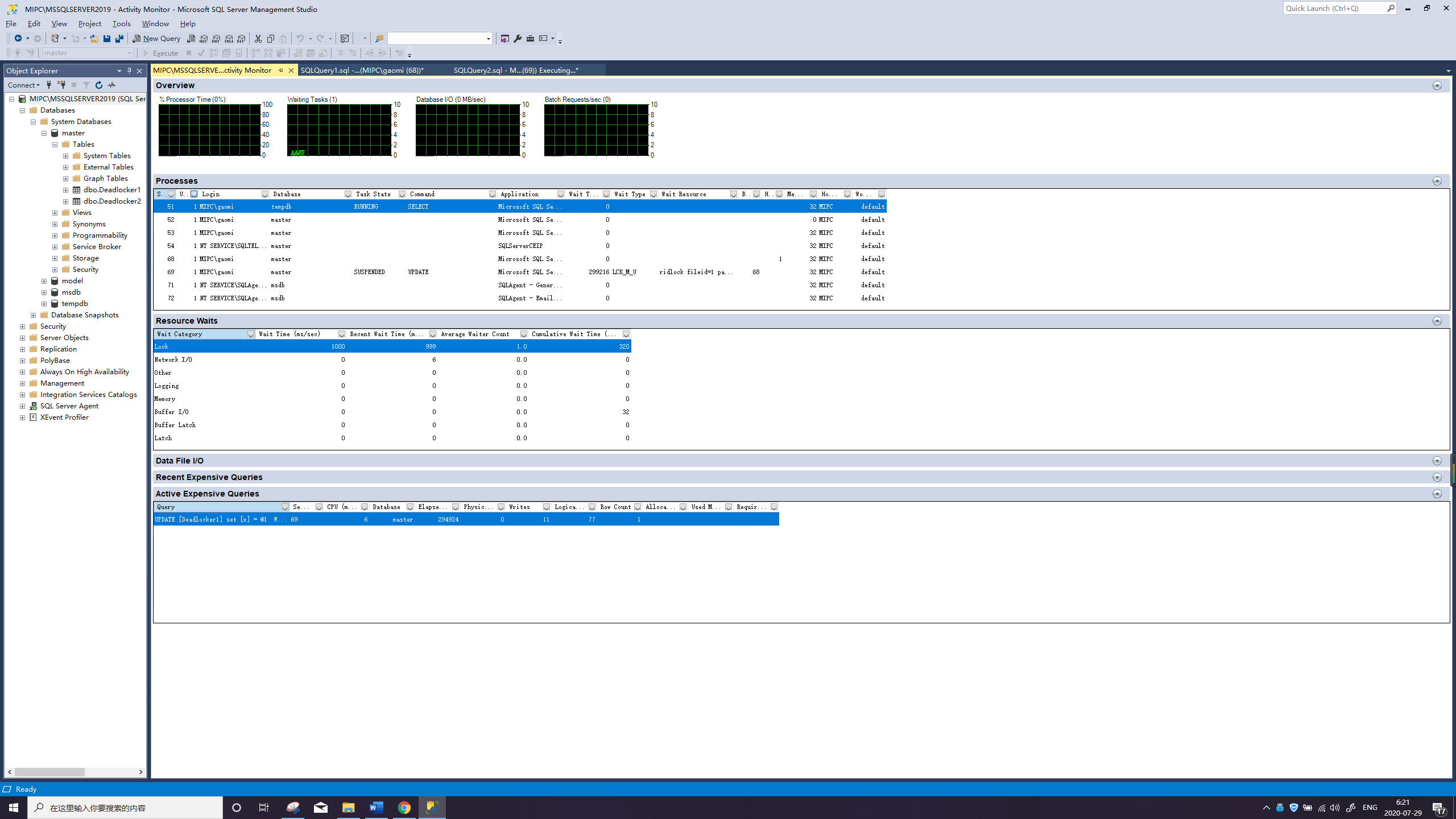
* In **Oracle SQL Developer:** Under Tools select “Monitor Sessions” and connect as user SYSTEM. A Session window will open. Look for the User who you logged in as (you can filter by the column) and note the UPDATE command running (under the Command Column).

[Take a screenshot to show your results and paste it here.]

* **In Microsoft SQL Server:** Open the Activity Monitor (One way is to hold down CTRL-ALT-A)

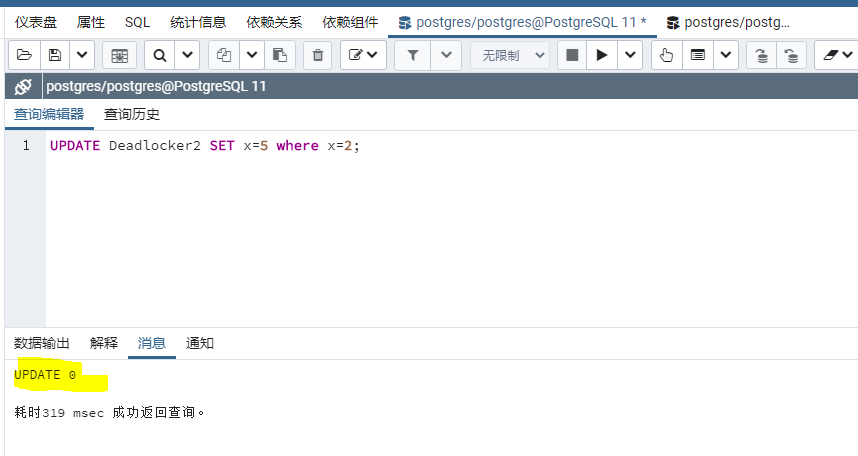
It is also available as an icon in the toolbar at the top. Expand the processes window. You should see one of the tasks suspended. That is the one that has the lock (examine the wait type and wait response).

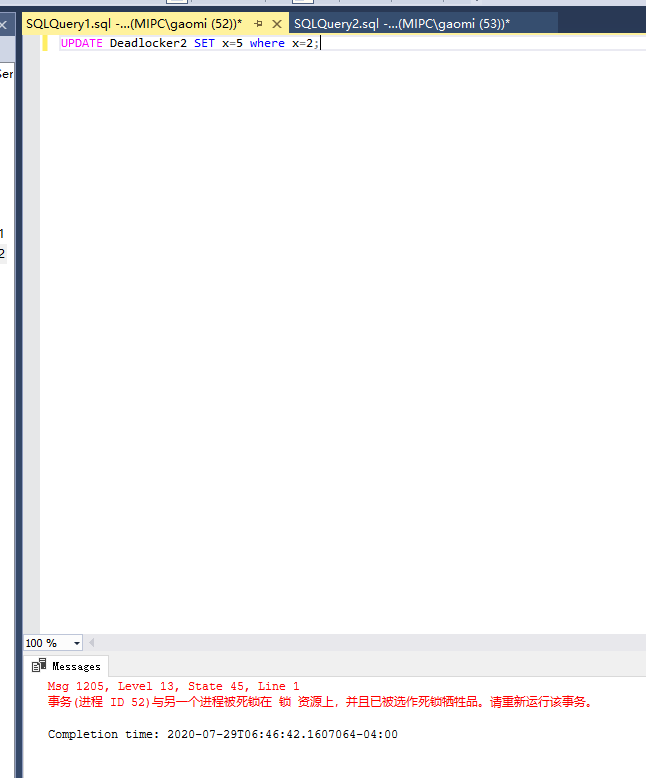




1. Now go back to the first session window, which is running the first transaction, and update the second Deadlocker2 table, updating the row where X=2.

UPDATE Deadlocker2 SET x=5 where x=2;



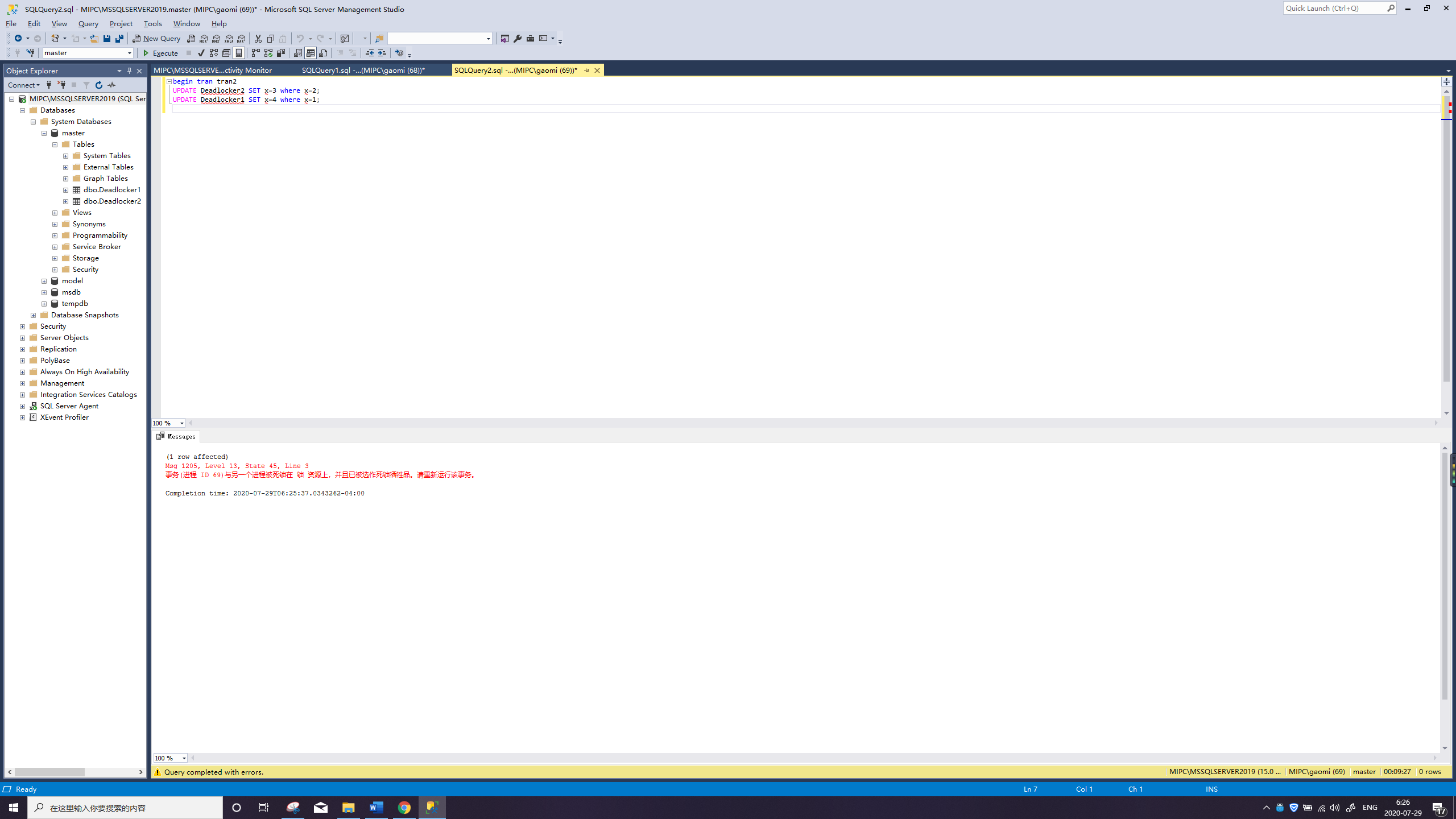


**If you mean update the second Deadlocker2 table in Session 1 immediately, session 1 will be sacrificed and session 2 will move on. Otherwise, session 2 will be sacrificed with the error.**

1. You will see an error message, which should tell you that the transaction was deadlocked for one of your transactions. Take a screenshot of this and attach it here.

Here is the other situation I said that session 2 has error exception; I just show out.

[Take a screenshot to show your results and paste it here.]

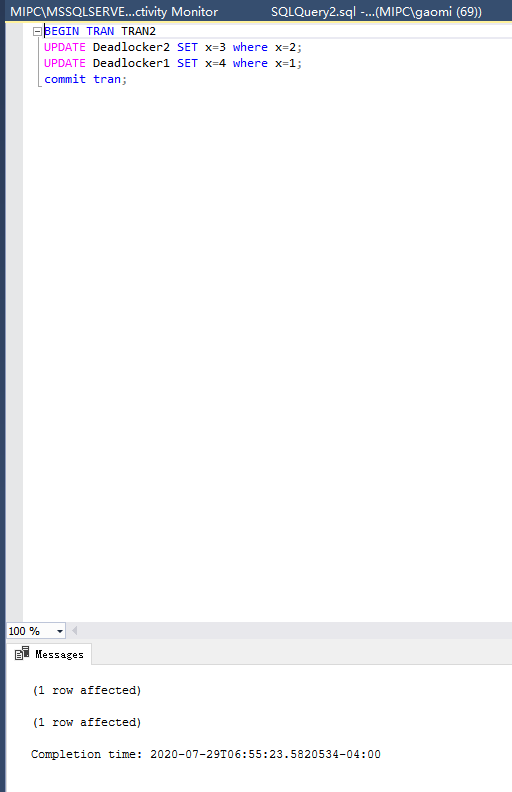


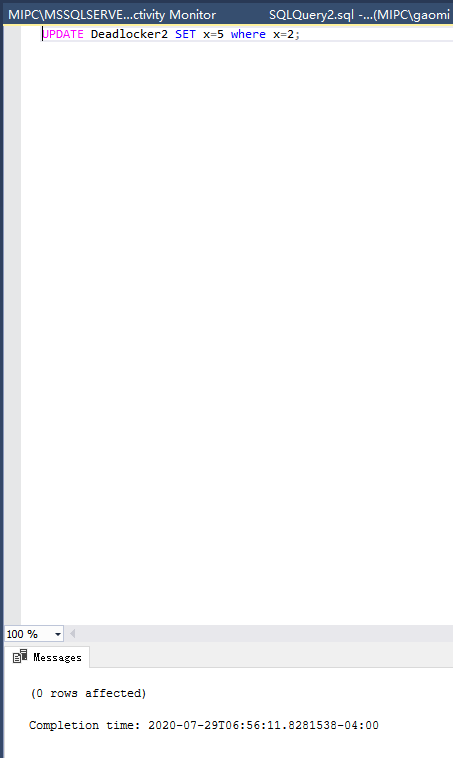
Anyway, this depends on which one will be the next/last one active once the deadlock occurred and begin to stack.

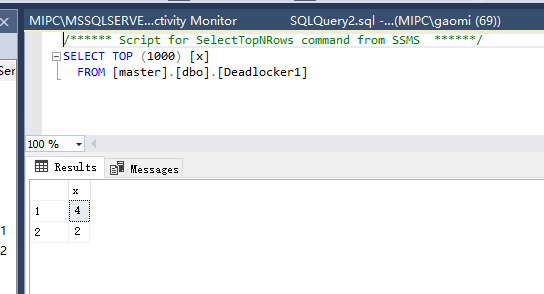
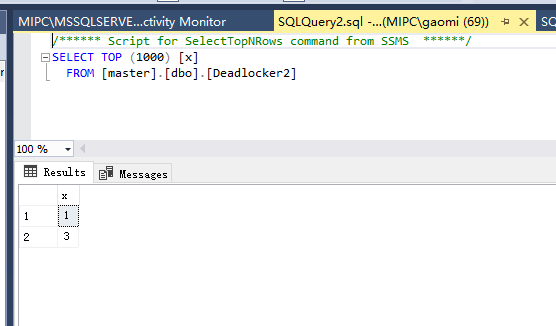
1. You can unlock the second session window by committing the transaction.



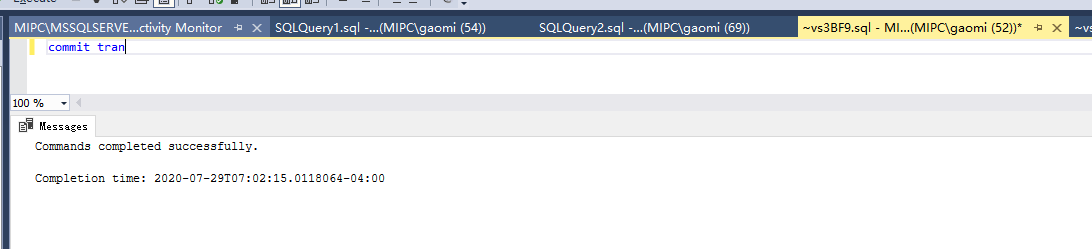
If you want to commit in session 2:



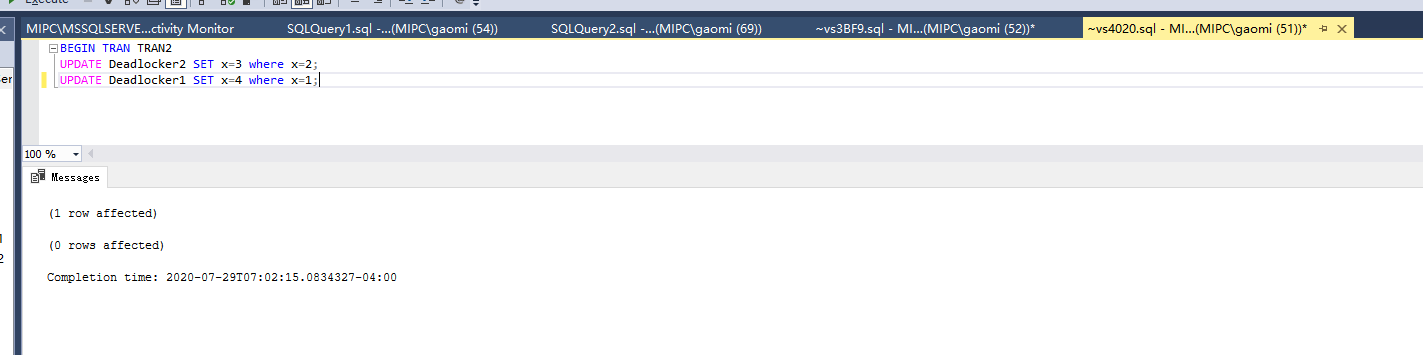


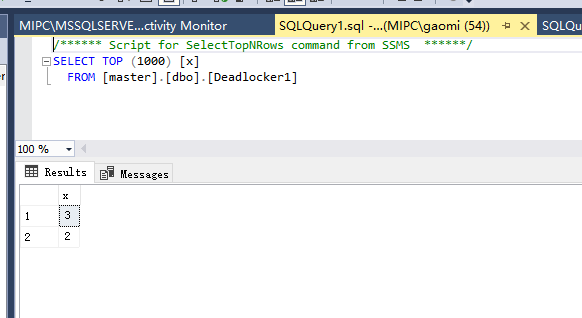
**However, let we rollback because you may want me to do both way. Therefore, I will also show what will happen if I just commit in session 1:**

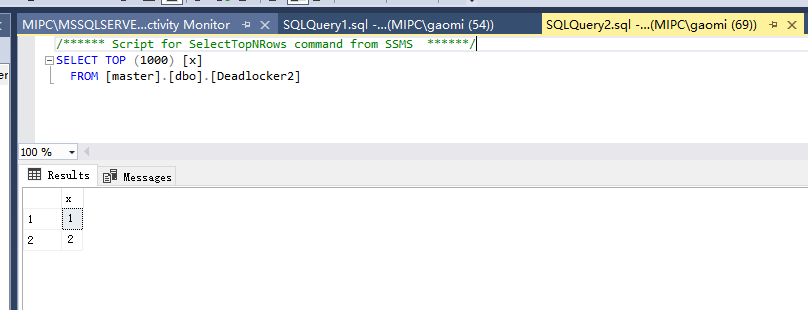


I did not UPDATE Deadlocker2 SET x=5 where x=2 this time because we all knew what is going to happen. I just show how the deadlock will be unlocked. So then:



Session 2 will be unlocked by committed in session 1.





Hope I have shown the situations fully to prevent the lost of my points.

1. [Optional] Explain which session was the deadlock victim and why this occurred. It may be helpful to create a little sequence table, and consider what is happening at each step, identifying when the deadlock occurs.

Generally, I think the victim of the deadlock is session 1 which did not commit its transaction. Then the deadlock occurs when transaction happen in session 2 because session 2 is going to update data in session 1. Also, this can be prove by querying SELECT once you’ve done the step 4; it will also be locked for same reason. Thus, the session 1 will take the responsibility as long as it does not commit the transaction.

1. [Optional]: If you would like you can try the same thing with shared locks, by selecting the rows instead of updating them. You shouldn’t get a deadlock in a DBMS that supports shared locks. In a DBMS with only binary locks you would get a deadlock.

Luck find! I just did this thing when I was going to double check the reason of who occurs the deadlock. Hope this one can also count as I completed.

1. [Optional] For a more advanced variation you can create a three-session deadlock. Hint: This requires three rows in Deadlocker and multiple Deadlocker tables.

CREATE TABLE Deadlocker3 (x INTEGER);

INSERT INTO Deadlocker3 VALUES(1);

INSERT INTO Deadlocker3 VALUES(2);

Then, when do step 5, do anything whether SELECT or UPDATE in session 3. For example,

Begin tran tran3

UPDATE Deadlocker3 SET x=3 where x=1

UPDATE Deadlocker2 SET x=5 where x=1

Then in session 1, try to update things in Deadlocker3.

Use the **Ask the Facilitators Discussion Board** if you have any questions regarding the how to approach this assignment.

Save your assignment as ***lastnameFirstname\_assign3.1.docx*** and submit it in the *Assignments* section of the course.

For help uploading files please refer to the *Technical Support* page in the syllabus.