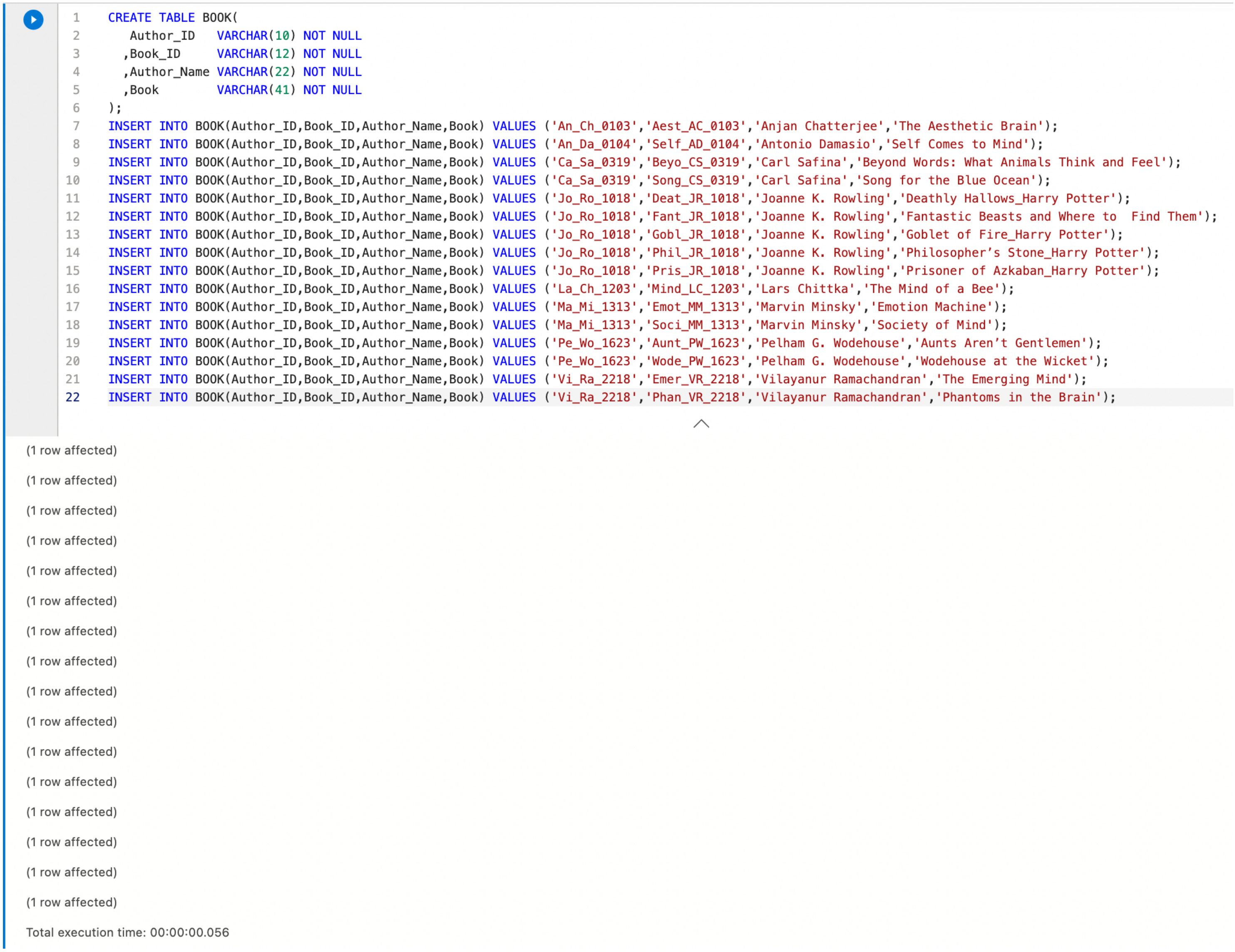
DBMS

LAB 8 and 9

- Rahul Barodia (B20CS047)

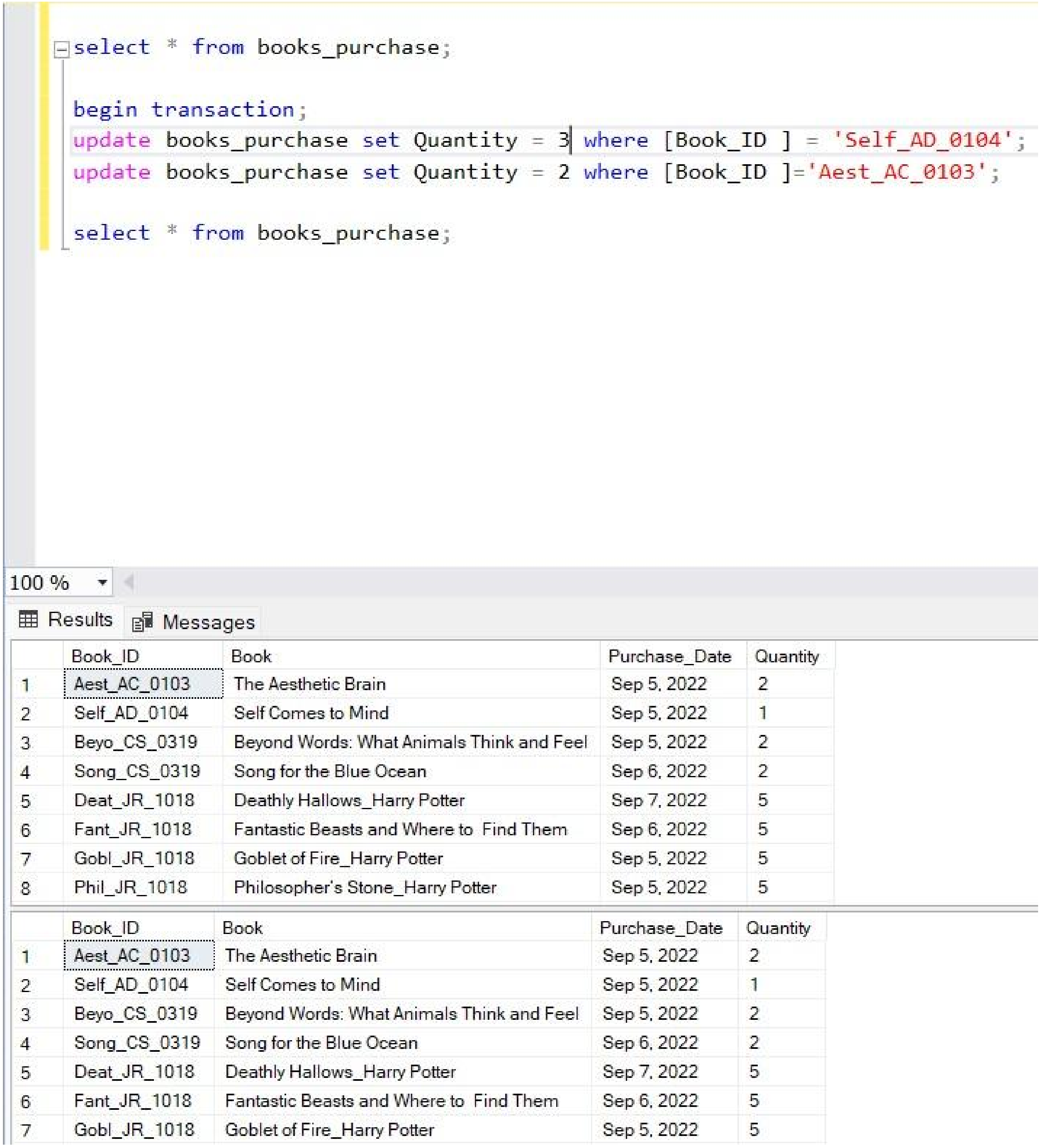
Q1)

a) We are asked to simulate an environment that allows multithreaded database access for multi-transaction handling. We can do it as follows,



```
CREATE TABLE BOOK(
   Author_ID   VARCHAR(10) NOT NULL
  ,Book_ID     VARCHAR(12) NOT NULL
  ,Author_Name VARCHAR(22) NOT NULL
  ,Book        VARCHAR(41) NOT NULL
);
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('An_Ch_0103','Aest_AC_0103','Anjan Chatterjee','The Aesthetic Brain');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('An_Da_0104','Self_AD_0104','Antonio Damasio','Self Comes to Mind');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Ca_Sa_0319','Beyo_CS_0319','Carl Safina','Beyond Words: What Animals Think and Feel');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Ca_Sa_0319','Song_CS_0319','Carl Safina','Song for the Blue Ocean');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Jo_Ro_1018','Deat_JR_1018','Joanne K. Rowling','Deathly Hallows_Harry Potter');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Jo_Ro_1018','Fant_JR_1018','Joanne K. Rowling','Fantastic Beasts and Where to  Find Them');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Jo_Ro_1018','Gobl_JR_1018','Joanne K. Rowling','Goblet of Fire_Harry Potter');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Jo_Ro_1018','Phil_JR_1018','Joanne K. Rowling','Philosopher’s Stone_Harry Potter');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Jo_Ro_1018','Pris_JR_1018','Joanne K. Rowling','Prisoner of Azkaban_Harry Potter');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('La_Ch_1203','Mind_LC_1203','Lars Chittka','The Mind of a Bee');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Ma_Mi_1313','Emot_MM_1313','Marvin Minsky','Emotion Machine');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Ma_Mi_1313','Soci_MM_1313','Marvin Minsky','Society of Mind');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Pe_Wo_1623','Aunt_PW_1623','Pelham G. Wodehouse','Aunts Aren’t Gentlemen');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Pe_Wo_1623','Wode_PW_1623','Pelham G. Wodehouse','Wodehouse at the Wicket');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Vi_Ra_2218','Emer_VR_2218','Vilayanur Ramachandran','The Emerging Mind');
INSERT INTO BOOK(Author_ID,Book_ID,Author_Name,Book) VALUES ('Vi_Ra_2218','Phan_VR_2218','Vilayanur Ramachandran','Phantoms in the Brain');
```

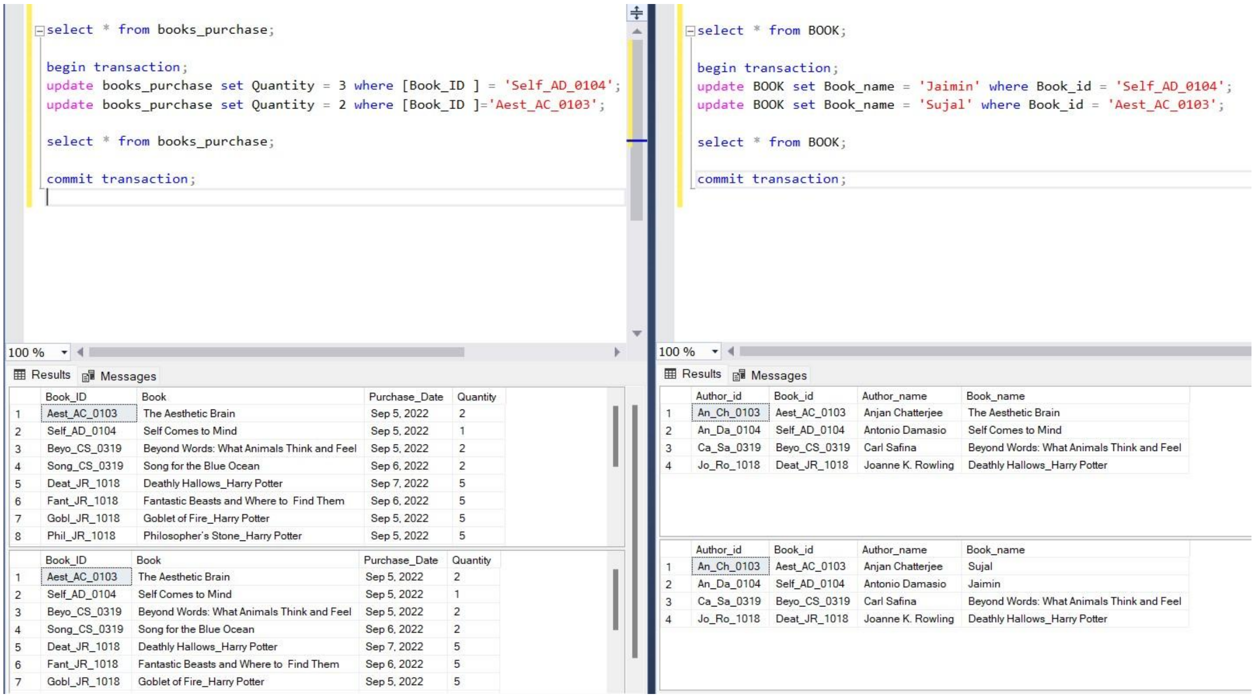
```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

Total execution time: 00:00:00.056
```



```
select * from books_purchase;

begin transaction;
update books_purchase set Quantity = 3 where [Book_ID ] = 'Self_AD_0104';
update books_purchase set Quantity = 2 where [Book_ID ]='Aest_AC_0103';

select * from books_purchase;
```

	Book_ID	Book	Purchase_Date	Quantity
1	Aest_AC_0103	The Aesthetic Brain	Sep 5, 2022	2
2	Self_AD_0104	Self Comes to Mind	Sep 5, 2022	1
3	Beyo_CS_0319	Beyond Words: What Animals Think and Feel	Sep 5, 2022	2
4	Song_CS_0319	Song for the Blue Ocean	Sep 6, 2022	2
5	Deat_JR_1018	Deathly Hallows_Harry Potter	Sep 7, 2022	5
6	Fant_JR_1018	Fantastic Beasts and Where to Find Them	Sep 6, 2022	5
7	Gobl_JR_1018	Goblet of Fire_Harry Potter	Sep 5, 2022	5
8	Phil_JR_1018	Philosopher's Stone_Harry Potter	Sep 5, 2022	5

	Book_ID	Book	Purchase_Date	Quantity
1	Aest_AC_0103	The Aesthetic Brain	Sep 5, 2022	2
2	Self_AD_0104	Self Comes to Mind	Sep 5, 2022	1
3	Beyo_CS_0319	Beyond Words: What Animals Think and Feel	Sep 5, 2022	2
4	Song_CS_0319	Song for the Blue Ocean	Sep 6, 2022	2
5	Deat_JR_1018	Deathly Hallows_Harry Potter	Sep 7, 2022	5
6	Fant_JR_1018	Fantastic Beasts and Where to Find Them	Sep 6, 2022	5
7	Gobl_JR_1018	Goblet of Fire_Harry Potter	Sep 5, 2022	5



```sql
select * from books_purchase;

begin transaction;
update books_purchase set Quantity = 3 where [Book_ID ] = 'Self_AD_0104';
update books_purchase set Quantity = 2 where [Book_ID ]='Aest_AC_0103';

select * from books_purchase;

commit transaction;
```

	Book_ID	Book	Purchase_Date	Quantity
1	Aest_AC_0103	The Aesthetic Brain	Sep 5, 2022	2
2	Self_AD_0104	Self Comes to Mind	Sep 5, 2022	1
3	Beyo_CS_0319	Beyond Words: What Animals Think and Feel	Sep 5, 2022	2
4	Song_CS_0319	Song for the Blue Ocean	Sep 6, 2022	2
5	Deat_JR_1018	Deathly Hallows_Harry Potter	Sep 7, 2022	5
6	Fant_JR_1018	Fantastic Beasts and Where to Find Them	Sep 6, 2022	5
7	Gobl_JR_1018	Goblet of Fire_Harry Potter	Sep 5, 2022	5
8	Phil_JR_1018	Philosopher's Stone_Harry Potter	Sep 5, 2022	5

	Book_ID	Book	Purchase_Date	Quantity
1	Aest_AC_0103	The Aesthetic Brain	Sep 5, 2022	2
2	Self_AD_0104	Self Comes to Mind	Sep 5, 2022	1
3	Beyo_CS_0319	Beyond Words: What Animals Think and Feel	Sep 5, 2022	2
4	Song_CS_0319	Song for the Blue Ocean	Sep 6, 2022	2
5	Deat_JR_1018	Deathly Hallows_Harry Potter	Sep 7, 2022	5
6	Fant_JR_1018	Fantastic Beasts and Where to Find Them	Sep 6, 2022	5
7	Gobl_JR_1018	Goblet of Fire_Harry Potter	Sep 5, 2022	5

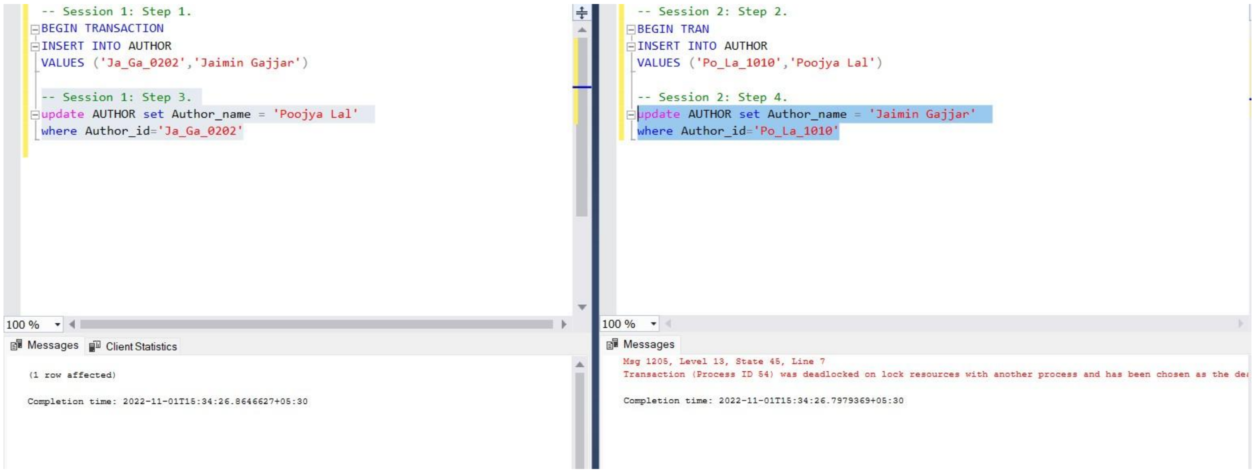
```sql
select * from BOOK;

begin transaction;
update BOOK set Book_name = 'Jaimin' where Book_id = 'Self_AD_0104';
update BOOK set Book_name = 'Sujal' where Book_id = 'Aest_AC_0103';

select * from BOOK;

commit transaction;
```

	Author_id	Book_id	Author_name	Book_name
1	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	The Aesthetic Brain
2	An_Da_0104	Self_AD_0104	Antonio Damasio	Self Comes to Mind
3	Ca_Sa_0319	Beyo_CS_0319	Carl Safina	Beyond Words: What Animals Think and Feel
4	Jo_Ro_1018	Deat_JR_1018	Joanne K. Rowling	Deathly Hallows_Harry Potter

	Author_id	Book_id	Author_name	Book_name
1	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	Sujal
2	An_Da_0104	Self_AD_0104	Antonio Damasio	Jaimin
3	Ca_Sa_0319	Beyo_CS_0319	Carl Safina	Beyond Words: What Animals Think and Feel
4	Jo_Ro_1018	Deat_JR_1018	Joanne K. Rowling	Deathly Hallows_Harry Potter

b) We are asked to simulate a deadlock situation derived from multiple transactions attempting to access & update the same tuple.

i)



```sql
-- Session 1: Step 1.
BEGIN TRANSACTION
INSERT INTO AUTHOR
VALUES ('Ja_Ga_0202','Jaimin Gajjar')

-- Session 1: Step 3.
update AUTHOR set Author_name = 'Poojya Lal'
where Author_id='Ja_Ga_0202'
```

```
(1 row affected)

Completion time: 2022-11-01T15:34:26.8646627+05:30
```

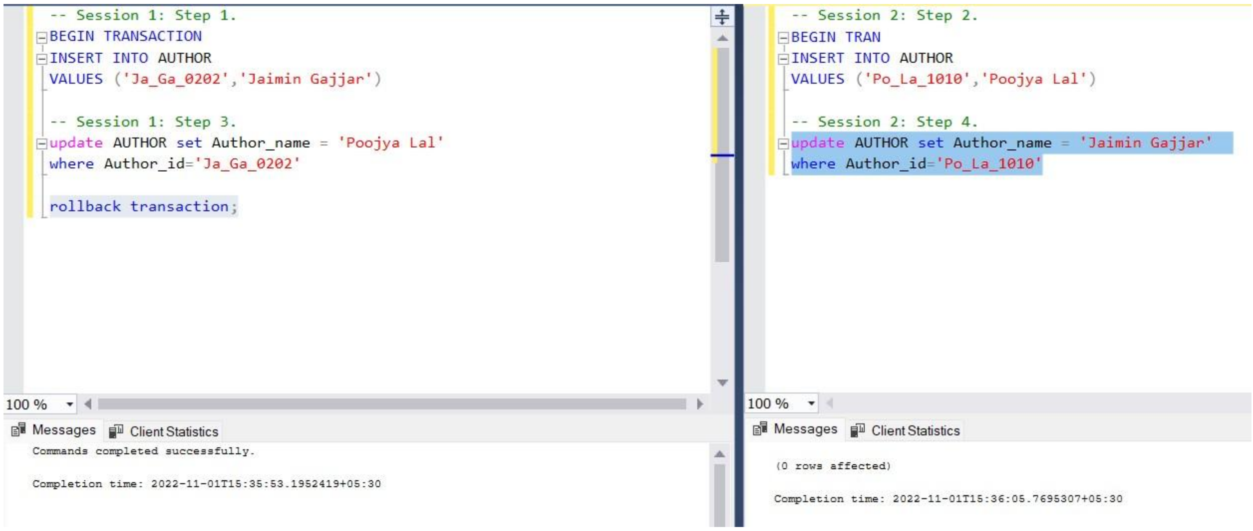
```sql
-- Session 2: Step 2.
BEGIN TRAN
INSERT INTO AUTHOR
VALUES ('Po_La_1010','Poojya Lal')

-- Session 2: Step 4.
update AUTHOR set Author_name = 'Jaimin Gajjar'
where Author_id='Po_La_1010'
```

```
Msg 1205, Level 13, State 45, Line 7
Transaction (Process ID 54) was deadlocked on lock resources with another process and has been chosen as the dea

Completion time: 2022-11-01T15:34:26.7979369+05:30
```

ii) My algorithm:

Since q1 arrived first, it should be given precedence over other transactions; thus I would have rolled back the second transaction, i.e., session 2 because its initial execution began after query 1's. This would have allowed query 1 to proceed.

iii)



```
-- Session 1: Step 1.
BEGIN TRANSACTION
INSERT INTO AUTHOR
VALUES ('Ja_Ga_0202','Jaimin Gajjar')

-- Session 1: Step 3.
update AUTHOR set Author_name = 'Poojya Lal'
where Author_id='Ja_Ga_0202'

rollback transaction;
```

```
Commands completed successfully.

Completion time: 2022-11-01T15:35:53.1952419+05:30
```

```
-- Session 2: Step 2.
BEGIN TRAN
INSERT INTO AUTHOR
VALUES ('Po_La_1010','Poojya Lal')

-- Session 2: Step 4.
update AUTHOR set Author_name = 'Jaimin Gajjar'
where Author_id='Po_La_1010'
```

```
(0 rows affected)

Completion time: 2022-11-01T15:36:05.7695307+05:30
```

Here, we can see that all operations are successfully executed after rolling back session 2.

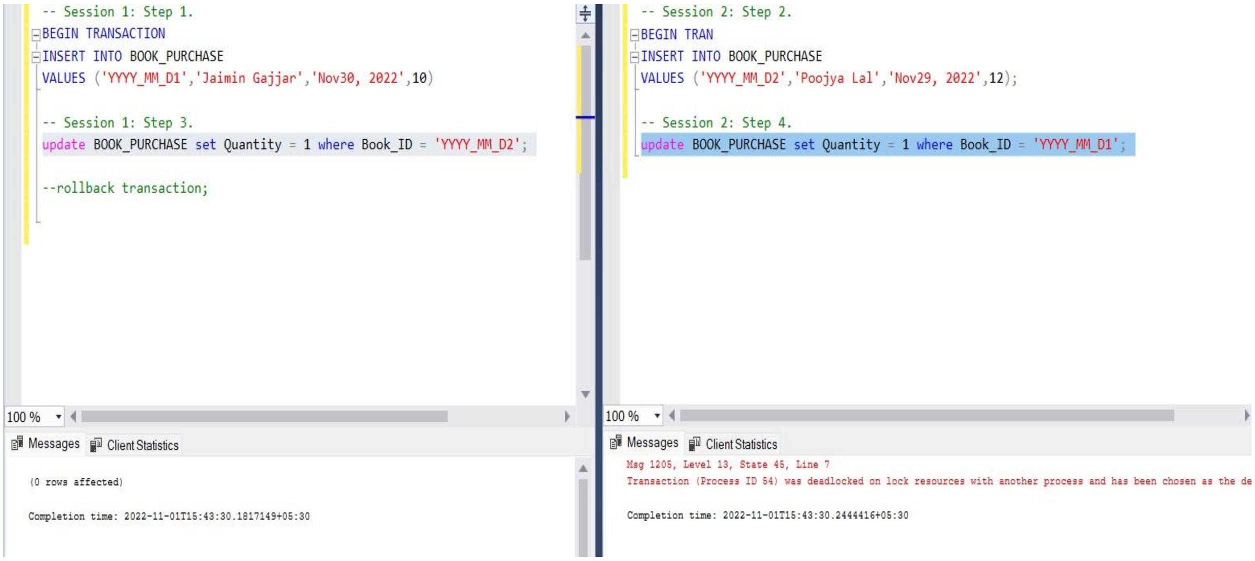
iv)

Algorithm is not serializable.
The reason is that it cannot be resolved into a comparable conflict schedule.

Q2) We are asked to simulate a deadlock situation derived from multiple transactions attempting to access & update the same table

a)
First of all, let's see what is meaning of Deadlock in a Database.
When two or more processes lock a resource and each demands a lock on the resource that another process has already locked, a deadlock takes place in a database. Since both transactions are awaiting the other to release the lock, neither may proceed at this time.

Now,



```
-- Session 1: Step 1.
BEGIN TRANSACTION
INSERT INTO BOOK_PURCHASE
VALUES ('YYYY_MM_D1','Jaimin Gajjar','Nov30, 2022',10)

-- Session 1: Step 3.
update BOOK_PURCHASE set Quantity = 1 where Book_ID = 'YYYY_MM_D2';

--rollback transaction;
```

```
(0 rows affected)

Completion time: 2022-11-01T15:43:30.1817149+05:30
```

```
-- Session 2: Step 2.
BEGIN TRAN
INSERT INTO BOOK_PURCHASE
VALUES ('YYYY_MM_D2','Poojya Lal','Nov29, 2022',12);

-- Session 2: Step 4.
update BOOK_PURCHASE set Quantity = 1 where Book_ID = 'YYYY_MM_D1';
```

```
Msg 1205, Level 13, State 45, Line 7
Transaction (Process ID 54) was deadlocked on lock resources with another process and has been chosen as the de

Completion time: 2022-11-01T15:43:30.2444416+05:30
```

Description of above:

A request from table 1 was made in order to update BOOK PURCHASE in query 1 (txn1), which was locked table 1. A request from table 2 was made in order to update BOOK in query 2 (txn2), which was locked table 2. A request from table 1 was also made in order to update BOOK PURCHASE in query 1 (txn1).

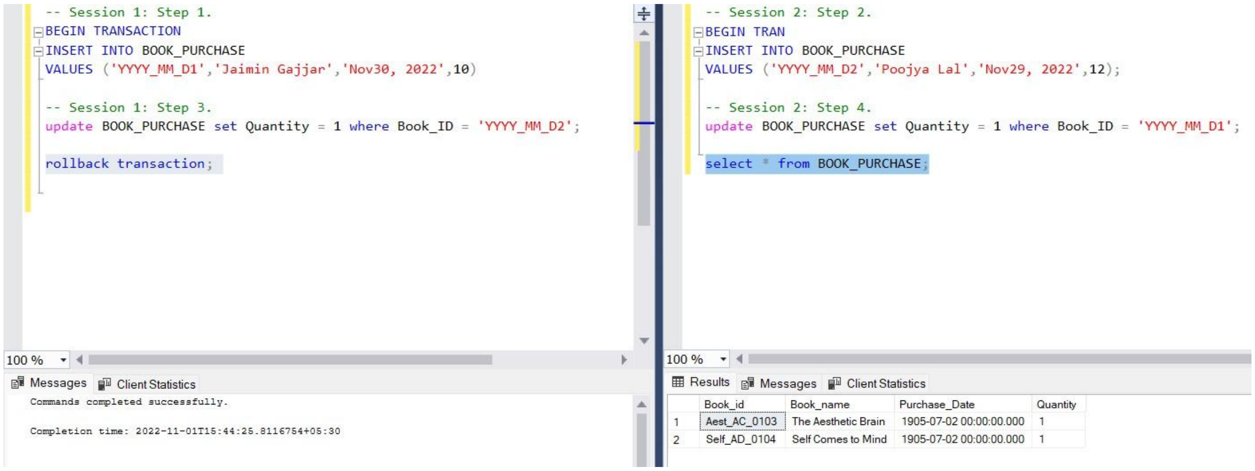
Since all the necessary conditions are seen to be satisfying, hence the situation is indeed a deadlock.

The system will select one of the processes as the deadlock target and rewind that process so that the other process may proceed.

b)

Since q1 arrived first, it should be finished first or given precedence over other transactions, thus I would have rolled back the second transaction, i.e., session 2, because its initial execution began after query 1's. This would have allowed query 1 to proceed.

c)



```
-- Session 1: Step 1.
BEGIN TRANSACTION
INSERT INTO BOOK_PURCHASE
VALUES ('YYYY_MM_D1','Jaimin Gajjar','Nov30, 2022',10)

-- Session 1: Step 3.
update BOOK_PURCHASE set Quantity = 1 where Book_ID = 'YYYY_MM_D2';

rollback transaction;
```

```
Commands completed successfully.

Completion time: 2022-11-01T15:44:25.8116754+05:30
```

```
-- Session 2: Step 2.
BEGIN TRAN
INSERT INTO BOOK_PURCHASE
VALUES ('YYYY_MM_D2','Poojya Lal','Nov29, 2022',12);

-- Session 2: Step 4.
update BOOK_PURCHASE set Quantity = 1 where Book_ID = 'YYYY_MM_D1';

select * from BOOK_PURCHASE;
```

	Book_id	Book_name	Purchase_Date	Quantity
1	Aest_AC_0103	The Aesthetic Brain	1905-07-02 00:00:00.000	1
2	Self_AD_0104	Self Comes to Mind	1905-07-02 00:00:00.000	1

Here, we can see that all operations are successfully executed after rolling back session 2.

d)

Not serializable because it cannot be resolved into a comparable conflict schedule.

Q3)

- See if there are any deadlocks in the system health session.
- In order to capture the deadlocks, create an extended event session.
- To identify the issue, examine the reports and graphs on deadlock.
- If there are any changes that can be made to the queries that are causing the deadlock.

The development team was using the following table to store the order numbers, and the following query was used to create the first row of the day

```
CREATE TABLE [TestTblCounter](
 [Id] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,
 [SerialNumber] [int] NULL,
 [LogDate] [datetime] NULL)
 GO
        IF NOT EXISTS(SELECT Id
                FROM TestTblCounter
                WHERE LogDate = CONVERT(VARCHAR(100), GETDATE(), 112))
        BEGIN
        INSERT INTO TestTblCounter
        VALUES
        ('1',
       CONVERT(VARCHAR(100), GETDATE(), 112)
        )
        END
```

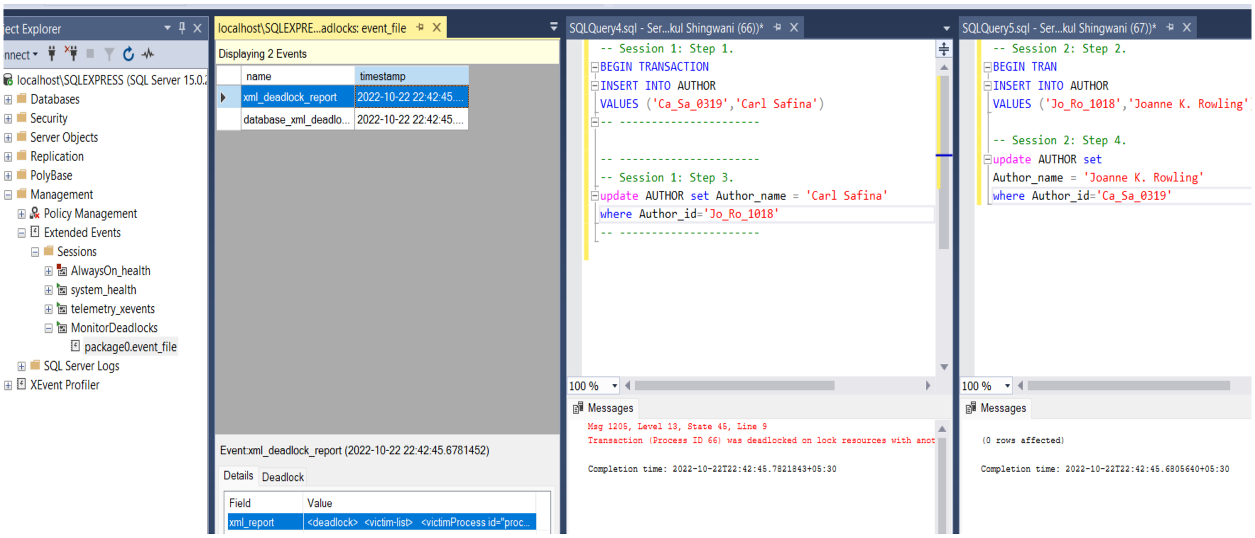
use the following stored procedure to create a new order number.

```
CREATE PROCEDURE CreateLogNo
AS
        DECLARE @LogNo AS VARCHAR(50), @LogCounter AS INT= 0;
        BEGIN TRAN;

        UPDATE TestTblCounter
        SET
        SerialNumber = SerialNumber + 1
        WHERE LogDate = CONVERT(VARCHAR(100), GETDATE(), 112);
        SELECT @LogCounter = SerialNumber
        FROM TestTblCounter WITH(TABLOCKX)
        WHERE LogDate = CONVERT(VARCHAR(100), GETDATE(), 112);
        SELECT @LogCounter AS LogNumber;
        COMMIT TRAN
```

The system_health is the default extended event session of the SQL Server, and it started automatically when the database engine starts. The system_health session collects various system data, and one of them is deadlock information. The following query reads the .xel file of the system_health session and gives information about the deadlock problems which were occurred. The system_health session can be a good starting point to figure out the deadlock problems. The below query helps to find out the deadlock problems which is captured by the system_health session

```
DECLARE @xelfilepath NVARCHAR(260)
SELECT @xelfilepath = dosdlc.path
FROM sys.dm_os_server_diagnostics_log_configurations AS dosdlc;
SELECT @xelfilepath = @xelfilepath + N'system_health_*.xel'
DROP TABLE IF EXISTS  #TempTable
SELECT CONVERT(XML, event_data) AS EventData
      INTO #TempTable FROM sys.fn_xe_file_target_read_file(@xelfilepath, NULL, NULL,
NULL)
            WHERE object_name = 'xml_deadlock_report'
SELECT EventData.value('(event/@timestamp)[1]', 'datetime2(7)') AS UtcTime,
         CONVERT(DATETIME, SWITCHOFFSET(CONVERT(DATETIMEOFFSET,
     EventData.value('(event/@timestamp)[1]', 'VARCHAR(50)')), DATENAME(TzOffset,
SYSDATETIMEOFFSET()))) AS LocalTime,
         EventData.query('event/data/value/deadlock') AS XmlDeadlockReport
            FROM #TempTable
            ORDER BY UtcTime DESC;
```

When we click any row of the XmlDeadlockReport column, the deadlock report will appear. The database administrator found some clues about the deadlock problem through the captured data by the system_health session. However, he thought that the system_health session shows the more recent events because of the file size limitations, so it cannot be reliable to detect all deadlocks in SQL Server. So, he decided to create a new extended event session that can capture all the deadlocks. Extended Event is a system monitoring tool that helps to collect events and system information from SQL Server. With the help of the XEvent, we can also capture deadlock information from SQL Server. Firstly, we will launch SQL Server Management Studio and navigate to Session, which is placed under the Management folder. Right-click on the Sessions folder and select New Session.



Displaying 2 Events

name	timestamp
xml_deadlock_report	2022-10-22 22:42:45....
database_xml_deadlo...	2022-10-22 22:42:45....

Event:xml_deadlock_report (2022-10-22 22:42:45.6781452)

Field	Value
xml_report	<deadlock> <victim-list> <victimProcess id="proc...

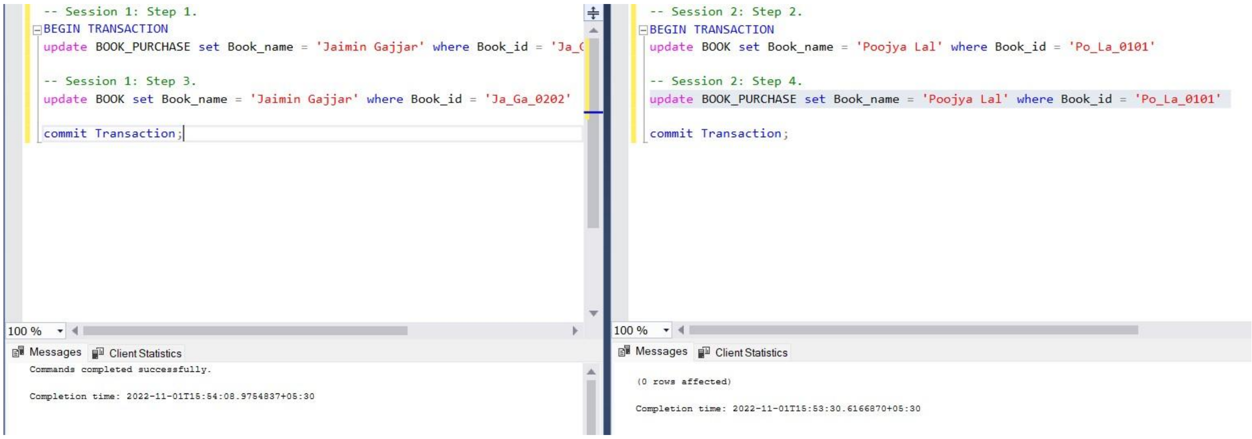
```sql
-- Session 1: Step 1.
BEGIN TRANSACTION
INSERT INTO AUTHOR
VALUES ('Ca_Sa_0319','Carl Safina')
-- ---------------------


-- ---------------------
-- Session 1: Step 3.
update AUTHOR set Author_name = 'Carl Safina'
where Author_id='Jo_Ro_1018'
-- ---------------------
```

```
Msg 1205, Level 13, State 45, Line 9
Transaction (Process ID 66) was deadlocked on lock resources with anot

Completion time: 2022-10-22T22:42:45.7821843+05:30
```

```sql
-- Session 2: Step 2.
BEGIN TRAN
INSERT INTO AUTHOR
VALUES ('Jo_Ro_1018','Joanne K. Rowling'

-- Session 2: Step 4.
update AUTHOR set
Author_name = 'Joanne K. Rowling'
where Author_id='Ca_Sa_0319'
```

```
(0 rows affected)

Completion time: 2022-10-22T22:42:45.6805640+05:30
```

Q4)



```sql
-- Session 1: Step 1.
BEGIN TRANSACTION
update BOOK_PURCHASE set Book_name = 'Jaimin Gajjar' where Book_id = 'Ja_(

-- Session 1: Step 3.
update BOOK set Book_name = 'Jaimin Gajjar' where Book_id = 'Ja_Ga_0202'

commit Transaction;
```

```
Commands completed successfully.

Completion time: 2022-11-01T15:54:08.9754837+05:30
```

```sql
-- Session 2: Step 2.
BEGIN TRANSACTION
update BOOK set Book_name = 'Poojya Lal' where Book_id = 'Po_La_0101'

-- Session 2: Step 4.
update BOOK_PURCHASE set Book_name = 'Poojya Lal' where Book_id = 'Po_La_0101'

commit Transaction;
```

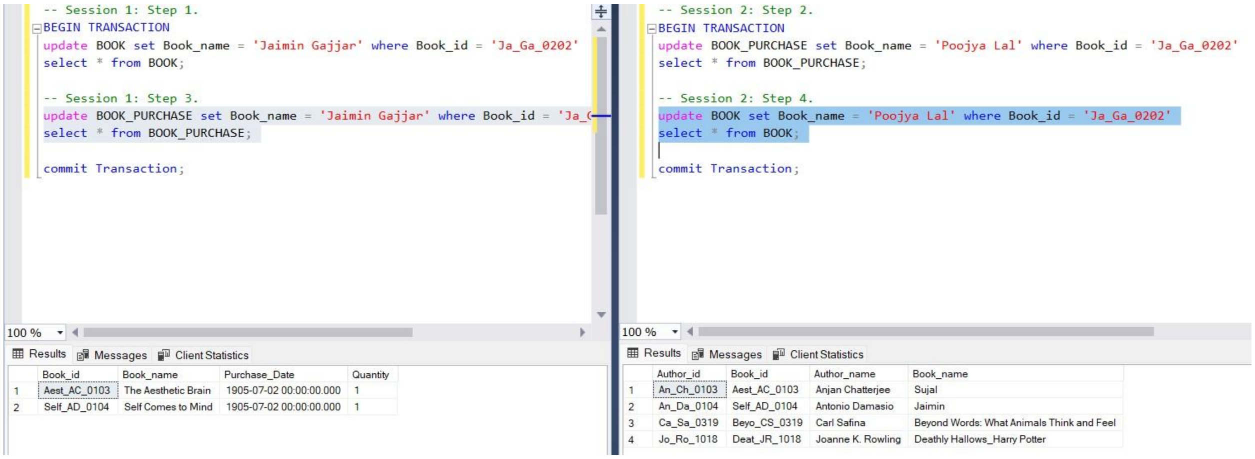
```
(0 rows affected)

Completion time: 2022-11-01T15:53:30.6166870+05:30
```

Same as Q2

Q5)

Using the same code as in Q2, we can see that a stalemate occurs. However, using rollback, the deadlock may be broken, as shown in the photos below.

Query



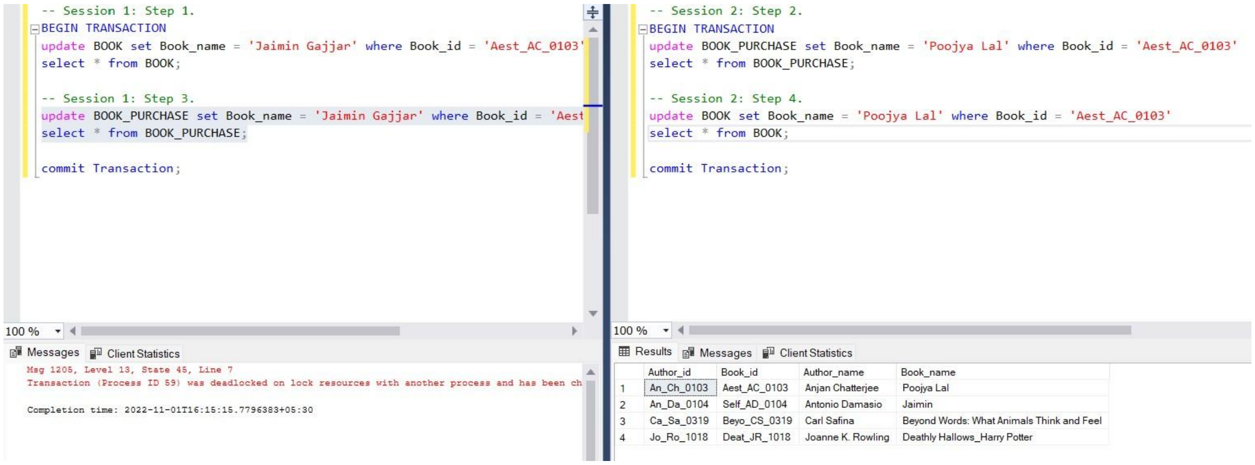
```sql
-- Session 1: Step 1.
BEGIN TRANSACTION
update BOOK set Book_name = 'Jaimin Gajjar' where Book_id = 'Ja_Ga_0202'
select * from BOOK;

-- Session 1: Step 3.
update BOOK_PURCHASE set Book_name = 'Jaimin Gajjar' where Book_id = 'Ja_G
select * from BOOK_PURCHASE;

commit Transaction;
```

	Book_id	Book_name	Purchase_Date	Quantity
1	Aest_AC_0103	The Aesthetic Brain	1905-07-02 00:00:00.000	1
2	Self_AD_0104	Self Comes to Mind	1905-07-02 00:00:00.000	1

```sql
-- Session 2: Step 2.
BEGIN TRANSACTION
update BOOK_PURCHASE set Book_name = 'Poojya Lal' where Book_id = 'Ja_Ga_0202'
select * from BOOK_PURCHASE;

-- Session 2: Step 4.
update BOOK set Book_name = 'Poojya Lal' where Book_id = 'Ja_Ga_0202'
select * from BOOK;

commit Transaction;
```

	Author_id	Book_id	Author_name	Book_name
1	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	Sujal
2	An_Da_0104	Self_AD_0104	Antonio Damasio	Jaimin
3	Ca_Sa_0319	Beyo_CS_0319	Carl Safina	Beyond Words: What Animals Think and Feel
4	Jo_Ro_1018	Deat_JR_1018	Joanne K. Rowling	Deathly Hallows_Harry Potter

Deadlock



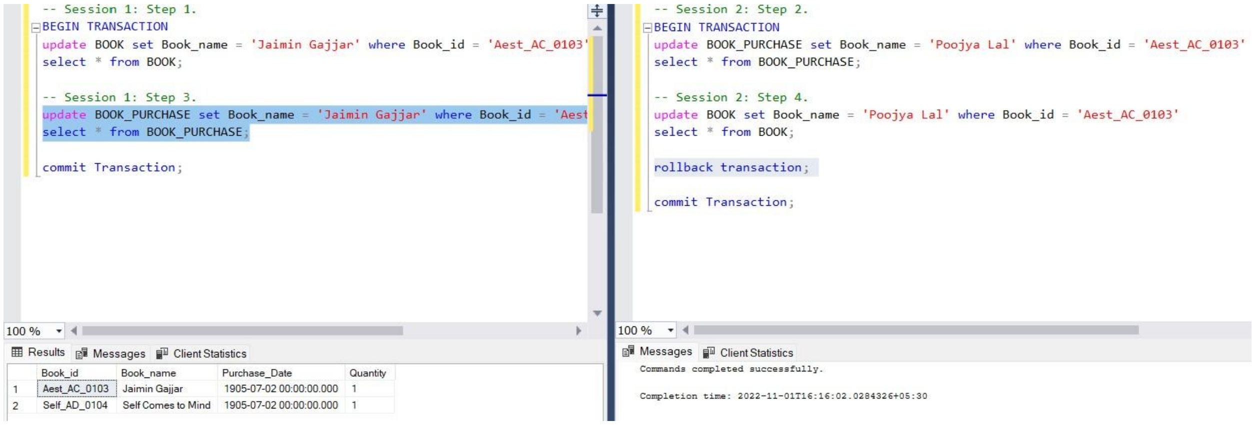
```sql
-- Session 1: Step 1.
BEGIN TRANSACTION
update BOOK set Book_name = 'Jaimin Gajjar' where Book_id = 'Aest_AC_0103'
select * from BOOK;

-- Session 1: Step 3.
update BOOK_PURCHASE set Book_name = 'Jaimin Gajjar' where Book_id = 'Aest
select * from BOOK_PURCHASE;

commit Transaction;
```

```
Msg 1205, Level 13, State 45, Line 7
Transaction (Process ID 59) was deadlocked on lock resources with another process and has been ch

Completion time: 2022-11-01T16:15:15.7796383+05:30
```

```sql
-- Session 2: Step 2.
BEGIN TRANSACTION
update BOOK_PURCHASE set Book_name = 'Poojya Lal' where Book_id = 'Aest_AC_0103'
select * from BOOK_PURCHASE;

-- Session 2: Step 4.
update BOOK set Book_name = 'Poojya Lal' where Book_id = 'Aest_AC_0103'
select * from BOOK;

commit Transaction;
```

	Author_id	Book_id	Author_name	Book_name
1	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	Poojya Lal
2	An_Da_0104	Self_AD_0104	Antonio Damasio	Jaimin
3	Ca_Sa_0319	Beyo_CS_0319	Carl Safina	Beyond Words: What Animals Think and Feel
4	Jo_Ro_1018	Deat_JR_1018	Joanne K. Rowling	Deathly Hallows_Harry Potter

Rollback



```
-- Session 1: Step 1.
BEGIN TRANSACTION
update BOOK set Book_name = 'Jaimin Gajjar' where Book_id = 'Aest_AC_0103'
select * from BOOK;

-- Session 1: Step 3.
update BOOK_PURCHASE set Book_name = 'Jaimin Gajjar' where Book_id = 'Aest
select * from BOOK_PURCHASE;

commit Transaction;
```

	Book_id	Book_name	Purchase_Date	Quantity
1	Aest_AC_0103	Jaimin Gajjar	1905-07-02 00:00:00.000	1
2	Self_AD_0104	Self Comes to Mind	1905-07-02 00:00:00.000	1

```
-- Session 2: Step 2.
BEGIN TRANSACTION
update BOOK_PURCHASE set Book_name = 'Poojya Lal' where Book_id = 'Aest_AC_0103'
select * from BOOK_PURCHASE;

-- Session 2: Step 4.
update BOOK set Book_name = 'Poojya Lal' where Book_id = 'Aest_AC_0103'
select * from BOOK;

rollback transaction;

commit Transaction;
```

```
Commands completed successfully.

Completion time: 2022-11-01T16:16:02.0284326+05:30
```

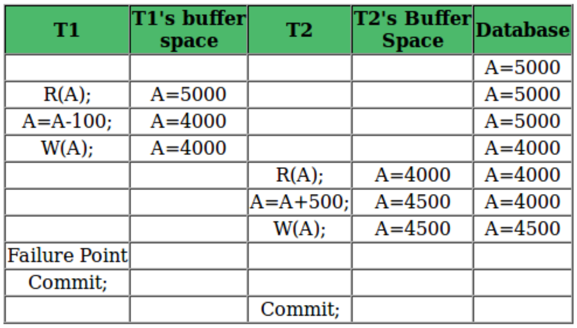
Q6)

Transaction Timestamp could offer a solution to the problem (TS). This solution assigns a timestamp, based on the time of arrival of each transaction, to the unique IDs for each transaction. They are established by the sequence in which Transactions are started. Consider the case when T1 starts before T2: TS(T1) TS (T2). Wound-wait and wait-die are two timestamp-based deadlock avoidance techniques.
Consider a scenario where there are two transactions, Ti and Tj. Ti attempts to lock an object X, but another Tj has already done so. As a result, there is a stalemate, therefore using the wait-die method: If TS(Ti) > TS(Tj), (Ti older than Tj) Ti may wait; if not, Ti (Ti younger than Tj) Ti shall be aborted and shall afterwards be restarted with the same timestamp. If TS(Ti) TS(Tj), then (Ti older than Tj) Ti is permitted to wait; otherwise, abort Ti (Ti younger than Tj) Ti and continue it later with the same parameters.
Ti and Tj lock mutex 1 and 2, respectively, in the real code.Now that Tj has locked mutex 2, Ti wants access to it, but Tj also wants access to mutex 1, which Ti has locked.
TS(Ti) TS because Ti's start time is earlier than Tj's (Tj). As a consequence, Tj is rolled back and the lock on mutex 2 is released, enabling Ti to wait while T may then acquire it.

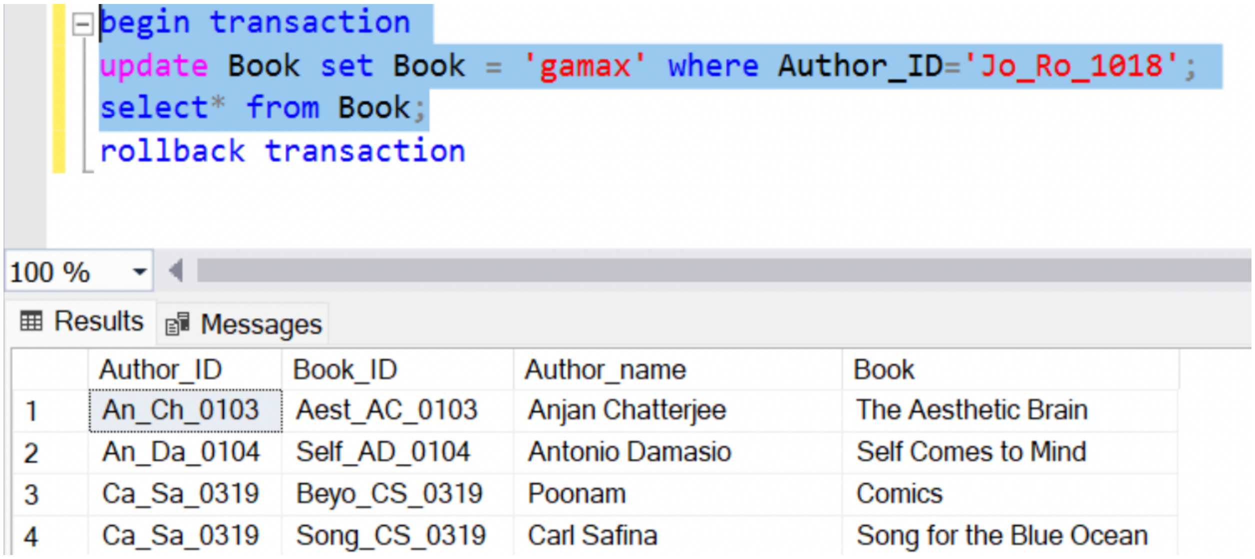
Q7)

A transaction becomes irreversible if it reads data that has already been written by an uncommitted transaction.

An irrecoverable schedule is one that involves a dirty read operation from an uncommitted transaction that commits before the transaction from which it has read the value.

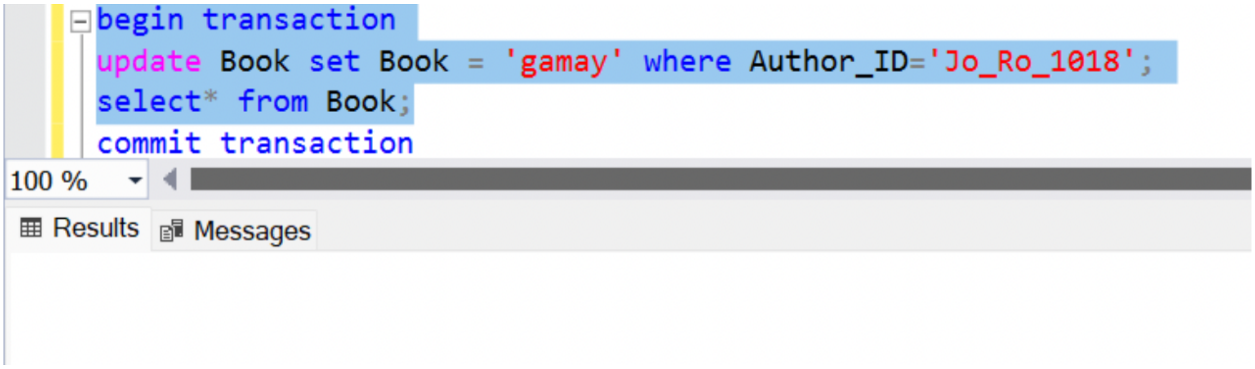


T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
A=A-100;	A=4000			A=5000
W(A);	A=4000			A=4000
		R(A);	A=4000	A=4000
		A=A+500;	A=4500	A=4000
		W(A);	A=4500	A=4500
Failure Point				
Commit;				
		Commit;		

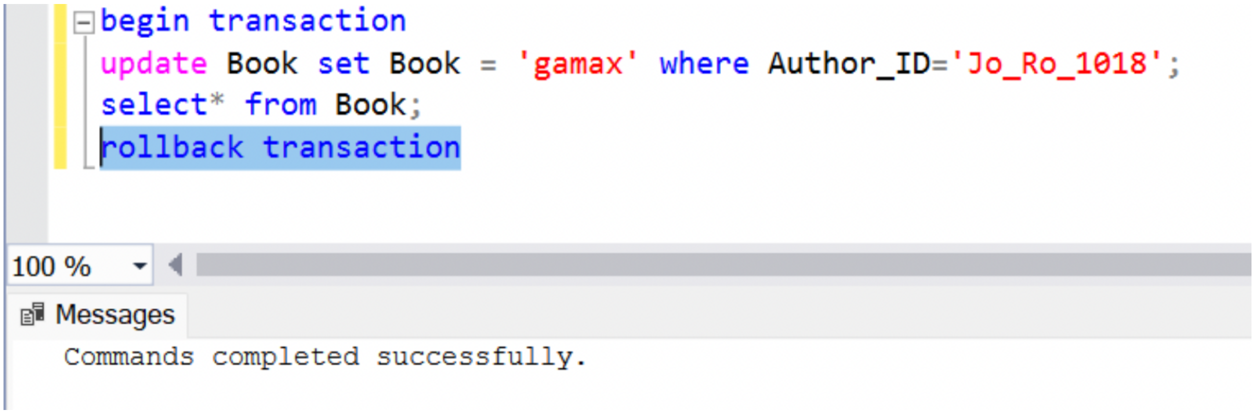


```
begin transaction
update Book set Book = 'gamax' where Author_ID='Jo_Ro_1018';
select* from Book;
rollback transaction
```

	Author_ID	Book_ID	Author_name	Book
1	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	The Aesthetic Brain
2	An_Da_0104	Self_AD_0104	Antonio Damasio	Self Comes to Mind
3	Ca_Sa_0319	Beyo_CS_0319	Poonam	Comics
4	Ca_Sa_0319	Song_CS_0319	Carl Safina	Song for the Blue Ocean

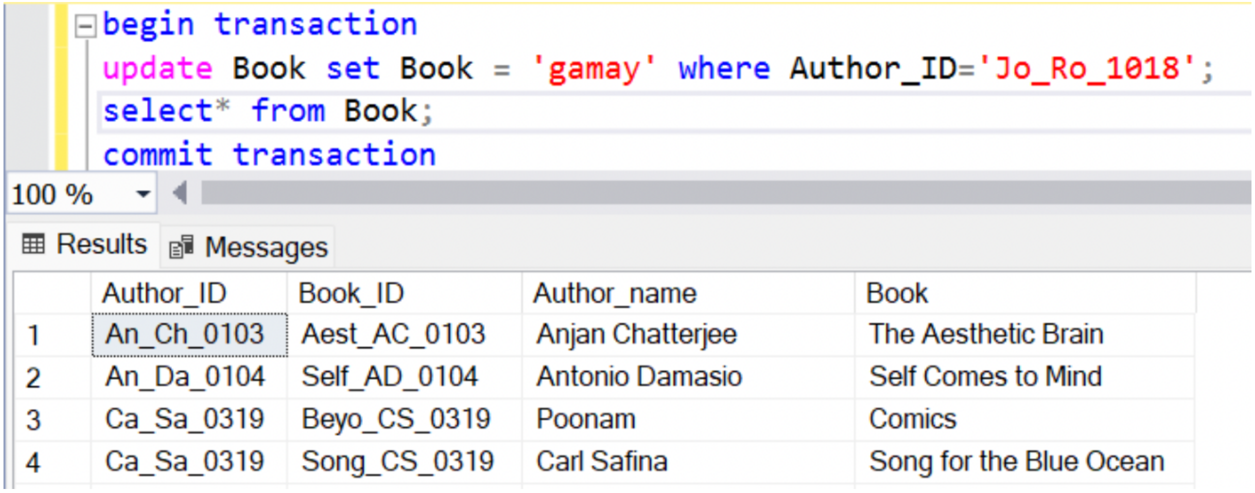


```
begin transaction
update Book set Book = 'gamay' where Author_ID='Jo_Ro_1018';
select* from Book;
commit transaction
```



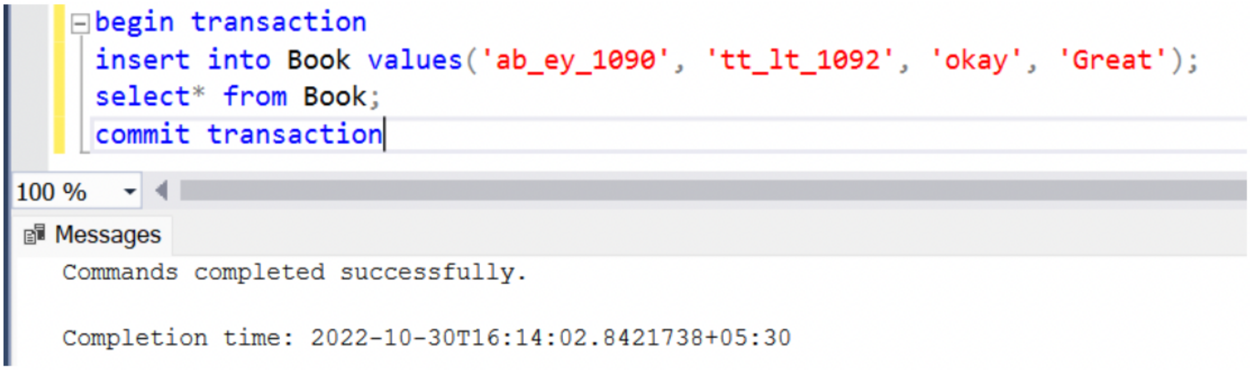
```
begin transaction
update Book set Book = 'gamax' where Author_ID='Jo_Ro_1018';
select* from Book;
rollback transaction
```

```
Commands completed successfully.
```



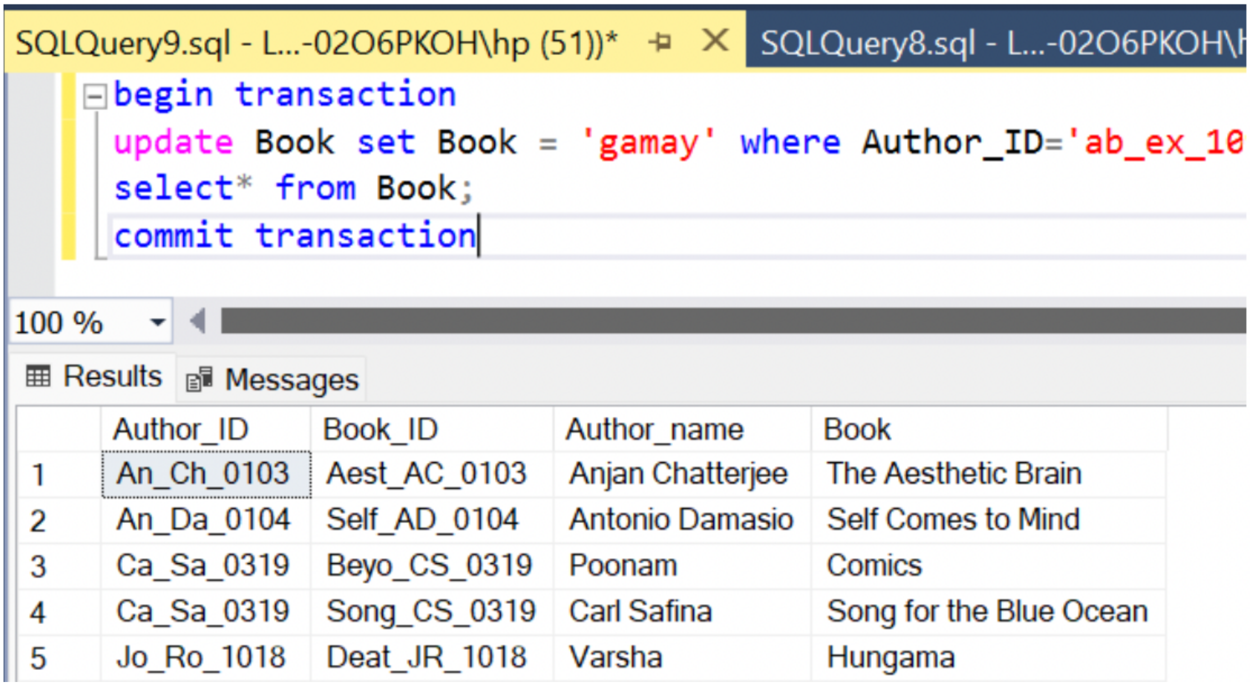
```
begin transaction
update Book set Book = 'gamay' where Author_ID='Jo_Ro_1018';
select* from Book;
commit transaction
```

	Author_ID	Book_ID	Author_name	Book
1	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	The Aesthetic Brain
2	An_Da_0104	Self_AD_0104	Antonio Damasio	Self Comes to Mind
3	Ca_Sa_0319	Beyo_CS_0319	Poonam	Comics
4	Ca_Sa_0319	Song_CS_0319	Carl Safina	Song for the Blue Ocean



```
begin transaction
insert into Book values('ab_ey_1090', 'tt_lt_1092', 'okay', 'Great');
select* from Book;
commit transaction
```

```
Commands completed successfully.

Completion time: 2022-10-30T16:14:02.8421738+05:30
```



SQLQuery9.sql - L...-02O6PKOH\hp (51))*
SQLQuery8.sql - L...-02O6PKOH\
100 %
Results
Messages

```
begin transaction
update Book set Book = 'gamay' where Author_ID='ab_ex_10
select* from Book;
commit transaction
```

	Author_ID	Book_ID	Author_name	Book
1	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	The Aesthetic Brain
2	An_Da_0104	Self_AD_0104	Antonio Damasio	Self Comes to Mind
3	Ca_Sa_0319	Beyo_CS_0319	Poonam	Comics
4	Ca_Sa_0319	Song_CS_0319	Carl Safina	Song for the Blue Ocean
5	Jo_Ro_1018	Deat_JR_1018	Varsha	Hungama

This situation will also lead to an irrecoverable schedule since session 2 updates on the newly entered row by session 1, which ultimately gets rolled back and lost.

To make them recoverable, we must employ a recoverable scheduling approach. A recoverable schedule is simply one in which a transaction's commit activity that performs a read operation is delayed until the uncommitted transaction commits or rolls back, or until all other transactions whose modifications they read commit.

Here, T2 was committed following T1's commit, or rolling back was an option. As a result, this has taken the shape of a recoverable timetable; the associated procedure is shown below.

Q8)

A recoverable schedule is one that cascades across time. A recoverable schedule is essentially one in which a specific transaction that conducts a read operation has its commit operation postponed until the uncommitted transaction commits or rolls backs.

When a transaction fails, a cascade rollback occurs, which results in the rollback of all dependent transactions. Cascading rollback's primary drawback is that it may waste CPU time.

An illustration of a cascading schedule is shown below -

T1	T2	T3	T4
Read(A)			
Write(A)			
	Read (A)		
	Write(A)		
		Read(A)	
		Write(A)	
			Read(A)
			Write(A)
Failure			

Cascadeless schedule: These sorts of schedules are known as cascadeless schedules because they prevent a transaction from reading data until the last transaction that wrote it has committed or aborted. Here, transaction T2 doesn't read the new value of X until after transaction T1 has committed.

When a transaction is not allowed to read data until the last transaction which has written it is committed or aborted, these types of schedules are called cascadeless schedules.

Given below is an example of a cascadeless schedule −

T1	T2
R(X)	
W(X)	
	W(X)
commit	
	R(X)
	Commit