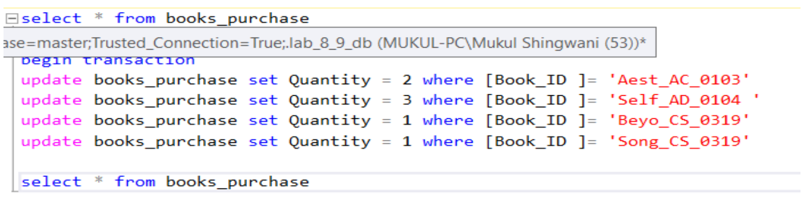
Data Engineering
Lab 8 and 9
Mukul Shingwani (B20AI023)

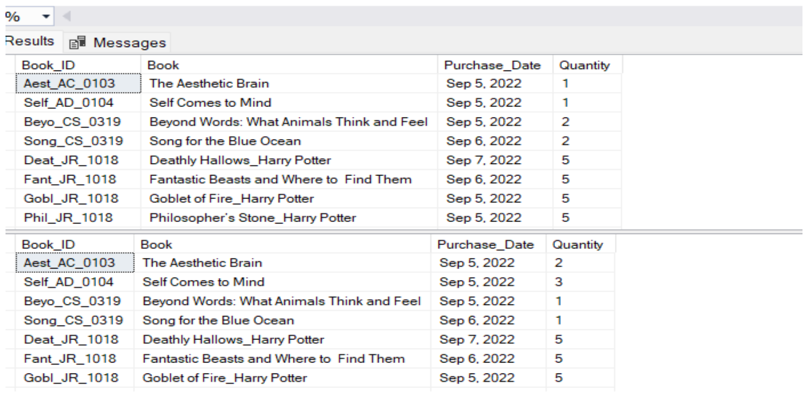
— ---

Q1) A)



```
ase=master;Trusted_Connection=True;.lab_8_9_db (MUKUL-PC\Mukul Shingwani (53))*
```

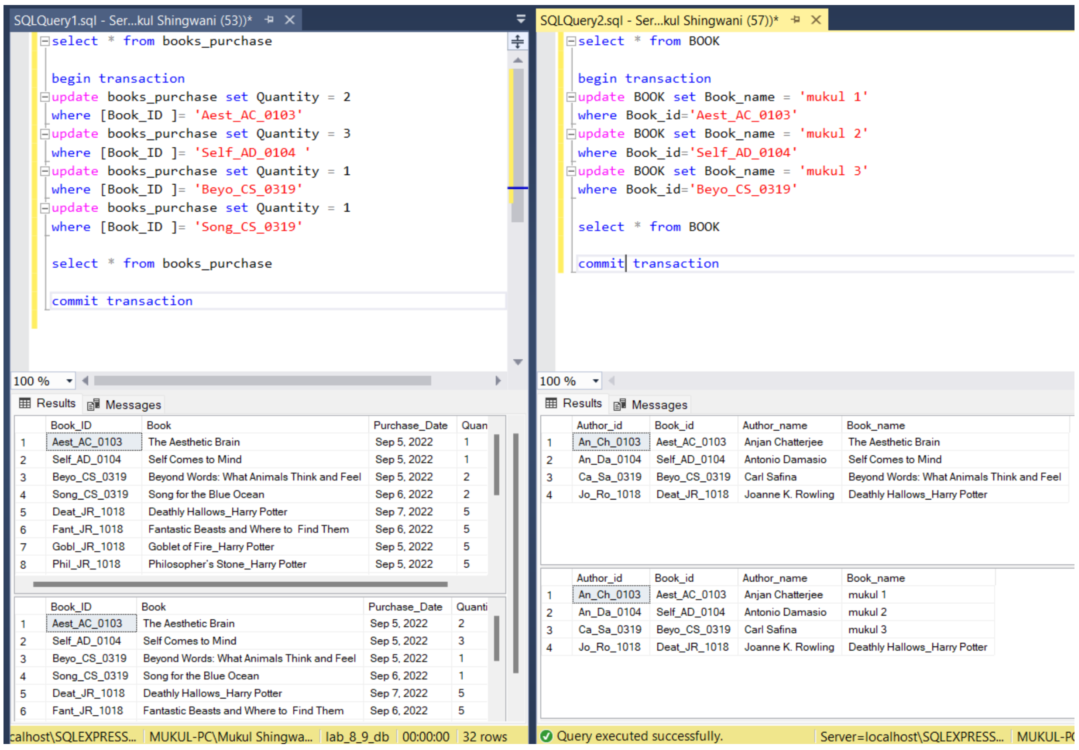
```sql
select * from books_purchase
begin transaction
update books_purchase set Quantity = 2 where [Book_ID ]= 'Aest_AC_0103'
update books_purchase set Quantity = 3 where [Book_ID ]= 'Self_AD_0104 '
update books_purchase set Quantity = 1 where [Book_ID ]= 'Beyo_CS_0319'
update books_purchase set Quantity = 1 where [Book_ID ]= 'Song_CS_0319'

select * from books_purchase
```



% | Results | Messages

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

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



SQLQuery1.sql - Ser...kul Shingwani (53))*

```sql
select * from books_purchase

begin transaction
update books_purchase set Quantity = 2
where [Book_ID ]= 'Aest_AC_0103'
update books_purchase set Quantity = 3
where [Book_ID ]= 'Self_AD_0104 '
update books_purchase set Quantity = 1
where [Book_ID ]= 'Beyo_CS_0319'
update books_purchase set Quantity = 1
where [Book_ID ]= 'Song_CS_0319'

select * from books_purchase

commit transaction
```

100 % | Results | Messages

	Book_ID	Book	Purchase_Date	Quan
1	Aest_AC_0103	The Aesthetic Brain	Sep 5, 2022	1
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	Quanti
1	Aest_AC_0103	The Aesthetic Brain	Sep 5, 2022	2
2	Self_AD_0104	Self Comes to Mind	Sep 5, 2022	3
3	Beyo_CS_0319	Beyond Words: What Animals Think and Feel	Sep 5, 2022	1
4	Song_CS_0319	Song for the Blue Ocean	Sep 6, 2022	1
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

calhost\SQLEXPRESS... | MUKUL-PC\Mukul Shingwa... | lab_8_9_db | 00:00:00 | 32 rows

SQLQuery2.sql - Ser...kul Shingwani (57))*

```sql
select * from BOOK

begin transaction
update BOOK set Book_name = 'mukul 1'
where Book_id='Aest_AC_0103'
update BOOK set Book_name = 'mukul 2'
where Book_id='Self_AD_0104'
update BOOK set Book_name = 'mukul 3'
where Book_id='Beyo_CS_0319'

select * from BOOK

commit transaction
```

100 % | Results | Messages

	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	mukul 1
2	An_Da_0104	Self_AD_0104	Antonio Damasio	mukul 2
3	Ca_Sa_0319	Beyo_CS_0319	Carl Safina	mukul 3
4	Jo_Ro_1018	Deat_JR_1018	Joanne K. Rowling	Deathly Hallows_Harry Potter

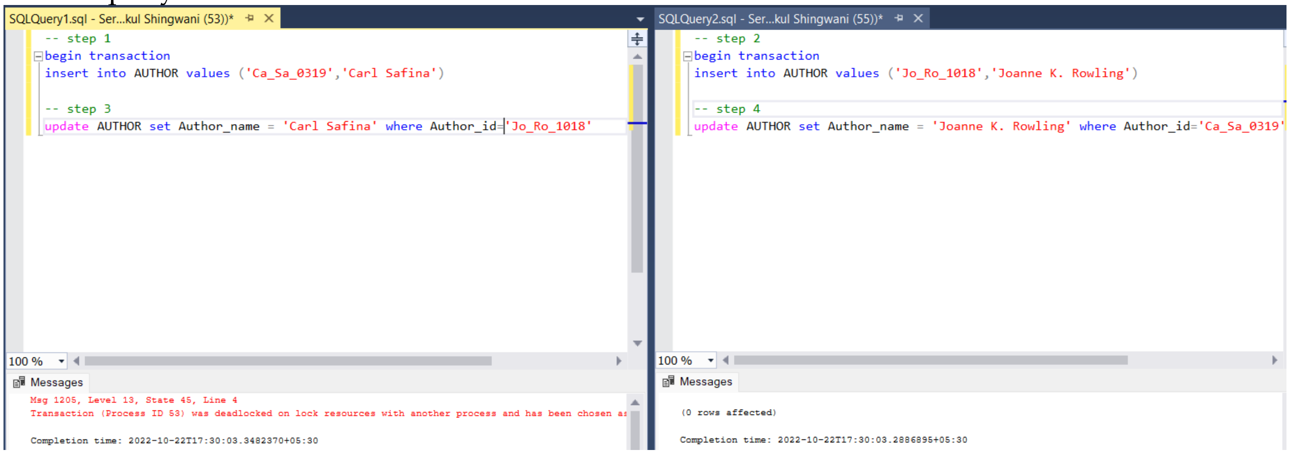
Query executed successfully. | Server=localhost\SQLEXPRESS... | MUKUL-P

From these images we can see, that multiple updates during one transaction, and multiple transaction at the same time are accessing the same database, thus showing the property of multi-threaded and multi-transaction handling

Q1) B) i)

Following were the steps of execution:-

- Insertion in Author was done in query 1 (txn1)
- Insertion in Author was done in query 2 (txn2)
- Tuple in AUTHOR was attempted to update where author_id = 'Jo_Ro_1018' in query 1
- Tuple in AUTHOR was attempted to update where author_id = 'Ca_Sa_0319' in query 2


SQLQuery1.sql - Ser...kul Shingwani (53))*

```
-- step 1
begin transaction
insert into AUTHOR values ('Ca_Sa_0319','Carl Safina')

-- step 3
update AUTHOR set Author_name = 'Carl Safina' where Author_id='Jo_Ro_1018'
```

Messages

```
Msg 1205, Level 13, State 45, Line 4
Transaction (Process ID 53) was deadlocked on lock resources with another process and has been chosen as

Completion time: 2022-10-22T17:30:03.3482370+05:30
```

SQLQuery2.sql - Ser...kul Shingwani (55))*

```
-- step 2
begin transaction
insert into AUTHOR values ('Jo_Ro_1018','Joanne K. Rowling')

-- step 4
update AUTHOR set Author_name = 'Joanne K. Rowling' where Author_id='Ca_Sa_0319'
```

Messages

```
(0 rows affected)

Completion time: 2022-10-22T17:30:03.2886895+05:30
```

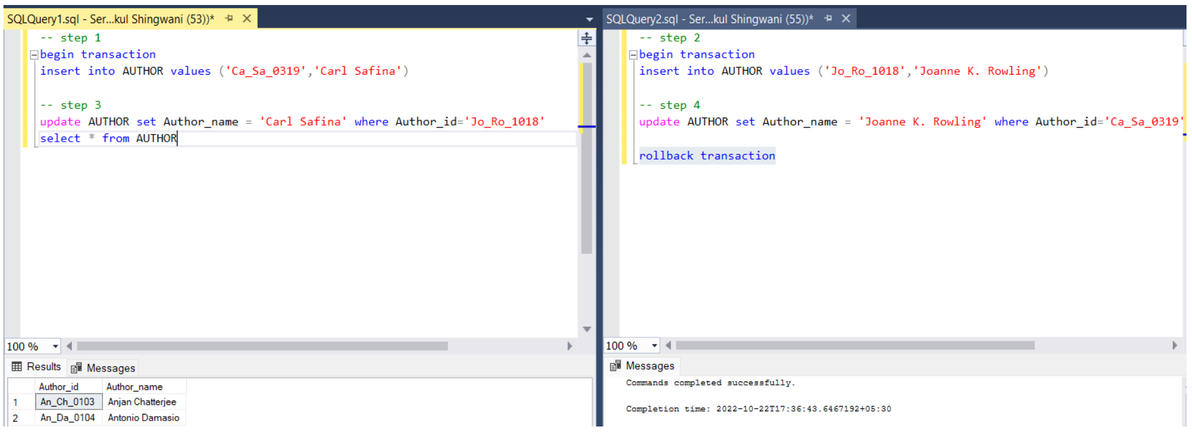
```
-- Session 1: Step 1.
BEGIN TRANSACTION
INSERT INTO AUTHOR
VALUES ('Ca_Sa_0319','Carl Safina')
-- ----------------------
-- Session 2: Step 2.
BEGIN TRAN
INSERT INTO AUTHOR
VALUES ('Jo_Ro_1018','Joanne K. Rowling')
-- ----------------------
-- Session 1: Step 3.
update AUTHOR set Author_name = 'Carl Safina' where Author_id='Jo_Ro_1018'
-- ----------------------
-- Session 2: Step 4.
update AUTHOR set Author_name = 'Joanne K. Rowling' where Author_id='Ca_Sa_0319'
```

Q1) B) ii)

I would have rolled back the second transaction i.e session 2 since it's first execution started later than query 1, so rolling it back will make query 1 run, this could be thought of as a FIFO (first-in-first-out) approach, since query 1 came first let it be completed first or given priority over other transactions.

Q1) B) iii)

As can been seen, after rolling back session 2, session 1 which was the victim of deadlock, successfully executed its operations.


SQLQuery1.sql - Ser...kul Shingwani (53))*

```sql
-- step 1
begin transaction
insert into AUTHOR values ('Ca_Sa_0319','Carl Safina')

-- step 3
update AUTHOR set Author_name = 'Carl Safina' where Author_id='Jo_Ro_1018'
select * from AUTHOR
```

Results | Messages

	Author_id	Author_name
1	An_Ch_0103	Anjan Chatterjee
2	An_Da_0104	Antonio Damasio

SQLQuery2.sql - Ser...kul Shingwani (55))*

```sql
-- step 2
begin transaction
insert into AUTHOR values ('Jo_Ro_1018','Joanne K. Rowling')

-- step 4
update AUTHOR set Author_name = 'Joanne K. Rowling' where Author_id='Ca_Sa_0319'

rollback transaction
```

Messages

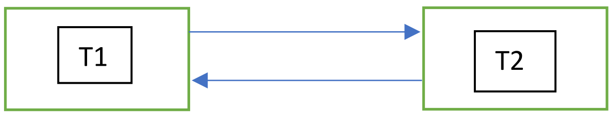
```
Commands completed successfully.

Completion time: 2022-10-22T17:36:43.6467192+05:30
```

Q1) B) iv)

This code is not conflict serializable, since it cannot be resolved into a serial schedule i.e. one of its conflict equivalent and neither its precedence graph is acyclic.

T1	T2
W(A)	
	W(A)
W(A)	
	W(A)

Precedence graph



Cyclic!!

Q2A)

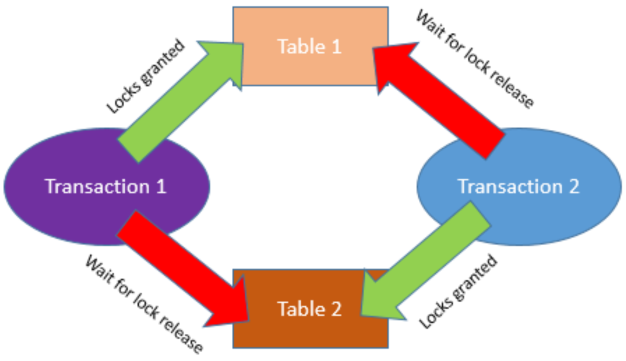
Following were the steps of execution :-

- Insertion in books_purchase was done in query 1 (txn1)
- Insertion in books_purchase was done in query 2 (txn2)
- Tuple in books_purchase was attempted to update where book_id = ‘pqrs_ZZ_10‘in query 1
- Tuple in books_purchase was attempted to update where book_id = ‘abcd_YY_12 ‘in query 2

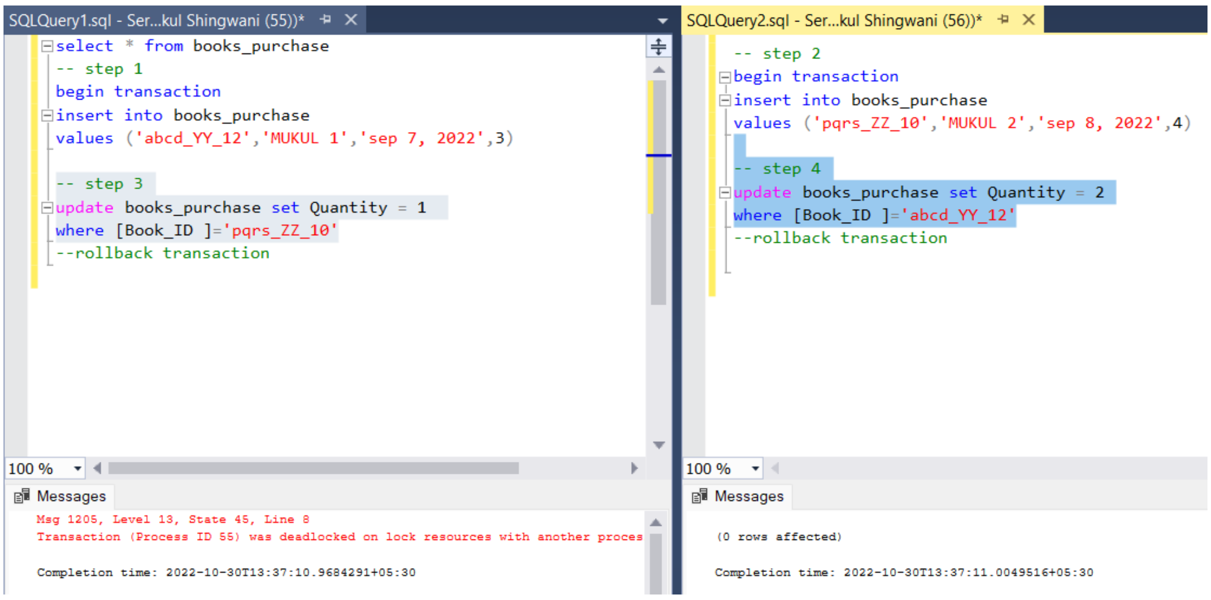
```sql
select * from books_purchase
-- T1 → step 1
begin transaction
insert into books_purchase
values ('abcd_YY_12','MUKUL 1','sep 7, 2022',3)
-----------------------------------
-- T2 → step 2
begin transaction
insert into books_purchase
```

```
values ('pqrs_ZZ_10','MUKUL 2','sep 8, 2022',4)
-----------------------------------
-- T1 → step 3
update books_purchase set Quantity = 1
where [Book_ID] = 'pqrs_ZZ_10'

select * from books_purchase
-----------------------------------
-- T2 → step 4
update books_purchase set Quantity = 2
where [Book_ID] = 'abcd_YY_12'
rollback transaction
```

In a database, a deadlock occurs when two or more processes have a resource locked, and each process requests a lock on the resource that another process has already locked. Neither of the transactions here can move forward, as each one is waiting for the other to release the lock



Since all the necessary conditions are seen to be satisfying, hence the situation is indeed of a deadlock, reference can be drawn from the diagram as well and query snapshots are also attached below.


SQLQuery1.sql - Ser...kul Shingwani (55))*

```
select * from books_purchase
-- step 1
begin transaction
insert into books_purchase
values ('abcd_YY_12','MUKUL 1','sep 7, 2022',3)

-- step 3
update books_purchase set Quantity = 1
where [Book_ID ]='pqrs_ZZ_10'
--rollback transaction
```

100 %

Messages

```
Msg 1205, Level 13, State 45, Line 8
Transaction (Process ID 55) was deadlocked on lock resources with another proces

Completion time: 2022-10-30T13:37:10.9684291+05:30
```

SQLQuery2.sql - Ser...kul Shingwani (56))*

```
-- step 2
begin transaction
insert into books_purchase
values ('pqrs_ZZ_10','MUKUL 2','sep 8, 2022',4)

-- step 4
update books_purchase set Quantity = 2
where [Book_ID ]='abcd_YY_12'
--rollback transaction
```

100 %

Messages

```
(0 rows affected)

Completion time: 2022-10-30T13:37:11.0049516+05:30
```

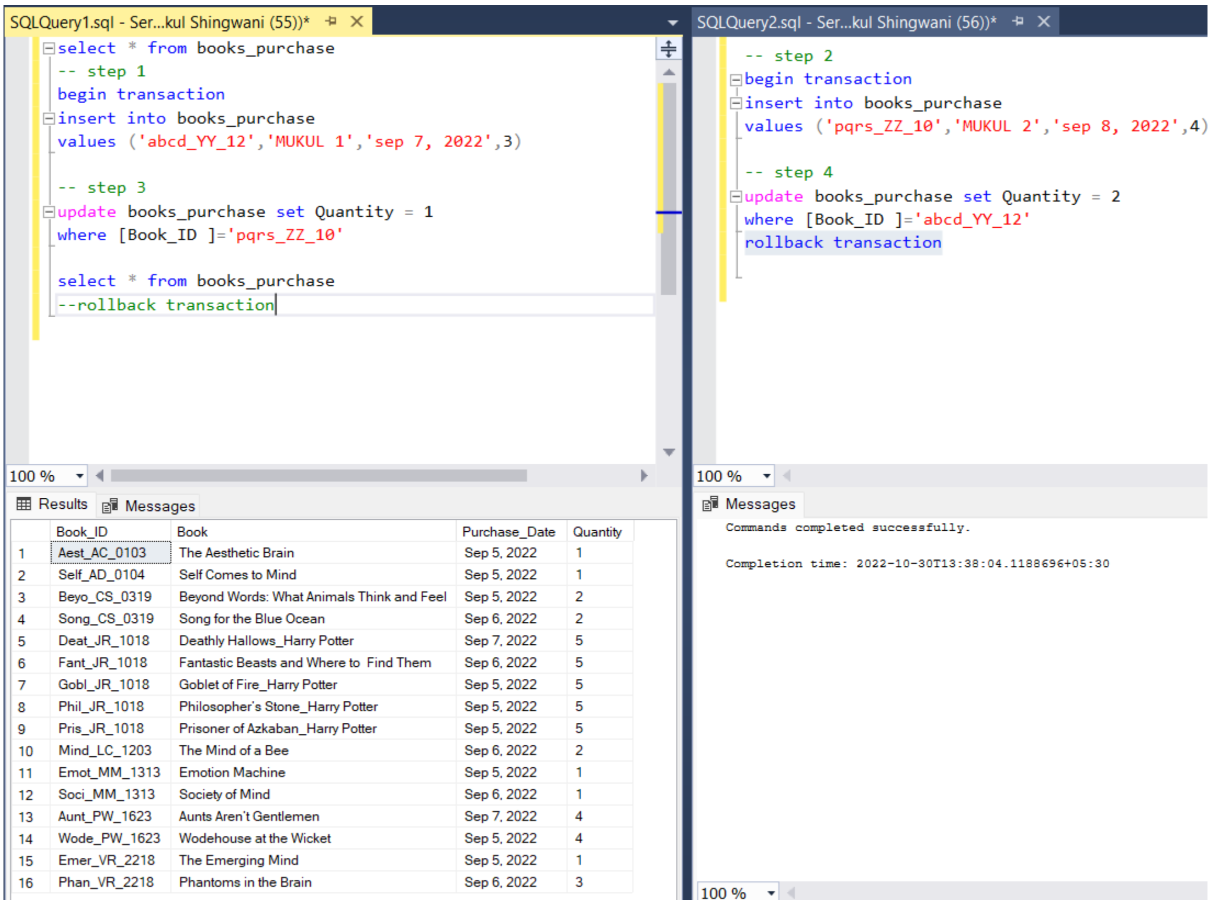
When deadlock occurs the system will choose one of the processes as the victim for deadlock, rollback that process so that the other could move forward.

Q2) B)

I would have rolled back the second transaction i.e. query 2 since it's first execution started later than query 1, so rolling it back will make query 1 run, this could be thought of as a FIFO (first-in-first-out) approach, since query 1 came first let it be completed first or given priority over other transactions.

Q2) C)

As can been seen below, after rolling back query 2, query 1 where deadlock caused a block, successfully executed its operations.



SQLQuery1.sql - Ser...kul Shingwani (55))*

```sql
select * from books_purchase
-- step 1
begin transaction
insert into books_purchase
values ('abcd_YY_12','MUKUL 1','sep 7, 2022',3)

-- step 3
update books_purchase set Quantity = 1
where [Book_ID ]='pqrs_ZZ_10'

select * from books_purchase
--rollback transaction
```

SQLQuery2.sql - Ser...kul Shingwani (56))*

```sql
-- step 2
begin transaction
insert into books_purchase
values ('pqrs_ZZ_10','MUKUL 2','sep 8, 2022',4)

-- step 4
update books_purchase set Quantity = 2
where [Book_ID ]='abcd_YY_12'
rollback transaction
```

	Book_ID	Book	Purchase_Date	Quantity
1	Aest_AC_0103	The Aesthetic Brain	Sep 5, 2022	1
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
9	Pris_JR_1018	Prisoner of Azkaban_Harry Potter	Sep 5, 2022	5
10	Mind_LC_1203	The Mind of a Bee	Sep 6, 2022	2
11	Emot_MM_1313	Emotion Machine	Sep 5, 2022	1
12	Soci_MM_1313	Society of Mind	Sep 6, 2022	1
13	Aunt_PW_1623	Aunts Aren't Gentlemen	Sep 7, 2022	4
14	Wode_PW_1623	Wodehouse at the Wicket	Sep 5, 2022	4
15	Emer_VR_2218	The Emerging Mind	Sep 5, 2022	1
16	Phan_VR_2218	Phantoms in the Brain	Sep 6, 2022	3

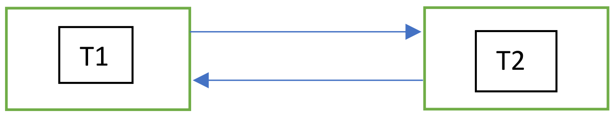
```
Commands completed successfully.

Completion time: 2022-10-30T13:38:04.1188696+05:30
```

Q2) D)

This code is not conflict serializable, since it cannot be resolved into a serial schedule i.e. one of its conflict equivalent and neither its precedence graph is acyclic.

T1	T2
W(A)	
	W(A)
W(A)	
	W(A)

Precedence graph



Cyclic!!

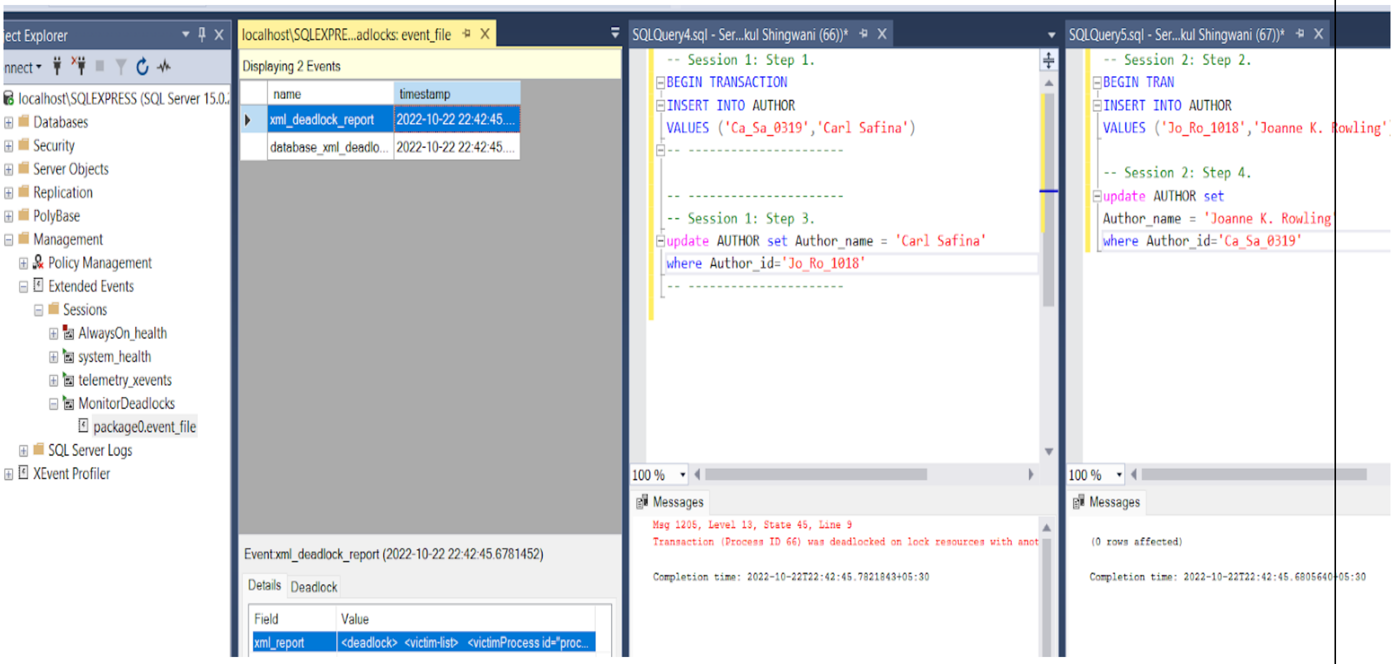
Q3)

- Check the ***system_health*** session for deadlocks
- Create an **extended event** session to capture the deadlocks
- Analyse the deadlock reports and graphs to figure out the problem
- If it is possible to make improvements or change the queries involved in the deadlock

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()))) A
S 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.


ect Explorer

localhost\SQLEXPRESS (SQL Server 15.0.

- Databases
- Security
- Server Objects
- Replication
- PolyBase
- Management
  - Policy Management
  - Extended Events
    - Sessions
      - AlwaysOn_health
      - system_health
      - telemetry_xevents
      - MonitorDeadlocks
        - package0.event_file
  - SQL Server Logs
- XEvent Profiler

localhost\SQLEXPRE...adlocks: event_file

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)

Details | Deadlock

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

SQLQuery4.sql - Ser...kul Shingwani (66))*

```
-- 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'
-- ---------------------
```

Messages

```
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
```

SQLQuery5.sql - Ser...kul Shingwani (67))*

```
-- 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'
```

Messages

```
(0 rows affected)

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

Q4

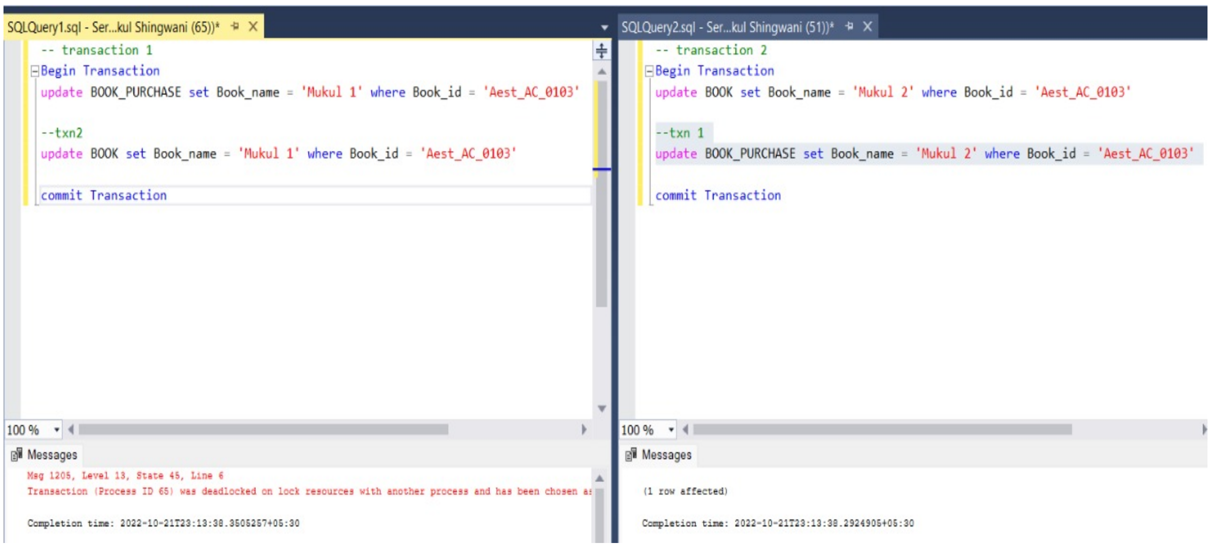
Following were the steps of execution :-

- BOOK_PURCHASE was updated in query 1 (txn1) → locked table 1
- BOOK was updated in query 2 (txn2) → locked table 2
- BOOK was attempted to update in query 1 → request from table 1
- BOOK_PURCHASE was attempted to update in query 2 → request from table 2

In a database, a deadlock occurs when two or more processes have a resource locked, and each process requests a lock on the resource that another process has already locked. Neither of the transactions here can move forward, as each one is waiting for the other to release the lock

```
-- Session 1: Step 1.
BEGIN TRANSACTION
-- txn 1
update BOOK_PURCHASE set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'
-- ----------------------
-- Session 2: Step 2.
BEGIN TRANSACTION
-- txn 2
update BOOK set Book_name = 'Mukul 2' where Book_id = 'Aest_AC_0103'
-- ----------------------
-- Session 1: Step 3.
--txn 2
update BOOK set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'
select * from BOOK
-- commit Transaction
-- ----------------------
-- Session 2: Step 4.
--txn 1
update BOOK_PURCHASE set Book_name = 'Mukul 2' where Book_id = 'Aest_AC_0103'
-- commit Transaction
```

Since all the necessary conditions are seen to be satisfying, hence the situation is indeed of a deadlock, reference can be drawn from the diagram as well and query snapshots are also attached below.



SQLQuery1.sql - Ser...kul Shingwani (65))*

```
-- transaction 1
Begin Transaction
update BOOK_PURCHASE set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'

--txn2
update BOOK set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'

commit Transaction
```

Messages

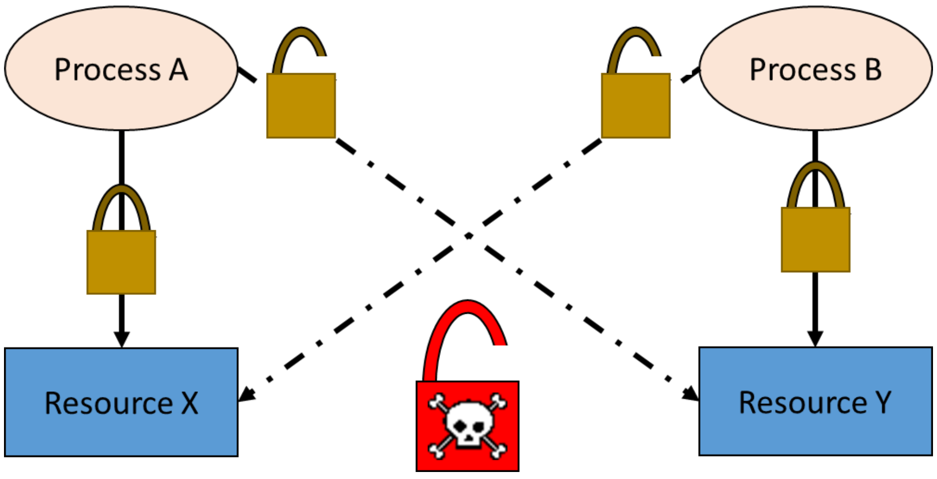
```
Msg 1205, Level 13, State 45, Line 6
Transaction (Process ID 65) was deadlocked on lock resources with another process and has been chosen as

Completion time: 2022-10-21T23:13:38.3505257+05:30
```

SQLQuery2.sql - Ser...kul Shingwani (51))*

```
-- transaction 2
Begin Transaction
update BOOK set Book_name = 'Mukul 2' where Book_id = 'Aest_AC_0103'

--txn 1
update BOOK_PURCHASE set Book_name = 'Mukul 2' where Book_id = 'Aest_AC_0103'

commit Transaction
```

Messages

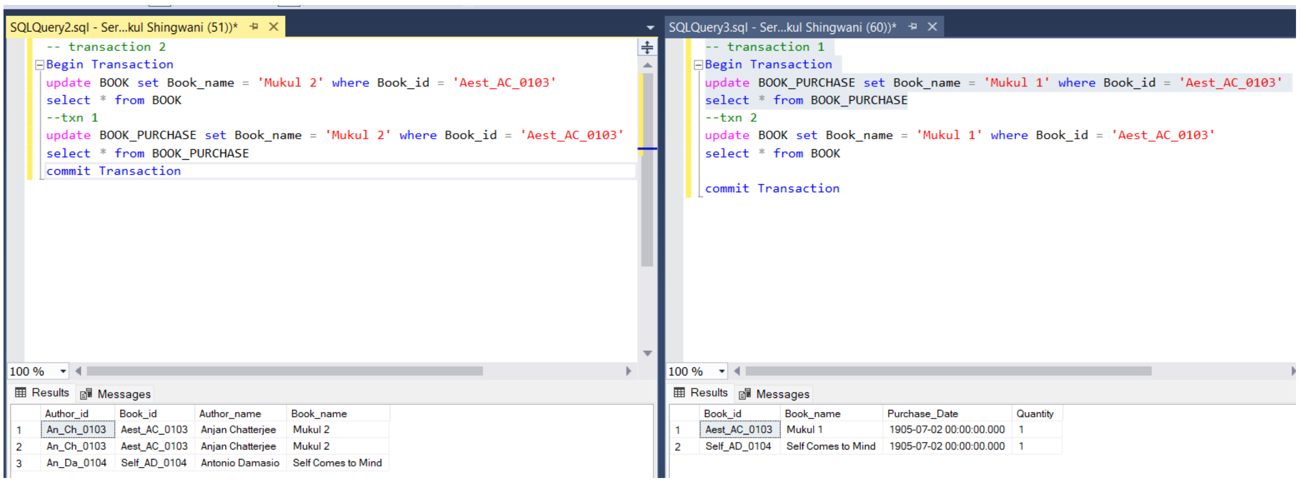
```
(1 row affected)

Completion time: 2022-10-21T23:13:38.2924905+05:30
```



Q5

Taking reference of the above code, we know there arises a deadlock situation, and using rollback it can be resolved as can be seen in the images below.

Query



SQLQuery2.sql - Ser...kul Shingwani (51))*

```
-- transaction 2
Begin Transaction
update BOOK set Book_name = 'Mukul 2' where Book_id = 'Aest_AC_0103'
select * from BOOK
--txn 1
update BOOK_PURCHASE set Book_name = 'Mukul 2' where Book_id = 'Aest_AC_0103'
select * from BOOK_PURCHASE
commit Transaction
```

Results

	Author_id	Book_id	Author_name	Book_name
1	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	Mukul 2
2	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	Mukul 2
3	An_Da_0104	Self_AD_0104	Antonio Damasio	Self Comes to Mind

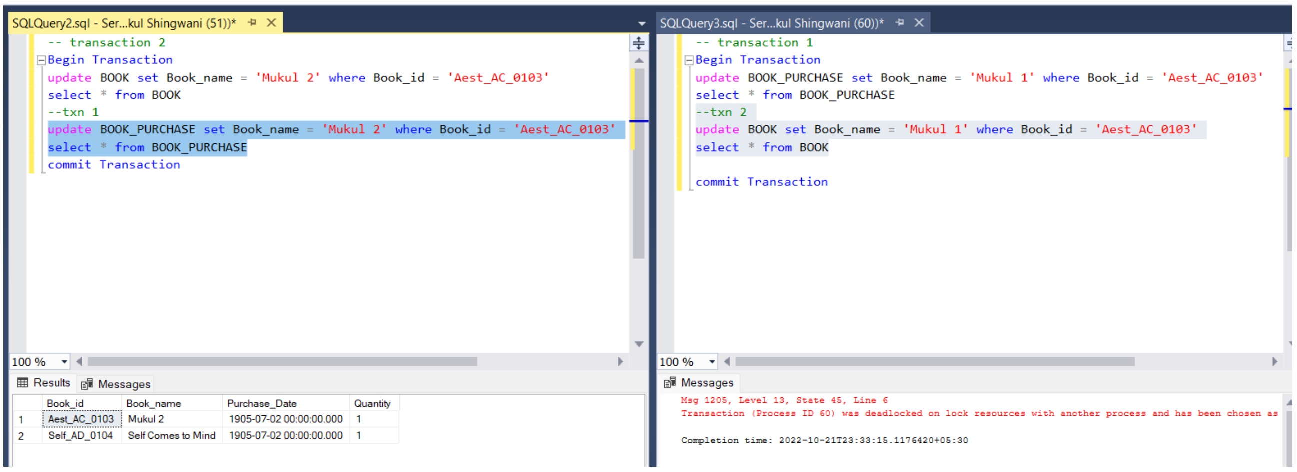
SQLQuery3.sql - Ser...kul Shingwani (60))*

```
-- transaction 1
Begin Transaction
update BOOK_PURCHASE set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'
select * from BOOK_PURCHASE
--txn 2
update BOOK set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'
select * from BOOK

commit Transaction
```

Results

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

Deadlock


SQLQuery2.sql - Ser...kul Shingwani (51))*
SQLQuery3.sql - Ser...kul Shingwani (60))*
100 %
Results
Messages

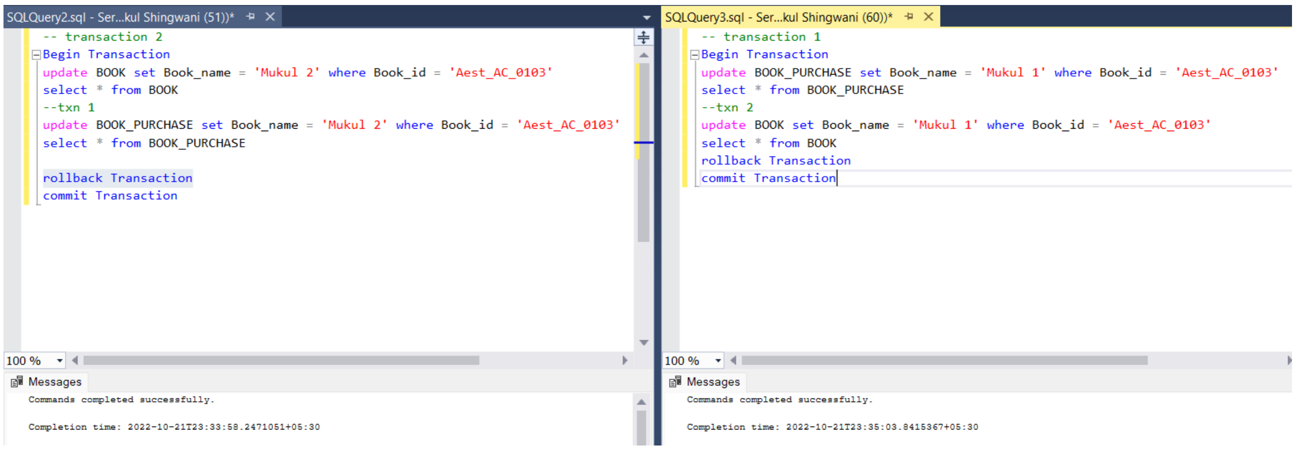
```
-- transaction 2
Begin Transaction
update BOOK set Book_name = 'Mukul 2' where Book_id = 'Aest_AC_0103'
select * from BOOK
--txn 1
update BOOK_PURCHASE set Book_name = 'Mukul 2' where Book_id = 'Aest_AC_0103'
select * from BOOK_PURCHASE
commit Transaction
```

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

```
-- transaction 1
Begin Transaction
update BOOK_PURCHASE set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'
select * from BOOK_PURCHASE
--txn 2
update BOOK set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'
select * from BOOK

commit Transaction
```

```
Msg 1205, Level 13, State 45, Line 6
Transaction (Process ID 60) was deadlocked on lock resources with another process and has been chosen as

Completion time: 2022-10-21T23:33:15.1176420+05:30
```

Rollback


SQLQuery2.sql - Ser...kul Shingwani (51))*
SQLQuery3.sql - Ser...kul Shingwani (60))*
100 %
Messages

```
-- transaction 2
Begin Transaction
update BOOK set Book_name = 'Mukul 2' where Book_id = 'Aest_AC_0103'
select * from BOOK
--txn 1
update BOOK_PURCHASE set Book_name = 'Mukul 2' where Book_id = 'Aest_AC_0103'
select * from BOOK_PURCHASE

rollback Transaction
commit Transaction
```

```
Commands completed successfully.

Completion time: 2022-10-21T23:33:58.2471051+05:30
```

```
-- transaction 1
Begin Transaction
update BOOK_PURCHASE set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'
select * from BOOK_PURCHASE
--txn 2
update BOOK set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'
select * from BOOK
rollback Transaction
commit Transaction
```

```
Commands completed successfully.

Completion time: 2022-10-21T23:35:03.8415367+05:30
```

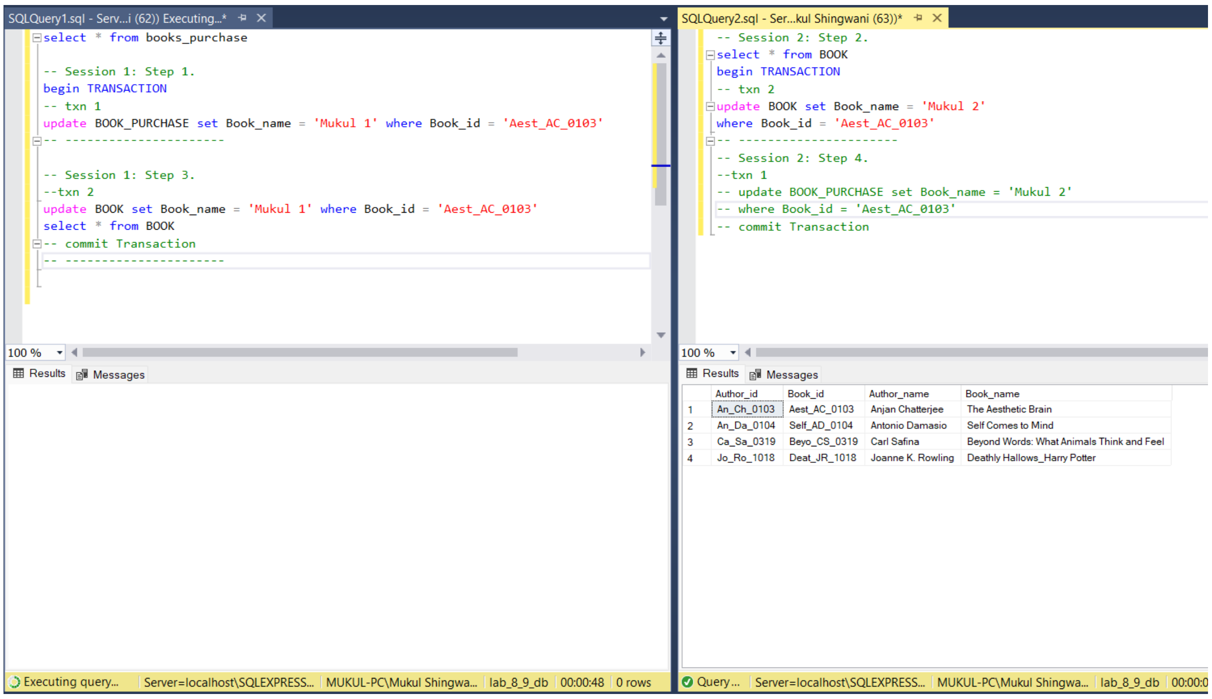
Q6

Wait and Die scheme

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

Let's assume there are two transactions Ti and Tj and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:

1. Check if TS(Ti) < TS(Tj) - If Ti is the older transaction and Tj has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.



```
SQLQuery1.sql - Serv...i (62)) Executing...*
select * from books_purchase

-- Session 1: Step 1.
begin TRANSACTION
-- txn 1
update BOOK_PURCHASE set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'
-- ---------------------

-- Session 1: Step 3.
--txn 2
update BOOK set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'
select * from BOOK
-- commit Transaction
-- ---------------------
```

```
Executing query... | Server=localhost\SQLEXPRESS... | MUKUL-PC\Mukul Shingwa... | lab_8_9_db | 00:00:48 | 0 rows
```

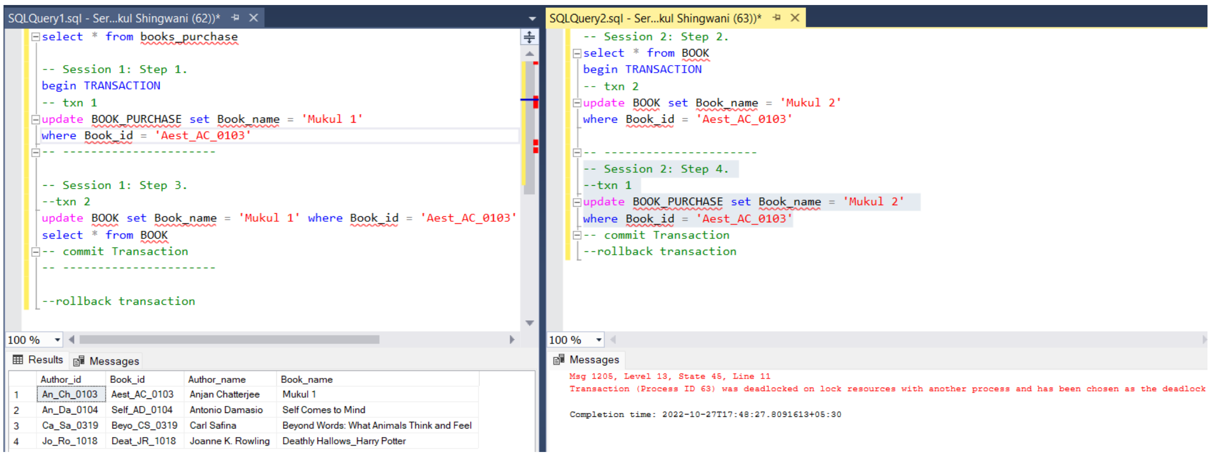
```
SQLQuery2.sql - Ser...kul Shingwani (63))*
-- Session 2: Step 2.
select * from BOOK
begin TRANSACTION
-- txn 2
update BOOK set Book_name = 'Mukul 2'
where Book_id = 'Aest_AC_0103'
-- ---------------------
-- Session 2: Step 4.
--txn 1
-- update BOOK_PURCHASE set Book_name = 'Mukul 2'
-- where Book_id = 'Aest_AC_0103'
-- 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

```
Query... | Server=localhost\SQLEXPRESS... | MUKUL-PC\Mukul Shingwa... | lab_8_9_db | 00:00:0
```

As can be seen in this image, T1 started earlier than T2 transaction, T1 had a lock on table BOOK_PURCHASE and T2 had a lock on BOOK table, Now, when T1 requested for BOOK table, it was allowed to wait, since it was the older transaction among the two.

2. Check if TS(T_i) < TS(Tj) - If Ti is older transaction and has held some resource and if Tj is waiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.



```
SQLQuery1.sql - Ser...kul Shingwani (62))*
select * from books_purchase

-- Session 1: Step 1.
begin TRANSACTION
-- txn 1
update BOOK_PURCHASE set Book_name = 'Mukul 1'
where Book_id = 'Aest_AC_0103'
-- ---------------------

-- Session 1: Step 3.
--txn 2
update BOOK set Book_name = 'Mukul 1' where Book_id = 'Aest_AC_0103'
select * from BOOK
-- commit Transaction
-- ---------------------

--rollback transaction
```

	Author_id	Book_id	Author_name	Book_name
1	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	Mukul 1
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

```
SQLQuery2.sql - Ser...kul Shingwani (63))*
-- Session 2: Step 2.
select * from BOOK
begin TRANSACTION
-- txn 2
update BOOK set Book_name = 'Mukul 2'
where Book_id = 'Aest_AC_0103'

-- ---------------------
-- Session 2: Step 4.
--txn 1
update BOOK_PURCHASE set Book_name = 'Mukul 2'
where Book_id = 'Aest_AC_0103'
-- commit Transaction
--rollback transaction
```

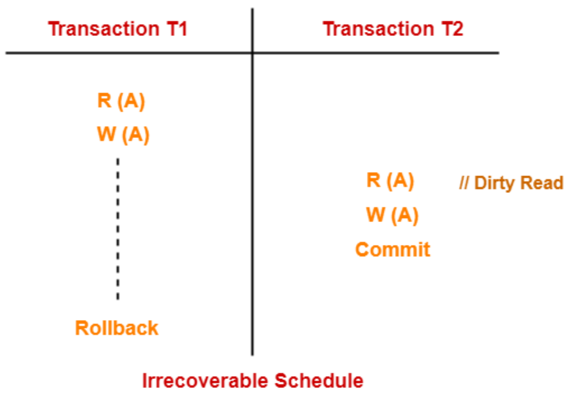
```
Msg 1205, Level 13, State 45, Line 11
Transaction (Process ID 63) was deadlocked on lock resources with another process and has been chosen as the deadlock

Completion time: 2022-10-27T17:48:27.8091613+05:30
```

As can be seen in this image, T1 started earlier than T2 transaction, T1 had a lock on table BOOK_PURCHASE and T2 had a lock on BOOK table, Now, when T1 requested for BOOK table, it was allowed to wait, but when T2 requested for a resource held by

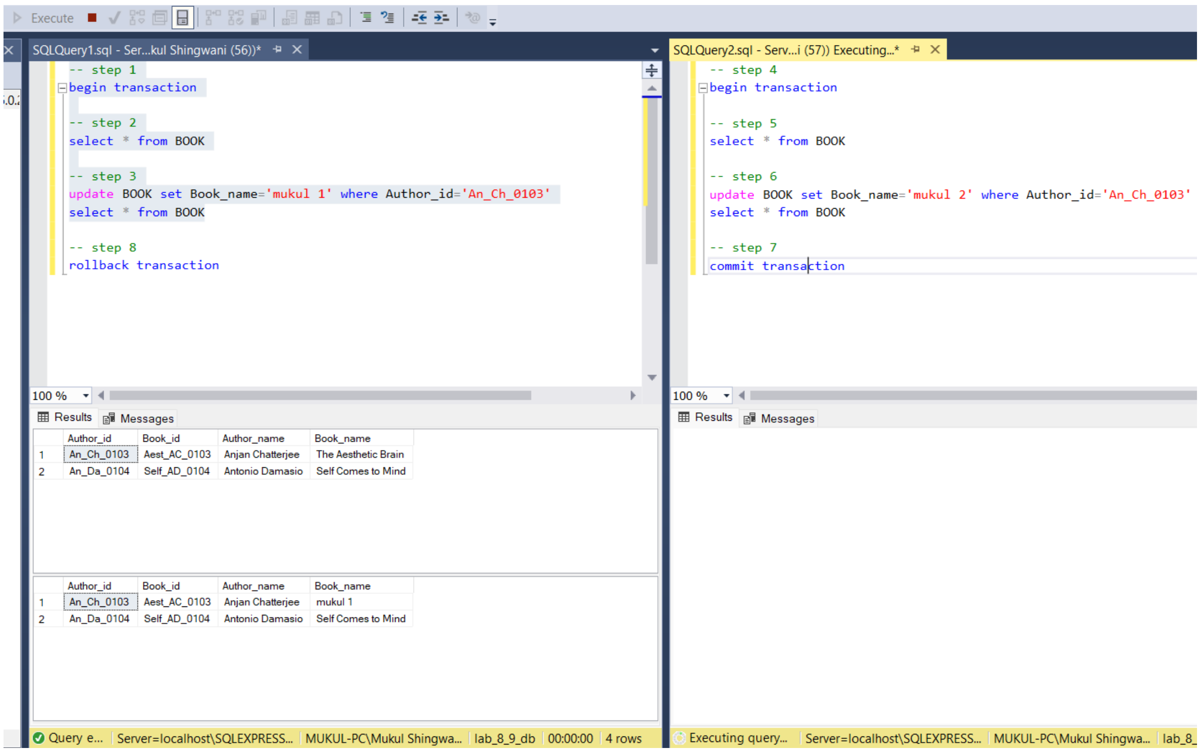
T1, T2 transaction got killed immediately and a deadlock warning came, since it was not older than T1 transaction.

Q7

If a transaction reads a data item after it is written by an uncommitted transaction it leads to an **irrevocable transaction**
i.e. a dirty read operation from an uncommitted transaction and commits before the transaction from where it has read the value, then such a schedule is called an irrecoverable schedule



Here such kind of a transaction flow was followed, where session 2 reads, writes and then commits before session 1 and finally session 1 rollbacks, but since session 2 has alread read the dirty values after write of session 1, it falls into the trap of Dirty Read problem, and this is an irrecoverable schedule since session 2 committed before session 1



SQLQuery1.sql - Ser...kul Shingwani (56))*

```
-- step 1
begin transaction

-- step 2
select * from BOOK

-- step 3
update BOOK set Book_name='mukul 1' where Author_id='An_Ch_0103'
select * from BOOK

-- step 8
rollback 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

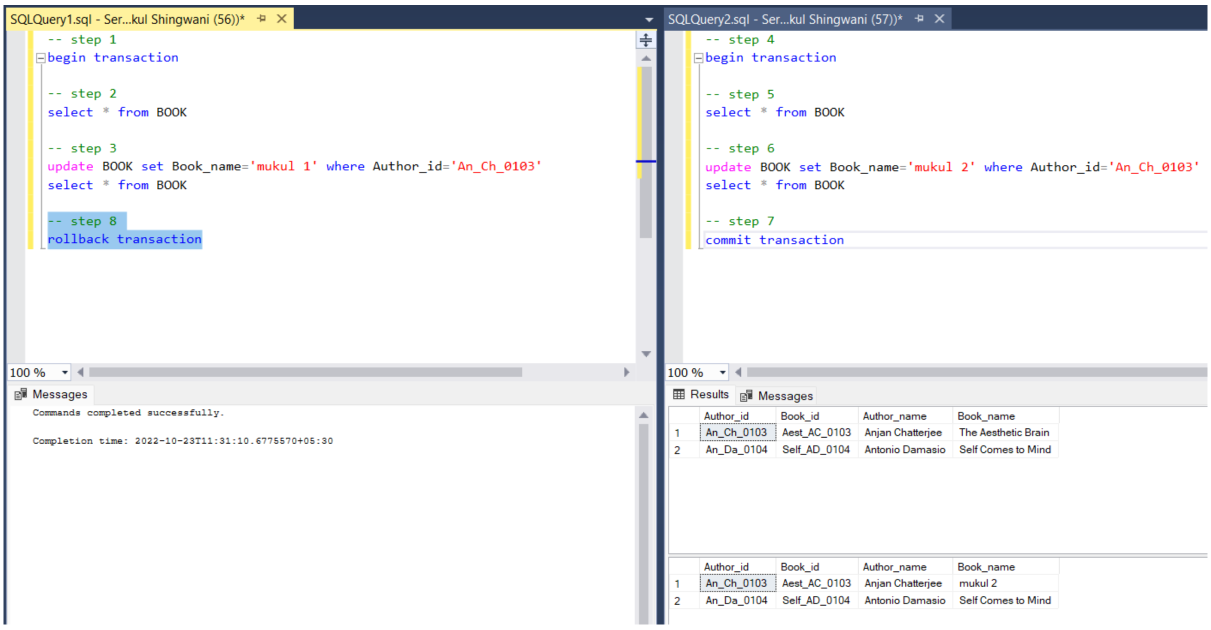
	Author_id	Book_id	Author_name	Book_name
1	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	mukul 1
2	An_Da_0104	Self_AD_0104	Antonio Damasio	Self Comes to Mind

Query e... | Server=localhost\SQLEXPRESS... | MUKUL-PC\Mukul Shingwa... | lab_8_9_db | 00:00:00 | 4 rows

SQLQuery2.sql - Serv...i (57)) Executing...*

```
-- step 4
begin transaction

-- step 5
select * from BOOK

-- step 6
update BOOK set Book_name='mukul 2' where Author_id='An_Ch_0103'
select * from BOOK

-- step 7
commit transaction
```

Executing query... | Server=localhost\SQLEXPRESS... | MUKUL-PC\Mukul Shingwa... | lab_8_


SQLQuery1.sql - Ser...kul Shingwani (56))*

```sql
-- step 1
begin transaction

-- step 2
select * from BOOK

-- step 3
update BOOK set Book_name='mukul 1' where Author_id='An_Ch_0103'
select * from BOOK

-- step 8
rollback transaction
```

Messages

```
Commands completed successfully.

Completion time: 2022-10-23T11:31:10.6775570+05:30
```

SQLQuery2.sql - Ser...kul Shingwani (57))*

```sql
-- step 4
begin transaction

-- step 5
select * from BOOK

-- step 6
update BOOK set Book_name='mukul 2' where Author_id='An_Ch_0103'
select * from BOOK

-- step 7
commit transaction
```

Results

	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

	Author_id	Book_id	Author_name	Book_name
1	An_Ch_0103	Aest_AC_0103	Anjan Chatterjee	mukul 2
2	An_Da_0104	Self_AD_0104	Antonio Damasio	Self Comes to Mind

T1

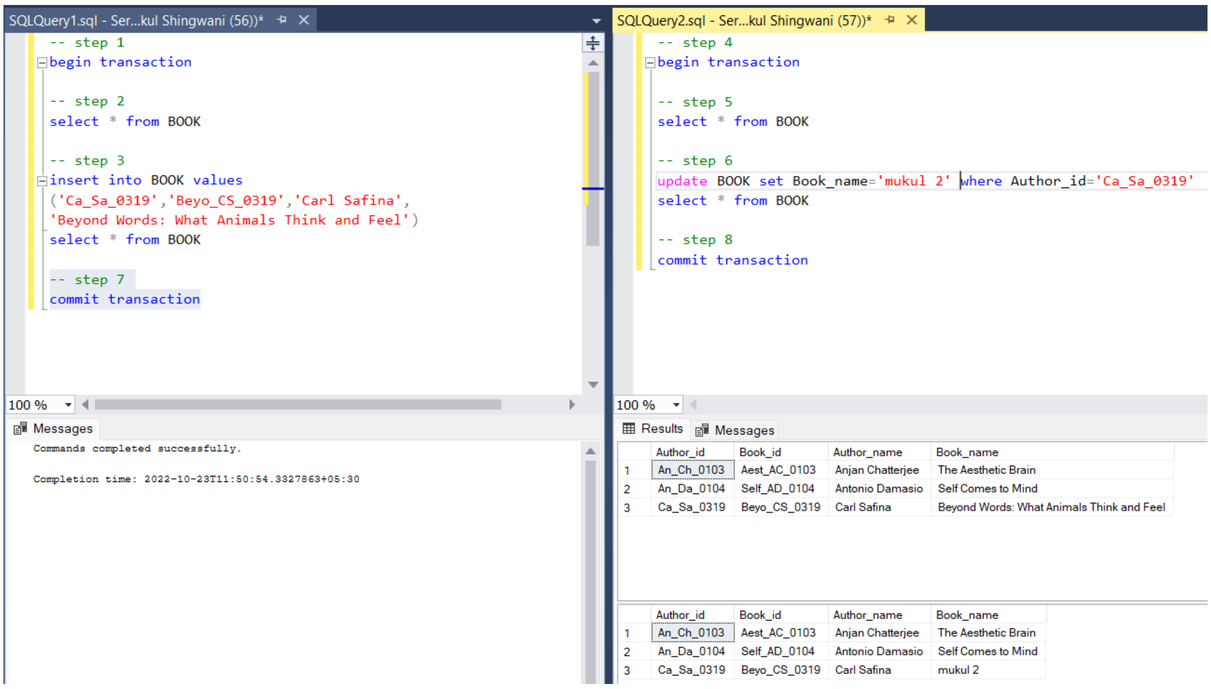
```sql
-- step 1
begin transaction


-- step 2

select * from BOOK



-- step 3
insert into BOOK values
('Ca_Sa_0319','Beyo_CS_0319','Carl Safina',
'Beyond Words: What Animals Think and Feel')
select * from BOOK

-- step 8
rollback transaction
```

T2

```sql
-- step 4

begin transaction

-- step 5
select * from BOOK

-- step 6
update BOOK set Book_name='mukul 2'
where Author_id='Ca_Sa_0319'
select * from BOOK



-- step 7
commit transaction
```

This example will also lead to an irrecoverable schedule, since T2 updates on the newly inserted row by T1, which finally gets rolled back and lost, so T2 updates are not meaningful.

To make them recoverable we need to follow a recoverable schedule approach
A **recoverable schedule** is basically a schedule in which the commit operation of a particular transaction that performs read operation is delayed until the uncommitted transaction either commits or roll backs
i.e. schedules in which transactions commit only after all transactions whose changes they read commits.


SQLQuery1.sql - Ser...kul Shingwani (56))*

```sql
-- step 1
begin transaction

-- step 2
select * from BOOK

-- step 3
insert into BOOK values
('Ca_Sa_0319','Beyo_CS_0319','Carl Safina',
'Beyond Words: What Animals Think and Feel')
select * from BOOK

-- step 7
commit transaction
```

Messages

```
Commands completed successfully.

Completion time: 2022-10-23T11:50:54.3327863+05:30
```

SQLQuery2.sql - Ser...kul Shingwani (57))*

```sql
-- step 4
begin transaction

-- step 5
select * from BOOK

-- step 6
update BOOK set Book_name='mukul 2' where Author_id='Ca_Sa_0319'
select * from BOOK

-- step 8
commit transaction
```

Results | Messages

	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

	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	mukul 2

Here, T2 was committed after T1 committed or rollback was also an option, Now this has shaped into a form of a recoverable schedule, and the workflow for the same is given below.

T1	T2

`-- step 1`
`begin transaction`	

	`-- step 4`
	`begin transaction`

`-- step 2`
`select * from BOOK`	

	`-- step 5`
	`select * from BOOK`

	`-- step 6`
	`update BOOK set Book_name='mukul 2'`
	`where Author_id='Ca_Sa_0319'`
	`select * from BOOK`

`-- step 3`
`insert into BOOK values`
`('Ca_Sa_0319','Beyo_CS_0319','Carl Safina',`
`'Beyond Words: What Animals Think and Feel')`
`select * from BOOK`	

`-- step 7`
`commit transaction`	

	`-- step 8`
	`commit transaction`

Q8)

A cascading rollback occurs in database systems when a transaction (T1) causes a failure and a rollback must be performed. Other transactions dependent on T1's actions must also be rollbacked due to T1's failure, thus causing a cascading effect. That is, one transaction's failure causes many to fail.

T1	T2	T3
R (A)		
W (A)		
	R (A)	
	W (A)	
		R (A)
		W (A)
Failure		

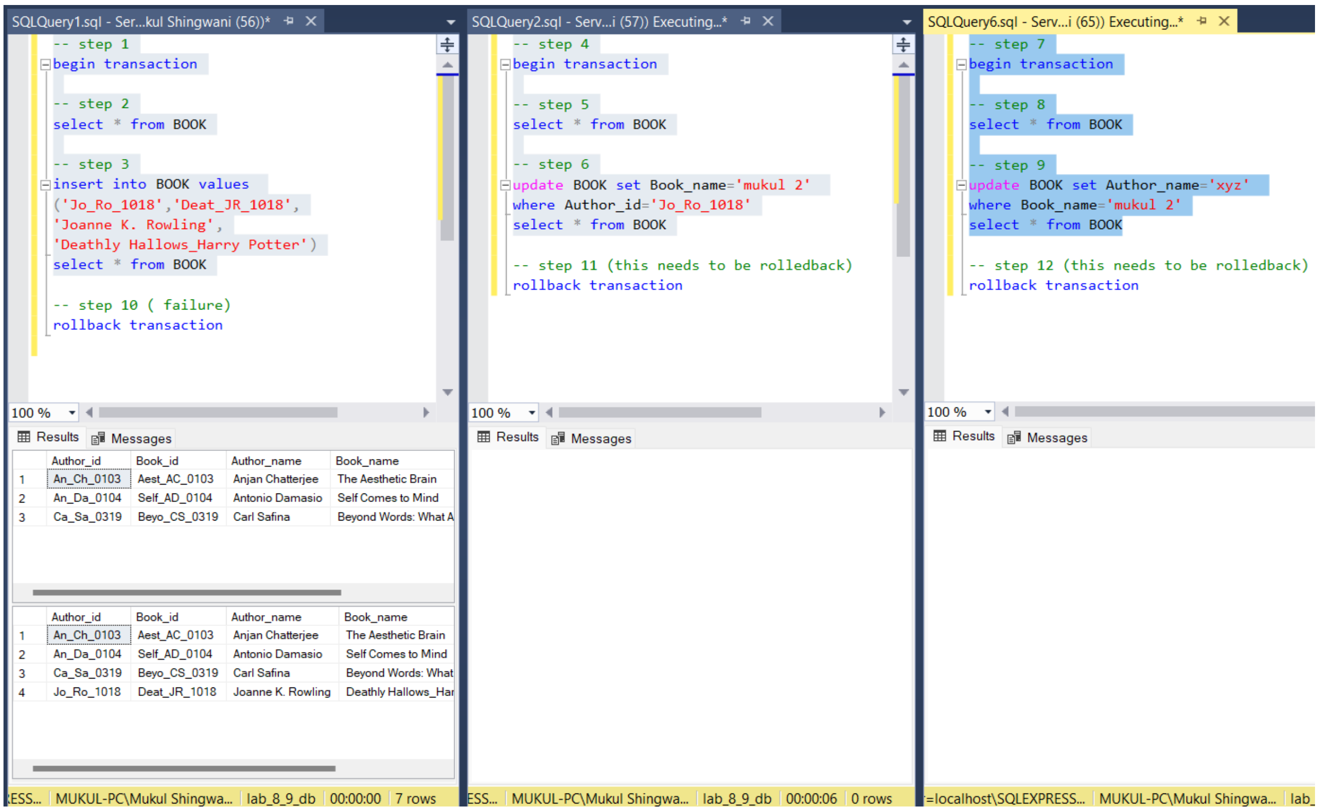
Here,

- Transaction T2 depends on transaction T1.
- Transaction T3 depends on transaction T2.

In this schedule,

- The failure of transaction T1 causes the transaction T2 to rollback.
- The rollback of transaction T2 causes the transaction T3 to rollback.

Such a rollback is called as a **Cascading Rollback**.



SQLQuery1.sql - Ser...kul Shingwani (56))*

```
-- step 1
begin transaction

-- step 2
select * from BOOK

-- step 3
insert into BOOK values
('Jo_Ro_1018','Deat_JR_1018',
'Joanne K. Rowling',
'Deathly Hallows_Harry Potter')
select * from BOOK

-- step 10 ( failure)
rollback 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 A

	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
4	Jo_Ro_1018	Deat_JR_1018	Joanne K. Rowling	Deathly Hallows_Har

```
ESS... | MUKUL-PC\Mukul Shingwa... | lab_8_9_db | 00:00:00 | 7 rows
```

SQLQuery2.sql - Serv...i (57)) Executing...*

```
-- step 4
begin transaction

-- step 5
select * from BOOK

-- step 6
update BOOK set Book_name='mukul 2'
where Author_id='Jo_Ro_1018'
select * from BOOK

-- step 11 (this needs to be rolledback)
rollback transaction
```

```
ESS... | MUKUL-PC\Mukul Shingwa... | lab_8_9_db | 00:00:06 | 0 rows
```

SQLQuery6.sql - Serv...i (65)) Executing...*

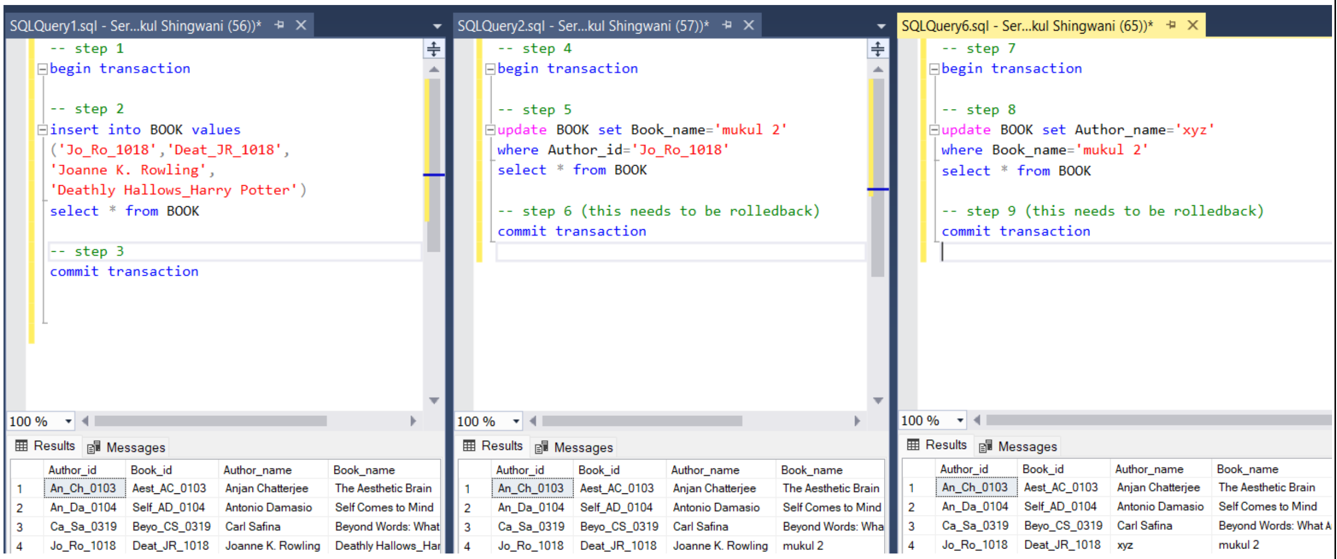
```
-- step 7
begin transaction

-- step 8
select * from BOOK

-- step 9
update BOOK set Author_name='xyz'
where Book_name='mukul 2'
select * from BOOK

-- step 12 (this needs to be rolledback)
rollback transaction
```

```
=localhost\SQLEXPRESS... | MUKUL-PC\Mukul Shingwa... | lab_
```

T1	T2	T3
-- step 1 begin transaction	-- step 4 begin transaction	-- step 7 begin transaction
-- step 2 select * from BOOK	-- step 5 select * from BOOK	-- step 8 select * from BOOK
-- step 3 insert into BOOK values	-- step 6 update BOOK set Book_name ='mukul 2' where Author_id ='Jo_Ro_1018' select * from BOOK	
('Jo_Ro_1018','Deat_JR_1018', 'Joanne K. Rowling', 'Deathly Hallows_Harry Potter') select * from BOOK		-- step 9 update BOOK set Author_name='xyz' where Book_name='mukul2' select * from BOOK
-- step 10 (failure) rollback transaction	-- step 11 rollback transaction	-- step 12 rollback transaction

Cascadeless schedule: 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. Here, the updated value of X is read by transaction T2 only after the commit of transaction T1

We converted the above cascading rollback schedule to a cascadeless schedule, it can be seen from the image attached below, I've also shown the code and flow of execution the transactions below, as can be seen it performs without any discrepancies or errors.


SQLQuery1.sql - Ser...kul Shingwani (56))*

```
-- step 1
begin transaction

-- step 2
insert into BOOK values
('Jo_Ro_1018','Deat_JR_1018',
'Joanne K. Rowling',
'Deathly Hallows_Harry Potter')
select * from BOOK

-- step 3
commit transaction
```

100 %

Results | Messages

	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
4	Jo_Ro_1018	Deat_JR_1018	Joanne K. Rowling	Deathly Hallows_Har

SQLQuery2.sql - Ser...kul Shingwani (57))*

```
-- step 4
begin transaction

-- step 5
update BOOK set Book_name='mukul 2'
where Author_id='Jo_Ro_1018'
select * from BOOK

-- step 6 (this needs to be rolledback)
commit transaction
```

100 %

Results | Messages

	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: Wha
4	Jo_Ro_1018	Deat_JR_1018	Joanne K. Rowling	mukul 2

SQLQuery6.sql - Ser...kul Shingwani (65))*

```
-- step 7
begin transaction

-- step 8
update BOOK set Author_name='xyz'
where Book_name='mukul 2'
select * from BOOK

-- step 9 (this needs to be rolledback)
commit transaction
```

100 %

Results | Messages

	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 A
4	Jo_Ro_1018	Deat_JR_1018	xyz	mukul 2

```
T1                                  T2                                  T3
---------------------------------------------------------------------------------------------------
-- step 1                           -- step 4                           -- step 7
begin transaction                   begin transaction                   begin transaction

-- step 2                           -- step 5
insert into BOOK values             update BOOK set Book_name
                                    ='mukul 2' where Author_id
                                    ='Jo_Ro_1018'
                                    select * from BOOK
('Jo_Ro_1018','Deat_JR_1018',
'Joanne K. Rowling',
'Deathly Hallows_Harry Potter')                                         -- step 8
select * from BOOK                                                      update BOOK set
                                                                        Author_name='xyz'
                                                                        where Book_name='mukul2'
                                                                        select * from BOOK


-- step 3                           -- step 6                           -- step 9
commit transaction                  commit transaction                  commit transaction
---------------------------------------------------------------------------------------------------
```

----------------- X ---------------------- END ---------------------- X -----------------