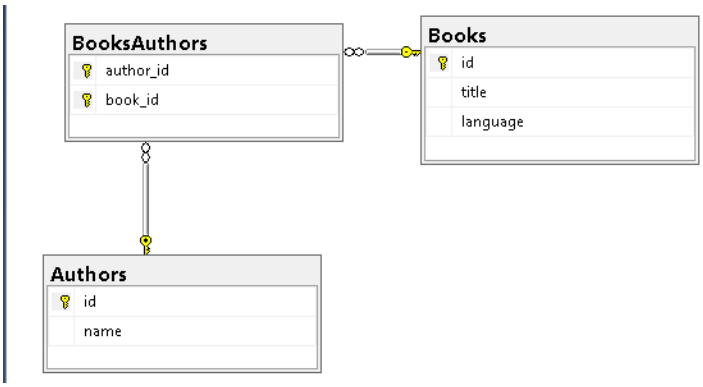
Lab 3

This homework will be solved in SQL SERVER. Each student will choose the tables desired from the own database.

- Create a stored procedure that inserts data in tables that are in a m:n relationship. If one insert fails, all the operations performed by the procedure must be rolled back. (grade: 3)

We consider the database



* Don't use IDs as input parameters for your stored procedures. Validate all parameters (try to use functions when needed).

The store procedure must include all the fields from the tables (3 tables) involved, except the id's of these tables (the primary key's, that can be extracted with MAX value introduced, SCOPE_IDENTITY(), …), and these fields must be validated.

Create functions for validation: for example - check the language to have some values (for the table Books)

```
CREATE FUNCTION uf_ValidateLanguage (@language varchar(100)) RETURNS INT AS
BEGIN
DECLARE @return INT
SET @return = 0
IF(@language  IN ('English','Romanian','French'))
SET @return = 1
RETURN @return
END
```

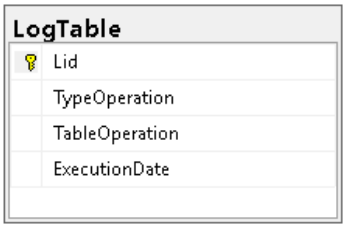
Create the stored procedure with the following restrictions:

- Do not take the Id's as parameters (here id from Authors, id from Books, author_id and book_id from BooksAuthors)
- Take the parameters all the rest of the fields from the tables (here title, language, name)
- Create validation functions for the parameters (all you consider necessary), like:
  - a field is in a set of values (language IN ('English','Romanian','French')))
  - the fields of varchar type be not null, start with a upper type, ..
  - the fields of int to be positive, …
  - validation functions for telephone numbers, e-mail, …
  - or, whatever do you need, or want

First we insert values in the tables Authors and Books (the order is not important) and, then, in BooksAuthors (the intermediate table), by taking the id from both of the tables. We can take the id from one of the tables in a variable or if the field is identity like the maximum value of that field .

* Setup a logging system to track executed actions for all scenarios.

For the log system, one can verify with SELECT or save in a log table.

SELECT * FROM Authors SELECT * FROM Books SELECT * FROM BooksAuthors EXEC …. SELECT * FROM Authors SELECT * FROM Books SELECT * FROM BooksAuthors	CREATE TABLE LogTable(Lid INT IDENTITY PRIMARY KEY, TypeOperation VARCHAR(50), TableOperation VARCHAR(50), ExecutionDate DATETIME) Where TypeOperation can be Update, Select, …. 
Or any other method…	

It's recommended to use TRY…CATCH to handle errors.

Next, we give an example for a stored procedure for table Books.

```
CREATE PROCEDURE AddBookAuthor @title varchar(50), @language varchar(50) AS
BEGIN

BEGIN TRAN
BEGIN TRY
IF(dbo.uf_ValidateLanguage(@language)<>1)
BEGIN
        RAISERROR('Language must be Romanian, English or French',14,1)
END
INSERT INTO Books (title, language) VALUES (@title, @language)
COMMIT TRAN
SELECT 'Transaction committed'
END TRY

BEGIN CATCH
ROLLBACK TRAN
SELECT 'Transaction rollbacked'
END CATCH
END
```

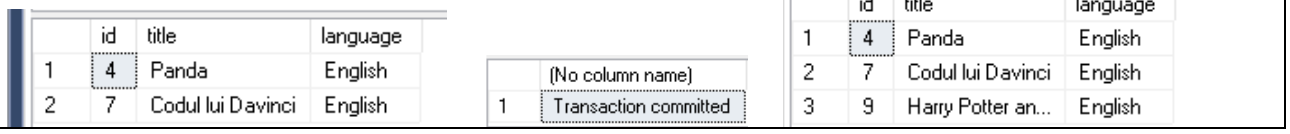
But, pay attention, you have to insert values in all the tables from the relation m-n considered. First, you insert data in the table(s) Books and Authors, and then in the intermediate table (BooksAuthors). All these Insert's operations (for all these 3 tables), will be done in a single transaction.

* Prepare test cases covering both the happy scenarios and the ones with errors (this applies to stored procedures). Be prepared to explain all your scenarios and implementations.

You have to prepare 2 scenarios for the verification of the stored procedure: one with commit and one with rollback. The rollback can be obtain from the validation conditions given by the validation functions.
You, also, have to return, the history of the operations executed. You can use Select/PRINT messages or use Select * from table_name or any other solution that you consider.
Execution:

- with success (commit)

```
Select * from Books
EXEC AddBookAuthor 'Harry Potter and The Chamber of Secrets','English'
Select * from Books
```



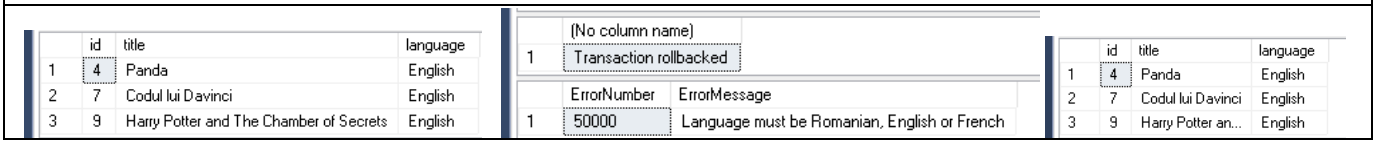
	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English

	(No column name)
1	Transaction committed

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter an...	English

- without success (rollback)

```
Select * from Books
EXEC AddBookAuthor 'Inferno','Spanish'
Select * from Books
```



	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English

	(No column name)
1	Transaction rollbacked

	ErrorNumber	ErrorMessage
1	50000	Language must be Romanian, English or French

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter an...	English

- create a stored procedure that inserts data in tables that are in a m:n relationship; if an insert fails, try to recover as much as possible from the entire operation: for example, if the user wants to add a book and its authors, succeeds creating the authors, but fails with the book, the authors should remain in the database (grade 5);

Here, the transaction (from the previous requirement) will be split into 3 transactions in the same stored procedure:

- First for the table Authors - with the validation also
- Second for the table Books – with the validation also
- Third for the table BooksAuthors – with the id's taken from both of the previous tables

The idea is that one can insert separately in each of the table. If we can add in Books, we add, and in Authors we cannot add, but this won't affect the add from Books. Each table is treat it separately and do not affect the add of the other(s) tables.
The execution has to be done for a success case and also for an un-success case (you have to prepare test cases for both happy and errors scenarios.

- create 4 scenarios that reproduce the following concurrency issues under pessimistic isolation levels: dirty reads, non-repeatable reads, phantom reads, and a deadlock; you can use stored procedures and / or stand-alone queries; find solutions to solve / workaround the concurrency issues (grade 9);

You need to consider a table in which you will analyze the concurrency execution. Here, I had chosen Books.
You must prepare scenarios for each case: “Transaction 1 with Transaction 2” – concurrency issue; and “Transaction 1 with Transaction 2” - solution. You have to create and save each of the transactions used. You can use one file for Transaction 1 and one file for Transaction 2 – that include the concurrency issue, but also the solution, or 2 files, saved suggestive. Or, you can organize the structure as you prefer, but to be clear. Or, you can have stored procedures. Also, prepare examples for each of the cases.

Try to run the transactions in the same time (or close): start Transaction 1 first, introduce a delay there, so that Transaction 2 can be executed in that time. Immediately that Transaction 1 was started, start also Transaction 2. (If you run the transactions converse, the result will also be converse).

In table Books we have

Results | Messages

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	10	Insomnii	English

For what follows: T1=Transaction 1 starts first. T2=Transaction start immediately after T1.

1. DIRTY READS – T1: 1 update + delay + rollback, T2: select + delay + select -> we see the update in the first select (T1 – finish first), even if it is rollback then

Isolation level: Read Uncommitted / Read Committed (solution)

--Dirty Reads Part 1 BEGIN TRANSACTION UPDATE Books SET language='Romanian' WHERE id = 7 WAITFOR DELAY '00:00:10' ROLLBACK TRANSACTION	--Dirty Reads Part 2 SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED BEGIN TRAN SELECT * FROM Books WAITFOR DELAY '00:00:15' SELECT * FROM Books COMMIT TRAN
Messages (1 row(s) affected)	Results Messages id title language 1 4 Panda English 2 7 Codul lui Davinci Romanian 3 9 Harry Potter and The Chamber of Secrets English 4 10 Insomnii English id title language 1 4 Panda English 2 7 Codul lui Davinci English 3 9 Harry Potter and The Chamber of Secrets English 4 10 Insomnii English

Solution: T1: 1 update + delay + rollback, T2: select + delay + select -> we don't see the update (that is also rollback) – T1 finish first

```
--Dirty Reads Part 1
BEGIN TRANSACTION
UPDATE Books SET language='Romanian'
WHERE id = 7
WAITFOR DELAY '00:00:10'
ROLLBACK TRANSACTION
```

```
--Solution: SET TRANSACTION ISOLATION LEVEL TO READ COMMITTED
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRAN
SELECT * FROM Books
WAITFOR DELAY '00:00:15'
SELECT * FROM Books
COMMIT TRAN
```

Messages

```
(1 row(s) affected)
```

Results Messages

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	10	Insomnii	English

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	10	Insomnii	English

2. NON-REPEATABLE READS – T1: insert + delay + update + commit, T2: select + delay + select -> see the insert in first select of T2 + update in the second select of T2, T1 finish first

Isolation level: Read Committed / Repeatable Read (solution). The result will contain the previous row version (before the finish of the transaction).

```
INSERT INTO Books(title, language) VALUES ('Sub dom','Spanish')
BEGIN TRAN
WAITFOR DELAY '00:00:05'
UPDATE Books SET language='English' WHERE title = 'Sub dom'
COMMIT TRAN
```

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRAN
SELECT * FROM Books
WAITFOR DELAY '00:00:05'
SELECT * FROM Books
COMMIT TRAN
```

Messages

```
(1 row(s) affected)

(1 row(s) affected)
```

Results Messages

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	10	Insomnii	English
5	11	Sub dom	Spanish

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	1...	Insomnii	English
5	1...	Sub dom	English

Solution: T1: insert + delay + update + commit, T2: select + delay + select -> see only the final result in both of the select of T2, T1 finish first

```
INSERT INTO Books(title, language) VALUES ('Sub dom','Spanish')
BEGIN TRAN
WAITFOR DELAY '00:00:05'
UPDATE Books SET language='English' WHERE title = 'Sub dom'
COMMIT TRAN
```

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRAN
SELECT * FROM Books
WAITFOR DELAY '00:00:05'
SELECT * FROM Books
COMMIT TRAN
```

Messages

```
(1 row(s) affected)

(2 row(s) affected)
```

Results | Messages

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	10	Insomnii	English
5	11	Sub dom	English
6	13	Sub dom	Spanish

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	10	Insomnii	English
5	11	Sub dom	English
6	13	Sub dom	Spanish

PHANTOM READS – T1: delay + insert + commit, T2: select + delay + select -> see the inserted value only at the second select from T2, T1 finish first. The result will contain the previous row version; the same number of rows (before the finish of the transaction – for example, 5 not 6). Isolation level: Repeatable Read / Serializable (solution)

```
--Phantom Reads Part 1
--DELETE FROM Books
BEGIN TRAN
WAITFOR DELAY '00:00:04'
INSERT INTO Books(title,language) VALUES ('Morometii','Romanian')
COMMIT TRAN
```

```
--Phantom Reads Part 2
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRAN
SELECT * FROM Books
WAITFOR DELAY '00:00:05'
SELECT * FROM Books
COMMIT TRAN
```

Messages

```
(1 row(s) affected)
```

Results | Messages

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	10	Insomnii	English
5	11	Sub dom	English
6	13	Sub dom	English

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	1...	Insomnii	English
5	1...	Sub dom	English
6	1...	Sub dom	English
7	1...	Morometii	Romani...

Solution: T1: delay + insert + commit, T2: select + delay + select -> the new inserted values are not visible at the end of T_2 and T1, only if we make a new select and test it.

```
--Phantom Reads Part 1
--DELETE FROM Books
BEGIN TRAN
WAITFOR DELAY '00:00:04'
```

```
--Solution: Set transaction isolation level to SERIALIZABLE
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRAN
SELECT * FROM Books
```

```
INSERT INTO Books(title,language) VALUES
('Morometii','Romanian')
COMMIT TRAN
```

```
Messages

(1 row(s) affected)
```

```
WAITFOR DELAY '00:00:05'
SELECT * FROM Books
COMMIT TRAN
```

Results Messages

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	10	Insomnii	English
5	11	Sub dom	English
6	13	Sub dom	English
7	19	Morometii	Romanian

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter an...	English
4	1...	Insomnii	English
5	1...	Sub dom	English
6	1...	Sub dom	English
7	1...	Morometii	Romani...

3. DEADLOCK – T1: update on table A + delay + update on table B, T2: update on table B + delay + update on table A

We update on table A (from T1 – that exclusively lock on table A), update on table B (from T2 – that exclusively lock on table B), try to update from T1 table B (but this transaction will be blocked because T2 has already been locked on table B), try to update from T2 table A (but this transaction will be blocked because T1 has already been locked on table A). So, both of the transactions are blocked. After some seconds T2 will be chosen as a deadlock victim and terminates with an error. After that, T1 will finish also. In table A and table B will be the values from T1.
The transaction that is chosen as a deadlock victim, is the one that has the deadlock_priority lower. If both of the transactions have the same deadlock_priority, the deadlock victim is the one less expensive at rollback. Otherwise, the deadlock victim is chosen random.

Here we consider 2 tables: Books, Authors.

Books

Results Messages

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	10	Insomnii	English
5	11	Sub dom	English
6	13	Sub dom	English
7	19	Morometii	Romanian
8	20	Morometii	Romanian

Authors

Results Messages

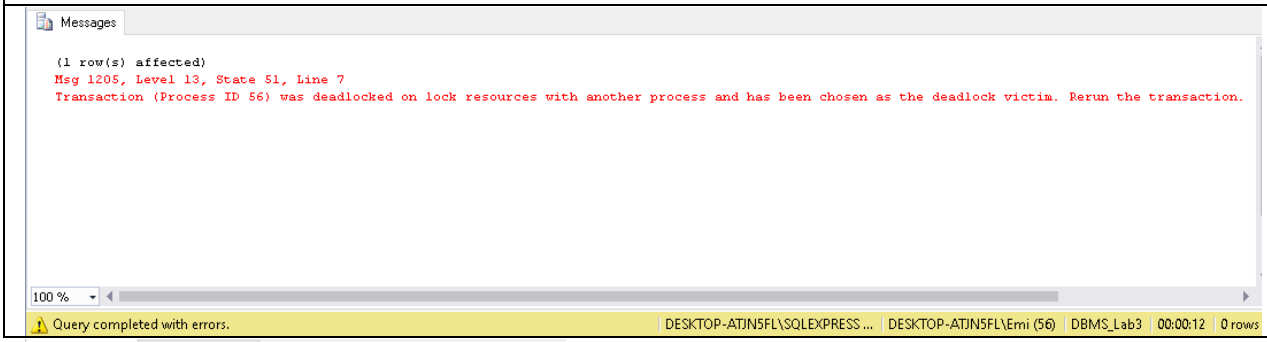
	id	name
1	4	J.K. Rowling
2	5	Irina Binder
3	6	2

```
-- transaction 1
begin tran
update Books set title='La cirese transaction 1' where id=20
-- this transaction has exclusively lock on table Books
waitfor delay '00:00:10'
update Authors set name='Petre Ispirescu transaction 1' where id=6
-- this transaction will be blocked because transaction 2 has already blocked our lock on table Authors
commit tran
```

```
Messages

(1 row(s) affected)

(1 row(s) affected)
```

```
-- transaction 2
```

```
begin tran
update Authors set name='Petre Ispirescu transaction 2' where id=6
-- this transaction has exclusively lock on table Authors
waitfor delay '00:00:10'
update Books set title='La cirese transaction 2' where id=20
-- this transaction will be blocked because transaction 1 has exclusively lock on table Books, so, both of the transactions are blocked
commit tran
-- after some seconds transaction 2 will be chosen as a deadlock victim and terminates with an error
-- in tables Books and Authors will be the values from transaction 1
```



Results Messages

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	10	Insomnii	English
5	11	Sub dom	English
6	13	Sub dom	English
7	19	Morometii	Romanian
8	20	La cirese transaction 1	Romanian

Results Messages

	id	name
1	4	J.K. Rowling
2	5	Irina Binder
3	6	Petre Ispirescu transaction 1

Solution: For deadlock, the priority has to be set (LOW, NORMAL, HIGH, or from -10 to 10). Implicit is NORMAL (0).
For example, here we set the DEADLOCK_PRIORITY to HIGH for T2, so that T1 be chosen as a deadlock victim (T1 will have a lower priority than T2 and it will finish first).

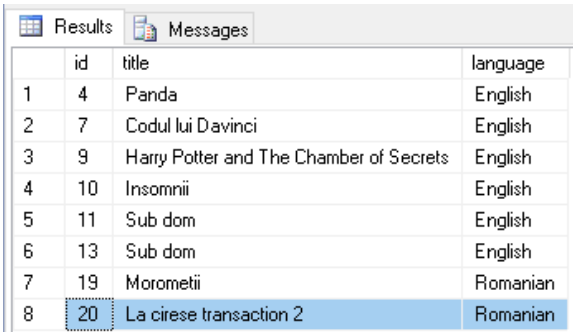
```
-- transaction 1
begin tran
update Books set title='La cirese transaction 1' where id=20
-- this transaction has exclusively lock on table Books
waitfor delay '00:00:10'
update Authors set name='Petre Ispirescu transaction 1' where id=6
commit tran
-- this transaction is chose as a deadlock, because it has the lowest priority level here (normal)
```

```
(1 row(s) affected)
Msg 1205, Level 13, State 51, Line 7
Transaction (Process ID 54) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.
```

```
-- transaction 2
SET DEADLOCK_PRIORITY HIGH
begin tran
update Authors set name='Petre Ispirescu transaction 2' where id=6
-- this transaction has exclusively lock on table Authors
waitfor delay '00:00:10'
update Books set title='La cirese transaction 2' where id=20
commit tran
-- this transaction has the higher priority level from here (set to HIGH)
-- transaction 1 finish with an error, and the  results are the ones from this transaction (transaction 2)
```

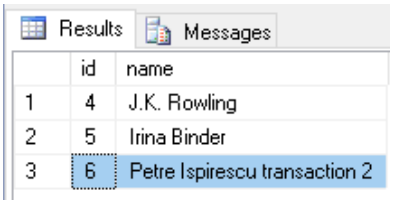
Messages

```
(1 row(s) affected)

(1 row(s) affected)
```



Results | Messages

	id	title	language
1	4	Panda	English
2	7	Codul lui Davinci	English
3	9	Harry Potter and The Chamber of Secrets	English
4	10	Insomnii	English
5	11	Sub dom	English
6	13	Sub dom	English
7	19	Morometii	Romanian
8	20	La cirese transaction 2	Romanian



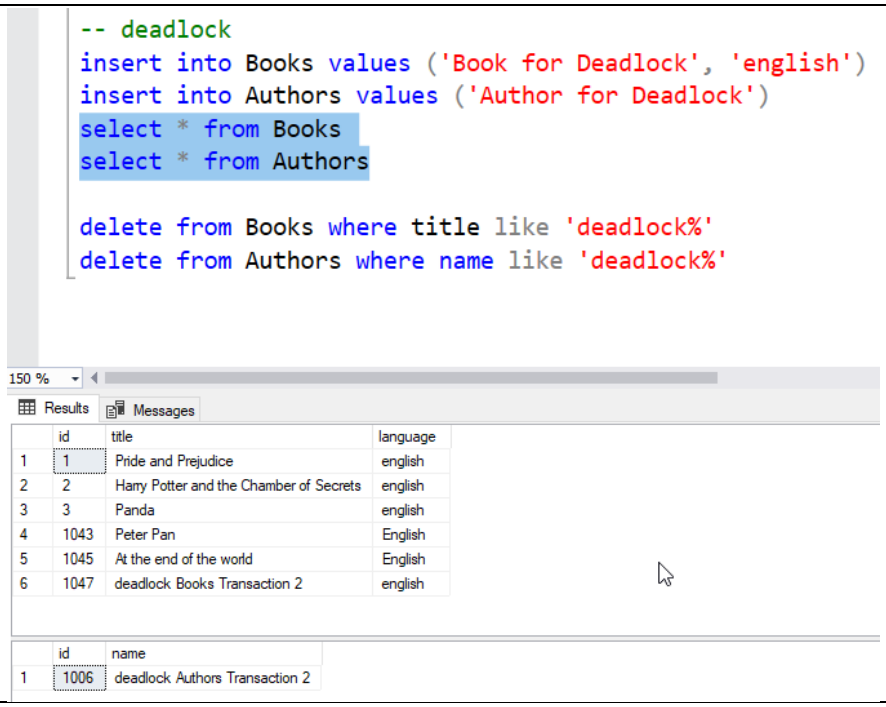
Results | Messages

	id	name
1	4	J.K. Rowling
2	5	Irina Binder
3	6	Petre Ispirescu transaction 2

Another possible solution for Deadlock is to execute the statements in the same order in both of the transactions. As result, first are performed the UPDATE's from the first transaction executed and then the UPDATE's from the second transaction executed.

```
-- transaction 1
begin tran
update Books set title='deadlock Books Transaction 1' where id=1047
-- this transaction has exclusively lock on table Books
waitfor delay '00:00:10'

update Authors set name='deadlock Authors Transaction 1' where id=1006
commit tran

-- (1 row affected)
-- (1 row affected)
```

```
-- transaction 2
begin tran
update Books set title='deadlock Books Transaction 2' where id=1047
-- this transaction has exclusively lock on table Books
waitfor delay '00:00:10'

update Authors set name='deadlock Authors Transaction 2' where id=1006
commit tran

-- (1 row affected)
-- (1 row affected)
```

select * from Books select * from Authors	

```
-- deadlock
insert into Books values ('Book for Deadlock', 'english')
insert into Authors values ('Author for Deadlock')
select * from Books
select * from Authors

delete from Books where title like 'deadlock%'
delete from Authors where name like 'deadlock%'
```

150 %

Results | Messages

	id	title	language
1	1	Pride and Prejudice	english
2	2	Harry Potter and the Chamber of Secrets	english
3	3	Panda	english
4	1043	Peter Pan	English
5	1045	At the end of the world	English
6	1047	deadlock Books Transaction 2	english

	id	name
1	1006	deadlock Authors Transaction 2

- create a scenario that reproduces the update conflict under an optimistic isolation level (grade 10).

First, we need to set the isolation level, to an optimistic one.

```
ALTER DATABASE DBMS_Lab3 SET ALLOW_SNAPSHOT_ISOLATION ON
OR
ALTER DATABASE DBMS_Lab3 SET READ_COMMITTED_SNAPSHOT ON
```

Transaction 1	Transaction 2

`-- transaction 1`
`use DBMS_Lab3`
`go`
 	
`waitfor delay '00:00:10'`
`BEGIN TRAN`
`UPDATE Books SET language = 'Israel' WHERE id=33;`
`-- language is now Israel`
`waitfor delay '00:00:10'`
`COMMIT TRAN`
	`--transaction 2`
	`Use DBMS_Lab3`
	`go`
	
	`SET TRANSACTION ISOLATION LEVEL SNAPSHOT`
	`BEGIN TRAN`
	`Select * from Books where id=33`
	`-- Bambi – French – the value from the beginning of the transaction`
	`Waitfor delay '00:00:10'`
	`select * from Books where id=4`
	`-- the value from the beginning of the transaction – Panda-English`
	`Update Books set language='Portugues' where id=33`
	`-- process will block`
	`-- Process will receive Error 3960.`
	`COMMIT TRAN`

Messages (1 row(s) affected)	(1 row(s) affected) (1 row(s) affected) Msg 3960, Level 16, State 3, Line 13 Snapshot isolation transaction aborted due to update conflict. You cannot use snapshot isolation to access table 'dbo.Books' direc Msg 3960, Level 16, State 3, Line 13 Snapshot isolation transaction aborted due to update conflict. You cannot use snapshot isolation to access table 'dbo.Books' directly or indirectly in database 'DBMS_Lab3' to update, delete, or insert the row that has been modified or deleted by another transaction.

	Retry the transaction or change the isolation level for the update/delete statement.

```
ALTER DATABASE DBMS_Lab3 SET ALLOW_SNAPSHOT_ISOLATION OFF
OR
ALTER DATABASE DBMS_Lab3 SET READ_COMMITTED_SNAPSHOT OFF
```