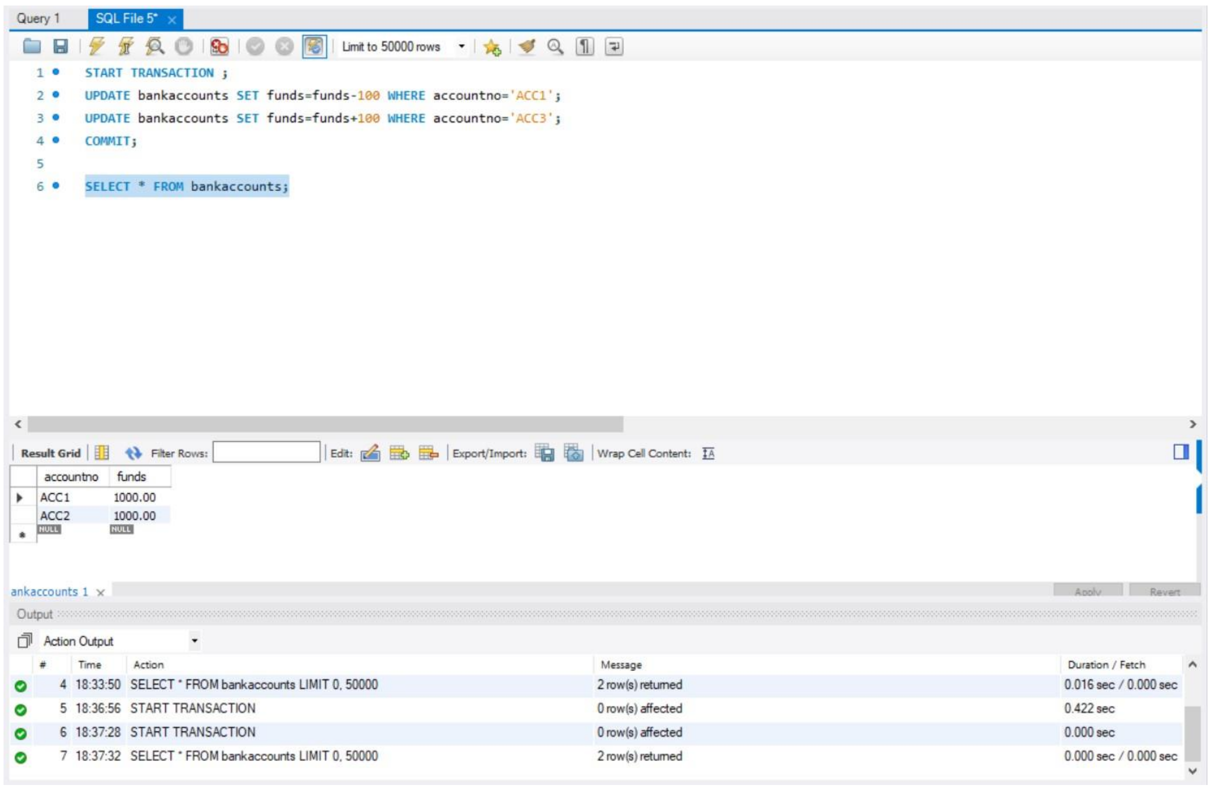
Atomicity

The **Atomicity Property of a Transaction in SQL Server** ensures that either all the DML Statements (i.e. insert, update, delete) inside a transaction are completed successfully or all of them are rolled back.

Lets take our example here, we have two bank accounts here in the transaction and one of them is correct and other one is not , and hence none of the changes take place and the table remains as the old one



```
START TRANSACTION ;
UPDATE bankaccounts SET funds=funds-100 WHERE accountno='ACC1';
UPDATE bankaccounts SET funds=funds+100 WHERE accountno='ACC3';
COMMIT;

SELECT * FROM bankaccounts;
```

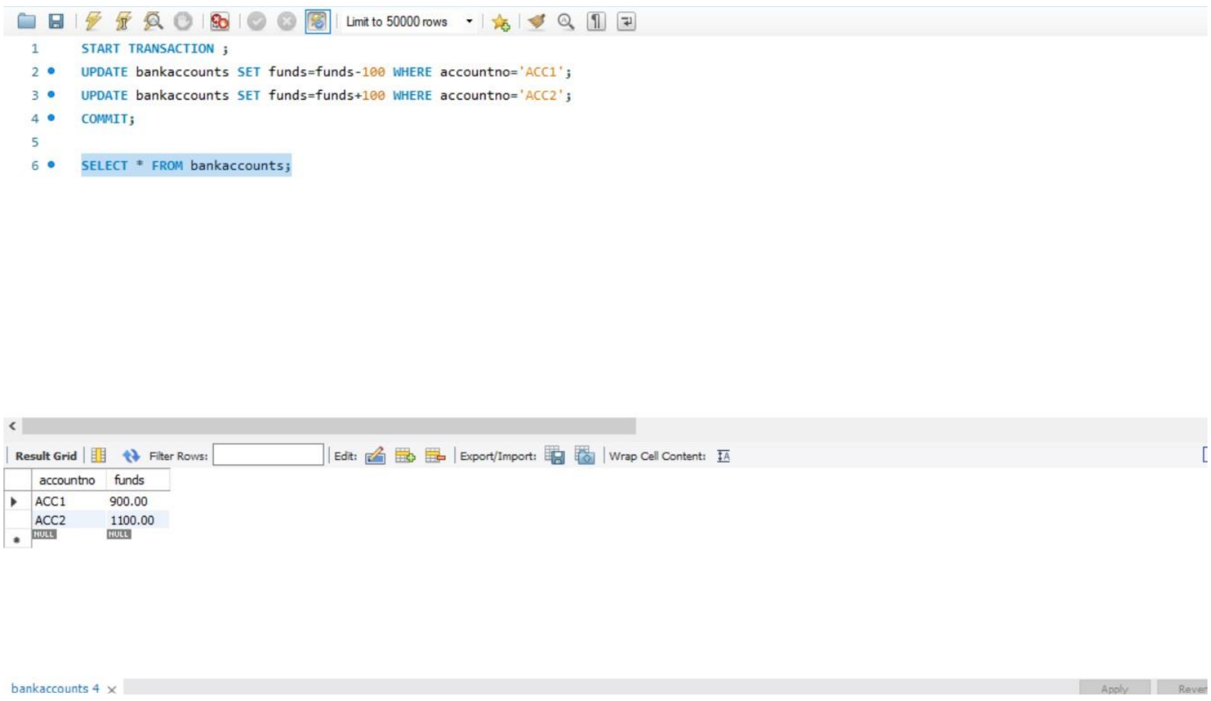
accountno	funds
ACC1	1000.00
ACC2	1000.00
NULL	NULL

```
#  Time      Action                                     Message              Duration / Fetch
4  18:33:50  SELECT * FROM bankaccounts LIMIT 0, 50000  2 row(s) returned    0.016 sec / 0.000 sec
5  18:36:56  START TRANSACTION                          0 row(s) affected    0.422 sec
6  18:37:28  START TRANSACTION                          0 row(s) affected    0.000 sec
7  18:37:32  SELECT * FROM bankaccounts LIMIT 0, 50000  2 row(s) returned    0.000 sec / 0.000 sec
```

Consistency

The **Consistency Property of a Transaction in SQL Server** ensures that the database data is in a consistent state before the transaction started and also left the data in a consistent state after the transaction is completed. If the transaction violates the rules then it should be rolled back.

Let’s take our example here, We have two bank accounts here and after the transaction we have a consistent table where the money transaction has taken place successfully



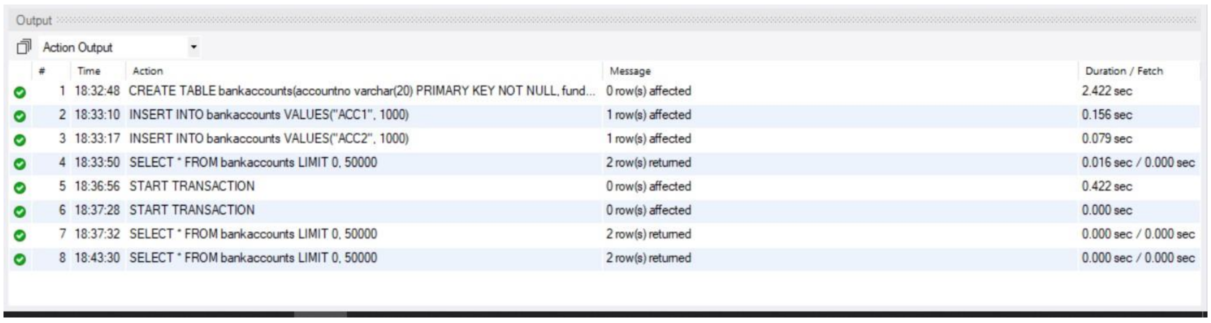
```
START TRANSACTION ;
UPDATE bankaccounts SET funds=funds-100 WHERE accountno='ACC1';
UPDATE bankaccounts SET funds=funds+100 WHERE accountno='ACC2';
COMMIT;

SELECT * FROM bankaccounts;
```

accountno	funds
ACC1	900.00
ACC2	1100.00
NULL	NULL

Isolation

The **Isolation Property of a Transaction in SQL Server** ensures that the intermediate state of a transaction is invisible to other transactions. The Data modifications made by one transaction must be isolated from the data modifications made by all other transactions. Most databases use locking to maintain transaction isolation.

Here that means we can't do two DML commands at the same time to the same database, each of the commands are isolated against each other.



#	Time	Action	Message	Duration / Fetch
1	18:32:48	CREATE TABLE bankaccounts(accountno varchar(20) PRIMARY KEY NOT NULL, fund...	0 row(s) affected	2.422 sec
2	18:33:10	INSERT INTO bankaccounts VALUES("ACC1", 1000)	1 row(s) affected	0.156 sec
3	18:33:17	INSERT INTO bankaccounts VALUES("ACC2", 1000)	1 row(s) affected	0.079 sec
4	18:33:50	SELECT * FROM bankaccounts LIMIT 0, 50000	2 row(s) returned	0.016 sec / 0.000 sec
5	18:36:56	START TRANSACTION	0 row(s) affected	0.422 sec
6	18:37:28	START TRANSACTION	0 row(s) affected	0.000 sec
7	18:37:32	SELECT * FROM bankaccounts LIMIT 0, 50000	2 row(s) returned	0.000 sec / 0.000 sec
8	18:43:30	SELECT * FROM bankaccounts LIMIT 0, 50000	2 row(s) returned	0.000 sec / 0.000 sec

Here you can see that the commands take place on after another and not parallel due to the isolation level in MYSQL

Durability

The **Durability Property of a Transaction in SQL Server** ensures that once the transaction is successfully completed, then the changes it made to the database will be permanent. Even if there is a system failure or power failure or any abnormal changes, it should safeguard the committed data.