T.C. İSTANBUL KÜLTÜR ÜNİVERSİTESİ

COM5041 - DATABASE DESIGN AND DEVELOPMENT

LAB 09 – How to manage transactions

After completing this Lab, you will be able to •

Understand the SQL transaction modes.

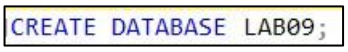
- How to create transaction.
- How to define transactions.
- How to work with nested transactions.
- How to work with save points.
- Create a real-world example for the transaction.

PROCEDURE 1 – How to manage transactions

A transaction is a group of database operations that you combine into a single logical unit. By combining operations in this way, you can prevent certain kinds of database errors. If a transaction is successful, all the changes made in that SQL transaction will apply to the table. If any single statement inside the transaction encounters an error, then changes made in that transaction will be erased or rolled back. The most suitable examples for transaction transactions are banking database transactions. It is important to plan and use work blocks correctly in this type of application, which is complex and consists of numerous transactions with many threads.

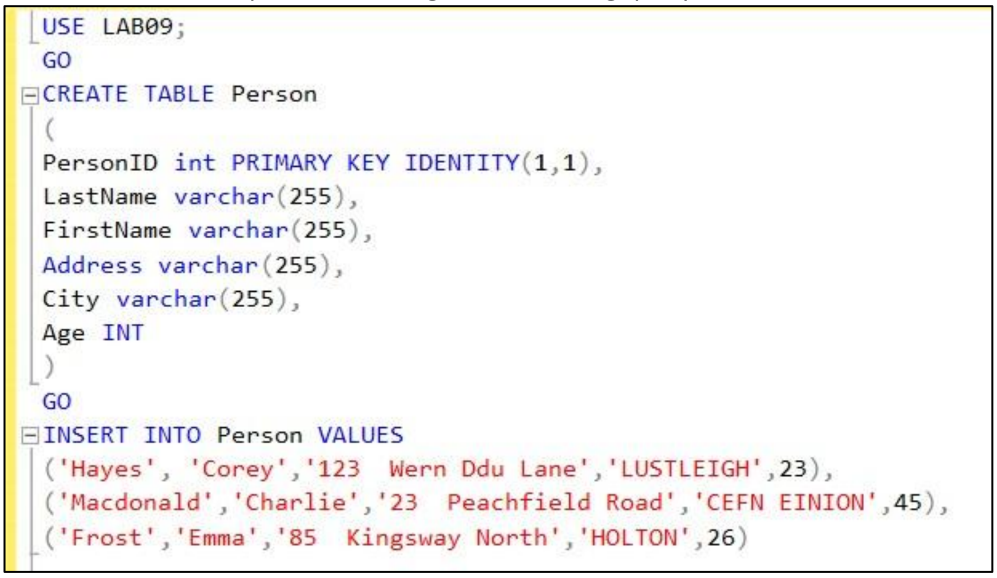
Step 1- Let's create a table named **Person** where the information of the people we will use to learn the transactions in SQL.

- Firstly, we will create a database named LAB09.



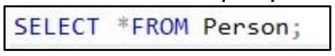
```sql
CREATE DATABASE LAB09;
```

- We will create a sample table through the following query and will insert some sample data.

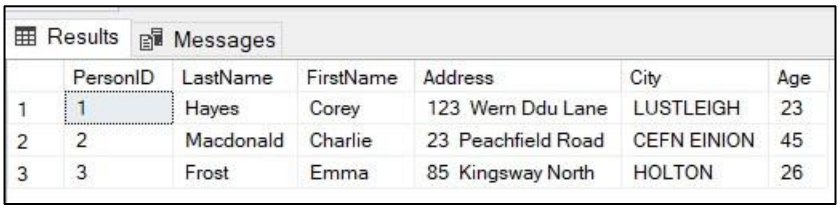


```sql
USE LAB09;
GO
CREATE TABLE Person
(
PersonID int PRIMARY KEY IDENTITY(1,1),
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255),
Age INT
)
GO
INSERT INTO Person VALUES
('Hayes', 'Corey','123  Wern Ddu Lane','LUSTLEIGH',23),
('Macdonald','Charlie','23  Peachfield Road','CEFN EINION',45),
('Frost','Emma','85  Kingsway North','HOLTON',26)
```

- Let's execute the query and see the data in the table.



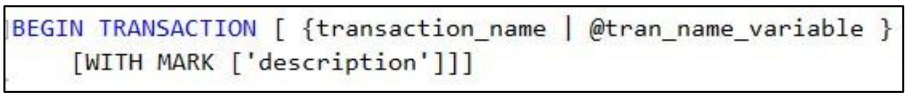
```sql
SELECT *FROM Person;
```



Results Messages

	PersonID	LastName	FirstName	Address	City	Age
1	1	Hayes	Corey	123 Wern Ddu Lane	LUSTLEIGH	23
2	2	Macdonald	Charlie	23 Peachfield Road	CEFN EINION	45
3	3	Frost	Emma	85 Kingsway North	HOLTON	26

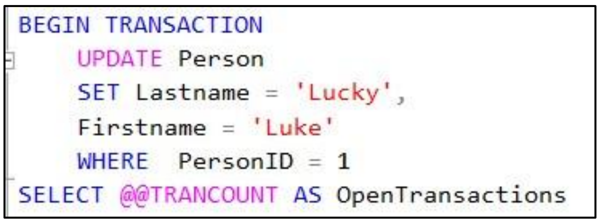
Step 2- Explicit transaction mode provides to define a transaction exactly with the starting and ending points of the transaction. To define an explicit transaction, we start to use the **BEGIN TRANSACTION** command because this statement identifies the starting point of the explicit transaction. It has the following syntax:



```
BEGIN TRANSACTION [ {transaction_name | @tran_name_variable }
    [WITH MARK ['description']]]
```

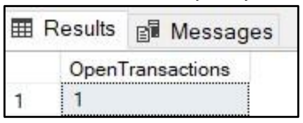
- **transaction_name** option is used to assign a specific name to transactions.
- **@trans_var** option is a user-defined variable that is used to hold the transaction name.
- **WITH MARK** option enable to mark a particular transaction in the log file.

The following statement starts a transaction and then it will change the name of a particular row in the Person table.



```
BEGIN TRANSACTION
    UPDATE Person
    SET Lastname = 'Lucky',
    Firstname = 'Luke'
    WHERE  PersonID = 1
SELECT @@TRANCOUNT AS OpenTransactions
```

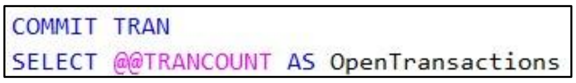
** Execute the query and review the results.



```
   OpenTransactions
1  1
```

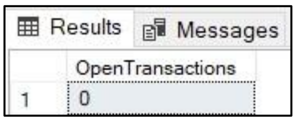
Note that; @@TRANCOUNT function returns the number of BEGIN TRANSACTION statements in the current session and we can use this function to count the open local transaction numbers in the examples.

Step 2- **COMMIT TRANSACTION** statement applies the data changes to the database and the changed data will become permanent. Now let's complete the open transaction with a COMMIT TRANSACTION statement.



```
COMMIT TRAN
SELECT @@TRANCOUNT AS OpenTransactions
```

** Execute the query and review the results.



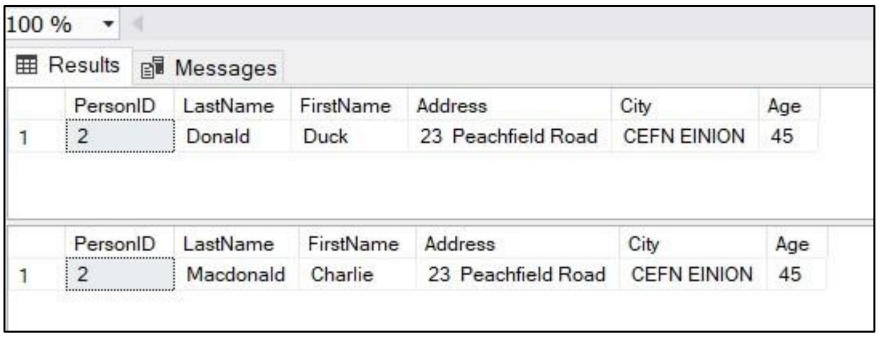
```
   OpenTransactions
1  0
```

Step

3 - The **ROLLBACK TRANSACTION** statement helps in undoing all data modifications that are applied by the transaction. In the following example, we will change a particular row, but this data modification will not persist.

```
BEGIN TRAN
    UPDATE Person
    SET Lastname='Donald',
    Firstname='Duck' WHERE PersonID=2
    SELECT * FROM Person WHERE PersonID=2
ROLLBACK TRAN
SELECT * FROM Person WHERE PersonID=2
```

** Execute the query and review the results.



100 %

Results | Messages

	PersonID	LastName	FirstName	Address	City	Age
1	2	Donald	Duck	23 Peachfield Road	CEFN EINION	45

	PersonID	LastName	FirstName	Address	City	Age
1	2	Macdonald	Charlie	23 Peachfield Road	CEFN EINION	45

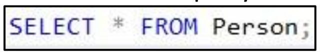
Step 4 - Savepoints can be used to rollback any particular part of the transaction rather than the entire transaction. So that we can only rollback any portion of the transaction where between after the save point and before the rollback command. To define a save point in a transaction we use the **SAVE TRANSACTION** syntax and then we add a name to the save point. The savepoint has the following syntax:

```
SAVE { TRAN | TRANSACTION } { savepoint_name | @savepoint_variable }
[ ; ]
```

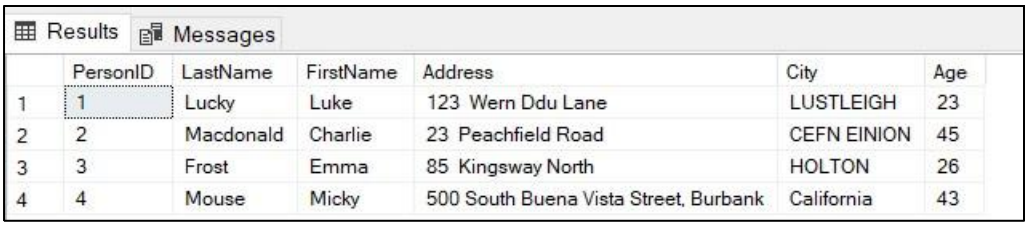
- Now, let's illustrates an example of savepoint usage. When we execute the following query, only the insert statement will be committed and the delete statement will be rolled back.

```
BEGIN TRANSACTION
    INSERT INTO Person
    VALUES('Mouse', 'Micky','500 South Buena Vista Street, Burbank','California',43)
    SAVE TRANSACTION InsertStatement
    DELETE Person WHERE PersonID=3
    SELECT * FROM Person
    ROLLBACK TRANSACTION InsertStatement
COMMIT
```

** Execute the query and review the results.



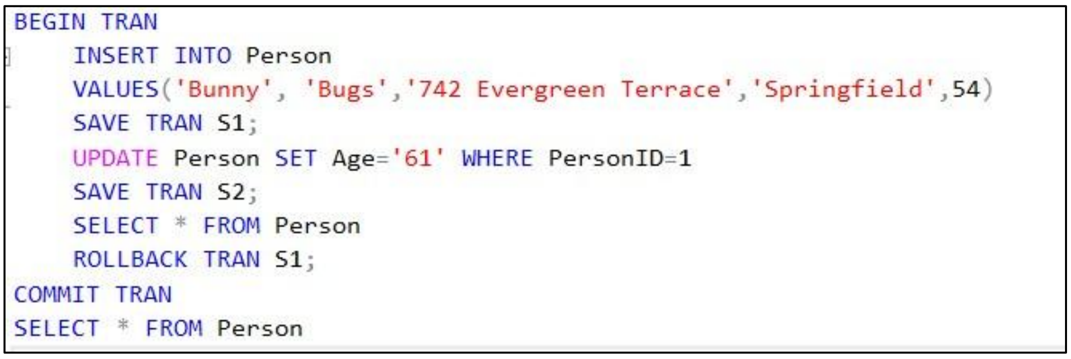
```
SELECT * FROM Person;
```



Results | Messages

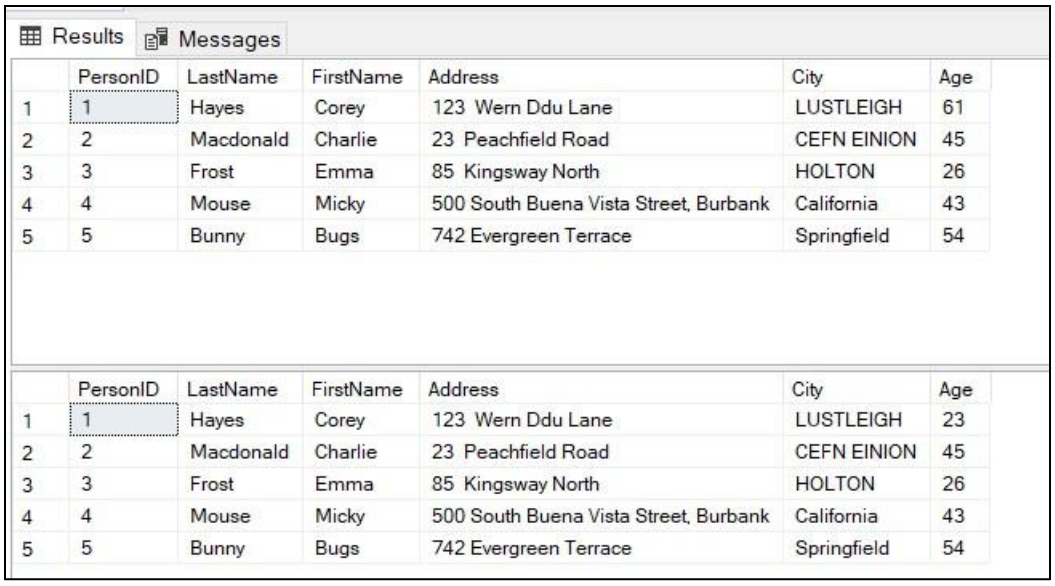
	PersonID	LastName	FirstName	Address	City	Age
1	1	Lucky	Luke	123 Wern Ddu Lane	LUSTLEIGH	23
2	2	Macdonald	Charlie	23 Peachfield Road	CEFN EINION	45
3	3	Frost	Emma	85 Kingsway North	HOLTON	26
4	4	Mouse	Micky	500 South Buena Vista Street, Burbank	California	43

- Let's create two savepoint transactions to insert a new value in the Person table. In the following SQL query, S1 save the insertion operation, the S2 save the insertion and updating operations.



```
BEGIN TRAN
    INSERT INTO Person
    VALUES('Bunny', 'Bugs','742 Evergreen Terrace','Springfield',54)
    SAVE TRAN S1;
    UPDATE Person SET Age='61' WHERE PersonID=1
    SAVE TRAN S2;
    SELECT * FROM Person
    ROLLBACK TRAN S1;
COMMIT TRAN
SELECT * FROM Person
```

**Execute the query and review the results.



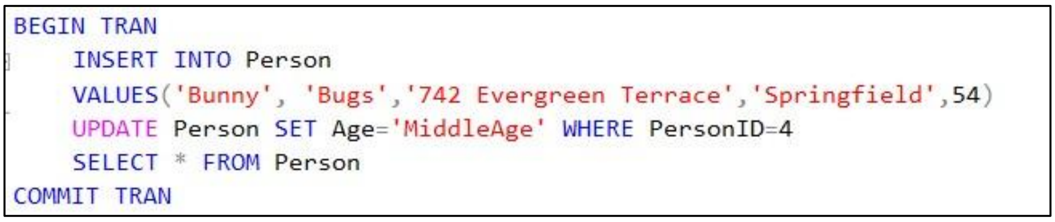
Results | Messages

	PersonID	LastName	FirstName	Address	City	Age
1	1	Hayes	Corey	123 Wern Ddu Lane	LUSTLEIGH	61
2	2	Macdonald	Charlie	23 Peachfield Road	CEFN EINION	45
3	3	Frost	Emma	85 Kingsway North	HOLTON	26
4	4	Mouse	Micky	500 South Buena Vista Street, Burbank	California	43
5	5	Bunny	Bugs	742 Evergreen Terrace	Springfield	54

	PersonID	LastName	FirstName	Address	City	Age
1	1	Hayes	Corey	123 Wern Ddu Lane	LUSTLEIGH	23
2	2	Macdonald	Charlie	23 Peachfield Road	CEFN EINION	45
3	3	Frost	Emma	85 Kingsway North	HOLTON	26
4	4	Mouse	Micky	500 South Buena Vista Street, Burbank	California	43
5	5	Bunny	Bugs	742 Evergreen Terrace	Springfield	54

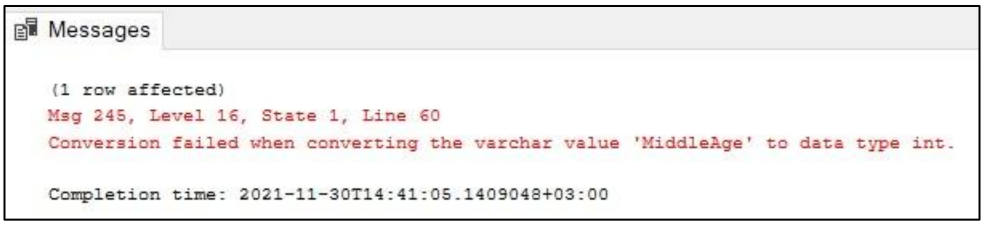
Step

5 - Generally, the transactions include more than one query. In this manner, if one of the SQL statements returns an error all modifications are erased, and the remaining statements are not executed. This process is called **Auto Rollback Transaction** in SQL. Now let's learn this principle with a very simple example.



```
BEGIN TRAN
    INSERT INTO Person
    VALUES('Bunny', 'Bugs','742 Evergreen Terrace','Springfield',54)
    UPDATE Person SET Age='MiddleAge' WHERE PersonID=4
    SELECT * FROM Person
COMMIT TRAN
```

** After we execute the transaction query, there was an error that occurred in the update statement due to the data type conversion issue. In this case, the inserted data is erased and the select statement did not execute.



Messages

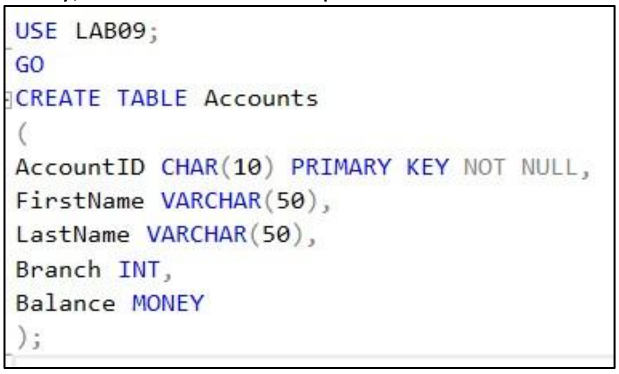
```
(1 row affected)
Msg 245, Level 16, State 1, Line 60
Conversion failed when converting the varchar value 'MiddleAge' to data type int.

Completion time: 2021-11-30T14:41:05.1409048+03:00
```

PROCEDURE 2 – Bank ATM Transaction Creation Example

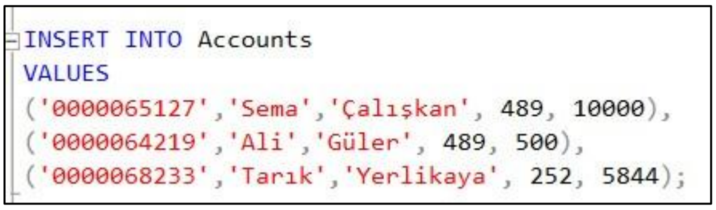
Step 1- Let's create an application that maintains user accounts in a bank database and performs wire transfers between two different bank accounts.

- Firstly, we will create a simple table that holds account information of bank customers.



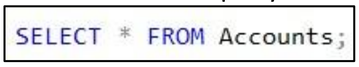
```
USE LAB09;
GO
CREATE TABLE Accounts
(
AccountID CHAR(10) PRIMARY KEY NOT NULL,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Branch INT,
Balance MONEY
);
```

- Let's insert new data in Accounts table.

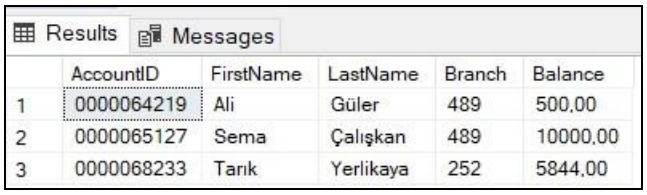


```
INSERT INTO Accounts
VALUES
('0000065127','Sema','Çalışkan', 489, 10000),
('0000064219','Ali','Güler', 489, 500),
('0000068233','Tarık','Yerlikaya', 252, 5844);
```

- Let’s execute the query and see the data in the table.

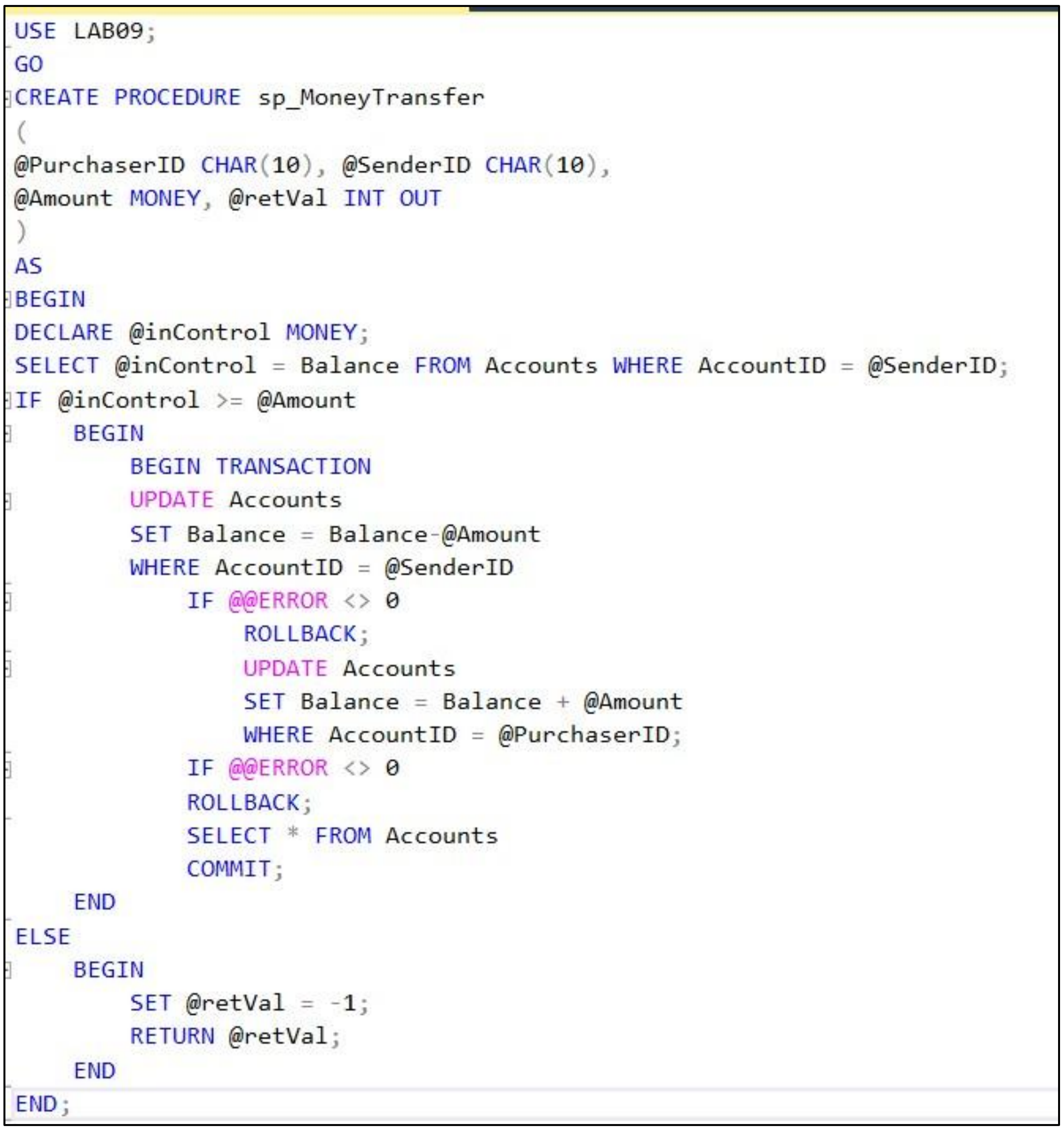


```
SELECT * FROM Accounts;
```



	AccountID	FirstName	LastName	Branch	Balance
1	0000064219	Ali	Güler	489	500,00
2	0000065127	Sema	Çalışkan	489	10000,00
3	0000068233	Tarık	Yerlikaya	252	5844,00

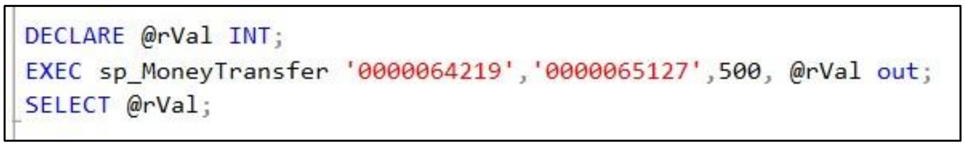
Step 2 - Now that we have added the necessary data for the query, we can now start managing our transaction by creating the relevant procedure.



```
USE LAB09;
GO
CREATE PROCEDURE sp_MoneyTransfer
(
@PurchaserID CHAR(10), @SenderID CHAR(10),
@Amount MONEY, @retVal INT OUT
)
AS
BEGIN
DECLARE @inControl MONEY;
SELECT @inControl = Balance FROM Accounts WHERE AccountID = @SenderID;
IF @inControl >= @Amount
    BEGIN
        BEGIN TRANSACTION
        UPDATE Accounts
        SET Balance = Balance-@Amount
        WHERE AccountID = @SenderID
            IF @@ERROR <> 0
                ROLLBACK;
                UPDATE Accounts
                SET Balance = Balance + @Amount
                WHERE AccountID = @PurchaserID;
            IF @@ERROR <> 0
            ROLLBACK;
            SELECT * FROM Accounts
            COMMIT;
    END
ELSE
    BEGIN
        SET @retVal = -1;
        RETURN @retVal;
    END
END;
```

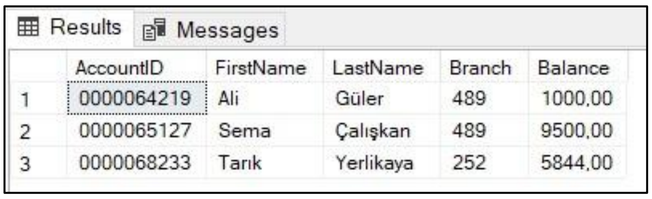
Step

3- Now, let's call the procedure that will enable ‘Sema Çalışkan’ to send 500 TL to ‘Ali Güler’.



```
DECLARE @rVal INT;
EXEC sp_MoneyTransfer '0000064219','0000065127',500, @rVal out;
SELECT @rVal;
```

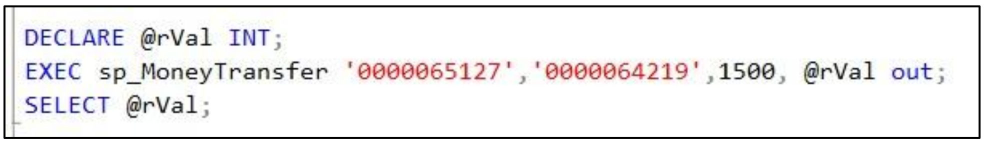
** Let’s execute the query and see the data in the table.



Results | Messages

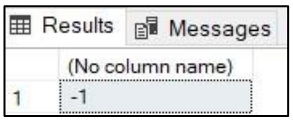
	AccountID	FirstName	LastName	Branch	Balance
1	0000064219	Ali	Güler	489	1000,00
2	0000065127	Sema	Çalışkan	489	9500,00
3	0000068233	Tarık	Yerlikaya	252	5844,00

Step 4- Let's call the procedure that will enable ‘Ali Güler’ to send 1500 TL to ‘Sema Çalışkan’.



```
DECLARE @rVal INT;
EXEC sp_MoneyTransfer '0000065127','0000064219',1500, @rVal out;
SELECT @rVal;
```

** Let’s execute the query and see the data in the table.



Results | Messages

	(No column name)
1	-1

Note that; the procedure returned the error message because there was not enough balance in Ali Güler's account.

SUMMARY

In this LAB, you’ve learned how SQL Server manages concurrent changes. You’ve learned how to combine related SQL statements into a single unit, called a transaction. You’ve learned the SQL transaction modes and how to create transaction. Then, you've understood to how to define transaction and worked with nested transactions. You’ve learned to how to work with save points. By learning these skills, you’ve learned to create a real-world example for the transaction.