(T19)討論 Transaction 和 ErrorHandling。討論 TryCatch
CourseGUID: e48417fc-9db5-4e99-822c-706c5ccef6cc
==========
(T19)討論 Transaction 和 ErrorHandling。討論 TryCatch
==========

==========

0. Summary

1.
We have to ensure a group of sql statement
can perform successfully together or unsuccessfully together.
Thus, we need SQL Transaction.
--BEGIN TRANSACTION;
BEGIN TRAN
...
--ROLLBACK TRANSACTION;
COMMIT TRAN;
2.
Prohibit to ROLLBACK any inner Transaction
No matter inner Transaction has name or not.
If you really want to roll back inner Transaction,
don't use inner Transaction, Use Savepoint with SavepointName
--BEGIN TRAN Tranl;
--PRINT @@TRANCOUNT; --1st TRANCOUNT, 1
--SAVE TRAN SavePoint;
--PRINT @@TRANCOUNT; --2nd TRANCOUNT, 1
--...
--ROLLBACK TRAN SavePoint;
--PRINT @@TRANCOUNT; --3rd TRANCOUNT, 1

----ROLLBACK TRAN Tranl
--COMMIT TRAN Tranl;
3.
When ROLLBACK Outter Transaction
No matter you have commit inner Transaction or not,
the inner Transaction will be forced to rollback too.
4.
--SELECT ERROR_NUMBER() AS [ERROR_NUMBER()] ,   --245
--   ERROR_MESSAGE() AS [ERROR_MESSAGE()] ,   --Conversion failed when converting the varchar value 'Account1' to data type int.
--   ERROR_PROCEDURE() AS [ERROR_PROCEDURE()] ,   --NULL
--   ERROR_STATE() AS [ERROR_STATE()] ,   --1
--   ERROR_SEVERITY() AS [ERROR_SEVERITY()] ,   --16
--   ERROR_LINE() AS [ERROR_LINE()]   --9
Reference:
https://docs.microsoft.com/en-us/sql/t-sql/functions/error-procedure-transact-sql
https://docs.microsoft.com/en-us/sql/t-sql/functions/error-state-transact-sql
4.1.
Each kind of Error has ONE Error number just like and id, and ONE ERROR_MESSAGE
In this case, ERROR_NUMBER is 245.
ERROR_MESSAGE is 'Conversion failed when converting the varchar value 'Account1' to data type int.'
4.2.
ERROR_PROCEDURE() returns the name of the stored procedure or trigger
where an error occurred that caused the CATCH block of a TRY…CATCH.
In this case, ERROR_PROCEDURE is NULL, because this is not stored procedure or trigger.
4.3.
ERROR_STATE is kind of flat for debugging.
Each specific condition that raises the error assigns a unique state code.
A SQL Server support engineer can also use the state code from an error to find the location
in the source code where that error is being raised,
which may provide additional ideas on how to diagnose the problem.
4.4.
ERROR_SEVERITY 16 means a general error.
This is kind of the category of error message.
4.5.
ERROR_LINE returns the lind number where an error occurred.
5.
We have to ensure a group of sql statement
can perform successfully together or unsuccessfully together.
Thus, we need SQL Transaction and try catch
--BEGIN TRY
--   --BEGIN TRANSACTION;
--   BEGIN TRAN
--   ...
--   --ROLLBACK TRANSACTION;
--   COMMIT TRAN;
--END TRY
--BEGIN CATCH
--   ...
--END CATCH
5.1.
--INSERT INTO BankTransaction
--   (FromBankAccountID ,
--   ToBankAccountID ,
--   Amount
--   )
--VALUES ('Account1' ,   -- datatype Error
--   'Account2' ,   --datatype Error
--   @TransferAmount
--   );
FromBankAccountID and ToBankAccountID need int type parameter,
but the input is character string.
This will raise an error and automaticly "ROLLBACK" to beginning of transaction.
and then jump to   BEGIN CATCH clause.

==========

1. CreateSampleData

```
--============================================================
--T019_01_CreateSampleData
--============================================================
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'BankTransaction' ) )
    BEGIN
        TRUNCATE TABLE BankTransaction;
        DROP TABLE BankTransaction;
    END;
GO -- Run the previous command and begins new batch
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'BankAccount' ) )
    BEGIN
        TRUNCATE TABLE BankAccount;
        DROP TABLE BankAccount;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE BankAccount
(
  BankAccountID INT PRIMARY KEY
                    IDENTITY(1, 1)
                    NOT NULL ,
  BankAccountName NVARCHAR(100) NULL ,
  BankAvailableBalance MONEY NULL,
 );
GO -- Run the previous command and begins new batch
INSERT  INTO BankAccount
VALUES  ( N'Account1', 41000 );
INSERT  INTO BankAccount
VALUES  ( N'Account2', 42000 );
GO -- Run the previous command and begins new batch
--Ch55toCh56_00_02
--Create [BankTransaction] table
CREATE TABLE BankTransaction
(
  BankTransactionID INT PRIMARY KEY
                        IDENTITY(1, 1)
                        NOT NULL ,
  FromBankAccountID INT FOREIGN KEY REFERENCES BankAccount ( BankAccountID )
                        NOT NULL ,
  ToBankAccountID INT FOREIGN KEY REFERENCES BankAccount ( BankAccountID )
                      NOT NULL ,
  Amount MONEY DEFAULT ( (0) )
                NULL,
 );
GO -- Run the previous command and begins new batch

SELECT  *
```

```
FROM    BankTransaction;
SELECT  *
FROM    BankAccount;
GO -- Run the previous command and begins new batch

=======================================================
```

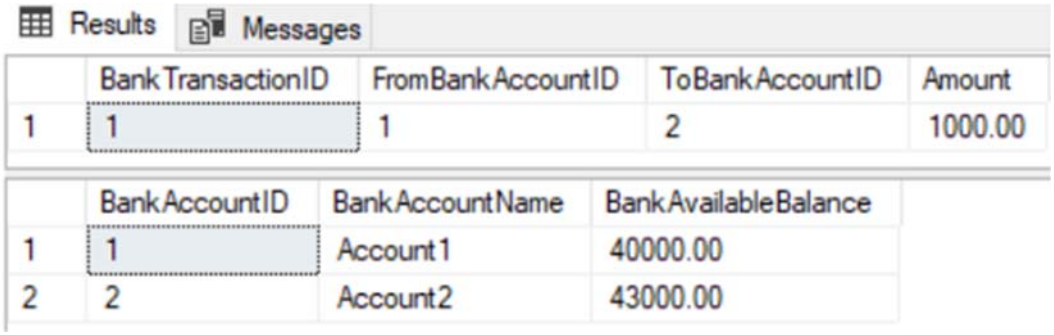
2. Transaction

```
--============================================================
--T019_02_Transaction
--============================================================
```

2.1. Reason to use Transaction

```
--============================================================
--T019_02_01
--Reason to use Transaction :
--We want a group of SQL statements perform successfully together or unsuccessfully together.
--E.g. Transfer $1000 From BankAccountID=1 to BankAccountID=2
DECLARE @TransferAmount INT = 1000;
--Adding new records to [BankTransaction] table
INSERT  INTO BankTransaction
        ( FromBankAccountID ,
          ToBankAccountID ,
          Amount
        )
VALUES  ( 1 ,
          2 ,
          @TransferAmount
        );
--Updating existing records
UPDATE  BankAccount
SET     [BankAvailableBalance] -= @TransferAmount
WHERE   [BankAccountID] = 1;
UPDATE  BankAccount
SET     [BankAvailableBalance] += @TransferAmount
WHERE   [BankAccountID] = 2;
GO -- Run the previous command and begins new batch
SELECT  *
FROM    [BankTransaction];
SELECT  *
FROM    dbo.BankAccount
WHERE   [BankAccountID] BETWEEN 1 AND 2;
GO -- Run the previous command and begins new batch
/*
1.
Transfer $1000 From BankAccountID=1 to BankAccountID=2
Before:
Account1 [BankAvailableBalance] = $41000
Account2 [BankAvailableBalance] = $42000
After:
Account1 [BankAvailableBalance] = $40000
Account2 [BankAvailableBalance] = $43000
2.
We have to ensure
--INSERT  INTO BankTransaction
and
--2 of the  UPDATE  BankAccount
```

```
have to peformed successfully together or unsuccessfully together.
Thus, we need SQL Transaction.
*/
```



Results | Messages

	BankTransactionID	FromBankAccountID	ToBankAccountID	Amount
1	1	1	2	1000.00

	BankAccountID	BankAccountName	BankAvailableBalance
1	1	Account1	40000.00
2	2	Account2	43000.00

2.2. BEGIN TRAN ... COMMIT TRAN

```
--============================================================
--T019_02_02
--BEGIN TRAN ... COMMIT TRAN
--Reason to use Transaction :
--We want a group of SQL statements perform successfully together or unsuccessfully together.
--E.g. Transfer $1000 From BankAccountID=1 to BankAccountID=2
--Begin a Transaction, and then commit the Transaction
--BEGIN TRANSACTION;
BEGIN TRAN
DECLARE @TransferAmount int = 1000;
--Adding new records to [BankTransaction] table
INSERT  INTO BankTransaction
        ( FromBankAccountID ,
          ToBankAccountID ,
          Amount
        )
VALUES  ( 1 ,
          2 ,
          @TransferAmount
        );
--Updating existing records
UPDATE  BankAccount
SET     [BankAvailableBalance] -= @TransferAmount
WHERE   [BankAccountID] = 1;
UPDATE  BankAccount
SET     [BankAvailableBalance] += @TransferAmount
WHERE   [BankAccountID] = 2;
GO -- Run the previous command and begins new batch
--COMMIT TRAN;
COMMIT TRAN;
SELECT  *
FROM    [BankTransaction]
SELECT  *
FROM    dbo.BankAccount
WHERE   [BankAccountID] between 1 AND 2
GO -- Run the previous command and begins new batch
/*
1.
Transfer $1000 From BankAccountID=1 to BankAccountID=2
Before:
Account1 [BankAvailableBalance] = $40000
Account2 [BankAvailableBalance] = $43000
After:
Account1 [BankAvailableBalance] = $39000
```

```
Account2 [BankAvailableBalance] = $44000
2.
We have to ensure
--INSERT  INTO BankTransaction
and
--2 of the  UPDATE  BankAccount
have to peformed successfully together or unsuccessfully together.
Thus, we need SQL Transaction.
--BEGIN TRANSACTION;
BEGIN TRAN
...
--COMMIT TRAN;
COMMIT TRAN;
*/
```

	BankTransactionID	FromBankAccountID	ToBankAccountID	Amount
1	1	1	2	1000.00
2	2	1	2	1000.00

	BankAccountID	BankAccountName	BankAvailableBalance
1	1	Account1	39000.00
2	2	Account2	44000.00

2.3. ROLLBACK TRAN

```
--============================================================
--T019_02_03
--ROLLBACK TRAN
--Reason to use Transaction :
--We want a group of SQL statements perform successfully together or unsuccessfully together.
--E.g. Transfer $1000 From BankAccountID=1 to BankAccountID=2
--Begin a Transaction, and then Rollback the Transaction
--BEGIN TRANSACTION;
BEGIN TRAN
DECLARE @TransferAmount int = 1000;
--Adding new records to [BankTransaction] table
INSERT  INTO BankTransaction
        ( FromBankAccountID ,
          ToBankAccountID ,
          Amount
        )
VALUES  ( 1 ,
          2 ,
          @TransferAmount
        );
--Updating existing records
UPDATE  BankAccount
SET     [BankAvailableBalance] -= @TransferAmount
WHERE   [BankAccountID] = 1;
UPDATE  BankAccount
SET     [BankAvailableBalance] += @TransferAmount
WHERE   [BankAccountID] = 2;
GO -- Run the previous command and begins new batch
--ROLLBACK TRANSACTION;
ROLLBACK TRAN;
SELECT  *
FROM    [BankTransaction]
SELECT  *
FROM    dbo.BankAccount
WHERE   [BankAccountID] between 1 AND 2
```

```
GO -- Run the previous command and begins new batch
/*
1.
Transfer $1000 From BankAccountID=1 to BankAccountID=2
Begin a Transaction, and then Rollback the Transaction
Before:
Account1 [BankAvailableBalance] = $39000
Account2 [BankAvailableBalance] = $44000
After:
Account1 [BankAvailableBalance] = $39000
Account2 [BankAvailableBalance] = $44000
2.
We have to ensure
--INSERT  INTO BankTransaction
and
--2 of the  UPDATE  BankAccount
have to peformed successfully together or unsuccessfully together.
Thus, we need SQL Transaction.
--BEGIN TRANSACTION;
BEGIN TRAN
...
--ROLLBACK TRANSACTION;
ROLLBACK TRAN;
In this case, a group of SQL statements perform unsuccessfully together.
*/
```

	BankTransactionID	FromBankAccountID	ToBankAccountID	Amount
1	1	1	2	1000.00
2	2	1	2	1000.00

	BankAccountID	BankAccountName	BankAvailableBalance
1	1	Account1	39000.00
2	2	Account2	44000.00

==========

3. Transaction Naming and Syntax

```
--BEGIN Transaction TransactionName
BEGIN TRAN TransactionName
--COMMIT Transaction TransactionName
COMMIT TRAN TransactionName
--BEGIN Transaction TransactionName
BEGIN TRAN TransactionName
--COMMIT Transaction
COMMIT TRAN
--BEGIN Transaction
BEGIN TRAN
--COMMIT Transaction
COMMIT TRAN
GO -- Run the previous command and begins new batch
-------------
--BEGIN Transaction TransactionName
BEGIN TRAN TransactionName
--ROLLBACK Transaction TransactionName
ROLLBACK TRAN TransactionName
--BEGIN Transaction TransactionName
BEGIN TRAN TransactionName
```

```
--ROLLBACK Transaction
ROLLBACK TRAN
--BEGIN Transaction
BEGIN TRAN
--ROLLBACK Transaction
ROLLBACK TRAN
GO -- Run the previous command and begins new batch

=========================================================================
```

4. Transaction Naming and Syntax

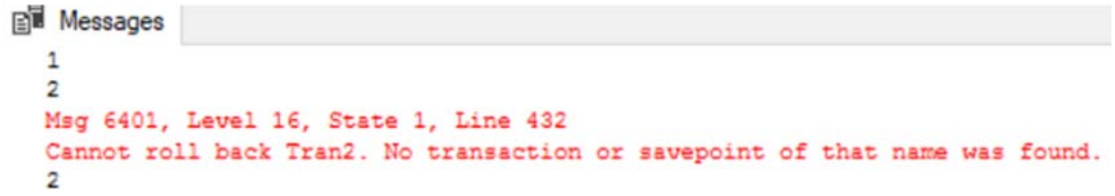
```
/*
1.
Prohibit to ROLLBACK any inner Transaction
No matter inner Transaction has name or not.
If you really want to roll back inner Transaction,
don't use inner Transaction, Use Savepoint with SavepointName
2.
When ROLLBACK Outter Transaction
No matter you have commit inner Transaction or not,
the inner Transaction will be forced to rollback too.
*/
```

4.1. Nested transactions basics

```
--=========================================================================
--T019_04_01
--Nested transactions basics
BEGIN TRAN Tranl;
PRINT @@TRANCOUNT;
       --1st TRANCOUNT, 1
BEGIN TRAN Tran2;
PRINT @@TRANCOUNT;
       --2nd TRANCOUNT, 2
COMMIT TRAN Tran2;
PRINT @@TRANCOUNT;
       --3rd TRANCOUNT, 1
COMMIT TRAN Tranl;
GO -- Run the previous command and begins new batch
/*
1.
@@TRANCOUNT
Reference:
https://docs.microsoft.com/en-us/sql/t-sql/functions/trancount-transact-sql
Returns the number of BEGIN TRANSACTION statements that have occurred on the current connection.
That means how many transaction has begun.
2.
This will return
--1
--2
--1
The 1st @@TRANCOUNT ==1, that means means 1 transaction has begun at that time.
The 2nd @@TRANCOUNT ==2, that means means 2 transaction has begun at that time.
The 1rd @@TRANCOUNT ==1, that means means 1 transaction has begun at that time.
Because at that time, the 2nd traction has been committed.
*/
```

4.2. ROLLBACK any inner Transaction

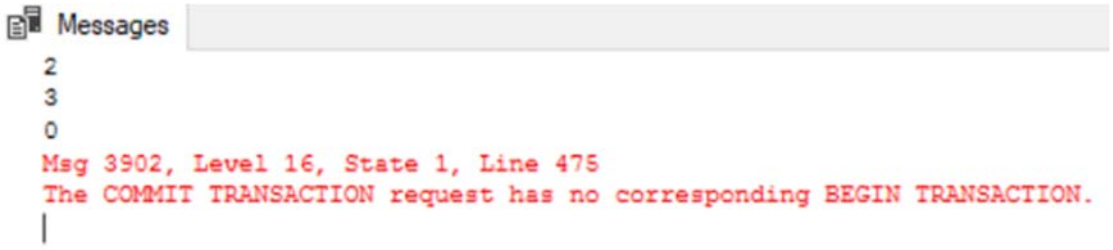
```
--============================================================================
--T019_04_02
/*
1.
Prohibit to ROLLBACK any inner Transaction
No matter inner Transaction has name or not.
If you really want to roll back inner Transaction,
don't use inner Transaction, Use Savepoint with SavepointName
*/
BEGIN TRAN Tranl;
PRINT @@TRANCOUNT;
       --1st TRANCOUNT, 1
BEGIN TRAN Tran2;
PRINT @@TRANCOUNT;
       --2nd TRANCOUNT, 2
ROLLBACK TRAN Tran2;
       -- * ROLLBACK Tran2 here, Error Message
PRINT @@TRANCOUNT;
       --3rd TRANCOUNT, 2
COMMIT TRAN Tranl;
GO -- Run the previous command and begins new batch
/*
1.
@@TRANCOUNT
Reference:
https://docs.microsoft.com/en-us/sql/t-sql/functions/trancount-transact-sql
Returns the number of BEGIN TRANSACTION statements that have occurred on the current connection.
That means how many transaction has begun.
2.
This will return
--1
--2
--Msg 6401, Level 16, State 1, Line 11
--Cannot roll back Tran2. No transaction or savepoint of that name was found.
--2
That means prohibit to ROLLBACK any inner Transaction.
*/
```



4.3. Savepoint

```
--============================================================================
--T019_04_03
/*
1.
Prohibit to ROLLBACK any inner Transaction
No matter inner Transaction has name or not.
If you really want to roll back inner Transaction,
don't use inner Transaction, Use Savepoint with SavepointName
```

```
*/
BEGIN TRAN Tranl;
PRINT @@TRANCOUNT;
	--1st TRANCOUNT, 1
BEGIN TRAN;
PRINT @@TRANCOUNT;
	--2nd TRANCOUNT, 2
ROLLBACK TRAN;
	-- * ROLLBACK here,  Error Message
PRINT @@TRANCOUNT;
	--3rd TRANCOUNT, 0
COMMIT TRAN Tranl;
GO -- Run the previous command and begins new batch
/*
1.
@@TRANCOUNT
Reference:
https://docs.microsoft.com/en-us/sql/t-sql/functions/trancount-transact-sql
Returns the number of BEGIN TRANSACTION statements that have occurred on the current connection.
That means how many transaction has begun.
2.
It will return
--1
--2
--0
--Msg 3902, Level 16, State 1, Line 14
--The COMMIT TRANSACTION request has no corresponding BEGIN TRANSACTION.
ROLLBACK TRAN will rollback both inner TRAN and outter TRAN.
Thus, no TRAN can be commit in the last line and output error message.
*/
```



4.4. ROLLBACK Outter Transaction

```
--============================================================================
--T019_04_04
/*
2.
When ROLLBACK Outter Transaction
No matter you have commit inner Transaction or not,
the inner Transaction will be forced to rollback too.
*/
SELECT  *
FROM    dbo.BankAccount
WHERE   BankAccountID = 1
--Nested transactions basics
BEGIN TRAN Tranl;
PRINT @@TRANCOUNT;
	--1st TRANCOUNT, 1
BEGIN TRAN;
UPDATE  dbo.BankAccount
SET     BankAccountName = 'NewName'
WHERE   BankAccountID = 1
PRINT @@TRANCOUNT;
	--2nd TRANCOUNT, 2
```

```
COMMIT TRAN;
             -- * commit Inner Transaction
PRINT @@TRANCOUNT;
       --3rd TRANCOUNT, 1
--COMMIT TRAN Tranl;
ROLLBACK TRAN Tranl
       -- * ROLLBACK outter Transaction
SELECT  *
FROM    dbo.BankAccount
WHERE   BankAccountID = 1
GO -- Run the previous command and begins new batch
/*
1.
@@TRANCOUNT
Reference:
https://docs.microsoft.com/en-us/sql/t-sql/functions/trancount-transact-sql
Returns the number of BEGIN TRANSACTION statements that have occurred on the current connection.
That means how many transaction has begun.
2.
It will return
--1
--2
--1
The last line will ROLLBACK every thing, so should return zero by logic.
Thus, ROLLBACK is hard to debug.
*/
```

Results Messages

```
(1 row affected)
1

(1 row affected)
2
1

(1 row affected)
```

	BankAccountID	BankAccountName	BankAvailableBalance
1	1	Account1	39000.00

	BankAccountID	BankAccountName	BankAvailableBalance
1	1	Account1	39000.00

4.5. Prohibit to ROLLBACK any inner Transaction

```
--=============================================================================
--T019_04_05
/*
1.
Prohibit to ROLLBACK any inner Transaction
No matter inner Transaction has name or not.
If you really want to roll back inner Transaction,
don't use inner Transaction, Use Savepoint with SavepointName
2.
When ROLLBACK Outter Transaction
No matter you have commit inner Transaction or not,
the inner Transaction will be forced to rollback too.
*/
SELECT  *
FROM    dbo.BankAccount
WHERE   BankAccountID = 1
```

```sql
BEGIN TRAN Tranl;
PRINT @@TRANCOUNT;
       --1st TRANCOUNT, 1
SAVE TRAN SavePoint;
PRINT @@TRANCOUNT;
       --2nd TRANCOUNT, 1
UPDATE  dbo.BankAccount
SET     BankAccountName = 'NewName'
WHERE   BankAccountID = 1
ROLLBACK TRAN SavePoint;
PRINT @@TRANCOUNT;
       --3rd TRANCOUNT, 1
--ROLLBACK TRAN Tranl
COMMIT TRAN Tranl;
SELECT  *
FROM    dbo.BankAccount
WHERE   BankAccountID = 1
GO -- Run the previous command and begins new batch
/*
1.
@@TRANCOUNT
Reference:
https://docs.microsoft.com/en-us/sql/t-sql/functions/trancount-transact-sql
Returns the number of BEGIN TRANSACTION statements that have occurred on the current connection.
That means how many transaction has begun.
2.
It will return
--1
--1
--1
1st @@TRANCOUNT == 1 because Tranl
2nd @@TRANCOUNT == 1 because still in Train 1
Then
--     ROLLBACK TRAN SavePoint;
This will only ROLLBACK to   SAVE TRAN SavePoint;
That means ROLLBACK
--     SAVE TRAN SavePoint;
--     PRINT @@TRANCOUNT;  --2nd @@TRANCOUNT
3rd @@TRANCOUNT == 1 because still in Train 1
*/
```

	BankAccountID	BankAccountName	BankAvailableBalance
1	1	Account1	39000.00

	BankAccountID	BankAccountName	BankAvailableBalance
1	1	Account1	39000.00

Results Messages

```
(1 row affected)
1
1

(1 row affected)
1

(1 row affected)
```

5. ErrorHandling, Transaction, TryCatch

```
--=============================================================================
--T019_05 : ErrorHandling, Transaction, TryCatch
--=============================================================================
```

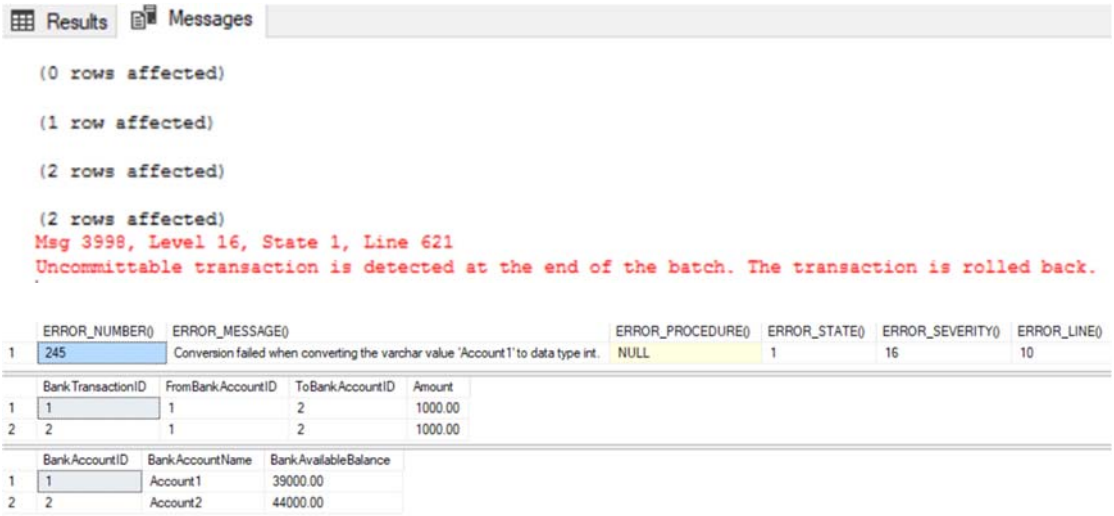
5.1. ErrorHandling, Transaction, TryCatch

```
--=============================================================================
--T019_05_01
--ErrorHandling, Transaction, TryCatch
BEGIN TRY
    BEGIN TRAN;
    DECLARE @TransferAmount INT = 1000;
        --Adding new records to [BankTransaction] table
    INSERT  INTO BankTransaction
            ( FromBankAccountID ,
              ToBankAccountID ,
              Amount
                )
    VALUES  ( 'Account1' , -- datatype Error
              'Account2' , --datatype Error
              @TransferAmount
                );
        --Updating existing records
    UPDATE  BankAccount
    SET     [BankAvailableBalance] -= @TransferAmount
    WHERE   [BankAccountID] = 1;
    UPDATE  BankAccount
    SET     [BankAvailableBalance] += @TransferAmount
    WHERE   [BankAccountID] = 2;

        --COMMIT TRANSACTION;
    COMMIT TRAN;
END TRY
BEGIN CATCH
    SELECT  ERROR_NUMBER() AS [ERROR_NUMBER()] ,
            ERROR_MESSAGE() AS [ERROR_MESSAGE()] ,
            ERROR_PROCEDURE() AS [ERROR_PROCEDURE()] ,
            ERROR_STATE() AS [ERROR_STATE()] ,
            ERROR_SEVERITY() AS [ERROR_SEVERITY()] ,
            ERROR_LINE() AS [ERROR_LINE()];
END CATCH;
SELECT  *
FROM    [BankTransaction];
SELECT  *
FROM    dbo.BankAccount
WHERE   [BankAccountID] BETWEEN 1 AND 2;
```

```
GO -- Run the previous command and begins new batch
/*
1.
--SELECT  ERROR_NUMBER() AS [ERROR_NUMBER()] , --245
--    ERROR_MESSAGE() AS [ERROR_MESSAGE()] ,         --Conversion failed when converting the varchar
value 'Account1' to data type int.
--    ERROR_PROCEDURE() AS [ERROR_PROCEDURE()] ,     --NULL
--    ERROR_STATE() AS [ERROR_STATE()] ,                    --1
--    ERROR_SEVERITY() AS [ERROR_SEVERITY()] , --16
--    ERROR_LINE() AS [ERROR_LINE()]                        --9
Reference:
https://docs.microsoft.com/en-us/sql/t-sql/functions/error-procedure-transact-sql
https://docs.microsoft.com/en-us/sql/t-sql/functions/error-state-transact-sql
1.1.
Each kind of Error has ONE Error number just like and id, and ONE ERROR_MESSAGE
In this case, ERROR_NUMBER is 245.
ERROR_MESSAGE is 'Conversion failed when converting the varchar value 'Account1' to data type int.'
1.2.
ERROR_PROCEDURE() returns the name of the stored procedure or trigger
where an error occurred that caused the CATCH block of a TRY…CATCH.
In this case, ERROR_PROCEDURE is NULL, because this is not stored procedure or trigger.
1.3.
ERROR_STATE is kind of flat for debugging.
Each specific condition that raises the error assigns a unique state code.
A SQL Server support engineer can also use the state code from an error to find the location
in the source code where that error is being raised,
which may provide additional ideas on how to diagnose the problem.
1.4.
ERROR_SEVERITY 16 means a general error.
This is kind of the category of error message.
1.5.
ERROR_LINE returns the lind number where an error occurred.
2.
We have to ensure a group of sql statement
can perform successfully together or unsuccessfully together.
Thus, we need SQL Transaction and try catch
--BEGIN TRY
--    --BEGIN TRANSACTION;
--       BEGIN TRAN
--       ...
--       --ROLLBACK TRANSACTION;
--       COMMIT TRAN;
--END TRY
--BEGIN CATCH
--     ...
--END CATCH
2.1.
--INSERT  INTO BankTransaction
--     (      FromBankAccountID ,
--           ToBankAccountID ,
--           Amount
--     )
--VALUES  ('Account1' ,    -- datatype Error
--           'Account2' , --datatype Error
--           @TransferAmount
--     );
FromBankAccountID and ToBankAccountID need int type parameter,
but the input is character string.
This will raise an error and automaticly "ROLLBACK" to beginning of transaction.
and then jump to   BEGIN CATCH clause.
3.
Transfer $1000 From BankAccountID=1 to BankAccountID=2
Begin a Transaction, and then Rollback the Transaction
Before:
Account1 [BankAvailableBalance] = $39000
Account2 [BankAvailableBalance] = $44000
After:
```

```
Account1 [BankAvailableBalance] = $39000
Account2 [BankAvailableBalance] = $44000
*/
```



Results Messages

```
(0 rows affected)

(1 row affected)

(2 rows affected)

(2 rows affected)
Msg 3998, Level 16, State 1, Line 621
Uncommittable transaction is detected at the end of the batch. The transaction is rolled back.
```

	ERROR_NUMBER()	ERROR_MESSAGE()	ERROR_PROCEDURE()	ERROR_STATE()	ERROR_SEVERITY()	ERROR_LINE()
1	245	Conversion failed when converting the varchar value 'Account1' to data type int.	NULL	1	16	10

	BankTransactionID	FromBankAccountID	ToBankAccountID	Amount
1	1	1	2	1000.00
2	2	1	2	1000.00

	BankAccountID	BankAccountName	BankAvailableBalance
1	1	Account1	39000.00
2	2	Account2	44000.00

5.2. ErrorHandling, Transaction, TryCatch, Raiserror

```
--============================================================================
--T019_05_02
--ErrorHandling, Transaction, TryCatch, Raiserror
BEGIN TRY
    BEGIN TRAN
    DECLARE @TransferAmount INT ,
        @FromBankAccountID INT ,
        @ToBankAccountID INT ,
        @FromBankAvailableBalance INT;
    SET @TransferAmount = 500000;
    SET @FromBankAccountID = 1;
    SET @ToBankAccountID = 2;
       --Declare @FromBankAvailableBalance int
    Select  @FromBankAvailableBalance = [BankAvailableBalance]
    from    BankAccount
    where   [BankAccountID] = 1;
       -- Throw an error if Not enough money available.
    if ( @FromBankAvailableBalance < @TransferAmount )
        Begin
            Raiserror('Not enough money available.',16,1)
        END
       --Adding new records to [BankTransaction] table
    INSERT  INTO BankTransaction
            ( FromBankAccountID ,
              ToBankAccountID ,
              Amount
                )
    VALUES  ( @FromBankAccountID ,
              @ToBankAccountID ,
              @TransferAmount
                );
       --Updating existing records
    UPDATE  BankAccount
    SET     [BankAvailableBalance] -= @TransferAmount
```

```
    WHERE    [BankAccountID] = @FromBankAccountID;
    UPDATE   BankAccount
    SET      [BankAvailableBalance] += @TransferAmount
    WHERE    [BankAccountID] = @ToBankAccountID;

        --COMMIT TRANSACTION;
    COMMIT TRAN;
END TRY
BEGIN CATCH
    SELECT  ERROR_NUMBER() AS [ERROR_NUMBER()] ,
            ERROR_MESSAGE() AS [ERROR_MESSAGE()] ,
            ERROR_PROCEDURE() AS [ERROR_PROCEDURE()] ,
            ERROR_STATE() AS [ERROR_STATE()] ,
            ERROR_SEVERITY() AS [ERROR_SEVERITY()] ,
            ERROR_LINE() AS [ERROR_LINE()]
END CATCH
SELECT  *
FROM    [BankTransaction]
SELECT  *
FROM    dbo.BankAccount
WHERE    [BankAccountID] between 1 AND 2
GO -- Run the previous command and begins new batch

/*
1.
--SELECT  ERROR_NUMBER() AS [ERROR_NUMBER()] , --50000
--    ERROR_MESSAGE() AS [ERROR_MESSAGE()] ,         --Not enough money available..
--    ERROR_PROCEDURE() AS [ERROR_PROCEDURE()] ,     --NULL
--    ERROR_STATE() AS [ERROR_STATE()] ,                    --1
--    ERROR_SEVERITY() AS [ERROR_SEVERITY()] , --16
--    ERROR_LINE() AS [ERROR_LINE()]                        --18
Reference:
https://docs.microsoft.com/en-us/sql/t-sql/functions/error-procedure-transact-sql
https://docs.microsoft.com/en-us/sql/t-sql/functions/error-state-transact-sql
1.1.
Each kind of Error has ONE Error number just like and id, and ONE ERROR_MESSAGE
1.2.
ERROR_PROCEDURE() returns the name of the stored procedure or trigger
where an error occurred that caused the CATCH block of a TRY…CATCH.
In this case, ERROR_PROCEDURE is NULL, because this is not stored procedure or trigger.
1.3.
ERROR_STATE is kind of flat for debugging.
Normally set to 1.
Each specific condition that raises the error assigns a unique state code.
A SQL Server support engineer can also use the state code from an error to find the location
in the source code where that error is being raised,
which may provide additional ideas on how to diagnose the problem.
1.4.
ERROR_SEVERITY 16 means a general error.
This is kind of the category of error message.
1.5.
ERROR_LINE returns the lind number where an error occurred.
2.
--if ( @FromBankAvailableBalance < @TransferAmount )
--    Begin
--        Raiserror('Not enough money available.',16,1)
--    END
2.1.
Throw an error if Not enough money available.
2.2.
--RAISERROR ( { msg_str | @local_variable }
--    { ,severity ,state }
```

```
--    [ ,argument [ ,...n ] ] )
--    [ WITH option [ ,...n ] ]
Reference:
https://docs.microsoft.com/en-us/sql/t-sql/language-elements/raiserror-transact-sql
2.2.1.
The first parameter, msg_str, is the error message.
2.2.2.
the second parameter, severity, is the severity level.
Severity level 16 means general errors and can be corrected by the user.
2.2.3.
The third parameter is state, and we should set default to 1.
RAISERROR only generates errors with state from 1 through 18.
Because the PDW engine may raise errors with state 0,
using a unique state number for different location
can help find which section of code is raising the errors.
3.
Transfer From BankAccountID=1 to BankAccountID=2
Begin a Transaction, and then Rollback the Transaction
Before:
Account1 [BankAvailableBalance] = $39000
Account2 [BankAvailableBalance] = $44000
After:
Account1 [BankAvailableBalance] = $39000
Account2 [BankAvailableBalance] = $44000
*/

==========
```

6. Store Procedure, ErrorHandling, Transaction, TryCatch, Raiserror

```
--=====================================================================
--T019_06 : Store Procedure, ErrorHandling, Transaction, TryCatch, Raiserror
--=====================================================================


/*
/// <summary>
/// Transfer amount of money from one account to another.
/// Rollback transaction if any error or
/// if the available amount in FromAccount is not enough.
/// </summary>
/// <param name="@FromBankAccountID">From bank account.</param>
/// <param name="@ToBankAccountID">To bank account.</param>
/// <param name="@TransferAmount">The amount of money you want to transfer.</param>
/// <returns>This is void method.</returns>
*/
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.ROUTINES
              WHERE     ROUTINE_TYPE = 'PROCEDURE'
                        AND LEFT(ROUTINE_NAME, 3) NOT IN ( 'sp_', 'xp_', 'ms_' )
                        AND SPECIFIC_NAME = 'spTransferMoneyFromTo' ) )
    BEGIN
        DROP PROCEDURE spTransferMoneyFromTo;
    END;
GO -- Run the previous command and begins new batch
CREATE PROC spTransferMoneyFromTo
    (
      @FromBankAccountID INT ,
```

```
    @ToBankAccountID INT ,
    @TransferAmount MONEY
)
AS
BEGIN
    BEGIN TRY
        BEGIN TRAN
            DECLARE @FromBankAvailableBalance INT;
            --Declare @FromBankAvailableBalance int
        Select  @FromBankAvailableBalance = [BankAvailableBalance]
        from    BankAccount
        where   [BankAccountID] = 1;
            -- Throw an error if Not enough money available.
        if ( @FromBankAvailableBalance < @TransferAmount )
            Begin
                Raiserror('Not enough money available.',16,1)
            END
            --Adding new records to [BankTransaction] table
        INSERT  INTO BankTransaction
            ( FromBankAccountID ,
              ToBankAccountID ,
              Amount
                )
        VALUES  ( @FromBankAccountID ,
              @ToBankAccountID ,
              @TransferAmount
                );
            --Updating existing records
        UPDATE  BankAccount
        SET     [BankAvailableBalance] -= @TransferAmount
        WHERE   [BankAccountID] = @FromBankAccountID;
        UPDATE  BankAccount
        SET     [BankAvailableBalance] += @TransferAmount
        WHERE   [BankAccountID] = @ToBankAccountID;

            --COMMIT TRANSACTION;
        COMMIT TRAN;
            -- Get out from stored procedure.
            RETURN;
    END TRY
    BEGIN CATCH
        SELECT  ERROR_NUMBER() AS [ERROR_NUMBER()] ,
            ERROR_MESSAGE() AS [ERROR_MESSAGE()] ,
            ERROR_PROCEDURE() AS [ERROR_PROCEDURE()] ,
            ERROR_STATE() AS [ERROR_STATE()] ,
            ERROR_SEVERITY() AS [ERROR_SEVERITY()] ,
            ERROR_LINE() AS [ERROR_LINE()]
    END CATCH
END;
GO -- Run the previous command and begins new batch
SELECT  *
FROM    [BankTransaction]
SELECT  *
FROM    dbo.BankAccount
WHERE   [BankAccountID] between 1 AND 2
EXEC spTransferMoneyFromTo 1, 2, 50000
```

```sql
SELECT  *
FROM    [BankTransaction]
SELECT  *
FROM    dbo.BankAccount
WHERE   [BankAccountID] between 1 AND 2
EXEC spTransferMoneyFromTo 1, 2, 1000
SELECT  *
FROM    [BankTransaction]
SELECT  *
FROM    dbo.BankAccount
WHERE   [BankAccountID] between 1 AND 2
GO -- Run the previous command and begins new batch
```

Results Messages

```
(2 rows affected)

(2 rows affected)

(1 row affected)
Msg 266, Level 16, State 2, Procedure spTransferMoneyFromTo, Line 0 [Batch Start Line 951]
Transaction count after EXECUTE indicates a mismatching number of BEGIN and COMMIT statements. Previous count = 0, current count = 1.

(2 rows affected)

(2 rows affected)

(1 row affected)

(1 row affected)

(1 row affected)

(3 rows affected)

(2 rows affected)
```

Results Messages

	BankTransactionID	FromBankAccountID	ToBankAccountID	Amount
1	1	1	2	1000.00
2	2	1	2	1000.00

	BankAccountID	BankAccountName	BankAvailableBalance
1	1	Account1	39000.00
2	2	Account2	44000.00

	ERROR_NUMBER()	ERROR_MESSAGE()	ERROR_PROCEDURE()	ERROR_STATE()	ERROR_SEVERITY()	ERROR_LINE()
1	50000	Not enough money available.	spTransferMoneyFromTo	1	16	31

	BankTransactionID	FromBankAccountID	ToBankAccountID	Amount
1	1	1	2	1000.00
2	2	1	2	1000.00

	BankAccountID	BankAccountName	BankAvailableBalance
1	1	Account1	39000.00
2	2	Account2	44000.00

	BankTransactionID	FromBankAccountID	ToBankAccountID	Amount
1	1	1	2	1000.00
2	2	1	2	1000.00
3	5	1	2	1000.00

	BankAccountID	BankAccountName	BankAvailableBalance
1	1	Account1	38000.00
2	2	Account2	45000.00

```sql
/*
1.
--SELECT  ERROR_NUMBER() AS [ERROR_NUMBER()] , --50000
--    ERROR_MESSAGE() AS [ERROR_MESSAGE()] ,         --Not enough money available..
--    ERROR_PROCEDURE() AS [ERROR_PROCEDURE()] ,     --NULL
--    ERROR_STATE() AS [ERROR_STATE()] ,                     --1
--    ERROR_SEVERITY() AS [ERROR_SEVERITY()] , --16
--    ERROR_LINE() AS [ERROR_LINE()]                         --18
Reference:
https://docs.microsoft.com/en-us/sql/t-sql/functions/error-procedure-transact-sql
https://docs.microsoft.com/en-us/sql/t-sql/functions/error-state-transact-sql
1.1.
```

```
Each kind of Error has ONE Error number just like and id, and ONE ERROR_MESSAGE
1.2.
ERROR_PROCEDURE() returns the name of the stored procedure or trigger
where an error occurred that caused the CATCH block of a TRY…CATCH.
In this case, ERROR_PROCEDURE is NULL, because this is not stored procedure or trigger.
1.3.
ERROR_STATE is kind of flat for debugging.
Normally set to 1.
Each specific condition that raises the error assigns a unique state code.
A SQL Server support engineer can also use the state code from an error to find the location
in the source code where that error is being raised,
which may provide additional ideas on how to diagnose the problem.
1.4.
ERROR_SEVERITY 16 means a general error.
This is kind of the category of error message.
1.5.
ERROR_LINE returns the lind number where an error occurred.
2.
2.1.
--EXEC spTransferMoneyFromTo 1, 2, 50000
Transfer From BankAccountID=1 to BankAccountID=2
Begin a Transaction, and then Rollback the Transaction
Before:
Account1 [BankAvailableBalance] = $39000
Account2 [BankAvailableBalance] = $44000
After:
Account1 [BankAvailableBalance] = $39000
Account2 [BankAvailableBalance] = $44000
2.2.
EXEC spTransferMoneyFromTo 1, 2, 1000
Transfer From BankAccountID=1 to BankAccountID=2
Begin a Transaction, and then commit the Transaction
Before:
Account1 [BankAvailableBalance] = $39000
Account2 [BankAvailableBalance] = $44000
After:
Account1 [BankAvailableBalance] = $38000
Account2 [BankAvailableBalance] = $45000
*/

==========
```

7. Clean up

```
--=====================================================================
--T019_07 : Clean up
--=====================================================================
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.ROUTINES
              WHERE     ROUTINE_TYPE = 'PROCEDURE'
                        AND LEFT(ROUTINE_NAME, 3) NOT IN ( 'sp_', 'xp_', 'ms_' )
                        AND SPECIFIC_NAME = 'spTransferMoneyFromTo' ) )
    BEGIN
        DROP PROCEDURE spTransferMoneyFromTo;
    END;
GO -- Run the previous command and begins new batch
-------------------------------------------------
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'BankTransaction' ) )
    BEGIN
        TRUNCATE TABLE BankTransaction;
```

```
        DROP TABLE BankTransaction;
    END;
GO -- Run the previous command and begins new batch
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'BankAccount' ) )
    BEGIN
        TRUNCATE TABLE BankAccount;
        DROP TABLE BankAccount;
    END;
GO -- Run the previous command and begins new batch
```