Myths and Truths about SQL Server Transaction

Simon Cho
Blog : Simonsql.com
Simon@simonsql.com

Please Support Our Sponsors

PASS

SQL Saturday is made possible with the generous support of these sponsors.
You can support them by opting-in and visiting them in the sponsor area.

PASS SQL saturday

Local User Groups

Orange County User Group
2rd Thursday of each month
bigpass.pass.org

Los Angeles User Group
3rd Thursday of each odd month
sql.la

Malibu User Group
3rd Wednesday of each month
sqlmalibu.pass.org

San Diego User Group
1st & 3rd Thursday of each month
meetup.com/sdsqlug
meetup.com/sdsqlbig

Los Angeles - Korean
Every Other Tuesday
sqlangeles.pass.org

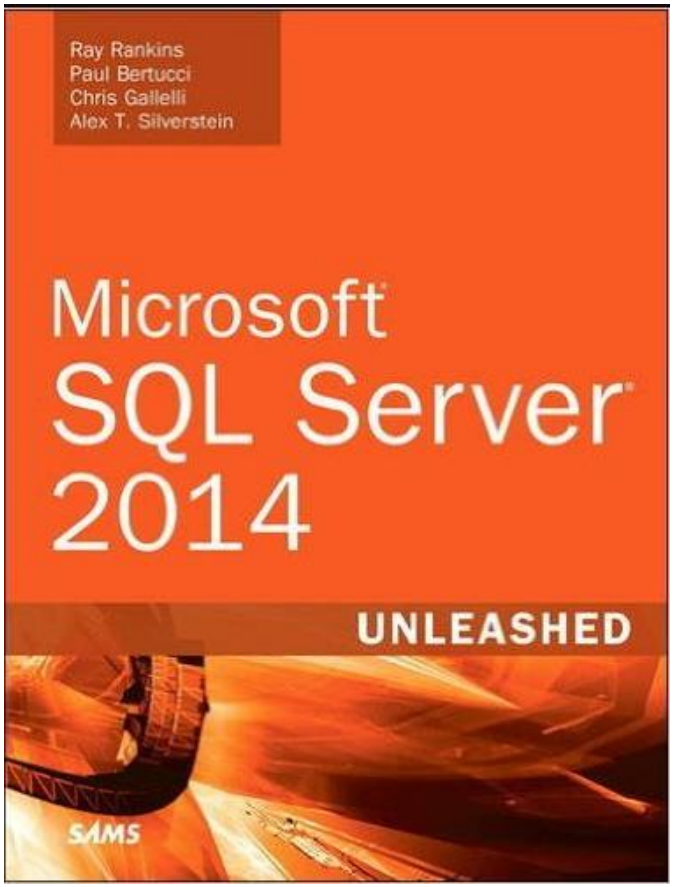
Who we are?

- SQLAngeles.com
  - Official Local Chapter group in SQLPASS.org
  - Only the community speak in Korean in SQL PASS.

- Blog : Simonsql.com/SQLmvp.kr
- Email : SQLAngeles@sqlpass.org, SQLAngeles@gmail.com

Study Book

- Microsoft SQL Server 2014 Unleashed
  - https://www.amazon.com/Microsoft-SQL-Server-2014-Unleashed/dp/0672337290



Question 1.

```
BEGIN TRAN A
   INSERT [Tbl] values('A')
   BEGIN TRAN B
      INSERT [Tbl] values('B')
   ROLLBACK TRAN B
COMMIT TRAN A
```

- Is it working?

Obviously Not.
Why? Rollback transaction will rollback everything.

Question 2.

```
BEGIN TRAN AAAA
   INSERT [Tbl] values('A')
   BEGIN TRAN BBBB
      INSERT [Tbl] values('B')
   COMMIT TRAN AAAA
COMMIT TRAN BBBB
```

- Is it working?

No? Answer is Yes.
Why? Transaction don't care the name.
Because nested transaction doesn't support.
How? Let's look at inside transaction lo

Question 3

```
BEGIN TRAN
   TRUNCATE TABLE [Tbl]
ROLLBACK TRAN
```

- Is it working?

Yes it is.

Why? Truncate table is no logging for each row deletion. But, it's fully logged for system table change.

Remember, this is RDBMS. Which mean, all DDL statement can be rollbacked.

Demo - Nested Transaction

Question 4.

- Which recovery mode is the fastest for this query?

```
Truncate table Table1
GO

INSERT INTO Table1
SELECT TOP 1000000 'A'
  FROM sys.objects a
  CROSS JOIN sys.objects b
  CROSS JOIN sys.objects c
  CROSS JOIN sys.objects d
  CROSS JOIN sys.objects e
  CROSS JOIN sys.objects f
```

Hint : This table doesn't have any index

A. SIMPLE Recovery mode
B. BULK_LOGGED Recovery mode
C. FULL Recovery mode

Definitely Simple??
Actually all same disregarding recovery mode.
Why? The Statement itself required full logged.
Hug?

Question 4 - Solution

- Minimal Logging within Simple or Bulk_Logged Mode.
  - Minimally Logged, No Logging.

```
Truncate table Table1
GO

INSERT INTO Table1  WITH(TABLOCK)
SELECT TOP 1000000 'A'
  FROM sys.objects a
  CROSS JOIN sys.objects b
  CROSS JOIN sys.objects c
  CROSS JOIN sys.objects d
  CROSS JOIN sys.objects e
  CROSS JOIN sys.objects f
```

How much faster?
Vary. Depends on system.

In my work station,
400 times less IO
10 times faster

A. SIMPLE Recovery mode
B. BULK_LOGGED Recovery mode
C. FULL Recovery mode

Demo – Minimal Logging

What is Recovery Mode?

- Full
  - Log backup required
  - Full logging everything
    - Note : Starting SQL 2008, some statement can be small logged even in full recovery mode.
- Simple
  - Automatically reclaims log space.
  - No log backup
    - Log chain is broken – Not able to restore Point-in-time
  - Fully Support minimal logging operation
- Bulk_Logged
  - Log backup required
    - Note : The log size pretty much same as full recovery mode even the minimal logging operation.
  - Fully Support minimal logging operation
  - Log chain is NOT broken. Means, Log backup required.
  - Purpose : Temporary change recovery mode for Minimal logging STMT in Full logging Database.
    - Note
      - Unfortunately, Mirrored database can't changed.
      - When transactional replication is enabled, most of statement fully logged even Bulk Logged mode

3 things for TX error handling

1. @@TranCount
   - It can determine nested transaction.
   - It doesn’t know un-committable transaction.

2. XACT_STATE()
   - 1 : Commitable active TX.
   - 0 : No active TX.
   - -1 : Uncommitable active TX.
   - It can determine un-committable transaction.
   - It doesn’t know nested transaction.

3 things for TX error handling - Cont

3. SET XACT_ABORT

- On : In case of run-time error, the entire TX is terminated and rolled back.

- Off(Default) : [in some cases only] the Transact-SQL statement that raised the error is rolled back and the transaction continues processing.
  - Note *: Depending upon the severity of the error, the entire transaction may be rolled back even when SET XACT_ABORT is OFF.

Rollback Transaction

- Rollback Tran
  - Do NOT print any error message
  - Rolls back all inner transactions
  - @@TRANCOUNT system function to 0

Question 5.

```
CREATE PROCEDURE USP_INNER_TRAN
@output varchar(255) OUTPUT
AS
BEGIN
SET NOCOUNT ON;
  BEGIN TRANSACTION

  BEGIN TRY
    SELECT 1/0 -- Bug
    COMMIT TRANSACTION

    SELECT @output = 'DONE'
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION
    SELECT @output = 'FAIL'
  END CATCH

Return 0
END
GO
```

Question 5. - Cont

```
BEGIN TRANSACTION
  BEGIN TRY
    DECLARE @output VARCHAR(255)
    EXEC USP_INNER_TRAN @output=@output OUTPUT

    SELECT 'InTry' AS WhereWeAre
         , @@TRANCOUNT
         , @@ERROR AS Error
         , ERROR_MESSAGE() AS ERR_MESSAGE
  COMMIT TRANSACTION
END TRY
BEGIN CATCH
  SELECT 'InCatch' AS WhereWeAre
       , @@TRANCOUNT
       , @@ERROR AS Error
       , ERROR_MESSAGE() AS ERR_MESSAGE
  ROLLBACK TRANSACTION
END CATCH
```

- Is it working?

Question 5. - Cont

```
BEGIN TRANSACTION
  BEGIN TRY
    DECLARE @output VARCHAR(255)
    EXEC USP_INNER_TRAN @output=@output OUTPUT

    SELECT 'InTry' AS WhereWeAre
         , @@TRANCOUNT
         , @@ERROR AS Error
         , ERROR_MESSAGE() AS ERR_MES
  COMMIT TRANSACTION
END TRY
BEGIN CATCH
  SELECT 'InCatch' A
       , @@TRANC
       , @@E
       , E
  ROLLBACK
END CATCH
```

- Is it working?

Results Messages

```
(1 row(s) affected)
Msg 3903, Level 16, State 1, Line 66
The ROLLBACK TRANSACTION request has no corresponding BEGIN TRANSACTION
```

SAGE

Question 6

- Anything wrong this SP?

```
CREATE PROC TestTran
AS
BEGIN
  DECLARE @Err INT
  BEGIN TRANSACTION
  BEGIN TRY
    SELECT * FROM [TypoNoTable]
    COMMIT TRAN
  END TRY
  BEGIN CATCH
    SET @ERR = @@ERROR
    IF @Err <> 0 BEGIN
      RAISERROR('Encountered an error, rollback', 16,1)
      IF @@TRANCOUNT <> 0 BEGIN
        ROLLBACK
      END
      RETURN (@ERR)
    END
  END CATCH
END
GO
EXEC TestTran
```

Demo – Exception Case

Best Practices to Compose SP

- Goal
  - Transaction match (Begin and Rollback/Commit)
    - It's not simple
    - Try catch isn't perfect.
    - Avoid Nested transaction for easy handling
  - Error logging for troubleshooting
    - Need to save Enough information in case of error
- What we need to do?
  - Define Return code and return it properly
  - Use Raiserror statement
  - Transaction Handling
  - Logging for error message

Best Practices to Compose SP

- Try Catch isn't enough for Error Handling
- Basic Rule
  - Set XACT_Abort on
  - Set Nocount on
  - Define return code SP and check the return code.
  - Avoid nested transaction
    - Inner SP need to check transaction status first before begin transaction
  - Transaction should be short enough
    - Execute plan, Minimal Logging
  - Commit and Rollback transaction at the end.
    - Use Goto statement to handle Commit/Rollback

Best Practices to Compose SP

- Errors
  - Expected Error
    - In case of parameter data wrong
    - Return is wrong
  - UnExpected Error
    - Not expected error
      - Logical bug
      - Any Unexpected system error : Server down…
- Consider Global Error Logging table
  - Global
    - Database should be online all the time.
    - Consolidated error table
- Error handling using GoTo statement.
  - This is old fashion. But, it’s still best I thought.

Demo - Best Practices SP Structure

Bulk Recovery Mode

- Protects against media failure.
- Normal transaction(Full logging Transaction)
  - Same as Full recovery mode.
- Minimal logging operation
  - Provides the best performance and least log space usage.
  - LDF file doesn’t have full traction log.
  - Point-in-time is NOT available.(StopAt doesn’t allow)
  - Log backup is contain whole page(extent) data.
- Please check below URL before change Bulk recovery mode.
  - https://technet.microsoft.com/en-us/library/ms190692.aspx
  - https://msdn.microsoft.com/en-us/library/ms179451.aspx

Bulk Recovery Mode - Cont

- There is risk.
- But, performance gain is very big.
- And it’s required only Target DB.
- Ex) how to use
  - Temporary change to Bulk recovery mode for Minimal logging STMT in Full logging Database.
  - Run the log backup more frequently.
  - Change back after minimal logging operation.

Q & A

Simon@simonsql.com

Welcome for any SQL question.
Please feel free to email me.