Research on SQL Server Bulk Operation Mechanism

Aiwu Li

Department of Computer Science, Guangdong Posts& Telecom Vocational Technology College, Guangzhou, China

*Corresponding author: aiwuli@gupt.edu.cn

Abstract. This paper illustrates the meaning of different database recovery modes, represented by the select into command, and verifies the differences in the redo data produced by bulk operations in different database recovery modes with a detailed experimental procedure.

1. Introduction
Bulk operations refers to select into, create index, insert into select, bcp, bulk insert and other similar commands, the common feature of these commands is that they can be executed once to add a large amount of data to the database. If each row of records added by the bulk operation is recorded in the redo file, the amount of redo data generated will generally exceed the actual amount of data added, which will cause the redo file size to grow dramatically.

For different database configuration and different bulk operations, the redo data generated by bulk operations vary greatly. If you set the appropriate database options and attach the appropriate table locking options when executing bulk operations, you can significantly reduce the amount of redo data it generates, i.e., generate redo data with minimal logging.

2. Impact on generating redo data of database recovery mode for bulk operations
Recovery mode is an important attribute of the database, this attribute can determine the way to generate redo data for bulk operations in the database and how to reuse the VLF in the log file.

Database recovery mode is divided into the following three types: full, bulk_logged, simple.

If the database is in full recovery mode, the data added by the bulk operation will be saved in the redo log, which is the meaning of full. Note here that after the database is set to full recovery mode or bulk_logged recovery mode, you also have to perform a full database backup to make it in the state of maintaining a full log sequence, otherwise, the database uses the redo log in the same way as if the database is in simple recovery mode.

The simple recovery mode of the database and the bulk_logged recovery mode have the same characteristics in terms of bulk operations that produce redo data. The difference between the simple mode and the other two recovery modes is that the VLF in the redo log file is constantly reused.

In the simple recovery mode and the bulk_logged recovery mode, the bulk operations generate redo data in the minimal-logging mode.

The minimal-logging mode means that only the number of extent or data page whose data is changed by the bulk operation are recorded in the log file, and not the data affected by the operation, which can significantly reduce the amount of redo data generated by the bulk operation.
3. Redo data generated by the select into command

The select into command is used to create a new table from an existing source table. Executing this command will copy the structure of the source table as the structure of the new table, and after getting the structure of the new table, it will also copy the data from the source table to the new table according to the additional conditions.

The following process examines the different characteristics of the select into command for producing redo data when the database is set to full recovery mode and bulkLogged log recovery mode, respectively. The testing platform is SQL Server 2019 on Windows 10, the client is sqlcmd.

3.1. Full recovery mode

First prepare the test data, create the test table t in the temporary database tempdb as the source table for the select into operation to get the data, set the length of its b column to 1500 bytes in order to easily verify the effect, and then add 100 rows to it.

1> use tempdb
2> go
1> create table t (a int identity, b char (1500) default 'xxxxx')
2> set nocount on
3> go
1> insert into t default values
2> go 100

Create the database dbFull and set its recovery mode to full, then perform a full database backup to put it in full log sequence maintenance mode.

1> create database dbFull
2> alter database dbFull set recovery full
3> backup database dbFull to disk ='d:\sqldata\dbFull.bak'
4> go

To get the redo data generated by the select into command, query the dbFull database for the current maximum LSN before executing the select into command.

1> use dbFull
2> go
1> select max ([current LSN]) max_lsn from fn_dblog (null, null)
2> go
max_lsn
-----------------------
'00000025:00000138:0003'

From the above result, we can confirm that the current maximum LSN of dbFull database is "00000025:00000138:0003". To use the fn_dblog function to query the redo data generated by the next operation, just attach the query condition: [Current LSN] > '00000025:00000138:0003'.

Execute the following select into operation to create a new table tt from the t table of the tempdb database.

1> use dbFull
2> go
1> select * into tt from tempdb.dbo.t
2> go

Each record in table t is larger than 1500 bytes in size. To verify that all the data added to table tt is included in the log records, simply query the newly generated redo data for those log records that are longer than 1500 bytes in length.

1> use dbFull
2> go
1> select operation, context, allocunitname, [log record length]
2> from fn_dblog (null, null)
where [current lsn] > '00000025:00000138:0003' and [log record length] > 1500

go

| Operation                | context     | allocunitname | log record length |
|--------------------------|-------------|---------------|-------------------|
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |
| LOP_FORMAT_PAGE          | LCX_HEAP    | dbo.tt        | 7752              |

(20 rows affected)

The 20 log records with operation type LOP_FORMAT_PAGE hold all the data added to the table, and each log record holds 5 rows, which can be verified by querying the [Log Record] value of its log record.

Since the [Log Record] value is too large, we execute the following query in SSMS to get its complete redo data.

```sql
Select top 1 [Log Record]
From fn_dblog (null, null)
Where [current lsn] > '00000025:00000138:0003' and [log record length] > 1500
And Operation = 'LOP_FORMAT_PAGE'
```

To facilitate viewing, copy the query results in the [Log Record] field value into a blank Word document, look for the appearance of the string "787878787878", we can find that it appears exactly 5 times, such as the first time in the form of the following.

```
…010000007878787878202020202020202020…
```

After careful observation, you can find that this happens to be the first row of records added to the tt table, where the 01000000, arranged in reverse byte order, yields 0000000001, i.e. 0x1, which is the value of the column a, i.e. integer 1, 7878787878 is the default value of the column b, i.e. "xxxxxxx", followed by a number of 20 indicates the ASCII code (decimal 32) of the space character appended to the string "xxxxxxx", in order to make its length reach the 1500 characters required by the type of column b, the second occurrence, the 01 in the above result becomes 02, that is the value of column a of the second row, the other data is the same, and so on. This confirms that all the data added by the select into command is saved in the redo data.

In order to compare the difference with the amount of redo data generated in bulk_logged mode, the following query is executed to get the total size of the redo data generated by the above select into command.

```sql
1> use dbFull
2> go
```
3.2. Bulk_logged recovery mode
Create a test database dbBulk, set its recovery mode to bulk_logged, and then perform a full database backup to put it in full log sequence maintenance mode.

```sql
1> create database dbBulk
2> alter database dbBulk set recovery bulk_logged
3> backup database dbBulk to disk = 'd: \sqldata\ dbBulk. bak'
4> go
```

To get the redo data produced by the select into command, first query the dbBulk database for the current maximum LSN.

```sql
1> use dbBulk
2> go
1> select max ([current LSN]) max_lsn from fn_dblog (null, null)
2> go
max_lsn
--------------------------------
00000025:00000158:000e
```

From the above query result, we can determine the maximum LSN of dbBulk database is "00000025:00000158:000e".

Execute the select into command in the dbBulk database, and create the tt table from the t table in tempdb.

```sql
1> use dbBulk
2> go
1> select * into tt from tempdb.dbo.t
2> go
```

After executing the above select into command, similar to the previous section, query the log records generated by this bulk operation that exceed 1500 bytes in size.

```sql
1> use dbBulk
2> go
1> select operation, context, allocunitname, [log record length]
2> from fn_dblog (null, null)
3> where [current lsn] > '00000025:00000158:000e' and [log record length] > 1500
4> go
```

From the query result above, we can know that there are no log records that satisfy the query condition, that is, the length of all newly generated log records is not more than 1500 bytes, and thus we can determine that the data added to the table is not saved to the redo data.

Remove the condition [log record length] > 1500, and re-execute the above query.

```sql
1> use dbBulk
2> go
1> select operation, context, [log record length]
2> from fn_dblog (null, null)
```
3> where [current lsn] > '00000025:00000158:000e'
4> go
Operation context log record length
------------------------ ------------------------ ---------------------
LOP_MODIFY_HEADER LCX_PFS 76
......
LOP_ROOT_CHAN LCX_CLUSTERED 96
LOP_SET_BITS LCX_ML_MAP 56
LOP_SET_BITS LCX_ML_MAP 56
LOP_SET_BITS LCX_ML_MAP 56
LOP_SET_BITS LCX_ML_MAP 56
LOP_COMMIT_XA LCX_NULL 84
(127 rows affected)
The lines marked in bold in the above result, the operation type is LOP_SET_BITS, the object type is LCX_ML_MAP, these log records correspond to the operation modifying the bitmap data in the BCM data page in the data file, record the information of the area where the data change occurred, the "ML" in the object type LCX_ML_MAP means "Minimal Logged", that is, the minimum logging method. The total size of the redo data that the above select into command produces can be queried as follows.
1> select sum ([Log Record Length]) sum_of_logrecord_length
2> from fn_dblog (null, null)
3> where [current lsn] > '00000025:00000158:000e'
4> and operation<>'LOP_IDENT_NEWVAL'
5> go
sum_of_logrecord_length
-----------------------
10132
Referring to the previous section, the total size of redo data generated by the same select into command in full recovery mode is 273616 bytes, which is about 27 times of the current 10132 bytes. This difference is mainly because in full recovery mode, all data added to the table is saved in the redo log, while in bulk-logged recovery mode, only the ID number of the extents where the data change occurred is saved. If more records are added using the select into command, the difference in the amount of redo data generated in the two log recovery modes becomes even more significant.

4. Conclusion
Bulk operations are logged to the redo file for each row in full recovery mode, and the size of the redo file grows dramatically. In bulk_logged mode, using minimal logging for most bulk operations, only the ID number of the extents that have changed are recorded, and only very little redo data is generated. This article uses the fn_dblog function to give a specific validation process for the select into operation, and other bulk operations can follow this example.

References
[1] Ken Henderson. Guru’s Guide to SQL Server Architecture and Internals, Addison Wesley Professional, 2004.
[2] Kalen Delaney. Inside Microsoft SQL Server 2005: the Storage Engine, Microsoft Press, 2007.
[3] Dmitri Korotkevitch, Pro SQL Server Internals, 2nd ed., Apress, 2016.
[4] Kalen Delaney. Inside Microsoft SQL Server 2005:Query Tuning and Optimization, Microsoft Press, 2007.