1. Introduction

SQL Server, as an enterprise-class database system, needs to ensure data consistency when the server fails and needs to be restarted, that is, if the data modified by the transaction submitted before the failure has not yet been written to disk, the relevant operation should be re-executed to ensure that the data is not lost, while the data modified by the uncommitted transaction should be undone if the data has been written to disk. The former operation is called redo and the latter is called undo, and the whole process is called instance recovery. In this paper, we study the process of instance recovery and focus on redo log record locating mechanism.

2. The physical structure of the redo log file

The log file starts with a file header of 8192 bytes in size, and the subsequent part is divided into several VLFs (Virtual Log File). The header of the VLF contains information about its size, so that the information can be quickly located to the starting position of the next VLF. Each VLF is coded with a file sequence number starting with 1 in the order in which it is used.

Execute the dbcc loginfo command to get the information of each VLF in the current database as shown below.

```
1> dbcc loginfo
2> go

| FileId | FileSize | StartOffset | FSeqNo | Status |
|--------|----------|-------------|--------|--------|
| 2      | 253952   | 8192        | 23     | 2      |
| 2      | 327680   | 262144      | 0      | 0      |
```

In the above result, FseqNo indicates the sequential number of the VLF, and status 2 indicates that the VLF is currently in use.

Each VLF is divided into data blocks of 512 bytes in size, which are numbered sequentially starting from 0 within the respective VLF, and several data blocks constitute log segments of variable size. SQL Server strives to store log records belonging to the same transaction in a single log segment.
3. Using LSN to locate redo log record
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 bulk_logged log recovery mode, respectively.

3.1. LSN of log record
Each log record has a Current LSN attribute, also known as the LSN of the log record (that is, Log Sequence Number), which is an integer of type numeric(25,0) and length of 10 bytes, divided into three parts, such as 000000d5:0000001241:0018.

The meaning of the three parts of LSN are as follows.
1) The VLF number where the log record is located, with a length of 4 bytes.
2) The starting block number of the log segment where the log record is located (i.e., the block number of the above 512-byte size), with a length of 4 bytes, and the block numbers in each log segment are numbered sequentially starting from 0.
3) The slot number where the log record is located, with a length of 2 bytes. The last part of the log segment in VLF is the list of slot offsets.

From the above conclusion, it can be found that LSN is essentially the physical address of log records.

Log records are added to the redo log file sequentially in the order they are generated, and the LSN of a new log record is always larger than the LSN of an old log record, so that the LSN of a log record not only indicates its physical address, but also reflects the order in which log records are added to the log file.

3.2. Mechanism for locating redo log records
Locating a specified log record in a redo log file requires knowing its LSN information. After knowing its LSN information, its position in the redo log file can be determined in the following order.
1) Decompose the LSN into three parts according to their respective lengths, and get its VLF number, starting data block number, and slot number in the log segment.
2) Determine the starting position of VLF in the whole redo log file. The starting position of the first VLF is 8192 bytes away from the beginning of the redo log file, and each VLF has its own size information stored in its header, so that the starting position of the next VLF can be located from this information.
3) Based on the second part of the LSN, i.e., the data block number, the starting location of the log segment where the log record is located is obtained. Assuming that the starting position of VLF is vlf_start and the data block number is block_no, the formula to calculate the starting position of the log segment where the log record is located is

\[ \text{vlf}\_\text{start} + \text{block}\_\text{no} \times 512 \] (1)

4) The information of its size is stored at the beginning of the log segment, from which the list of slot offsets at the end of the log segment can be located, i.e. the starting position of each log record in the log segment. According to the third part of the LSN, i.e., the slot number of the log segment, the specific location of the log record can be obtained from the slot offset list.

3.3. Log record location example
The following command gets the LSN of the first 5 lines of log records of the log file.
1> select top 5 [current lsn] from fn_dblog(null,null)
2> go

| current lsn |
|-------------|
| 00000017:000001b2:0001 |
| 00000017:000001b2:0002 |
| 00000017:000001b2:0003 |
| 00000017:000001b2:0004 |
| 00000017:000001b3:0001 |

From the above query result, we can see that the first 4 lines of log records are saved in the same log segment. The first part of its LSN 00000017 is converted to hexadecimal as 23, and the starting position of VLF 23 can be found out using the dbcc loginfo command as 8192.

1> dbcc loginfo
2> go

| FileId | FileSize | StartOffset | FSeqNo |
|--------|----------|-------------|--------|
| 2      | 253952   | 8192        | 23     |
| 2      | 327680   | 262144      | 0      |

The second part 000001b2 is the data page number, which corresponds to the decimal value of 434, in accordance with the calculation formula 1 in section 3.1 to obtain the starting position of this log segment: $8192 + 434 \times 512 = 230400$.

230400 converted to hexadecimal, the result is 38400, that is, the above four lines of log records where the starting location of the log segment is 38400.

Use a hexadecimal editing tool (e.g. WinHex) to open the log file of this database.

| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|--------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00039400 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00039410 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00039420 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00039430 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00039440 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00039450 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00039460 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00039470 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

**Figure 1.** Open the redo log file of the database in WinHex
The first two bytes of the first box in Figure 1 represent the number of slots in this log segment, and the last two bytes represent the size of this log segment. The first two bytes are converted to 0004, which means that this log segment contains 4 slots, i.e., 4 lines of log records, and the last two bytes are converted to 0180, which means that the size of this log segment is 0x180. Thus, it can be seen that the data at 3857F (i.e., 2800) is the last part of the log segment, and the 4 bytes in the second box represent the slot offset list. The second part of the box, from right to left, is the starting position of the first slot, the second slot, the third slot and the fourth slot in this log segment.

- 2800 is converted to 0x28, which means the starting position of the 1st slot is 38400+28=38428.
- 9800 is converted to 0x98, which means the starting position of the 2nd slot is 38400+98=38498.
- D000 is 0xD0 after position conversion, which means the starting position of the third slot is 38400+D0=384D0.
- 4001 is converted to 0x140, which means the starting position of the 4th slot is 38400+140=38540.

Once the starting position of the log record is determined, each line of log record information can be correctly located by combining it with its own length information stored in the log record.

4. Locating Redo Log Record in Instance Recovery

4.1. Main components of redo log record
Each log record contains the number of the transaction it is in and the LSN of the previous log record in this transaction, so that the log records belonging to the same transaction are gradually composed into a complete chain to complete the rollback. Note that the log records do not contain the LSN of the next log record in the same transaction, when you need to redo, you can directly execute them in order of the log records.

For the common DML statements, the corresponding log records will contain the data page number and the slot number of the data page they affect. In addition, the log records generated by the insert statement will contain the data added to the table, the log records generated by the delete statement will contain the data deleted, and the log records generated by the update statement will contain the data before and after the update, which are necessary to complete the redo or undo operation when the instance recovery is performed.

4.2. Function of checkpoints
SQL Server performs checkpoint operations as needed. The purpose of performing checkpointing is twofold.

1) Write dirty data pages from the data buffer to disk to reduce the time required for the database instance recovery process.

2) Determine the starting LSN for the instance recovery, which is called MinLSN, and is the smaller of the LSN at the time of checkpoint initiation and the starting LSN of the earliest currently active transaction. When executing the checkpoint, the MinLSN in the redo log file is written to the log record corresponding to this checkpoint operation, and finally the LSN at the beginning of this checkpoint operation is written to the boot page of the database, which is the data page 9 of the database master data file. In addition, in simple recovery mode, the VLFs that do not contain active transactions in the redo log file are marked as truncable, so that the VLFs can be reused.

4.3. Locating Redo Log Record in undo Operations
When the server is restarted due to a failure, a redo operation is first performed on each database.

If the server is restarted due to a failure after the transaction is committed and before the checkpoint is started, at this time, the data modified by the committed transaction has not yet been written to disk, and in order to ensure data consistency, the data not written to disk is to be redone through the log records in the redo log file, which is the execution of the so-called redo process. The whole redo process is as follows.
1) Read the start LSN of the last checkpoint from the start data page of the database.
2) Locate the corresponding log record by this LSN value in the redo log file and extract the MinLSN in the log record.
3) Execute log records sequentially from MinLSN until the last log record of the redo log file is reached.

4.4. Locating Redo Log Records in undo Operations
After the execution of the redo operation to the database is completed, then the undo operation is executed. Each unfinished transaction, starting from its last log record, executes the operation with the opposite effect of the corresponding operation of the log record to complete the rollback of the corresponding operation, such as the opposite operation of the insert operation is the delete operation, and the opposite operation of the delete is the insert operation, and the opposite operation of the update operation is to execute the update operation again to undo the original update operation effect.

In each log record, the LSN number of the previous log record in its transaction is recorded, and after completing the undo operation on itself, it can then roll back its previous log record in turn until every operation in the outstanding transaction has been undone.

5. Conclusion
When SQL Server performs instance recovery using log files, it performs a redo operation starting from MinLSN and then performs a undo operation. Throughout the process, the key to locating the log records is its LSN.

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.