```
 LSN   Transaction ID    Type     Page ID
----- ---------------- -------- ---------
   1   T1                UPDATE   P3
   2   T2                UPDATE   P2
   3   T1                COMMIT
   4   CHECKPOINT
   5   T3                UPDATE   P1
   6   T2                UPDATE   P3
   7   T2                COMMIT
```

## ARIES

ARIES starts by finding log records for operations that were not written to disk on crash and then replays all of them. This even includes transactions that need to rolled back. It brings the database to the same state as it was before crash. This process is called "repeating history". Note that this is same mechanism that is used for database replication in normal course of action. Once database is brought up to date, all the transactions that need to be rolled back are undone.

ARIES is a three phase algorithm:

1. Analysis phase: This phase reads the last checkpoint record in the log to figure out active transactions and dirty pages at point of crash/restart. A page is considered dirty if it was modified in memory but was not written to disk. This information is used by next two phases.

2. REDO phase: In this phase operations in log are reapplied to bring the state of the database to current.

3. UNDO phase: This phase proceeds from the end of log reverting back operations for uncommitted transactions. This has impact of rolling them back. Note that this is same procedure that database performs when rolling back a transaction in normal mode of operation for STEAL policy.

ARIES maintains two data structures and adds one more field to log record:

1. Transaction table: It contains all the transactions that are active at any point of time (i.e. are started but not committed/aborted). The table also stores the LSN of last log record written by the transaction in "lastLSN" field.

2. Dirty page table: Contains an entry for each page that has been modified but not written to disk. The table also stores the LSN of the first log record that made the

associated page dirty in a field called “recoveryLSN” (also called “firstLSN”). This is the log record from which REDO need to restart for this page.

3. In addition log records are also updated to contain a field called “prevLSN” which points to previous log record for the same transaction. This creates a linked list of all log records for a transaction. When a new log record is created, “lastLSN” from transaction table is filled into its “prevLSN” field. And the LSN of current log record becomes the “lastLSN” in transaction table. Here is updated log record table with prevLSN filled in:

| LSN | Prev LSN | Transaction ID | Type | Page ID |
|---|---|---|---|---|
| 1 | NIL | T1 | UPDATE | P3 |
| 2 | NIL | T2 | UPDATE | P2 |
| 3 | 1 | T1 | COMMIT | |
| 4 | CHECKPOINT | | | |
| 5 | NIL | T3 | UPDATE | P1 |
| 6 | 2 | T2 | UPDATE | P3 |
| 7 | 6 | T2 | COMMIT | |

During checkpointing, a checkpoint log record is created. This log record contains the content of both “Transaction table” and “Dirty page table”. “Analysis” phase starts by reading last checkpoint log record to get the information about active transactions and dirty pages. Here is content of “Transaction table” and “Dirty page table” at the checkpoint stage in above table at LSN 4:

**Transaction Table**

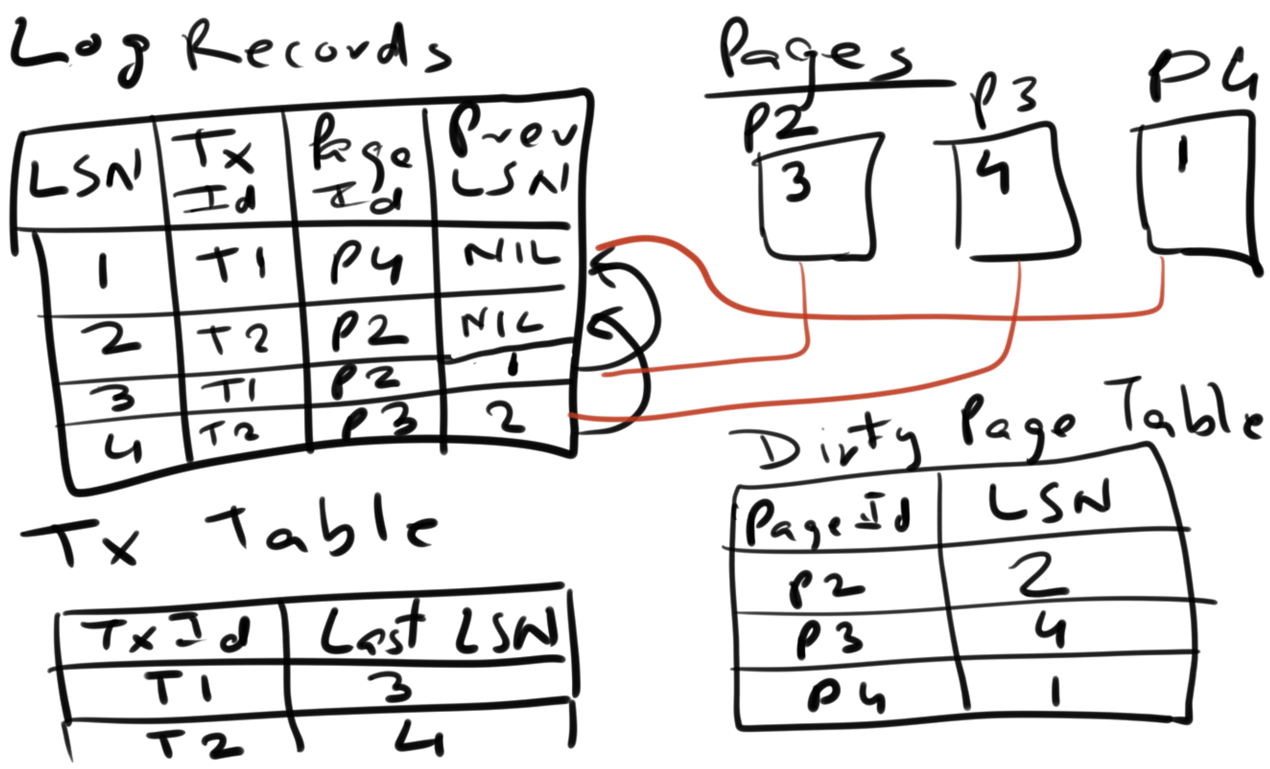
| Transaction ID | Last LSN | Status |
|---|---|---|
| T1 | 3 | Commit |
| T2 | 2 | In Progress |

**Dirty Page Table**

| Page ID | Recovery LSN |
|---|---|
| P3 | 1 |
| P2 | 2 |

This whole setup can be visualized in following picture, pay attention to LSN for P2 in the “pages” list and in dirty table (dirty page table has the first LSN, whereas the

P2 page has the last LSN):



ARIES Data Structures

## Analysis Phase

Analysis phase reads the "checkpoint log record" for the latest checkpoint and then scans the log forward to create list of all active transactions and dirty pages. When scanning log forward:

1. If a new transaction is started, it is added to the transaction table.
2. If a commit/abort is seen for a transaction, its entry is removed from transaction table.
3. If a page is updated, a new entry is added to dirty page table and "recoveryLSN" is set to the LSN of the log record.

Hers is how the Transaction table and Dirty page table will look like after analysis is done:

```
**Transaction Table (Rows with [x] are deleted, but are left here to
show full process)**

  Transaction ID   Last LSN     Status
 ---------------- ---------- -------------
*[x]   T1                3         Commit*
```

```
[x]  T2               7         Commit
     T3               5         In Progress
```

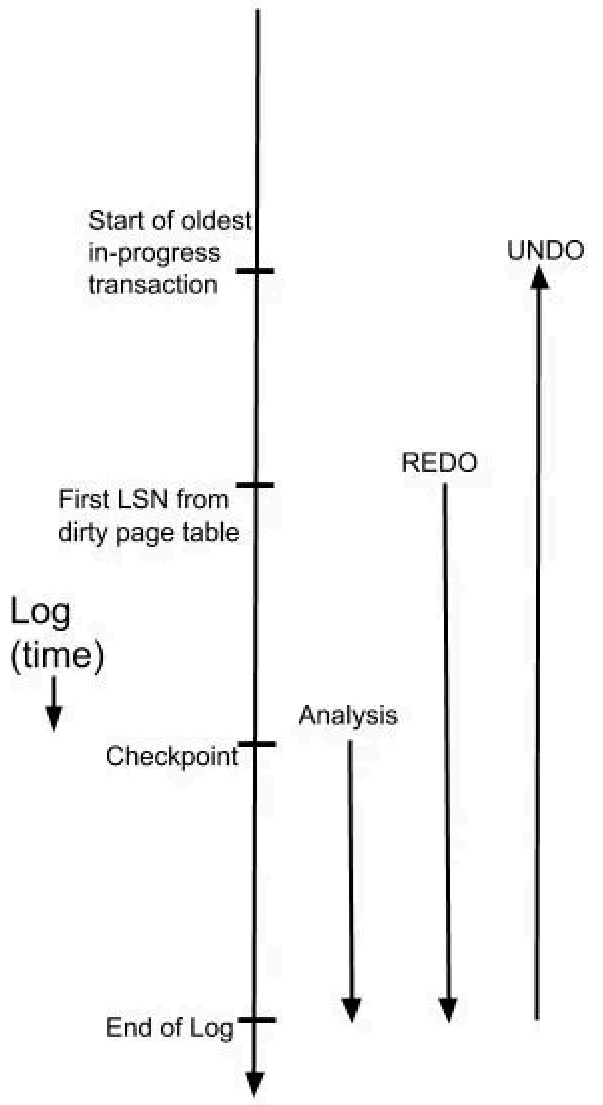
```
Dirty Page Table

   Page ID   Recovery LSN
  --------- --------------
   P1          5
   P2          2
   P3          6
```

## REDO phase

At the end of analysis phase, the first LSN that caused a page to become dirty is known (its the lowest LSN value in dirty page table). REDO phase starts at this point in the log and processes all log entries from that point:

1. If the log record point to a page that is not dirty, ignore that log record.
2. If the log record point to a dirty page, but its "recoveryLSN" is more than current record's LSN, it means that the page was committed/aborted and a later transaction then updated the page again. So ignore this log record.
3. Otherwise check the LSN on the page (as discussed before, each page contains LSN that last updated the page). If the LSN of page is greater than this logs record's LSN, again ignore the record as this situation indicates that page made to disk successfully for a later transaction.
4. Otherwise, redo the action specified in the log record.

At the end of this phase database is in same state as it was before crash. Here is a timeline showing how three phases make a pass through the log.


Start of oldest in-progress transaction
UNDO
REDO
First LSN from dirty page table
Log (time)
Analysis
Checkpoint
End of Log

**UNDO phase**

This phase starts from end of the log and goes backward. This phase undoes all transactions that were active at time of restart/crash. Information about these transactions are available in the transaction table. The lastLSN field in transaction table also tells the first record that need to be undone. And as we saw before, “prevLSN” field in log record creates a linked list of operations in the transaction. So all log records in this linked list need to be undone to rollback the transaction. Another good property of UNDO is that unlike REDO it doesn’t need to check any state, it need to unconditionally undo log records of all active transactions.

To avoid repeating undo during repeated restart, each time a log record is undone, a “Compensation Log Record” is added to the log. In addition to the information about the undo, the record also contains the next log record for the transaction that need to be undone. And the next log record that needs to be undone is stored in the "prevLSN" of the current log being undone.

After that UNDO becomes a simple operation of going in reverse through the linked list of log records for an active transaction and undoing it. It need to do this for all active transactions in the transaction table.

## Conclusion

Database systems have evolved and there are many new ones that have become popular. Key Value stores, Search indexing, new query execution engines like Impala, Tez, Spark, Flink etc are used to process data. However using logs for recovery and replication is pretty common. Systems like Kafka are developed using similar principle of maintaining partitioned log. Even though these mechanisms are developed few decades back, they have turned out to be pretty durable and still form underpinning of new storage and query systems.

"Concurrency Control and Recovery" paper by Michael Franklin explains this in greater detail and is easy to understand, so if you want to dig more, head there.

Database

V

Follow

Written by Vikas Singh

23 Followers

Interested in sports, finance and technology.