# Crash Recovery:ARIES Data Structures

## The Log

In order to insure recoverability of operations, DBMS maintains **the log**. It contains records necessary for successful recovery.

**Log Records:**

1. Update
2. Undo (Compensation Log Record)
3. Commit
4. Abort
5. End
6. begin_checkpoint
7. end_checkpoint

**Structure of Log Records**

**commit, abort, end:** (LSN, prevLSN, TransID, type)

**update:** (LSN, prevLSN,TransID, type, pageID, length, offset, before, after)

**undo:** ((LSN, prevLSN,TransID, type, pageID, length, offset, restored, nextUndoLSN) (CLR - Compensation Log Record)

**begin_checkpoint, end_checkpoint:** See below

**Log Attributes**

**LSN:** Log Sequence Number, unique id for each log record.

**TransID:** Id of the transaction which caused the record to be added to the log.

**prevLSN:** LSN of the previous log record for the same transaction

**type:** Type of the record. Possible values are update, undo, commit, abort, end, begin_checkpoint, end_checkpoint.

**PageID:** Pointer to the page affected by the update (or undo).

**length:** Number of bytes to be written to the page by an update/undo operation.

**offset:** Position on the page where the value needs to be written to.

**before:** Value prior to the update.

**after:** Value after the update.

**restored:** Value, restored by the undo operation.

**nextUndoLSN:** In a CLR, the pointer to the next record for current transaction which is to be undone.

### More Data Structures

**Transaction Table (T-Tab):** Contains the list of all currently active transactions. Contains many attributes. For recovery, the following attributes for each transaction record are interesting:

**TransID:** Transaction ID

**lastLSN:** Pointer to the last record in the Log for the transaction.

**Dirty Page Table (DPT):** Contains the list of *dirty* buffer pages. Attributes:

**PageID:** ID of the buffer page

**recLSN** LSN of the **first** record in the log that made the page *dirty*.

**Note:** A buffer page is called ***dirty*** if it contains updates that had not yet been flushed to stable storage.

**LOG**

| LSN | transID | prevLSN | type | pageID | length | offset | before | after/restored | undoNextLSN |
|---|---|---|---|---|---|---|---|---|---|
| 10 | T1 | **NULL** | update | P1 | 1 | 500 | 200 | 100 | |
| 20 | T2 | **NULL** | update | P2 | 2 | 134 | 3100 | 4000 | |
| 30 | T1 | 10 | update | P1 | 1 | 501 | 100 | 200 | |
| 40 | T3 | **NULL** | update | P3 | 3 | 101 | abc | dog | |
| 50 | T2 | 20 | commit | | | | | | |
| 60 | T1 | 30 | update | P3 | 1 | 201 | a | z | |
| 70 | T3 | 40 | update | P3 | 3 | 121 | pqr | red | |
| 80 | T3 | 70 | abort | | | | | | |
| 90 | T2 | 50 | end | | | | | | |
| 100 | T3 | 80 | undo | P3 | 3 | 121 | | pqr | 40 |

**Transaction Table**

| transID | status | lastLSN |
|---|---|---|
| T1 | active | 60 |
| T3 | aborted | 100 |

**Dirty Page Table**

| pageID | recLSN |
|---|---|
| P1 | 10 |
| P2 | 20 |
| P3 | 40 |

# ARIES Features

### Commit force-writes the log

- A commit record causes the current *tail* of the log to be **forced** to stable storage. This *immediately* creates a permanent record about any committed transaction.

  Flushing dirty pages written by the committed transaction occurs as a background process afterwards and is **not** reflected in the log.

### Write-Ahead Logging (WAL)

A more general rule (that causes commit records to force-write the log) is represented by the principle of **write-ahead logging**.

**Write Ahead Logging** Before any *database page* is flushed to disk (stable storage), all log entries documenting changes on this page **must** be flushed (forced) to the stable storage.

**Write Ahead Logging** insures that the log always contains accurate and up-to-date record of changes in the database.

### Checkpointing

At certain moments of time, DBMS performs a checkpointing operation. Two records are inserted into the log:

**begin_checkpoint:** a simple record is inserted to indicate the moment for which checkpoint information is collected.

**end_checkpoint:** a record containing the contents of Transaction Table and Dirty Page Table at the begin_checkpoint moment is inserted into the log.

### Fuzzy Checkpoints

end_checkpoint record can be large and creating it may take some time. During this time DBMS can stop accepting any actions ("rigid" checkpoints) or it can keep accepting action requests from transactions and log their actions (*fuzzy* checkpoints).

In the latter case, the end_checkpoint record contains the correct information *as of the time when begin_checkpoint record had been added to the log.*

# ARIES in a nutshell

ARIES consists of three phases:

**Analysis:** the log is retrieved from stable storage and analyzed to find the starting point for the next phase and the state of the data structures at the moment of the crash.

**Redo:** the sequence of actions prior to the crash is redone, producing (alomst) the state of the database prior to the crash.

**Undo:** all transactions, active at the time of the crash are aborted, their actions undone and the undos logged. When this phase ends, the database recovers to a consistent state induced by the results of all transactions committed prior to the crash.

## Analysis

- Starts by recovering T-Table and DPT from the last end_checkpoint record.
- Proceeds analyzing all log entries after the last begin_checkpoint records and updating T-Table and DPT accordingly.
- After the last log entry is analyzed, T-Table and DPT are restored to their *pre-crash* state.

From this information, **analysis** phase determines:

- The starting point for the **Redo** phase. This will be the log record corresponding to the **smallest LSN** recorded in the DPT. This corresponds to the first buffer page update *not guaranteed to be written to stable storage.*
- A (superset) of dirty pages in the buffer. This is determined by the contents of DPT.
- A set of *active* transactions. Determined by the contents of T-Table.

## Redo

**Redo** stage

- starts at the point determined by **Analysis** stage. (minimum of recLSN records from the final DPT).
- Reapplies all the updates for both committed and uncommited transactions to the database.
- For transactions aborted prior to the crash, thier undo actions are also reapplied.

Each update or undo operation must be reapplied **unless**:

- The affected page is **not** is DPT
- The affected page is in DPT but its recLSN is greater than the LSN of current action.
- The pageLSN (stored on the page) is *greater than or equal to* the LSN of the current log record.

When action is redone, pageLSN record is changed to the actions LSN.

## Undo

**Undo** stage

- Performs actions necessary to abort **all** active and aborted transactions.

- T-Table computed during **Analysis** contains the list of transactions to be aborted (**loser** transactions).

**Undo** Algorithm:

- Create (and maintain) ToUndo data structure. Undo should contain lastLSNs from all **loser** transactions.
- On each step, Log record with largest LSN is chosen (and removed) from ToUndo. Actions depend on its type:
  - Update: Write a CLR and undo corresponding action. Insert prevLSN into ToUndo.
  - Undo: If UndoNextLSN is not NULL, add UndoNextLSN to ToUndo. If UndoNextLSN is NULL, add end record to log and delete current transaction for T-Table.
  - Other: Insert prevLSN into ToUndo.