ARIES recovery

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | * 1. Log combined REDO/UNDO record before each update to database   2. Log COMMIT record when a transaction finishes execution   3. Modified data pages can be written to disk any time after the corresponding REDO/UNDO record (the most flexible)   4. **Recovery**: UNDO uncommitted transactions (flashed before commit) and REDO committed (but not written o disk) transactions | | | | |
| **No Steal**: Dirty pages **stay** in DRAM until the transaction commits  **Steal**: Dirty pages can be flushed to disk before transaction commits | | | | **No Force**: more flexible. Dirty pages **may stay** in memory after transaction commits  **Force**: All dirty pages must be flushed (random) before the transaction commits | | | | |
| No Force/Steal may stay in memory after commit (REDO)/can be flash before commit (UNDO) | | | | Modified data pages can be written to disk any time after the corresponding REDO/UNDO record | | | | |
| **Write**: Flush REDO/UNDO to log; update the page  **Commit**: Write COMMIT to log  **Recovery: Forward** scan of entire log: **redo** all records   * 1. **Backward** scan of entire log: **undo** only **uncommitted** transactions   How to identify transactions that did not commit before crash?   * 1. Those that do not have a commit record   2. **Naïve** not optimized **approach we build log data structures**   -If see transaction T’s log record, add T to **Transaction Table;**  -if see T’s commit record (abort), remove T from **Transaction Table**   * 1. In the backward scan, undo only transactions in Transaction Table | | | | LSN seq. # position in log file | | | | Data page exist in memory and disk  Transactions that do not commit before crash do not have commit log  ??  REDO tie sto **imaat** comit ? **not on disk**  UNDO tie sto nemaat commit (skip tie so commit, poednostavno izbrisi ih od transaction table) |
| **Limitations: Inefficiency in the REDO process**   * 1. -Unnecessary to redo all records (start from beginning).   2. -Redo only records that are **not reflected in data pages (not on disk)**   **Inefficiency in UNDO process**.   * Unnecessary to scan the entire log. * Can skip records that **do not belong** to **uncommitted** transactions.   **Lack of checkpointing**.  -Unnecessary to start from the beginning of log   * 1. -Start with the first log record that is not reflected in data pages | | | | (UNDO skip committed records without reading them)) | | | | **Optimize REDO** Process,  Inefficiency in REDO  -Unnecessary to redo all records   * 1. -**Need to redo** only records that are **not reflected in the data page.** |
| **Optimize REDO**  **Solution**: add a version number to each page   * 1. **- pageLSN**: LSN of the log record that describes latest update to page.   2. **- REDO scan**: Apply REDO only if **record.LSN > page.pageLSN**   3. **-Write**: update pageLSN for the page in buffer pool for each write   4. (if transaction modify more pages, each will have log) | | | |  | | | | samo record belong to uncomited  each write in transaction has own log record |
| **Optimise UNDO Process**  Inefficiency in UNDO process.   * Unnecessary to scan the entire log. * Can skip records that **do not belong** to **uncommitted** transactions   **Solution**: link records from the same transaction   * 1. **prevLSN**: preceding log record written by the same transaction   2. **lastLSN**: LSN of the last log record written by the transaction   3. **UNDO scan**: Follow lastLSN and prevLSN to undo records   4. **REDO scan**: update lastLSN in Transaction Table based on the last update of the transaction | | | |  | | | Recognize uncommited transactions    Have to UNDO in order, not bloo ones, red ones  nejasno | |
| **Lack of checkpointing**   * 1. -Unnecessary to start from the beginning of log   2. -Start with the first log record that is not reflected in data pages   3. (all that is on disk and committed do not need redo/undo??)   **Solution**: Maintain a dirty page table  (DP in memory should be rid during recovery)   * 1. **pageID**: ID of the dirty page   2. **recLSN**: LSN of the first log record since when the page is dirty   3. **Fuzzy Checkpoint:** log DPT and TT asynchronously   4. **REDO scan**: start from the smallest LSN in DP | | | |  | | | Compensation Log Record (CLR)  Dirty page should be read during recovery  Action only to dirty page with smallest LST  If it is not in DP it was reflected to disk | |
|  | | | | Compensation Log Record (CLR)  The action of applying UNDO leads to a **CLR**   * 1. - In undo scan, do not **reapply UNDO** if **CLR exists**   2. **- UndoNxtLSN**: LSN of the next record to be processed during undo scan   3 and 3’….. undo the same tr. Do not reapply if CLR exist. So do not undo 3’ but undo 3  CLR only can have redo not undo  CLR are log record that show changes during undo process  **Redo committed**, **Undo uncommited** | | | | |
| * 1. **ARIES BIG Picture** | | | |  | | | | |
|  | | **Goal**: Bring the database to the state before the crash (REDO phase) and **rollback uncommitted transactions** (UNDO phase)  Start from the last complete checkpoint   * 1. **Analysis phase**: rebuild transaction table (**for undo phase**)   2. and **dirty page table (for redo phase**)   3. **REDO phase**: redo transactions whose effects may   4. not be persistent before the crash   5. **UNDO phase**: undo transactions that did not   6. commit before the crash | | | | | Crash Recovery – **Analysis Phase**  Goal: Rebuild transaction table (for undo phase) and dirty page table (for redo phase) based on the ones in the last checkpoint  (**update transaction table)** For each log record:   * 1. - If ‘update’ or ‘CLR’: insert to transaction table if not exists   2. - If ‘end’: delete from transaction table   **(update dirty page table)** For each log record:   * 1. -If ‘update’ or ‘CLR’: insert to dirty page table if not exists (PageID, RecLSN) | |
| **Analysis phase**    Because not committed write in Transaction table | | | |  | | CLR has Lst 40 but for P5 we keep 10 | | |
|  | We put T2 50 in Transaction page,  but keep the oldest LSN for P5  🡨 redo\_recordLSN< Data pages\_LSN ??  Nejasno, dali se odnesuba na records in log < of DPT.LSN | | | | | In analyze phase we build these 2 tables,  Dirty page table is now used for redo process  And Transaction will use in undo proces | | |
| **Crash Recovery – REDO Phase**  Repeat history to reconstruct state at crash   * 1. Reapply **all updates** (even of aborted transactions), redo CLRs   **Where to start?**   * 1. - From log record containing **smallest RecLSN** in the **dirty page table (10)**   2. - Before this LSN, all redo records have been reflected in data pages on disk   Observation: can **skip a redo record** for the following 3 cases where the corresponding page has already been flushed before the crash   * 1. -The page is not in dirty page table (DPT)   2. -The page is in DPT but redo\_record.LSN < DPT[page].recLSN   3. -After fetching the data page, redo\_record.LSN ≤ page.page\_LSN | | | | | | LSN on disk is 40 for P5 so this incorporate newer change | | |
|  | | | **<-is in memory**  **Update P3 in buff pool, not need to flush to disk P3** | | | Read from disk, not need to update | | |
| Update P1 in buffer pool | | | | | |  | | |
| **Crash Recovery – UNDO Phase**  **Rollback uncommitted transactions**  **Repeat until transaction table is empty:** (actually undo all from Transaction table)   * 1. - Choose **largest LastLSN** among transactions in the transaction table   2. - If the log record is an ‘update’: Undo the update, write a CLR, add record.prevLSN to transaction table   3. - If the log record is an ‘CLR’: add CLR.UndoNxtLSN to transaction table   4. - If prevLSN and UpdoNxtLSN are NULL, remove the transaction from transaction table | | | | | |  | | |
| righy to left next is 50 for T3 there is no previous so remove it | | | | | | | | |
| There is no previous so remove T3 | | | | | | | | |
|  | | | | |  | | | |

|  |  |
| --- | --- |
|  | 1. <https://www.youtube.com/watch?v=-yIV_vSRE3I> |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| A database contains two pages P1 and P2. P1 contains two elements A and B. P2 contains two elements C and D. | * 1. P1 (A,B)   2. P2 (C,D) |
| • Transaction T1 writes A.  • Transaction T2 writes B.  • Transaction T2 writes C.  • The system flushes the log to disk and also flushes page P2 to disk. • Transaction T1  • Transaction T1 commits. The system writes a commit log record and f lushes the tail of the log to disk.  • Transaction T2 writes B.  • The system writes an END log record for T1.  • The system crashes. |  |
|  |  |