CSE 5331 – Database Systems II

Project 1 Documentation (updated)

Team Members:

Agrawal, Saurabh V (UTA ID: 1000954351)
Parekh, Shivang (UTA ID: 1000990285)

Project Definition: Implementing a program that simulates the behavior of the UNDO/REDO protocol for recovery from failure.

Programming Language: Java v. 7

Program Input & Output: Input to the program will be a text file which will be read by the program, line by line, and will produce an output text file containing the description of the operations done.

How to run the program:

1. Only one class to run named ‘MainClass.java’ in ‘Recovery’ project.
2. Preferably use Eclipse IDE.
3. Specify the **path** to input & output files on **lines 72 and 73** in the code.
4. Run the program as a Java Application.
5. The output file will be created in the path specified above.

Assumption: The program is written with the assumption, that whenever a failure occurs, all the tables in the memory (log buffer, transaction table & cache table) are lost. Thus, the UNDO/REDO recovery process is done only by examining and scanning the log file on disk.

System Tables: To keep track of the data/transactions/log, we will be making several system tables –

Table Name	Location	Purpose	Columns	Data Structure
Cache Table	memory	This table keeps track of data items present in main memory buffers, and other relevant information	• Buffer Number (Range from 0 … 9) • Data Item (Single letters from A … Z) • Dirty Bit (Either 0 (= unmodified) or 1 (= modified)) • Value (Latest value of that data item)	HashMap <String, List <String>>
Transaction Table	memory	This table keeps track of the transactions currently under progress and other relevant information.	• Sr. No. (starting from 1, to keep track of records) • Transaction ID (will hold the ids of the transactions running/committed/aborted) • Last LSN (it will hold the last log sequence number of the operation by a particular transaction) • Status (can be in progress/ committed or aborted)	HashMap <Integer, List <String>>
Log Buffer	memory	This table will contain log records for each operation that is performed on input file. It has a capacity of holding 4 records.	• Log Sequence Number (Unique identifier for each log record) • Transaction ID (ID of the transaction) • Last LSN (Last LSN of the transaction) • Operation Type (Type of operation that the transaction is executing. Operation types can be: b = begin, r = read item, w = write item, e = end transaction, c = commit transaction, a = abort transaction) • Data Item (Data item on which the operation is performed) • BFIM (Before Image of the data item) • AFIM (After Image of the data Item)	HashMap <Integer, List <String>>
Log File	Disk	This table will hold log records, but on the disk. It will have 20 blocks, each holding 4 records, thus can hold total of 80 records.	• It will have the same columns as the log buffer in main memory.	HashMap <Integer, List <String>>

Data Table	Disk	The data block on disk will hold 26 records (A, B … Z) and their corresponding values.	• Sr. No. (Serial numbers to keep track) • Data Item (A through Z) • Value (value of each data item. Zero/null in the beginning)	HashMap <String, Integer>

High Level Pseudo-code: A high level pseudo code of the program is provided below –

- During execution of the program, the input file will be read line by line and output file will be written accordingly.
- The input file will contain one operation on each line. The operations will be terminated by semicolons, which will be used as a delimiter when reading.
- The program will read the first letter and recognize it. Based on the sample input file given in definition, it will do the following in each operation –
    - **Read 'b1'** –
        - 'b' means **begin** of a transaction
        - It will get the transaction number
        - Update log buffer in memory
        - Update transaction table
    - **Read 'r1(Y)'** –
        - 'r' means **read** of a data item
        - It will get the transaction number & data item
        - Update the log buffer in memory
        - Update cache table. If data item is not found in cache, fetch it from data table on disk
        - Update transaction table
    - **Read 'w1(Y,2,0)'** –
        - 'w' means **write** of a data item
        - It will get the transaction number & data item, along with its old & new value
        - Update the log buffer in memory
        - Update cache table. If data item is not found in cache, fetch it from data table on disk
        - Update transaction table

- **<u>Read ‘c1’</u>** –
  - ‘c’ means **commit** of a transaction
  - It will get the transaction number after that
  - Update the log buffer in memory
  - Update transaction table

- **<u>Read ‘C’</u>** –
  - ‘C’ means **Checkpoint**
  - Update the log buffer in memory with a ‘begin checkpoint’ record *(following the Write Ahead Logging (WAL) protocol)*
  - Update the cache table
  - Update the data table on disk
  - Update the log buffer in memory with a ‘end checkpoint’ record

- **<u>Read ‘F’</u>** –
  - ‘F’ means **failure**
  - There can be no further operations after failure
  - The system (program) checks the log file on disk
  - If a checkpoint record is found, it looks for all the transactions that committed after the checkpoint. It applies REDO on those transactions & UNDO on uncommitted ones.
  - If a checkpoint record is not found, it looks in the entire log to determine committed & uncommitted transactions, & applies REDO & UNDO operations accordingly.
  - Update cache table
  - Update data table on disk

- **<u>Read ‘E1’</u>** –
  - ‘E’ means **end** of a transaction, similar to commit.
  - It will get the transaction number after that
  - Update the log buffer in memory
  - Update transaction table

- **<u>Read ‘A1’</u>** –
  - ‘A’ means **abort** of a transaction
  - It will get the transaction number after that
  - Will undo all the writes performed by the aborted transaction
  - Update log file

    - Update transaction table

- **<u>Read 'L'</u>** –
    - 'L' means **Force Write**
    - This system call will force write the log buffer to the log file on disk, even if it is not full.
    - The buffer is written again to the same disk location when it becomes full.

<u>Conclusion:</u> By following pseudo code we have implemented the simulation of UNDO/REDO protocol for recovery from failure (allowing only strict schedules).