



Introduction to Project 4



Logging and Recovery

- Implementing a recovery manager that uses
 - Write-ahead logging
 - ARIES as Recovery algorithm



Project 4 Overview

- Textbook sections
- Setup
- Submissions
- Grading



Read

- Sections 16.7 through 16.7.2
- Sections 18.1 through 18.6

This is very important!!

"The recovery manager is one of the hardest components of a DBMS to design and implement." (Chapter 18)



Setup

- Download the recovery simulator from Ctools and unzip it
- Directory structure: Demo!
- Create LogMgr.cpp in the StudentComponent folder
- Implement ARIES in LogMgr.cpp
- Disk is represented by a files. It has .db extension
- Log written to disk is written in .log file



Few comments...

- Can use C+11 STLs
- Do not add any header files
- Create helper functions in LogMgr.cpp
- Do not use anything that reads from or write to disk
- Do not add any static variables/functions to LogMgr.cpp



Grading

- 10 autograder test cases – 100 points
 - Your log and db files will be compared against ours
 - 10 points per test case
 - 5 points for correct log file
 - 5 points for correct db file
 - You have the output from PUBLIC testcases for debugging purposes.
 - Passing the public test cases doesn't guarantee full marks on the autograder



Submission

- Submit only one file: LogMgr.cpp
- Turn in your code via Autograder
- Autograder will be out by Saturday
- Each username gets 3 submissions per day
- Last submission will be graded
- Submit partner note through Ctools



Introduction to Logging

- Log is a history of actions executed by the DBMS
- Kept as a list of log records
- Most recent part, called log tail, kept in memory
- Certain events require appending log tail to log file on disk



Log Records

- Data structure for storing information about history
- Each record gets a log sequence number (LSN)
- Basic information (stored for all types of log records):

LSN	previous LSN	transaction ID	type
-----	-----------------	-------------------	------



Types of Log Records

- Commit, abort, end: use just the basic fields
- Update: information about writing to a page
- Compensation Log Record (CLR): information about undoing an update
- Checkpoints: Snapshot of what transactions are active and what pages have been written

Example: Update Record

Update sname for sid 17 to
"Captain Hook"

Page #342

14 Rusty
15 Dusty
16 Lubber
17 Ahab
18 Starbuck

DBMS

LSN	Previous LSN	TX #	Type	Page	Offset	Before Image	After Image
974	965	15	Update	342	32	"Ahab"	"Captain Hook"



Example: Update Record

LSN	Previous LSN	TX #	Type	Page	Offset	Before Image	After Image
974	965	15	Update	342	32	"Ahab"	"Captain Hook"

```
logtail.push_back(new UpdateLogRecord  
    (974, 965, 15, 342, 32, "Ahab",  
    "Captain Hook"));
```

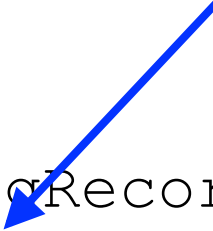
Your Turn: Commit Record

LSN	Previous LSN	TX #	Type
975	974	15	Commit

Look at LogRecord.h, and figure out how to add a record of a commit to the log tail.

enum TxType

```
logtail.push_back(new LogRecord  
    (975, 974, 15, COMMIT));
```





Log Records, Strings, and Files

- Do NOT use file IO to write your log to disk.
- Instead:

```
string my_str =  
    my_rec->toString();  
my_str.append(rec2->toString());  
se->updateLog(my_str);
```



Log Records, Strings, and Files

- Do NOT use file IO to read your log from disk.
- Instead:

```
string logstring = se->getLog();  
vector<LogRecord*> log =  
    stringToLRVector(logstring);
```




Log Records, Strings, and Files

- In your `stringToLRVector` function:

```
vector<LogRecord*> result;  
  
istringstream stream(logstring);  
string line;  
while (getline(stream, line)) {  
    //WHAT GOES HERE?  
    result.push_back(lr);  
}  
return result;
```



Log Records, Strings, and Files

- In your `stringToLRVector` function:

```
vector<LogRecord*> result;

istringstream stream(logstring);
string line;
while (getline(stream, line)) {
    LogRecord* lr =
    LogRecord::stringToRecordPtr(line);
    result.push_back(lr);
}
return result;
```



Tips for Success

- Read the textbook!
- Start with `write()`, `flushLogTail()`, and `commit()`
- The recovery functions -- `analyze()`, `redo()`, and `undo()` -- and `abort()` will be the hardest part. Next week, we will help you map out how they work. Try to get your other functions working before then