COSC6340: Database Systems
Mini-DBMS Project

Instructor: Carlos Ordonez

1 Introduction

In this project you will develop a small DBMS in C++ supporting queries and transaction processing. Your program will be focused on evaluating SPJA queries on large amounts of data. As a result, your small DBMS will require you to use binary files. You will have to submit this program in two phases. The 1st phase is focused on storage and querying data (SELECT statement). The 2nd phase is performing on-line transaction processing (concurrent access, locking).

2 Program requirements

Your program should be coded in C++ using the Borland C++ v. 5.5 Compiler. The program needs to efficiently manage large amounts of data. Therefore, your application cannot use regular text files to store the data. Instead, you are required to use binary files (with efficient random access). Here is an overview:

- Phase 1: DDL, INSERT, $\pi, \sigma, \bowtie$ as supported in SQL without any index support.
- Phase 2: UPDATE, INSERT, COMMIT/ROLLBACK. 2 PL. Index support for primary key or external sort on primary key.

1st phase of the project requires you to manage two modes of entering SQL queries: via an SQL console and using a batch file. Your SQL console should start if the user decides to give as, an argument, an SQL query, and once it has finished executing, allow the user to input more SQL queries by displaying a prompt ("$SQL >$"). The system will finalize when the user types the "quit;" command. Additionally, the 1st phase of the project requires a catalog table that contains all the schema information for all the tables. This catalog table is going to be maintained and managed in main memory, and updated when the application closes (in a lazy update). The schema information of a table includes the table names, the column names (and their types), primary keys, size of the record, table and number of records. All temporary tables should be included in the memory catalog table, but not in the text catalog.

Some other SQL commands that are required for the 1st phase are the following. YOu can assume all statement are terminated by ";" in a script. "SHOW TABLE", which returns the DDL information of the specified table. CREATE TABLE, which creates a permanent table in the DBMS with simple primary keys (only one column). INSERT INTO should insert one record to the target table. The SELECT/FROM statement should return the specified columns and result rows without duplicates (i.e. it should work like "SELECT DISTINCT .."). The SELECT/FROM/WHERE query will be similar to the SELECT/FROM operation but also considers a comparison given in the WHERE clause ($<, >, =$) with AND/OR (two comparisons). For the JOIN operation indicated with the JOIN/ON clause, you are only required to join two tables with a single column as the join condition. Do not specify the join operation in the WHERE clause.

INSERT INTO would insert all the data of a SELECT operation in the specified table (use SQL server syntax). If you are trying to insert data into a table that does not exist you should return an error.

Your program will be tested with scripts where CREATE TABLE & SELECT are chained. That is, the output of a query is stored in a table than can later be used as input in another table.

2.1 Program Specification

2.1.1 Phase 1

In the initial phase, you are required to call your main program 'dbms.exe", from which you can submit an SQL query or a batch file containing a set of SQL queries. Your application should be able to store and select data in an efficient manner. Therefore, you are required to store each table in a separate binary file. Each binary file will have the name of the table and a ".tbl" extension. The data types that you should manage include: integers (4 bytes, 32 bit) and strings (1 byte per char). Every time you complete an operation, you are expected to return the result/data of such operation (e.g. Created TABLE T Successfully; SELECT successful).

In order to efficiently manage all your database collection and data, you are required to have a file containing all the information of your database schema (table metadata). This catalog should be maintained (updated) with a lazy policy. As a result, your text file will only be updated with the required information when the program finishes executing. Hence, the metadata of each table should be included in text file (catalog.txt) with a table descriptor per line. The table metadata includes: tablename, columns, primary key, record size in bytes, total number of bytes of the table, and the total number of records in the table. The columns should be divided by "," and the type of the column should be divided by a ":". At the beginning of your program, you can load the whole catalog in memory, and only modify this memory loaded catalog. Once your program finishes executing, you can then dump the new catalog information (discarding the information regarding temporary tables). Due to the fact that you are expected to keep your temporary tables during the execution of the program in the catalog, you also have to keep the related binary files and allow the user the capability of also querying (SELECT) this temporary tables.

For example, consider the following catalog file:

```
File: catalog.txt
-----------------------------------
tablename=T
columns=C1:INT,C2:CHAR(5),C3:INT
primary key=C1
recordsize=13
totalsize=65
records=5
tablename=T1
columns=B1:INT,B2:CHAR(255)
primary key=B1
recordsize=259
totalsize=2590
records=10
```

2.1.2 Phase 2

You will develop transaction processing with 2 phase locking (2PL). You need to support serializable isolation level in SQL. You have to implement COMMIT and optionally ROLLBACK to break deadlocks and handle unexpected errors. COMMIT is required and ROLLABACK is optional for extra credit. Each transaction must end with COMMIT, and optionally must have logic to end with ROLLBACK.

Your program must work with multiple threads (5 threads minimum, defaul=10 threads). Create an internal constant in the C++ code to specify the number of threads with #define (call it NUMBER_OF_THREADS). Your program must support 2PL. Locks should be obtained and released one at a time as this results in mreo interleaving in concurrent processing. Your program must issue a lock immediately before updating an item. Locks should be released together at the end of the transaction or after the updated item is no longer needed. Your program must support concurrent updates on up to 3 tables with 3 columns each (one of them is the primary key, the rest are non-key columns). Your program must support one transaction; multiple transactions are optional. The table should be updated on disk only when the COMMIT instruction is found. In other words, locks and variable updates will happen in RAM.

You can assume UPDATE will have a WHERE clause on the primary key (only one record updated). You have two alternatives to efficiently search for the primary key: (1) add an index (B-tree, hash, or multilevel; HARDER, extra credit) or (2) you can sort the input table with a merge sort and then use a binary search at transaction processing time (EASIER).

The multithreaded program should work as follows. The first time the DBMS starts it create the threads and reserves RAM for shared access. All variables should be maintained in such RAM space. Subsequent calls can be assumed to process one transaction per DBMS call. You can know if the DBMS is already started by updating a control file. There should be a program that is constantly checking for input transactions. When one transaction comes it routes to the appropriate thread. You should program a queue to handle waiting transactions. Locks should be managed with a hash table for O(1) access, but AVL trees are acceptable.

Input SQL program: Your program will identify tranasction processing when it encounters ”BEGIN TRANSACTION”. Your program must handle temporary variables to store values retrieved by a SELECT. Then there will be a sequence of ”UPDATE” statements, with some optional ”SELECT” statements in between. At a minimum there will be one UPDATE. The transaction end will be marked with COMMIT or ROLLBACK. You can optionally handle multiple transactions in the same script in which case a transaction name must be specified. Processing of tables will happen with T-SQL syntax using the UPDATE statment. You will search for a record with a with an equality predicate on the primary key specified in the UPDATE statement (you can assume there are no duplicdates absed on the search value; only one). You will update only one column of the given tables (one column in the SET assignment). UPDATE will not have join conditions. In other words, it will act on a single table.

Testing: Your program will be tested with several concurrent transactions (calling ”dbms” several times in multiple windows) trying to update a set of tables at the same time with the same script or different scripts. You can assume each DBMS call processes one transaction. You can assume the scripts have only one transaction. Your program should detect deadlocks in which case the transaction gets aborted. Your program must execute interleaved execution of transactions steps, provided locks are issued/released with 2PL. The interleaving order is arbitrary and expected to be different in each execution. There will be SELECT * to query the updated tables.

3 Program call and Output

Here is a specification of input/output.

- Provide prompt, Start threads: dbms
- Evaluate queries or transactions in scripts: dbms script=$filename$

If no arguments are provided the threads are started. If the a script is given, but there are no threads then the program rejects script execution. The default testing mode will be with script files.

3.1 Interactive mode

```
c:\>dbms

SQL>CREATE TABLE T ( C1 INT, C2 CHAR(5), C3 INT,PRIMARY KEY(C1));
TABLE T created successfully
SQL> SELECT C1 FROM T;
SQL> quit;
```

3.2 Script mode

dbms script=xyz.txt $> output.txt$[1]

The script file will contain a combination of:

Phase 1: CREATE TABLE, SELECT,

Phase 2: CREATE TABLE, SELECT, UPDATE, BEGIN TRANSACTION, COMMIT, ROLLBACK

[1]The output.txt file should contain all the results and messages of the SQL queries in the xyz.txt file.