COSC6340: Database Systems
Mini-DBMS Project

Instructor: Carlos Ordonez

1 Introduction

In this project you will develop a small DBMS in C++. Your program will be focused on managing SPJ queries in 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 initial phase is focused on storage and selection of data (SELECT and JOIN). the second phase is based on sorting and grouping techniques (GROUP BY and ORDER BY). Each submission will be graded on the specific aspects of the related phase. However, you are expected to fix the bugs or missing functionalities from previous phases in order to finish future phases.

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.

The first 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 application will finalize when the user types the "quit;" command.

Additionally, your initial phase of the project requires a catalog table that contains all the metadata information for all the tables. This catalog table is going to be maintained and managed in main memory, and updated when the application closes (lazy update). The metadata 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 commands that are required for the initial part of the project are the following. "SHOW TABLE", which will return the DDL information of the specified table. CREATE TABLE, which will create a table in the DBMS. INSERT INTO should add given record to the required specified table. The SELECT/FROM operator should return the specified columns without duplicates. The SELECT/FROM/WHERE query will be similar to the SELECT/FROM operation but, also considers the comparison criterion given in the where clause ($<, >, =$). For the JOIN operation, you are only required to cross two tables with only a single attribute as a join condition. If you have repeated table names, they should include the source table name (e.g. T1_A , T2_A). INSERT INTO would insert all the data of a SELECT operation in the specified table. If you are trying to insert data into a table that does not exist you should return an error.

For the second part of the project, you are required to implement an ORDER BY operation of a specified table (using an efficient algorithm like merge sort). In addition, a new operation called INSERT INTO/ORDER BY will be implemented, with the requirement that it should store the data retrieved by the SELECT in an ordered fashion in the specified table. Finally, a SELECT/GROUP BY operation will be performed, where you are required to exploit your sorting algorithm from the previous operation to obtain this functionality. You can assume that only a SUM aggregation will be performed on numerical data.

You will be given +2 points (on a 10 scale) if you can include (AND,OR and NOT) operations in the WHERE clause for the first phase. Also, you will be given +1 point (second phase) if you can include an indexing technique by using any library of your choice that includes B+Trees or Hash indexing.

2.1 Programming details

2.1.1 Phase 1

In the initial phase, you are required to call your application as “uhdbms.exe”, from which you can send 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) and chars (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).

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

In the second phase, your programming requirements include modifying your application to sort data. This sorting is the base for the other two important functionalities of the second phase: GROUP BY and BINARY

Table 1: Queries.

Operator	Phase	Parameters	Example
CREATE TABLE	1	Table Name, Columns and Types	CREATE TABLE T (C1 INT, C2 CHAR(5), C3 INT, PRIMARY KEY(C1));
INSERT INTO	1	Data	INSERT INTO T VALUES(1,‘some text’,5);
SELECT/FROM	1	Columns	SELECT C3 FROM T;
SELECT/FROM/WHERE	1	Columns $(=, >, <)$	SELECT C1 FROM T WHERE C2=‘some text’;
SELECT/FROM/JOIN	1	Table Names	SELECT T.A FROM T JOIN T1 ON (T.A = T1.B);
INSERT INTO	1	Table, Select Query	INSERT INTO T SELECT B FROM T1;
SHOW TABLES	1	None	SHOW TABLES;
SHOW TABLE	1	Table Name	SHOW TABLE T;
QUIT	1	None	QUIT;
SELECT/FROM/ORDER BY	2	Columns, Table, Columns	SELECT C1, C2 FROM T ORDER BY C1;
INSERT INTO/ORDER BY	2	Table, SELECT ORDER BY Query	INSERT INTO T3 SELECT T_A FROM T1 ORDER BY B;
SELECT/FROM/GROUP BY	2	Columns, Table, Columns	SELECT C1, SUM(C3) FROM T GROUP BY C1;
SELECT/FROM/WHERE (Binary search)	2	Columns, Table, Columns	SELECT * FROM T3 WHERE C2=‘some text’; [1]

SEARCH. Sorting should be performed using an in-file sorting approach (a large table will not fit in memory). Therefore, you will have to merge binary files if you decide to include the merge sort algorithm. You can assume that there will be at most one attribute in the sorting function. Once you have coded your ORDER BY operation, you will have to perform a GROUP BY on a previously sorted table. This GROUP BY operation may imply that you have to create temporary tables in order to present the final aggregation. Only consider the SUM aggregation in numerical attributes, and a single attribute in the GROUP BY statement. The second phase of the project also includes a special operation (that you will not find in a DBMS), which is called INSERT INTO/ORDER BY. This operation requires you to store the data in sorted fashion. For this statement, only consider a simple SELECT with an ORDER BY. The last operation that you will have to code for the second phase implies that you will have to modify your SELECT/FROM/WHERE function to allow a faster search in previously sorted tables. Hence, you are required to perform a binary search on a sorted table.

A summary of the operations (including the submission phase) that you are required to implement are described in Table 1.

Some restrictions that you need to take into consideration include:

- Do not consider NULLs for this project.
- Only two tables are going to be considered in the JOIN statement.
- Only one condition will be considered in the WHERE statement.
- Return an error message for every invalid operation (e.g. load data into a non-existent table.)
- Do not consider nested queries for this project.

3 Program call and Output

Here is a specification of input/output.

- uhdbms script=$filename$
- uhdbms $< query >$

Example of call:

1) Example 1:
uhdbms CREATE TABLE T (C1 INT, C2 CHAR(5), C3 INT,PRIMARY KEY(C1));

[1] In a sorted table, you are required to perform a binary search.

Created TABLE T successfully
$SQL >$ SELECT C1 FROM T
$SQL >$ quit;

2) Example 2: uhdbms script=xyz.txt $> output.txt$[1]

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