COSC6340: Database Systems
Mini-DBMS Project

Instructor: Carlos Ordonez

1 Introduction

You will develop a small DBMS in C++ supporting SQL with nested SPJ queries. The small DBMS will require you to use binary files for efficient processing. You will have to submit this program in two phases. You are expected to fix the bugs or missing functionaly from the previous phase. If you have doubts about how an SQL statement should work try it on a commercial DBMS. If you want to implement extra SQL functionality, especially related to complex query optimization feel free to send email.

2 Program requirements

Your program should be coded in C++ using the Borland C++ v. 5.5 Compiler (other compilers are discouraged due to automated grading). You can use an IDE or editor. The program needs to efficiently manage large tables. Therefore, your application cannot use regular text files. Instead, you are required to use binary files (with efficient random access). Here is an overview:

- Phase 1: DDL, DML INSERT, simple SQL queries (WHERE). Queries can be nested (SQL subqueries).
- Phase 2: required: join, push selection over join. optional: binary search, group-by aggregations, order-by (sorting).

Phase 1 requires supporting two modes of entering SQL queries: (1) an SQL console, interactive (2) a script file, batch mode. The 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 enter more SQL queries by displaying the prompt ("$SQL >$"). The system will stop when the user types the "quit;" command. Additionally, Phase 1 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 program stops (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. The script file option allows the execution of a long SQL script combining DDL and DML statements; this feature will be essential for testing.

The SQL commands that are required for Phase 1 are the following. "SHOW TABLE", which returns the DDL information of the specified table. CREATE TABLE, which creates a table in the DBMS. Nested SELECTs in FROM clause (up to 5 levels). INSERT INTO/VALUES should insert one row into the target table. The SELECT/FROM statement should return the specified columns and result rows (optional: eliminate duplicates with DISTINCT). The SELECT/FROM/WHERE query will be similar to the SELECT/FROM operation but also considers a comparison given in the WHERE clause (=). 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.

Phase 2: Support join queries (up to 2 joins on 3 tables per SELECT). YOu must use the "JOIN Table ON condition" syntax (do not join tables in the WHERE clause). Query optimization by pushing selection over join, at the deepest level possible (traversing the tree). Optional: Implement ORDER BY, GROUP BY (using an efficient external sort algorithm: merge sort). For JOIN, you are required to join two tables with a single column as the join condition (equi-join). If there are repeated column names, they should include the alias table name (e.g. T1.A , T2.A).

2.1 Program Specification

Phase 1

The main program ashould be called ‘dbms.exe”, from which you can submit an SQL query or a batch file containing a set of SQL queries. The program 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 char(n) (strings, 1 byte per char). Every time you complete a statement, you are expected to display sucess of such operation like a DBMS (e.g. Created TABLE T Successfully). To parse nested SELECTs you can build your own parser with a stack or you can use a compiler tool (lex/yacc). You are not required to parse and evaluate arithmetic expressions or boolean expressions with negation.

In order to efficiently manage the database, 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 one table header (description) 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 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
```

Phase 2

To evaluate joins you can use a nested-loop or sort-merge join algorithm. To push σ given in the WHERE clause push it as deep as possible in the query plan tree.

Optional: sorting feature is the foundation for important functionality: GROUP BY, ORDER BY and binary search. Sorting should be performed using an external sort algorithm (i.e. a large table will not fit in memory). Therefore, the system has to merge binary files if using a merge sort algorithm. You can assume that there will be only one column in for sorting. 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. For query optimization you should push a GROUP BY before a JOIN when possible.

A summary of the SQL statements (including the submission phase) that you are required to program are described in Table 1.

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

Table 1: SQL Queries: Specification and examples.

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/VALUES	1	Data	INSERT INTO T VALUES(1,'string',5);
SELECT/FROM	1	Columns	SELECT C3 FROM T;
SELECT/FROM/WHERE	1	Columns (=)	SELECT A,B FROM T WHERE A=1;
nested SELECTs	1		SELECT A FROM (SELECT B FROM T1)T2;
INSERT INTO/SELECT	1	Table, Select Query	INSERT INTO T SELECT B FROM T1;
SHOW TABLE/SHOW TABLES	1	name/None	SHOW TABLE T; SHOW TABLES;
QUIT	1		QUIT;
SELECT/FROM/JOIN	2	Table Names	SELECT T.A FROM T JOIN T1 ON T.A = T1.B;
SELECT/FROM/JOIN/WHERE	2	Table Names, conditions	SELECT A,B,K FROM T1 JOIN T2 ON T1.K = T2.KB WHERE B=1;
SELECT/FROM/ORDER BY	2	Columns, Table, Columns	SELECT C1, C2 FROM T ORDER BY C1;
INSERT INTO/SELECT/ORDER BY	2	Table, SELECT/ORDER BY	INSERT INTO T3 SELECT T_A FROM T1 ORDER BY B;

Some important considerations and limits include: Up to five levels of nested SELECTs. No NULLs: do not consider NULLs. Up to two joins, up to three tables can be joined per SELECT statement (no self join). Consider ambiguous column names for join operations (qualification). Only one comparison in the WHERE statement (=). Return an error message for every invalid operation (e.g. non-matching data type in an insertion; invalid join/where condition; load data into a non-existent table.)

Requirements

Use C++ and any available libraries for parsing. If you download code from the Internet you must acknowledge it in your submission (in the README file). However, you cannot downaload and modify an existing DBMS source code (like MySQL, PostgreSQL). It is expected you will develop most of your code and you will submit a small zip file. It is not expected you program provides advanced I/O capabilities like blocked record storage, buffer management and multi-threaded parallel processing, but you can send e-mail to the Instructor about it.

3 Program call and Output

Here is a specification of input/output. Notice quotes are necessary on the command line.

- dbms script=$filename$
- dbms ”SQL query”

Example of call:

1) Example 1:
dbms ”CREATE TABLE T (C1 INT, C2 CHAR(5), C3 INT,PRIMARY KEY(C1));”
TABLE T created successfully
$SQL >$ SELECT C1 FROM T;
$SQL >$ quit;

2) Example 2:
dbms script=xyz.txt
$> output.txt$
The output.txt file should contain all the results and messages of the SQL queries in the ”xyz.txt” file.