**CS157B Spring 2022**

**Project: DBMS Implementation**

**Due Dates: Part 1 - April 12 (120 points); Part II – May 09 (130 points)**

Description:

You are responsible to implement a simple DBMS system from the code given in from the class. This DBMS will allow the user to type in simple DDL statements and build a system catalog (set of packed descriptors). Once the table definition is in place, the user can insert, update, delete, and select from a table. All the commands are passed into the CLP. Items 1-4 source code will be given out.

Project Specification:

CLP : db.exe (Windows) db (Mac/Linux) (Usage: db "DDL or command statement" )

Statement syntax and its function:

Note: { ... } means the content of ... within {} pair can be repeated separated by a comma ','.

Note: [ ... ] means the content within [] is optional.

Note: The '{', '}', '[', ']' symbols are NOT part of the syntax.

1) CREATE TABLE table\_name ( { column\_name <data\_type> [NOT NULL] } )

<data\_type> : INT, CHAR(n)

- Create a table packed descriptor (TPD) and column descriptor(s) (CD) in the database file (DBF) which contains all the table information.

- The DBF must be named dbfile.bin

2) DROP TABLE table\_name

- Drop the TPD from the DBF.

3) LIST TABLE

- List all the tables in the DBF.

4) LIST SCHEMA FOR table\_name [TO report\_filename]

- Display all the information within the TPD for any given table in a report format, on screen or to a file.

- The report\_filename is given by the user and can be any valid file name (identifier) without an extension.

5) INSERT INTO table\_name VALUES ( { data\_value } )

- <date\_value> can be any <string literal>, integer literal, or the keyword NULL

- <string literal> : ‘any string enclosed with single quotes’

- String literal is case sensitive.

- Must enforce the column NOT NULL violation during insertion.

- e.g. db “insert into tab1 values (‘Student Name’, 12345, NULL)”

6) SELECT \* FROM table\_name [ NATURAL JOIN table\_name2 ]

7) DELETE FROM table\_name [ WHERE column\_name <relational\_operator> data\_value ]

- <relation\_operator> can be >, <, or =

- Delete 0 or more rows depending on the search condition.

- Must return a warning if no row is found.

- e.g. db “delete from tab1 where gender = ‘F’

8) UPDATE table\_name SET column = data\_value [ WHERE column\_name <relational\_operator> data\_value ]

- Update 0 or more rows depending on the search condition.

- Must return a warning if no row is found.

9) SELECT { column\_name } FROM table\_name [ NATURAL JOIN table\_name2 ]

[ WHERE column\_name <condition> [(AND | OR) column\_name <condition>] ]

[ ORDER BY column\_name [DESC] ]

|

SELECT <aggregate>(column\_name) FROM table\_name [ NATURAL JOIN table\_name2 ]

[ WHERE column\_name <condition> [(AND | OR) column\_name <condition>] ]

[ ORDER BY column\_name [DESC] ]

- <condition> : <relational\_operator> data\_value

- <condition> : IS NULL

- <condition> : IS NOT NULL

- Return 0 or more rows of data which matches the search condition.

- The column\_name in the select list can be replaced by the \* symbol meaning all the columns.

- If the ORDER BY clause is not there, then display the records in the storage order.

- <aggregate> can be SUM, AVG, COUNT

- SUM & AVG are only valid on integer column. The \* symbol is not a valid substitution.

- COUNT can be used in any column type or \*, it always count the # of rows depending on the condition.

Other details:

-- Each statement can have many combination of valid and invalid syntax, all syntax errors must be handled.

-- For this project, an identifier is defined as follow:

must start with 'A'..'Z' or 'a'..'z'

can contain any letter, number, or the underscore ('\_')

must not be greater than 16 characters long

-- Any keyword and type name can also be a valid identifier.

-- Implementation details such as file format, data structure, and report format will be given.

Getting started:

Step 1: Understand the get\_token() source code.

-- Identify all the keywords, type names, and symbols from the command syntax:

Keywords, type and function names:

"int", "char", "create", "table", "not", "null", "drop", "list", "schema",

"for", "to", "insert", "into", "values", "delete", "from", "where",

"update", "set", "select", "order", "by", "desc", "is", "and", "or",

"sum", "avg", "count"

Symbols:

"(", ")", ",", “\*”, “=”, “<”, “>”

-- Identify all types of numeric literal used. In this case, only integer is valid. Therefore a token 123.5 is consider an invalid token, use token class INVALID.

-- Use meaningful enum as the token class value for each keyword and symbol.

e.g. CREATE, LEFT\_PAREN, RIGHT\_PAREN, INT\_LITERAL, IDENTIFIER

-- Parse each command and generate a linked list of tokens.

e.g. db "create table tab1(name char(50), total\_score int)"

token string token class token class value

create keyword K\_CREATE

table keyword K\_TABLE

tab1 identifier IDENT

( symbol S\_LEFT\_PAREN

name identifier IDENT

char type\_name T\_CHAR

( symbol S\_LEFT\_PAREN

50 constant INT\_LITERAL

) symbol S\_RIGHT\_PAREN

, symbol S\_COMMA

total\_score identifier IDENT

int type\_name T\_INT

) symbol S\_RIGHT\_PAREN

<empty> terminator EOC

-- If there is an invalid token found, then token class should be "error" and token class value should be "INVALID".

-- If the user uses a keyword where in place of an identifier, display the class as keyword and value as the K\* value.

Step 2: After the call to get\_token(...), now start looking at each token.

-- Group the first 2 tokens together and you will have different combinations.

Example:

if ((cur\_token->token\_value == K\_CREATE) && (cur\_token->next->token\_value == K\_TABLE))

{

cur\_command = CREATE\_TABLE; /\* it must be a create table statement \*/

cur\_token = cur\_token->next->next; /\* consumed 2 tokens, therefore advance 2 tokens ahead \*/

}

...

switch(cur\_command)

{

case CREATE\_TABLE:

{

...

break;

}

case DROP\_TABLE:

{

...

break;

}

}

Step 1 - Add the new requirement to given source code.

1. Assume that you can only have a max of 100 rows for this project. Keep all the rows in memory.
2. At the end of the create table statement, you must now add the code to create a new file call <table\_name>.tab.

(This is the point where you need to calculate the record\_size). Even with an empty table, you still need the file header structure defined below when first creating the file.

1. Add to the DROP TABLE code to delete the table.tab file when completed.
2. Find the maximum size by first calculating the record length. E.g. CREATE TABLE statement:

Create Table students ( id int, first char(10), last char(10), age int )

Then your record size is (1+4) + (1+10) + (1+10) + (1+4) = 32 bytes; then you will need to allocate 3200 bytes of storage with malloc(). We use 1 byte for length which implies your string is <=255 bytes. The INT data type’s length is always 4 bytes. To represent a NULL value, you will use length=0. Otherwise it must have a length for any string. That also implied that you can’t have an empty string.

1. Malloc() to get memory block; Memset() everything to 0 so there are no garbage.
2. Read from table file to see if there are any records.

Table file structures:

typedef struct table\_file\_header\_def

{

int file\_size; // 4 bytes

int record\_size; // 4 bytes

int num\_records; // 4 bytes

int record\_offset; // 4 bytes

int file\_header\_flag; // 4 bytes

tpd\_entry \*tpd\_ptr; // 8 bytes. (e.g. 64-bit pointer)

} table\_file\_header; // minimum size = 28

Step 2: Implement the INSERT statement.

- Always add a record to the end of the file. Make sure you increment the num\_records fields in the header.

Step 3: Implement the following SELECT statement for checking the insert result.

- SELECT \* from table\_name

At this point, you have a semi-working table with insert and select \*.

Note:

The size of each record is fixed for this project but the record size depends on the table definition. Therefore you must follow these rules:

- Use 1 unsigned byte as a length tag for each field.

- The total record\_size must be rounded to a 4-byte boundary.

For example:

column 1 is char(10)

column 2 is int

column 3 is char(22)

Then the total size is (1+10) + (1+4) + (1+22) = 39, must round this to 40.

The tpd\_ptr is there just to ease the searching during run-time. You MUST refresh this value each time the table file is read from disk. Also you must 0 out this field before writing the data file to disk.