CS-6360 Database Design

Programming Project #2: Files and Indexing

Instructor: Chris Irwin Davis

1. Overview

The goal of this project is to implement a (very) rudimentary database engine that is loosely based on MySQL, which I call **DavisBase**. Your implementation should operate entirely from the command line (no GUI).

2. Requirements

2.1. Prompt

Upon launch, your engine should present a prompt similar to the **`mysql>`** prompt, where interactive commands may be entered. The DavisBase prompt should be:

```
davisql>
```

2.2. Supported Commands (Summary)

Your database engine must support the following high-level commands. All commands should be terminated by a semicolon (;).

- `SHOW SCHEMAS` – Displays all schemas defined in your database.
- `USE` – Chooses a schema.
- `SHOW TABLES` – Displays all tables in the currently chosen schema.
- `CREATE SCHEMA` – Creates a new schema to hold tables.
- `CREATE TABLE` – Creates a new table schema, i.e. a new empty table.
- `INSERT INTO TABLE` – Inserts a row/record into a table.
- ~~`DELETE FROM` – Deletes one *or more* rows/records from a table~~. Requirement removed
- `DROP TABLE` – Remove a table schema, and all of its contained data.
- "`SELECT-FROM-WHERE`" -style query
- `EXIT` – Cleanly exits the program and saves all table and index information in non-volatile files.

2.3. Supported Commands (Detail)

The detailed syntax for the above commands is described below.

```
SHOW SCHEMAS;
```

Display a list all database schemas by name, including the system **information_schema**.

```
USE schema_name;
```

This determines the schema that is currently in use (i.e. active). All other table-specific commands should consider only tables in the database schema that is currently active. When DavisBase is launched, the currently active schema should default to **information_schema**. This behavior differs from MySQL, which does not have a default active schema when it is launched.

```
SHOW TABLES;
```

Display a list all table names in the currently used schema.

```
CREATE SCHEMA schema_name;
```

Create a new schema.

```
CREATE TABLE table_name (
    column_name1 data_type(size) [primary key|not null],
    column_name2 data_type(size) [primary key|not null],
    column_name3 data_type(size) [primary key|not null],
    ...
);
```

Create the table schema information for a new table. It will be created in the currently schema. In other words, add appropriate entries to the system **information_schema** tables that define the described **CREATE TABLE**.

Your table definition should support the following data types. All numbers should be represented as bytes in Big Endian order.

Data Type	Data Size (bytes)	Description
BYTE	1	A signed two's compliment byte: range -128 to 127
SHORT INT, SHORT	2	A signed two's compliment short integer: range -32768 to 32767
INT	4	A signed two's compliment integer: range -2147483648 to 2147483647
LONG INT, LONG	8	A signed two's compliment long integer: range -2^{63} to $2^{63} - 1$
CHAR(*n*)	*n*	A fixed length ASCII string of *n* characters, including the string terminator \n (i.e. 0x00). Strings less than *n* are padded with \0's.
VARCHAR(*n*)	*variable*	A variable length ASCII string with a maximum of *n* characters. *n* may be **0-127**. Each instance is prepended with an unsigned byte indicating the number of ASCII characters that follow.
FLOAT	4	A single precision IEEE 754 floating point number
DOUBLE	8	A double precision IEEE 754 floating point number
DATETIME	8	An unsigned long int that represents the specified number of milliseconds since the standard base time known as "the epoch". It should display as: YYYY-MM-DD_hh:mm:ss, e.g. `2016-03-23_13:52:23`.
DATE	8	A datetime whose time component is 00:00:00, but does not display.

CHAR中‘\0’不占用字符

long类型的最大值是 9223372036854775807；
用long类型可以更节约空间！！！

DATETIME 和 DATE 在创建表格的时候不需要什么修改和转换，
但是在插入时，把字符串转换成long，
在select显示时，需要把long型转换成字符串。

The only table constraints that you are required to support are PRIMARY KEY and NOT NULL (to indicate that NULL values are not permitted for a particular column). All primary keys are single column keys. If a column is a primary key, its **`information_schema.COLUMNS.COLUMN_KEY`** attribute will be "PRI", otherwise, it will be the empty string. If a column is defined as NOT NULL, then its **`information_schema.COLUMNS.IS_NULLABLE`** attribute will be "NO", otherwise, it will be "YES".

You are *not* required to support FOREIGN KEY, since multi-table queries are not supported in DavisBase.

```
INSERT INTO ~~TABLE~~ table_name VALUES (value1,value2,value3,…);
```

Insert a new record into the indicated table.

If *n* values are supplied, they will be mapped onto the first *n* columns. Prohibit inserts that do not include the primary key column or do not include a NOT NULL column. For columns that allow NULL values, INSERT INTO TABLE should parse the keyword NULL in the values list as the special value NULL.

```
~~DELETE FROM table_name WHERE column_name operator value;~~
```

~~Delete (remove) an existing record from a table.~~

~~This should set the record's active bit to 0 (false). Therefore the record will still physicially exist int he table, but should be excluded from all queries.~~

You do not have to support DELETE.

```
SELECT *
FROM table_name
WHERE column_name operator value;
```

Query syntax is similar to formal SQL. The result set should display to stdout (the terminal) formatted like a typical SQL query. The differences between DavisBase query syntax and SQL query syntax is described below.

`SELECT` only needs to support the * wildcard, which will display all columns in **`ORDINAL_POSITION`** order.

You only need to support one filter condition in the **`WHERE`** clause. Note that the **`WHERE`** clause is optional (as in MySQL). Omitting it will result in the delete removing *all* rows/records from the table.

3. File Formats

Both table data and index data must be saved to files so that your database state is preserved after you exit the database. When you re-launch **DavisBase**, your database engine should load the previous state from table data and index files.

3.1. Table Files

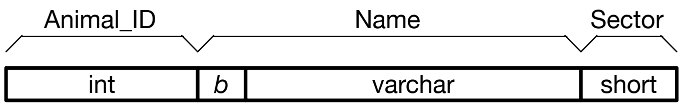
Tables files should store table data in binary format. Table files should not include any delimiters between records or between columns. i.e. No linefeeds (`\n`). No carriage returns (`\r`). No string terminators (`\0`).

Table data files should use the naming convention: ***`schema_name`***.***`table_name`***.**`tbl`**.

Consider the following table definition.

```
CREATE TABLE Zoo (
  Animal_ID INT PRIMARY KEY,
  Name      VARCHAR(20),
  Sector    SHORT INT
);
```

Each record is the following binary format, where **int**=4-bytes, ***b***=1-byte (varchar size byte) + **varchar** ASCII string of length *b*, **short**=2-bytes



By example, the following is the binary representation of four records using the above table schema. Color coding is provided to assist in visualizing the correspondence between the raw text data and database binary data.

`schema_name`.**`Zoo.tbl`**

57,giraffe,9
12,elephant,5
23,lion,4
17,hippo,5

```
Byte Address            Binary Data (as hex)

      0       00000039 07676972 61666665 00090000
     16       000C0865 6C657068 616E7400 05000000
     32       17046C69 6F6E0004 00000011 05686970
     48       706F0005
```

You are highly encouraged to use a Hex Editor to examine files for debugging your code.

3.2. Index Files

For this assignment, index files must be created for all columns in a table. This allows efficient search (binary lookup) on any field. Therefore, each table insert should append a new record to the end of a data file and concurrently update all associated index files.

Index files should use the naming convention: ***`schema_name`***`.`***`table_name`***`.`***`column_name`***`.ndx`.

The file format must be binary with each index entry being a **key-value** pair. The **key** is the column value. The **value** is a list of location(s) where the associated record is in the data file. Each record location in the list is a 4-byte integer that indicates the number of bytes offset from the beginning of the data file. The value list begins with a 4-byte integer that indicates how many values follow.

`Zoo_schema.Zoo.Animal_ID.ndx`

12,01,14
17,01,40
23,01,29
57,01,00

```
0000000C 00000001 0000000E 00000011
00000001 00000028 00000017 00000001
0000000D 00000039 00000001 00000000
```

`Zoo_schema.Zoo.Name.ndx`

elephant,01,14
giraffe,01,00
hippo,01,40
lion,01,29

```
08656C65 7068616E 74000000 01000000
0E076769 72616666 65000000 01000000
00056869 70706F00 00000100 00002804
6C696F6E 00000001 0000001D
```

`Zoo_schema.Zoo.Sector.ndx`

4,01,29
5,02,14,40
9,01,00

```
00000004 00000001 0000001D 00000005
00000002 0000000E 00000028 00000009
00000001 00000000
```

4. System Tables: information_schema

The following three system tables are defined to always exist in the **`information_schema`**. They hold schema definitions for all tables, including themselves. These are based on the information_schema tables used by MySQL. You are encouraged to examine the information_schema tables and their data in an actual MySQL instance for reference.

The **`DESCRIBE`** ***`table_name`*** syntax below is for informational purposes only. You do not have to support this command.

Structure of the SCHEMATA table.

```
DESCRIBE SCHEMATA; -- an abbreviated "describe"
+--------------------+--------------+------+-----+
| Field              | Type         | Null | Key |
+--------------------+--------------+------+-----+
| SCHEMA_NAME        | varchar(64)  | NO   |     |
+--------------------+--------------+------+-----+
```

Structure of the TABLES table.

```
DESCRIBE TABLES; -- an abbreviated "describe"
+-----------------+---------------------+------+-----+
| Field           | Type                | Null | Key |
+-----------------+---------------------+------+-----+
| TABLE_SCHEMA    | varchar(64)         | NO   |     |
| TABLE_NAME      | varchar(64)         | NO   |     |
| TABLE_ROWS      | long int            | YES  |     |
+-----------------+---------------------+------+-----+
```

Structure of the COLUMNS table.

```
DESCRIBE COLUMNS; -- an abbreviated "describe"
+------------------+---------------------+------+-----+
| Field            | Type                | Null | Key |
+------------------+---------------------+------+-----+
| TABLE_SCHEMA     | varchar(64)         | NO   |     |
| TABLE_NAME       | varchar(64)         | NO   |     |
| COLUMN_NAME      | varchar(32)         | NO   |     |
| ORDINAL_POSITION | int unsigned        | NO   |     |
| COLUMN_TYPE      | varchar(64)         | NO   |     |
| IS_NULLABLE      | varchar(3)          | NO   |     |
| COLUMN_KEY       | varchar(3)          | NO   |     |
+------------------+---------------------+------+-----+
```

Initial values in the information_schema tables. Upon launch, these three information_schema tables will need to be created if they don't exist and initialized with the data about the information_schema table themselves.

```
SELECT * FROM SCHEMATA;
+--------------------+
| SCHEMA_NAME        |
+--------------------+
| information_schema |
+--------------------+

SELECT * FROM TABLES;
+--------------------+--------------+-----------+
| TABLE_SCHEMA       | TABLE_NAME   | TABLE_ROWS |
+--------------------+--------------+-----------+
| information_schema | SCHEMATA     | 1          |
| information_schema | TABLES       | 3          |
| information_schema | COLUMNS      | 7          |
+--------------------+--------------+-----------+

SELECT * FROM COLUMNS;
+--------------------+-------------+------------------+------------------+-------------+-------------+-----------+
| TABLE_SCHEMA       | TABLE_NAME  | COLUMN NAME      | ORDINAL_POSITION | COLUMN_TYPE | IS_NULLABLE | COLUMN_KEY |
+--------------------+-------------+------------------+------------------+-------------+-------------+-----------+
| information_schema | SCHEMATA    | SCHEMA_NAME      | 1                | varchar(64) | NO          |            |
| information_schema | TABLES      | TABLE_SCHEMA     | 1                | varchar(64) | NO          |            |
| information_schema | TABLES      | TABLE_NAME       | 2                | varchar(64) | NO          |            |
| information_schema | TABLES      | TABLE_ROWS       | 3                | long int    | NO          |            |
| information_schema | COLUMNS     | TABLE_SCHEMA     | 1                | varchar(64) | NO          |            |
| information_schema | COLUMNS     | TABLE_NAME       | 2                | varchar(64) | NO          |            |
| information_schema | COLUMNS     | COLUMN_NAME      | 3                | varchar(64) | NO          |            |
| information_schema | COLUMNS     | ORDINAL_POSITION | 4                | int         | NO          |            |
| information_schema | COLUMNS     | COLUMN_TYPE      | 5                | varchar(64) | NO          |            |
| information_schema | COLUMNS     | IS_NULLABLE      | 6                | varchar(3)  | NO          |            |
| information_schema | COLUMNS     | COLUMN_KEY       | 7                | varchar(3)  | NO          |            |
+--------------------+-------------+------------------+------------------+-------------+-------------+-----------+
```

The following is an example of how the information schema tables would be updated in the case of a new schema and table were created. This example does not include the creation of entries into that new table. Note that the number of TABLES_ROWS is initially zero before records are inserted.

```
CREATE SCHEMA Zoo_schema;
CREATE TABLE Zoo (
  Animal_ID INT PRIMARY KEY,
  Name      VARCHAR(20),
  Sector    SHORT INT
);

SELECT * FROM SCHEMATA;
+--------------------+
| SCHEMA_NAME        |
+--------------------+
| information_schema |
| Zoo_schema         |
+--------------------+

SELECT * FROM TABLES;
+--------------------+--------------+-----------+
| TABLE_SCHEMA       | TABLE_NAME   | TABLE_ROWS |
+--------------------+--------------+-----------+
| information_schema | SCHEMATA     | 1          |
| information_schema | TABLES       | 3          |
| information_schema | COLUMNS      | 7          |
| Zoo_schema         | Zoo          | 0          |
+--------------------+--------------+-----------+

SELECT * FROM COLUMNS;
+--------------------+-------------+------------------+------------------+-------------+-------------+-----------+
| TABLE_SCHEMA       | TABLE_NAME  | COLUMN NAME      | ORDINAL_POSITION | COLUMN_TYPE | IS_NULLABLE | COLUMN_KEY |
+--------------------+-------------+------------------+------------------+-------------+-------------+-----------+
| information_schema | SCHEMATA    | SCHEMA_NAME      | 1                | varchar(64) | NO          |            |
| information_schema | TABLES      | TABLE_SCHEMA     | 1                | varchar(64) | NO          |            |
| information_schema | TABLES      | TABLE_NAME       | 2                | varchar(64) | NO          |            |
| information_schema | TABLES      | TABLE_ROWS       | 3                | long int    | NO          |            |
| information_schema | COLUMNS     | TABLE_SCHEMA     | 1                | varchar(64) | NO          |            |
| information_schema | COLUMNS     | TABLE_NAME       | 2                | varchar(64) | NO          |            |
| information_schema | COLUMNS     | COLUMN_NAME      | 3                | varchar(64) | NO          |            |
| information_schema | COLUMNS     | ORDINAL_POSITION | 4                | int         | NO          |            |
| information_schema | COLUMNS     | COLUMN_TYPE      | 5                | varchar(64) | NO          |            |
| information_schema | COLUMNS     | IS_NULLABLE      | 6                | varchar(3)  | NO          |            |
| information_schema | COLUMNS     | COLUMN_KEY       | 7                | varchar(3)  | NO          |            |
| Zoo_schema         | Zoo         | Animal_ID        | 1                | int         | NO          | PRI        |
| Zoo_schema         | Zoo         | Name             | 2                | varchar(20) | YES         |            |
| Zoo_schema         | Zoo         | Sector           | 3                | short       | YES         |            |
+--------------------+-------------+------------------+------------------+-------------+-------------+-----------+
```

Drop Table的时候，需要把它给冲刷掉；并且把所有的.dat文件和.ndx文件删掉