Programming Project, Part 1:
Database Files and Indexing

CS-6360 Database Design
Instructor: Chris Irwin Davis

1. Overview

The goal of this project is to implement a (very) rudimentary database engine that is based on a simplified file-per-table variation on the SQLite file format, which we call **DavisBase**. Your implementation should operate entirely from the command line and possibly API calls (no GUI).

Like MySQL's InnoDB data engine (SDL), your program will use *file-per-table* approach to physical storage. Each database table will be physically stored as a separate single file. Each table file will be subdivided into logical sections of fixed equal size call *pages*. Therefore, each table file size will be exact increments of the global `page_size` attribute, i.e. all data files must share the same `page_size` attribute. You may make `page_size` be a configurable attribute, but your implementation must capable of supporting a page size of **512 Bytes**. The test scenarios for grading will be based on a page_size of 512B. Once a database is initialized, your are *not* required to support a reformat change to its `page_size` (but you may implement such a feature if you choose).

Your team may choose any langage for implementation, but all examples will be provided only in **Java**.

DavisBase data is encoded in two different kinds of database files—tables files and index files. Each database file is stored as a single file in the underlying OS.

Each DB file is comprised of one or more *pages* (a virtual subdivision of the file). All pages of a file are the same size. For example, if the page size is set to 1024 bytes (1kb), then each DB file size is some multiple of 1024 bytes.

- Each page in a Table file (interior or leaf) is a node in a B^{+1} tree.
- Each page in a Table file (interior or leaf) is a node in a B tree.

The location of each element in a page is referenced with a "page offset" value (i.e. the number of bytes from the beginning of the page that the element is located.

2. Requirements

2.1. Prompt

Upon launch, your engine should present a prompt similar to the MySQL **`mysql>`** prompt or SQLite **`sqlite>`** prompt, where interactive commands may be entered. Your prompt text may be hardcoded string or user configurable. It should appear something like:

```
davisql>
```

2.2. Summary of Required Supported Commands

Your database engine must support the following DDL, DML, and DQL commands. All commands should be terminated by a semicolon (;). Each one of these commands will be tested during grading.

DDL (Data Definition Language)

- **Show tables** – displays a list of all tables in DavisBase.
- **Create table** – creates a new table file, its associated meta-data, and indexes (if they exist).
- **Drop table** – removes a table file, its associated meta-data, and indexes (if they exist).
- **Create index** – creates an index file that is associated with a table file. Note that DavisBase only allows indexes to be createed on *single columns*.
- **Exit** – Cleanly exits DavisBase and saves all table, index, and meta-data information to disk in non-volatile files.

  Note that you **do not** have to implement `ALTER TABLE` schema change commands.

  The database catalog (i.e. meta-data) shall be stored in two special tables that should exist by default: **`davisbase_tables`** and **`davisbase_columns`**.

DML (Data Manipulation Language)
- **Insert** – inserts a new record into a table file and updates any associated meta-data and indexes.
- **Delete** – removes a record from a table file and updates any associated meta-data and indexes.
- **Update** – modifies an existing record in a table file and updates any associated meta-data and indexes.

- `INSERT INTO` *`table_name`* `[(`*`column_list`*`)] VALUES (`*`value_list`*`);`
  - Inserts a single record into a table.
- `DELETE FROM` *`table_name`* `[WHERE` *`condition`*`];`
  - Deletes one *or more* records from a a table.
- `UPDATE` *`table_name`* `SET` *`column_name`* `=` *`value`* `[WHERE` *`condition`*`];`
  - Modifies one *or more* records in a table.

DQL (data query language)
- **Select-From-Where** – performs a standard SQL ***select-from-where*** format query and displays the result to the screen, but does not support nested queries or complex where conditions (e.g. IN, EXISTS, etc.). You additionally do not have to support SQL commands ORDER BY, GROUP BY, HAVING, or AS.

  - You **do not** have to support multiple WHERE conditions. The condition is a single *`column comparison_operator value`* clause.
  - You **do not** have to support nested queries.
  - You **do not** have to support `ORDER BY`, `GROUP BY`, `HAVING`, or `AS` alias.

3. Details of Required Supported Commands

The detailed syntax for the above commands is described below.

3.1. DDL (Data Definition Language) Commands

Show Tables

```
SHOW TABLES;
```

Displays a list of all table names in the database. Note: this is equivalent to the query:

```
SELECT table_name FROM davisbase_tables;
```

Create Table

Create a table schema

```
CREATE TABLE table_name (
    column_name1 data_type1 [NOT NULL][UNIQUE],
    column_name2 data_type2 [NOT NULL][UNIQUE],
    ...
);
```

Create the table schema information for a new table. In other words, add appropriate entries to the system **davisbase_tables** and **davisbase_columns** tables that define the described **CREATE TABLE** and create the associated *table_name*.**tbl** data file.

Note that every table in DavisBase automatically creates an additional unique "hidden" column named **rowid**. This extra column is stored in a specially designated place in each record. It is represented as a 4-byte two's complement integer. Row IDs are unique over the lifetime of a table and are never re-used. Once a record has been deleted, its **rowid** is never repurposed. They begin at the value 1 and increase monitonically. Note that **rowid** is separate from any user-defineed PRIMARY KEY.

The only table constraints that you are required to support are PRIMARY KEY, UNIQUE, and NOT NULL (to indicate that NULL values are not permitted for a particular column). If a column is the primary key, its **davisbase_columns.column_key** attribute will be the string "**PRI**". If a column is unique (but the primary key), its **davisbase_columns.column_key** attribute will be the string "**UNI**". If a column is neither a primary key, nor otherwise unique, its **davisbase_columns.column_key** attribute will be **NULL**.

If a column is defined as **NOT NULL** in the table schema, then its **davisbase_columns.is_nullable** attribute will be the string "**NO**", otherwise, it will be "**YES**".

You are *not* required to support any type of **FOREIGN KEY** constraint, since multi-table queries (i.e. Joins) are not required to be supported in your project.

Drop Table

```
DROP TABLE table_name;
```

Removes a table file, its meta-data, and any associated indexes it may have.

Create Index

```
CREATE INDEX table_name (column_name);
```

Creates new index file (B-tree) based on the given table and column name. All DavisBase indices are only based on a single column, i.e. no compound indices. The file name in the underlying OS file system should be `table_name.column_name.ndx`.

3.2. DML (Data Manipulation Language) Commands

Insert Row Into Table

```
INSERT INTO TABLE (column_list) 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 Record

```
DELETE FROM TABLE table_name [WHERE condition];
```

Delete one or more records from a table given an optional WHERE condition.

Update Record

```
UPDATE table_name SET column_name = value WHERE condition;
```

Modify one or more records from a table given a WHERE condition. Note that you will not be required to update TEXT columns to values that will be longer than the original column string length.

3.3. DQL (Data Query Language) Commands

Query Table

```
SELECT *
FROM table_name
WHERE [NOT] condition;
```

Query syntax is similar to formal SQL. The result set should display to stdout (the terminal) formatted like a typical SQL query. Note that a WHERE condition may apply to multiple records.

You do not have to support the following types of queries:

- Nested queries
- Join conditions
- Complex boolean WHERE conditions, i.e. just a single column. Although you must support equalities (=), inequalities (>, ≥, <, ≤, <>), and negation (**`NOT`**).

If **`SELECT`** has the * wildcard, it will display all columns in **`ORDINAL_POSITION`** order, not including the hidden column **`rowid`**.

4. SDL (Storage Definition Language)

This information is in a separate DavisBase File Format Guide.