Team Programming Project

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 Summary

2.1. Prompt

Upon launch in a system terminal (MacOS) or command (`cmd`) window (Windows), your engine should present an 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) Commands

- `SHOW TABLES;`
  - Displays a list of all tables in DavisBase.
- `CREATE TABLE (column_list);`
  - Creates a new table file, its associated meta-data, and index files for the `PRIMARY KEY` or `UNIQUE` columns (if they exist). The syntax is similar to ISO standard SQL.
- `DROP TABLE table_name;`
  - Removes a table file, its associated meta-data, and indexes (if they exist).
- `CREATE INDEX index_name ON table_name (column_name);`
  - Creates a new index file and its associated meta-data. The syntax is similar to ISO standard SQL, except that multi-column indexes need not be supported.
- `DROP INDEX index_name;`
  - Removes an index file and its associated meta-data.
- `EXIT;`
  - Cleanly exits DavisBase and saves all table, index, and meta-data information to disk in non volatile files.• Show tables – displays a list of all tables in DavisBase.

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

DML (Data Manipulation Language) Commands

- `INSERT INTO table_name 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 from a a table.

DQL (data query language) Commands

- `SELECT column_list FROM table_name [WHERE condition];`
  - Performs a standard SQL select-from-where format query and displays the result to the screen in a tabular (rows and columns) ASCII format. Only a small subset of SQL query options (defined elsewhere) are required.

3. Requirement Details

The detailed syntax for the above commands is described below.

3.1. Data Definition Language (DDL) Commands

Show Tables

```
SHOW TABLES;
```

Displays a list of all table names in the database as a single column.

Create Table

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

This command creates a new table file and its associated meta data in the data dictionary. Table files are structured as page-based B^{+1}-tree files.[1] The file name in the underlying OS file system should be *table_name*.ndx.

Only one column may be labeled `PRIMARY KEY`. The only other column constraint keywords in the `CREATE TABLE` command are `NOT NULL` and `UNIQUE`. You do not need to support `DEFAULT`, `ASSERT`, or other `CREATE TABLE` column constraint keywords.

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. `rowid`'s begin at the value 1 and increase monitonically. Note that `rowid` is distinct and separate from any user-defineed `PRIMARY KEY`.

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.

[1] B^{+1}-Tree file format details are described in the DavisBase Storage Definition Language (SDL) document.

Create Index

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

This command creates new index file. Index files are structured as page-based B-tree files.[2] All DavisBase indexes are only based on a single column, i.e. no multi-column indexes. The file name in the underlying OS file system should be `index_name.ndx`.

Drop Index

```
DROP INDEX index_name ON table_name;
```

This command deletes an existing index, its associated `.ndx` index file, and any associated meta data in the data dictionary.

3.1. Data Manipulation Language (DML) Commands

Insert Row Into Table

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

This command inserts a new record into the indicated table. If the table has *n* number of columns then the command must have *n* values are supplied. You *do not* need to implement syntax that allows supplying column names and fewer than *n* values. If a column should have no value, then supply the keyword `NULL` in the associated place in the value list. Prohibit inserts that have a `NULL` value for the `PRIMARY KEY` column or any `NOT NULL` constraint column. The insert command should automatically generate an associated value for the "hidden" `rowid` column that is auto-incremented by *exactly one more* than the most recent row inserted into the table.

Delete Row(s) From Table

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

This command deletes one or more records from the indicated table based on the `WHERE` condition. The deletion of a record entails *only* switching the deletion byte of the record to be `true`. When executing a query, any record with a `true` deletion byte should *not* be displayed in the result set. Details of the `WHERE` clause syntax can be found under the Data Query Language (DQL) section.

[2] B-Tree file format details are described in the DavisBase Storage Definition Language (SDL) document.

3.2. Data Query Language (DQL) Commands

Query A Result Set From A Table

Query syntax is similar to ISO standar SQL. The result set should display to the terminal standard output (STDOUT) in a tabular ASCII format (rows and columns). The result set should include column names in the first row of the output. Your implementation should support any number of valid columns in the table displayed in any designated order. Your implementation should support use of the * column wildcard that should display all table columns in the same order they appear in the table's schema definition. The * character should *not include* the `rowid` column. The otherwise hidden `rowid` column should be displayed in a query result only when it is specifically listed by name. It may be combined with any other column name(s) or with *.

The following example will display all columns (including `rowid`) and all rows from ***`table_name`***.

```
SELECT rowid, * FROM table_name;
```

The `WHERE` condition needs only to support ***comparison conditions***, their ***negation***, or ***combinations of comparisons***, i.e. you *do not* need to support nested queries inside the `WHERE` clause. Comparison conditions are of the form ***`column_name comparison_op value`***. Newlines in SQL statements should be ignored. Multi-line statements don't terminate until a semicolon is reached. Some examples:

```
SELECT * FROM table_name WHERE column_name=value;
SELECT * FROM table_name WHERE column_name1>value1 AND column_name2>=value2;
SELECT * FROM table_name WHERE NOT column_name=value;

SELECT *
FROM table_name
WHERE column_name1 > value1 OR NOT column_name2 >= value2;
```

4. Storage Definition Language (SDL)

The DavisBase Storage Definition Language (SDL) details are in a separate DavisBase File Format Guide.