

# **CS 6360- Database Design Final Project**

**by**

Diya Chandra (dxc190009)

## **AUDIENCE**

This document can be used as a reference for developers, students and teachers working on learning the various aspects of relational database systems (RDBMS).

This includes the details of implementation and inner workings of a rudimentary database system. This system draws inspiration from the working mechanism of popular open source database system, namely SQLite.

## Table of Contents

1.	Introduction.....	1
2.	Content.....	1
3.	Implementation.....	1
3.1.	Database Catalog (meta-data).....	1
3.2.	B-tree Pages.....	2
3.3.	Implementation Logic .....	3
3.3.1	Database initialization .....	3
3.3.2	Query Handling .....	3
3.3.3	Database Shutdown .....	3
4.	Working .....	3
4.1.	Prompt .....	3
4.2.	Supported Commands.....	4
4.2.1.	DDL (Data Definition Language) .....	4
4.2.2.	DML (Data Manipulation Language) .....	7
4.2.3.	VDL (View Definition Language).....	8
5.	Conclusion .....	10

## 1. Introduction

The goal of this project is to implement a (very) rudimentary database engine that is loosely based on a hybrid between MySQL and SQLite, which is called **DavisBase**. This implementation operates entirely from the command line and API calls (no GUI).

## 2. Content

This database supports actions on a single table at a time, without joins or nested queries. Like MySQL's InnoDB data engine (SDL), this project uses *file-per-table* approach to physical storage. Each database table is physically stored as a separate file. Each table file is subdivided into logical sections of fixed equal size call *pages*. Therefore, each table file size is exact increments of the global `page_size` attribute, i.e. all data files share the same `page_size` attribute. This program supports a page size of **512 Bytes**.

## 3. Implementation

### 3.1. Database Catalog (meta-data)

The DavisBase Catalog consists of two tables containing meta-data about each of the user table. You may optionally choose to include meta-data about the two catalog files in the catalog itself. These two tables (and their associated implmentation files) have the following table schema, as if they had been created via the normal **CREATE** command.

```
CREATE davisbase_tables (  
  rowid INT,  
  table_name TEXT,  
  record_count INT, -- optional field, may help your implementation  
  avg_length SMALLINT -- optional field, may help your implementation  
);
```

```
CREATE davisbase_columns (  
  rowid INT,  
  table_name TEXT,  
  column_name TEXT,  
  data_type TEXT,  
  ordinal_position TINYINT,  
  is_nullable TEXT  
);
```

### 3.2. B-tree Pages

The following tables replaces the “B-tree Pages Header Format” in SQLite File Format document.

Offset from beginning of page	Content Size (bytes)	Description
0x00	1	The one-byte flag at offset 0 indicating the b-tree page type. <ul style="list-style-type: none"> <li>• A value of 2 (0x02) means the page is an interior index b-tree page (<i>not used</i>).</li> <li>• A value of 5 (0x05) means the page is an interior table b-tree page.</li> <li>• A value of 10 (0x0a) means the page is a leaf index b-tree page (<i>not used</i>).</li> <li>• A value of 13 (0x0d) means the page is a leaf table b-tree page.</li> </ul> Any other value for the b-tree page type is an error.
0x01	1	The one-byte integer at offset 3 gives the number of cells on the page.
0x02	2	The two-byte integer at offset 5 designates the start of the cell content area. A zero value for this integer is interpreted as 65536.
0x04	4	The four-byte page number at offset 8 is the right-most pointer. This value appears in the header of interior b-tree pages only and is omitted from all other pages.

The format of a cell depends on which kind of b-tree page the cell appears on. The following info shows the elements of a cell, in order of appearance, for the various b-tree page types.

#### Table B-Tree Leaf Cell (in pages whose page type header is 0x0D):

- A 2-byte SMALLINT which is the total number of bytes of payload
- A 4-byte INT which is the integer key, a.k.a. "rowid"
- The payload.
  - 1-byte TINYINT that indicates the number of columns n.
  - n-bytes which are Serial Type Codes, one for each of n columns - binary column data
- DavisBase does not support overflow payload.

#### Table B-Tree Interior Cell (in pages whose page type header is 0x05):

- A 4-byte INT page number which is the left child pointer.
- An INT which is the integer key

### 3.3. Implementation Logic

#### 3.3.1 Database initialization

- a) On first run of the program, the data directories are checked for catalog files.
- b) Catalog files are initialized with 2 default tables, namely,
  - database\_tables
  - databse\_columns

#### 3.3.2 Query Handling

- a) Prompt allows user to input queries which conform to the supported commands.
- b) Input query is parsed by query parser and split into tokens.
- c) Based on input tokens, respective functions are called and processed.

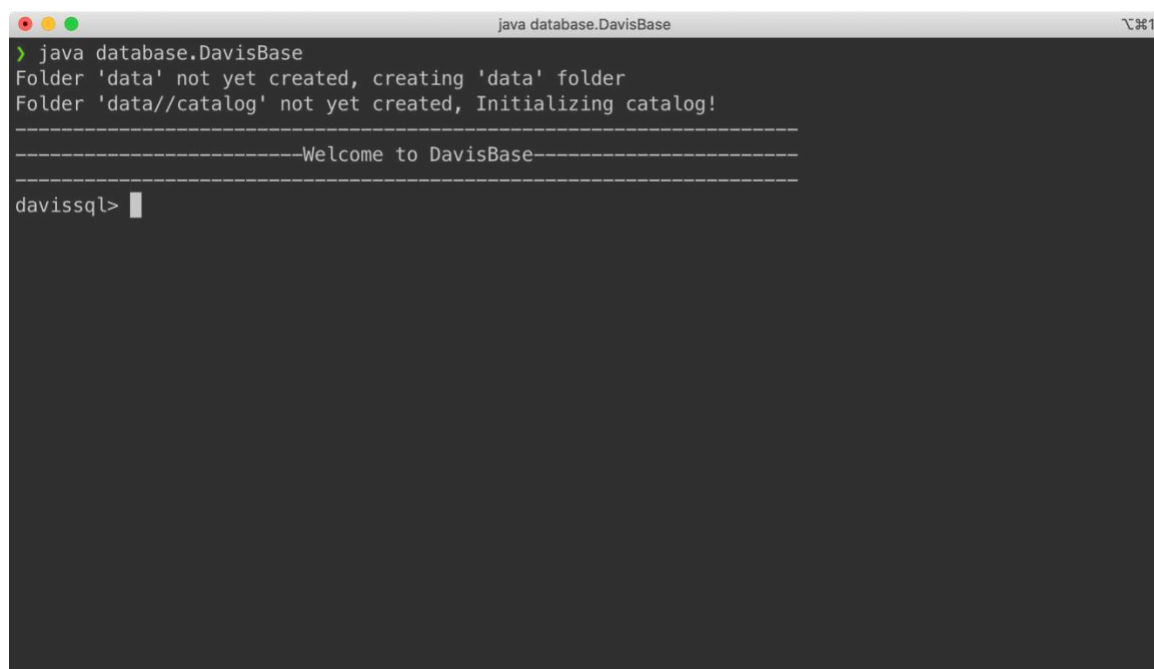
#### 3.3.3 Database Shutdown

- a) When exit command is encountered, the database shutdown routine gracefully writes all the volatile files to the secondary storage of the PC for example hard drive.
- b) 'Exiting from DavisBase' message is displayed.

## 4. Working

### 4.1. Prompt

Upon launch, the engine presents a prompt similar to the **mysql>** prompt, where interactive commands may be entered. The prompt text may be hardcoded string or user configurable. It appears like this:



```
java database.DavisBase
> java database.DavisBase
Folder 'data' not yet created, creating 'data' folder
Folder 'data//catalog' not yet created, Initializing catalog!
-----Welcome to DavisBase-----
davissql> |
```

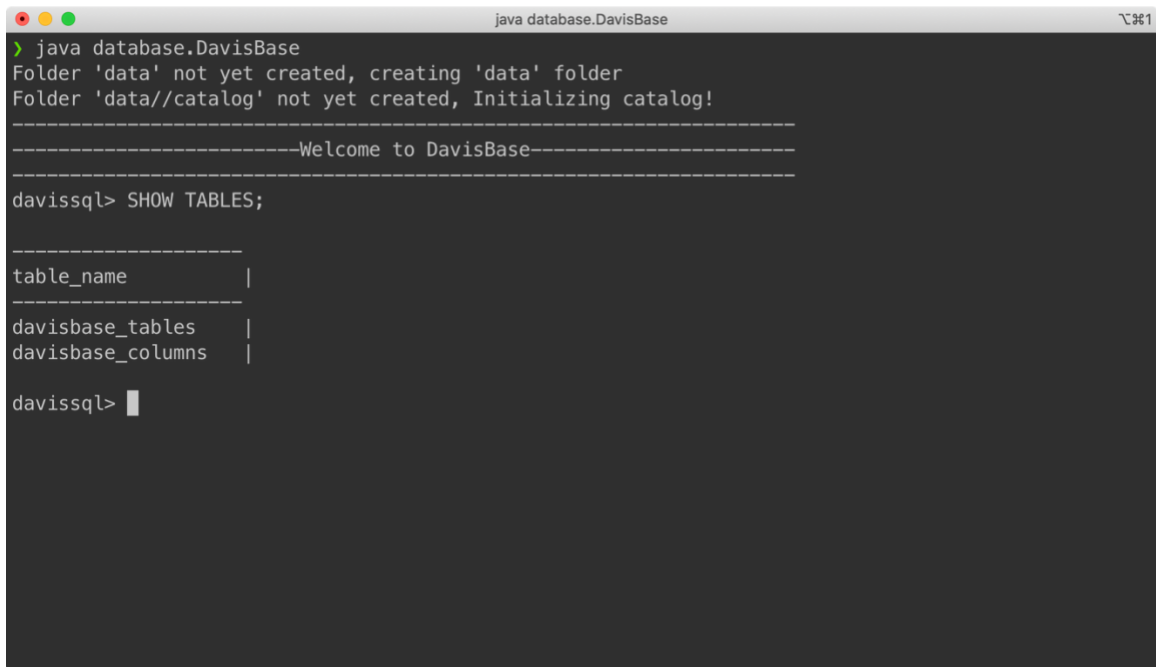
## 4.2. Supported Commands

The database engine supports the following DDL, DML, and VDL commands. All commands should be terminated by a semicolon (;).

### 4.2.1. DDL (Data Definition Language)

#### 4.2.1.1. SHOW TABLES

Displays a list of all tables in DavisBase. Initially the two catalog tables are shown upon running this command.



```
java database.DavisBase
> java database.DavisBase
Folder 'data' not yet created, creating 'data' folder
Folder 'data//catalog' not yet created, Initializing catalog!
-----Welcome to DavisBase-----
davissql> SHOW TABLES;

-----
table_name |
-----
davisbase_tables |
davisbase_columns |
davissql> 
```

When additional tables are created, running this command displays all the user created tables along with the initial two catalog tables.

```
java database.DavisBase
> java database.DavisBase
-----Welcome to DavisBase-----
davissql> CREATE TABLE animals (
id INT PRIMARY KEY,
name TEXT NOT NULL,
type TEXT NOT NULL);
Table animals created successfully.

davissql> SHOW TABLES;

-----
table_name      |
-----
davisbase_tables |
davisbase_columns |
user_data.animals |

davissql> 
```

#### 4.2.1.2. CREATE TABLE

This command creates a table with the given name and specified columns according to their types. Upon successful creation it displays the message – ‘Table created successfully’.

```
java database.DavisBase
> java database.DavisBase
-----Welcome to DavisBase-----
davissql> CREATE TABLE animals (
id INT PRIMARY KEY,
name TEXT NOT NULL,
type TEXT NOT NULL);
Table animals created successfully.

davissql> 
```

#### 4.2.1.3. DROP TABLE

It removes a table schema, and all of its contained data.

```
java database.DavisBase
table_name |
-----|
davisbase_tables |
davisbase_columns |
user_data.animals |

davissql> EXIT;

Exiting from DavisBase
> java database.DavisBase
-----Welcome to DavisBase-----

davissql> DROP TABLE animals;
Table animals dropped successfully.

davissql> SHOW TABLES;

table_name |
-----|
davisbase_tables |
davisbase_columns |

davissql> 
```



## 4.2.2. DML (Data Manipulation Language)

### 4.2.2.1. INSERT INTO TABLE

It inserts a single record at a time into the table.

```
java database.DavisBase
> java database.DavisBase
-----
-----Welcome to DavisBase-----
-----
davissql> SHOW TABLES;

-----
table_name      |
-----
davisbase_tables |
davisbase_columns |
user_data.animals |

davissql> INSERT INTO animals VALUES (
1,Lion,Wild);
Inserted Successfully

davissql> 
```

We can also insert multiple rows by executing the INSERT INTO command multiple times.

```
java database.DavisBase
> java database.DavisBase
-----
-----Welcome to DavisBase-----
-----
davissql> INSERT INTO animals VALUES (2,Goat,Domestic); INSERT INTO animals VALUES (3,Rabbit,Domestic);
Inserted Successfully

davissql> Inserted Successfully

davissql> 
```

### 4.2.3. VDL (View Definition Language)

#### 4.2.3.1. SELECT FROM WHERE

```

> java database.DavisBase
-----
-----Welcome to DavisBase-----
-----
davissql> INSERT INTO animals VALUES (2,Goat,Domestic); INSERT INTO animals VALUES (3,Rabbit,Domestic);
Inserted Successfully

davissql> Inserted Successfully

davissql> SELECT * FROM animals;
-----
id  |name  |type  |
-----
1   |lion  |wild  |
2   |goat  |domestic  |
3   |rabbit|domestic  |

davissql>

```

```

davissql> INSERT INTO animals VALUES (2,Goat,Domestic); INSERT INTO animals VALUES (3,Rabbit,Domestic);
Inserted Successfully

davissql> Inserted Successfully

davissql> SELECT * FROM animals;
-----
id  |name  |type  |
-----
1   |lion  |wild  |
2   |goat  |domestic  |
3   |rabbit|domestic  |

davissql> SELECT name FROM animals;
-----
name  |
-----
lion  |
goat  |
rabbit|

davissql>

```

```

-----
id  |name  |type  |
-----
1   |lion  |wild  |
2   |goat  |domestic |
3   |rabbit|domestic |

davissql> SELECT name FROM animals;
-----
name  |
-----
lion  |
goat  |
rabbit|

davissql> SELECT name FROM animals WHERE type=wild;
-----
name  |
-----
lion  |

davissql>

```

#### 4.2.3.2. EXIT

```

diychandra@Diyas-MacBook-Pro: ~/Old Materials/Database Design- Chris Davis/Projects/Project 2/vxi170230_Project2/DavisBase/src
> java database.DavisBase
-----
-----Welcome to DavisBase-----
-----

davissql> SHOW TABLES;

-----
table_name  |
-----
davisbase_tables |
davisbase_columns |
user_data.animals |

davissql> EXIT;

Exiting from DavisBase

~ / 0 / Datab / P / Project 2 / vxi170230 _ Project2 / D / src 9s 16:04:31

```

## 5. Conclusion

RDBMS is a widely used system. It has proved itself to be an indispensable part of data storage for large corporations across the world. The comprehensive RDBMS system requires a lot of system resources which make it not feasible to be used with small systems like IoT devices, mobile phones, etc. DavisBase is a file-based database system which is light on resources requiring less than 25 MBs of main memory for runtime. It is developed in Java making it cross-platform compatible. These properties make DavisBase an excellent choice for data storage in small systems.