# TEAM PROJECT

BUNKER: Team Chandrasekhar

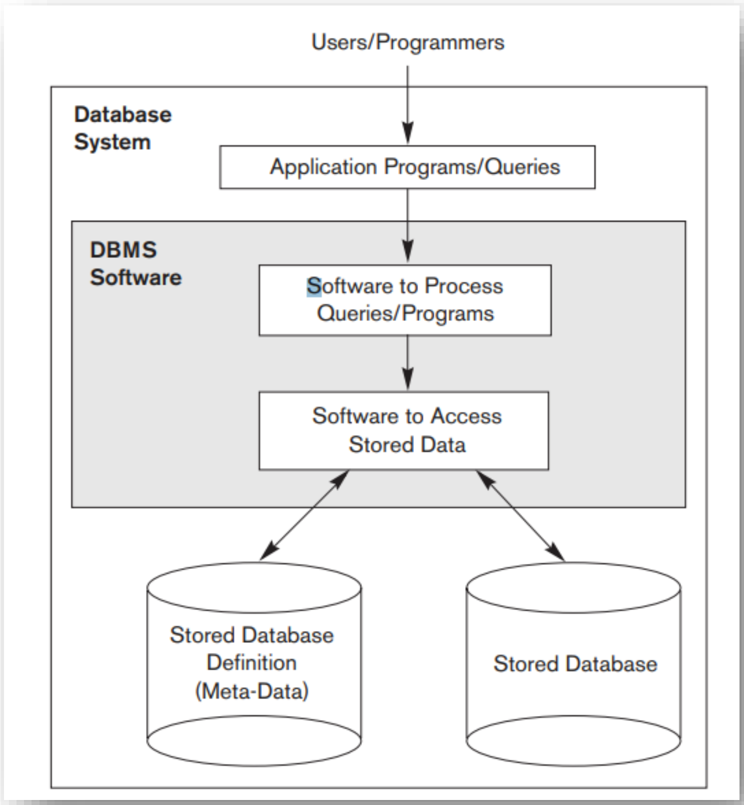
## Team Members

| NetID | Name |
|---|---|
| AXB210119 | Abhinava Bharamasagara Nanjundaiah |
| HXS210045 | Hari Venkata Deepak Sanka |
| JXD210021 | Jeevan Desouza |
| RXM210063 | Rakshitha Mahesh Kumar |
| SXG200139 | Sarthak Gupta |

## Description

A database management system (DBMS) is system software for creating and managing databases. A DBMS makes it possible for end users to create, read, update, and delete data in a database. Database applications are widely used, and different vendors have their own in which they store and process data to optimize query performance. It looks very simple for an end user to perform various database operations in daily life, but behind the scenes, a very complicated process goes into this. In this project, we would be creating a new version of the database, the DavisBase. DavisBase will have basic functionality like database creation, table creation, and various other commands that are a key part of any DBMS.

## Structure/Architecture


Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

## Programming language

- Java – 1.8
- Dart (if time permits)

## Library/framework

- **Junit-4.13.2**
- **Flutter (if time permits for UI)**

## Standout Implementational Features

Bunker is a multi-Database storage solution. Bunker system truly embraces the Object-Oriented Approach of the Java Programming Language. By doing so we reduce unnecessary code redundancy as well as human errors across the code base. We offer a simple and straightforward solution to upgrade the system for future advancements, as each part of the Database system follows Object oriented approach, in other words, each class does what it is supposed to do.

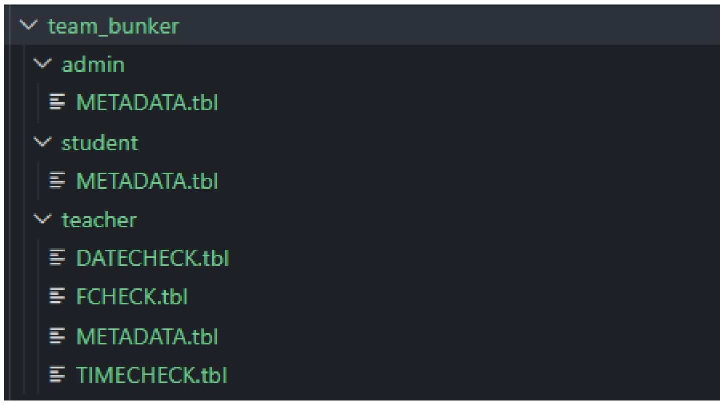
### Brief Introduction to Code Base

We have one entry point to our Bunker Database Management System which is by using DBEngine Class.

1. DBEngine.java: Loads individual database along with the Metadata required for basic operation within the database
2. Table.java: handles all the operations related to the Table such as Table SELECT, CREATE, UPDATE, INSERT INTO, DELETE, etc.
3. Metadata.java: inherited from the table class but altered to handle the metadata.
4. SupportedDataTypeConst.java: Holds all the datatype supported by Bunker Database Management System.
5. BPlusTreeController.java: This is the only place direct File Handling is done. This class is responsible for Inserting, Deleting, Updating, and Creating the binary file using the BPlusTree Insert, Update, Delete and Create functionality using read and write using Page.java Class.
6. Page.java: Loading the data inside of the binary file is done here, handling offsets, type of Page, Page header, Storing the data and reading the data, and updating the data is done here. Every data manipulation in the Page Object is written back to the table file. Page Object acts as the one and only entry point to reading/write .tbl file directly as bytes data. This in turn reduces the code complexity of BPlusTree as this class would only call appropriate functions rather than directly handling Byte level code.
7. ColumnField.java: Store information of the column of the table required for reading and validating the type of data being stored and retrieved.
8. ValueField.java: This is Inherited from ColumnField.java class but has an extra variable to store the value of the data of that particular cell, here sanity checks and data validation of the data being stored are carried out.
9. Index.java: On-demand index creation when the user wants to create an index using the create index command. Based on the column specified the indexes are created.

### Database File Structure

Our multi-database approach was possible using this file structure shown below.



```
team_bunker
  admin
    METADATA.tbl
  student
    METADATA.tbl
  teacher
    DATECHECK.tbl
    FCHECK.tbl
    METADATA.tbl
    TIMECHECK.tbl
```

Fig: File structure of the Bunker, multi-database system

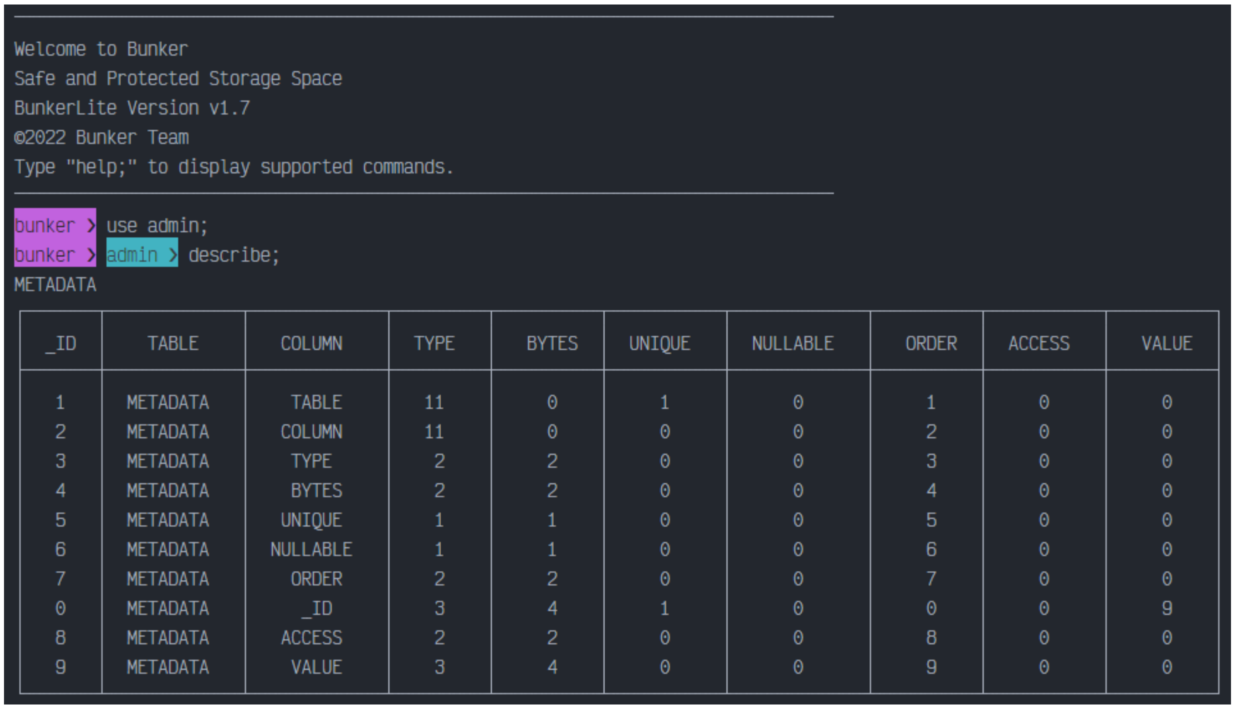
The root folder team_bunker stores all the database files and admin, student, and teacher are the 3 databases that were created. Each database has a METADATA table to maintain the metadata of respective tables in the database.

Related Commands for database operation

1. Creation of the database can be done using **CREATE DATABASE <database_name>;** command.
2. Our database also supports the creation of multiple databases, the listing of the database can be done using the **SHOW DATABASES;** command.
3. Just as in MySQL when you want to make use of or work on a database we use the **USE** command. The use command in our database is similar to MySQL **USE** database_name; command.
4. Deleting a database command DROP DATABASE <database_name>

## Database Metadata Structure

We follow in the footsteps of Postgres whilst storing the metadata of each database. We separate metadata of the tables and data itself by using a “**Metadata.tbl**” Table to store metadata info and <new_table>.tbl file to store data.



```
Welcome to Bunker
Safe and Protected Storage Space
BunkerLite Version v1.7
©2022 Bunker Team
Type "help;" to display supported commands.

bunker > use admin;
bunker > admin > describe;
METADATA

| _ID |  TABLE   |  COLUMN  | TYPE | BYTES | UNIQUE | NULLABLE | ORDER | ACCESS | VALUE |
|-----|----------|----------|------|-------|--------|----------|-------|--------|-------|
|  1  | METADATA |   TABLE  |  11  |   0   |   1    |    0     |   1   |   0    |   0   |
|  2  | METADATA |  COLUMN  |  11  |   0   |   0    |    0     |   2   |   0    |   0   |
|  3  | METADATA |   TYPE   |   2  |   2   |   0    |    0     |   3   |   0    |   0   |
|  4  | METADATA |   BYTES  |   2  |   2   |   0    |    0     |   4   |   0    |   0   |
|  5  | METADATA |  UNIQUE  |   1  |   1   |   0    |    0     |   5   |   0    |   0   |
|  6  | METADATA | NULLABLE |   1  |   1   |   0    |    0     |   6   |   0    |   0   |
|  7  | METADATA |   ORDER  |   2  |   2   |   0    |    0     |   7   |   0    |   0   |
|  0  | METADATA |    _ID   |   3  |   4   |   1    |    0     |   0   |   0    |   9   |
|  8  | METADATA |  ACCESS  |   2  |   2   |   0    |    0     |   8   |   0    |   0   |
|  9  | METADATA |   VALUE  |   3  |   4   |   0    |    0     |   9   |   0    |   0   |
```

Fig: Describing the present database

Above Metadata table shows the present database's information and metadata of itself. Also, we store auto-incrementing ID field information for the metadata table itself.

Related Command for Information retrieval

1. To describe the database information use DESCRIBE
2. To get the information of individual table use DESCRIBE TABLE <table_name>

## Database Data Operations

All Database data are stored as rows and columns in a Table. While we can create table, manage table, and delete entries if need, we support 11 types of data which can be stored in our table. Namely,

| Supported Data Types | Space Occupied in Bytes |
|---|---|
| **NULL** | 0 |
| **TINY INT** | 1 |
| **SMALL INT** | 2 |
| **INT** | 4 |
| **BIG INT** | 8 |
| **FLOAT** | 4 |
| **DOUBLE** | 8 |
| **YEAR** | 1 |
| **TIME** | 4 |
| **DATE TIME** | 8 |
| **DATE** | 8 |
| **TEXT** | 0 |

Fig: Above table describes Supported Data types with their respective byte capacity. (NOTE: text bytes size is calculated dynamically)

| Data Constraints Supported | |
|---|---|
| **Unique** | Boolean |
| **Nullable** | Boolean |

Fig: Each table column can have unique and nullable data.

Related Command for Table Operations:

1. Once you select the database you can use the create table command to create a table in the current database. The command is as follows **CREATE TABLE <tabel_name> column_name data_type, ….;**
2. **UPDATE** table_name **SET** conditions WHERE conditions; Updates a record from a table.
3. **DESCRIBE TABLE** <table_name>**;** Displays table schema. **DROP TABLE** <table_name>**;** Deletes a table data and its schema.
4. INSERT INTO <table_name> col1 type, col2 type, col3 type…; Inserts a record into the table.
5. **SELECT** <column_list> **FROM** table_name **WHERE** column_name1= <value1>, column_name2= <value2>**;** Primitive select operations on the data that is present.
6. **DELETE** FROM table_name **WHERE** conditions; Deletes a record from a table.
7. To list all the tables in the database, we also support the **SHOW TABLES;** command.

All the commands are mentioned in the README.md file along with syntax and examples.

## OTHER COMMANDS:

1. **HELP**; Displays help information regarding the database options etc.
2. **EXIT**; Close database

## Responsibilities:

| | |
|---|---|
| Abhinava Bharamasagara Nanjundaiah | DDL queries, Designed Page Read/Write, took responsibility for designing source code. |
| Hari Venkata Deepak Sanka | DML, DQL, Query conversion. |
| Jeevan Desouza | Utility methods, Exception handling. |
| Sarthak Gupta | Helper functions and Exception handling |
| Rakshitha Mahesh Kumar | Datatypes, Handling data type conversions. |

## Future Work

- We will try to include a simple user interface for this project by choosing the best user interface we used for individual projects if time permits.
- If time permits, we will include a simple user interface and make the application a terminal and web application. We will make use of the spring framework to render the information through APIs.