Download and extract contents from -
https://drive.google.com/drive/folders/1pLRC5DFDDf_X9RouQPVZaEdVdVm7vnIb?usp=sharing

The above link has following files and folders

- **Database Engine Files:**
  - DbStorage.py
  - QueryParser.py
  - QueryProcessor.py

- **Data Directory:**
  - Inserted_Data/
    - Sales/
    - College/

- **Sample Queries:**
  - Queries/
    - Sales.txt
    - College.txt
    - Custom.txt

- **Instructions to run:**
  1. Place **DbStorage.py**, **QueryParser.py**, **QueryProcessor.py** inside an empty directory where you would want to run engine and store data - let's call this ***root*** directory
  2. From **Inserted_Data** directory, copy **Sales** and **College** folders inside the ***root*** folder
  3. Run command with ***root*** as current working directory:
     python3 QueryParser.py
  4. You will be prompted to input queries with following prompt:
     fsDB>
  5. Sample queries for **Sales** and **College** datasets are provided in the **Queries** directory. Text in quotes (‘” ‘”) are comments about queries, while everything else are actual queries

FILES

QueryParser.py: Runs the interactive shell to get user input, tokenizes the query to extract keywords, content and create key-value pairs, validates the order of keywords to check if requested sequence of operations is allowed and display results

QueryProcessor.py: This class receives a dictionary of query keywords and data from the QueryParser class. It processes the user input (query text) and converts it to data structures

that can be used by the DbStorage class. It supports the following important functions: DDL, Insert, Delete, Update, DML, Join. More information could be found in the project report.

DbStorage.py: This class receives processed input from QueryProcessor. This class implements functions to interact with the underlying file system to read, write, update and delete data. Data is only loaded in batches and is kept in the form of Pandas Dataframe object in memory for processing. Data is serialized to binary format when being written into file. The DbStorage class incorporates many important functions that implement filtering, joining, sorting, group by and aggregations on data. More information could be found in the project report.

Inserted_Data: Contains **Sales** and **College** directory, which has a subset of data already inserted for Sales and College datasets. Paste **Sales** and **College** folder in your *root* directory and you can start working with these datasets. Sample queries for these datasets are given in **Queries** directory
Link to Sales dataset - https://www.kaggle.com/datasets/dataceo/sales-and-customer-data
Link to College dataset - https://www.kaggle.com/datasets/jessemostipak/college-tuition-diversity-and-pay

Queries: Directory containing sample query text for Sales dataset, College dataset - **Sales.txt** and **College.txt** respectively. **Custom.txt** has queries to create database and tables from scratch and test referential-integrity-constraint operations

Query syntax for all supported operations:

- Create/Select database:
  @database => <db_name>
  @batchsize => <numeric_batch_size>;

- Create Table:
  @create => <table_name>
  @columns => <col1> <data_type> <data_size>, <col2> <data_type> <data_size>
  @constraint => [table_name.col1 references foreign_table1.col2] [table_name.col2, table_name.col3 references foreign_table2.col4, foreign_table2.col1] ;

- Insert:
  @populate => <table_name>
  @values => [row1_value1 | row1_value2 | row1_value3 | row1_value4] [row2_value1 | row2_value2 | row2_value3 | row2_value4];

○ Update:

@update => <table1_name>
@values => <col1> = "new value"
@filter => (<table1_name>.<col1> == "old value");

○ Delete:

@delete => <table1_name>
@filter => (<table1_name>.<col1> == "old value");

○ Projection:

@use => <table_name>
@project => <table_name>.<col1_name>, <table_name>.<col2_name>, <table_name>.<col3_name>;

○ Filter:

@use => <table_name>
@filter => (((<table_name>.<col1_name> == 600.00) AND (<table_name>.<col2_name> == “xxx xxx”)) OR (<table_name>.<col2_name> == 100))
*@project => *;*

○ Only Join:

@use => <table1_name>, <table2_name>
@combine => <table1_name>.<col1> = <table2_name>.<col2>
@project => <table1_name>.<col1>, <table1_name>.<col2>, <table2_name>.<col1>, <table2_name>.<col2>;

○ Join + Filter:

@use => <table1_name>, <table2_name>
@combine => <table1_name>.<col1> = <table2_name>.<col2>
@filter => (<table1_name>.<col1> > 10)
*@project => *;*

○ Join + Filter + sort:

@use => <table1_name>, <table2_name>
@combine => <table1_name>.<col1> = <table2_name>.<col2>
@filter => (<table1_name>.<col1> != 10)
*@project => **
@sort => <table1_name>.<col1> DESC, <table2_name>.<col2> ASC;

○ Only Categorize(group by):

@use => <table1_name>, <table2_name>
@combine => <table1_name>.<col1> = <table2_name>.<col2>
@categorize => <table1_name>.<col1>
@project => COUNT:<table1_name>.<col2>, AVG:<table2_name>.<col3>;

- Join + Filter + Categorize:
  @use => <table1_name>, <table2_name>
  @combine => <table1_name>.<col1> = <table2_name>.<col2>
  @filter => (<table1_name>.<col1> != 10)
  @categorize => <table1_name>.<col1>
  @project => COUNT:<table1_name>.<col2>, AVG:<table2_name>.<col3>;

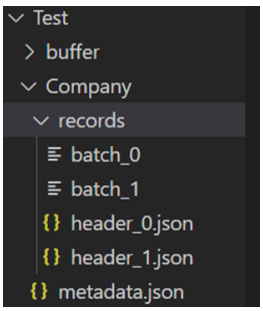
- Join + Filter + Categorize + Sort:
  @use => <table1_name>, <table2_name>
  @combine => <table1_name>.<col1> = <table2_name>.<col2>
  @filter => (<table1_name>.<col1> != 10)
  @categorize => <table1_name>.<col1>
  @project => COUNT:<table1_name>.<col2>, AVG:<table2_name>.<col3>
  @sort => AVG(<table2_name>.<col3>) DESC , COUNT(<table1_name>.<col2>) ASC;

- No double quotes are needed for strings while inserting data. Engine uses **|** as delimiter b/w different column values. Each row data is enclosed within square brackets while inserting data **[ ]**
- Parenthesis must have space b/w them in filter expression
- Inside filter expression strings must be surrounded by double quotes ● Spaces are handled by trimming preceding and trailing spaces in input values while inserting data
- To project all fields use **@project => ***
- Categorization must be followed by a project with aggregate expression ● Characters **@**, **[**, **]**, **=>** are not supported to be passed as input data, as they are special characters used by the engine
- Null values are not supported
- Supported data types: **str**, **float**, **int**
- Queries end with semicolon
- Aggregation allowed : MIN, MAX, COUNT, AVG, SUM
- Foreign key constraints while creating table using **@constraint** are optional

Directory Structure for a database

- For each database, the engine creates a directory *<db_name>/*
- Inside *<db_name>/,*
  - engine creates directory *buffer/* to store intermediate results of query operations

- for each table, engine will create a directory *<table_name>/*
- For each table directory, engine have the following files and directories:
  - *Metadata.json*
  - *records/*
  - Inside *records/* engine will create batch files to store actual data. Data for batch *i* is stored using 2 files*:*
    - *batch_i*
    - *header_i.json*



Directory structure of “Test” database with “Company” table