Bonus Assignment: 10 Points

W4111 - Introduction to Databases
Due: 12/6/2019 11:59 PM

Overview

This bonus assignment is long and will require a bit of effort. This assignment is for those who are interested in going above and beyond to improve their coding skills and understanding of database architecture; we hope that it will be rewarding. Furthermore, it is *infinitely* valuable to be thrown into a ton of code that you haven't written, make sense out of, and *improve* it. This is what the real world is like.

You can receive either 5 points or 10 points of Extra Credit on this assignment, no other score will be assigned.

This assignment provides further insight into the implementation of a database engine by implementing a very small, simple database engine to use on top of CSVFiles. On a much smaller scale, we want you to build something similar to SQL. Think about what information SQL requires (Metadata, table names, column names, primary keys…) and how SQL leverages the data that it contains to make data processing faster and better (column type constraints, indexing, query optimization…)

There are two parts to this bonus assignment. The first MUST be completed before the second.

1. Build a simple database catalog (a table that contains the metadata of all tables in a schema)
2. Build a query engine that uses the data of the catalog

Breakdown

The zipfolder contains the following files:

- Appearances.csv
- Batting.csv
- People.csv
- CSVTable.py
- CSVCatalog.py
- Unit_test_catalog.py (empty)
- Unit_test_csv_table.py (empty)

Part 1

The first step is to complete the catalog. Think of the catalog as defining and containing all of the relevant information about the table, columns, and indexes along the lines of:

The tables we want you to use are the included csv files (appearances, batting, people)

ex_table_name	ex_column_name	ex_primary_key	ex_path...
...	...	...	...

This information can be stored in a table(s) in sql and accessed as necessary or stored in a csv file(s) that you create.

The Catalog defines 3 types:

1. A *TableDefinition* represents metadata information about a CSVDataTable. The database engine will maintain the data in a CSV file. The catalog contains information about:
    1. The path/file name for the data.
    2. Column names, types and whether or not NULL is an allowed value. The column names defined via the catalog API are a subset of column headers in the underlying CSV file.
    3. Columns that comprise the primary key.
    4. A set of one or more index definitions. An index definition has a name, type of index (PRIMARY, UNIQUE, INDEX) and columns that comprise the index value.
2. ColumnDefinition: A class defining a column.
3. IndexDefinition: A class defining an index.

I have filled out the methods for the column defintion. You are responsible for filling out the rest of the methods in this file. The catalog supports defining a new table, dropping an existing table definition, loading a previous defined table definition, adding and removing columns and indices from a table definition.

These methods should be tested in the Unit_test_catalog.py file using the tables we included.

Part 2

The second step is to complete the CSVTable which leverages information from the catalog. There are only two methods you must write in this file, execute_smart_join() and dumb_join().

Dumb_join looks at each row in the left table and iteratively says “Does this row match this row in the right table?” If it does, let’s make a new entry appending the two dictionaries together. If it does not, keep looking! After aggregating the two tables, dumb_join applies the where_template to the data and returns the output. A dumb join of people.csv and batting.csv on playerID should take 30 minutes.

execute_smart_join() designates a scan table and a probe table (HINT: leverage a method I’ve included), finds the sub_where_template for each table (push down the where clause if applicable) and leverages the index to join the tables together. An optimized join of people.csv and batting.csv on playerID with a where clause should take under a minute.

Show testing and comment how long it took in the unit_test_csv_table.py file. Include screenshots if necessary.