COMS W4111: Introduction to Databases
Homework 2: Implement a Very Simple Database
(Programming Track - 15 points)

Due: October 22, 11:59 pm ET

Overview

Focus on the general understanding of this assignment before starting to code.

In this assignment you are tasked with implementing a simple database in python. You will be creating and maintaining metadata tables, creating and using indexes, and implementing joins and join optimizations. This homework will give you a rudimentary understanding of how a relational database server like MySQL works.

This is an extension of CSVDataTable, all the data for the tables will be from CSV files. The metadata tables will be stored in your AWS SQL server.

There is a lot of coding you must do. There is a lot of code that we provide to help you.

Submission details will be at the end.

You should read these instructions thoroughly.

I'll say it again.

You should read these instructions thoroughly.

Part 1: Written/SQL Questions

Answer the questions in the accompanying notebook. This will provide some conceptual overview for the programming assignment.

Part 2: CSVCatalog

CSVCatalog implements a simple database catalog for any CSV files you give it. If you don't know what Metadata tables are, stop here and read up on them a bit.

The catalog defines four classes:

1. A ***TableDefinition*** represents metadata information about a CSVTable. This is the only class you will have to complete in the CSVCatalog file. The database engine will maintain the metadata in tables on your AWS MySQL Server. While we could ask

you to maintain these metadata tables in CSV files, integrity checking would get overwhelming so we will take the middle ground by having you manually maintain the metadata using MySQL's integrity checking. 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. (Implementation provided)
3. ***IndexDefinition***: A class defining an index. (Implementation provided)
4. ***CSVCatalog***: A class to create, drop, and retrieve a table. (Implementation provided)

The catalog supports:

- Defining a new table.
- Dropping an existing table definition.
- Loading a previous defined table definition.
- Adding and removing columns from a table definition.
- Adding and removing indexes from a table definition.

You will save the catalog information in a set of tables you create in your AWS MySQL Database. There is a create.sql script included that will create the CSVCatalog schema and associated tables. You must run this script in DataGrip,

Methods to complete in TableDefinition:

- load_columns(self)
- load_indexes(self)
- load_core_definition(self)
- drop_col_in_sql(self, cn)
- define_index(self, index_name, columns, type="index")
- get_index(self, ind_name)
- drop_index(self, index_name)

Please read through CSVCatalog for more details on each method. Please note: If you make any additional helper methods you MUST add documentation in the same style as the rest of the code is provided. If you add any methods please make note of this in your readme and include a brief explanation.

Complete the tests in the unit_test_csv_catalog.py to ensure your metadata tables are working properly. Some tests will be provided, some you will need to implement as specified. If you test more than specified, please detail it in your README and make it clear what you are testing.

The tests that you must complete are:

- drop_table_test()
- add_column_test()
- add_index_test()
- col_drop_test()
- index_drop_test()

The tests that are provided that you must execute are:

- create_table_test()
- column_name_failure_test()
- column_type_failure_test()
- column_not_null_failure_test()
- describe_table_test()

Examples for test outputs are provided in unit_test_catalog.txt. You must format the output of your tests in the same format as the example. Test within PyCharm while developing/debugging. Execute your tests within your notebook once everything works. You will need to save your .py files and restart your jupyter kernel so it will use the most updated versions.

Make sure everything is working before moving onto Part 3.

Part 3: CSVTable

In this part of the assignment you will build a rudimentary MySQL engine capable of performing SELECT statements and JOINs. You will not have to implement insert, delete, or update for CSVTable, partially because you have already done a version in CSVDataTable in HW1 and to reduce the amount of work for this already substantial assignment.

There are two main methods you must get functioning.

1. *find_by_template()* determines if there is an applicable/best index (access path) that can be used to accelerate a find_by_template(). If there is an applicable index, find_by_template used the index by calling __find_by_template_index__(). You must implement the find using indexes as well as the logic to determine if there is an applicable index. Find_by_template() is given but the helper methods it calls are not. You must implement:
    a. __get_access_path__(self, fields)
        i. Figures out if there is an index that can be used
        ii. If multiple indexes can be used , selects the most selective index
    b. __find_by_template_scan__(self, t, fields=None)
        i. If there are no applicable indexes, just scans through all the rows
    c. __find_by_template_index__(self, t, idx_name, fields=None)
        i. Uses the best index to find rows that match the template.
2. *__smart_join__()* performs a join of the table (self) with the input table. Based on reviewing the lecture material (or Google), you should easily be able to identify two vastly different, significant optimizations. Your solution should document the optimizations in comments and implement the optimizations. The method

> dumb_join() is provided as well as its helper methods as an example and can be used for guidance.

Complete the tests in the unit_test_csv_table.py to ensure your metadata tables are working properly. Some tests will be provided, some you will need to implement as specified. If you test more than specified, please detail it in your README and make it clear what you are testing. You don't need to add these tests in your jupyter notebook.

The tests you must complete are:

- update_people_columns()
- update_appearances_columns()
- update_batting_columns()
- add_index_definitions()
- add_other_indexes()
- sub_where_template_test()
- test_find_by_template_index()
- smart_join_test()
- Any methods you might make to do the join comparison test specified in the notebook

The tests you must execute are:

- create_lahman_tables()
- test_load_info()
- Test_get_ column_names()
- load_test()
- dumb_join_test()

Examples for test outputs are provided in unit_test_csv_table.txt. You must format the output of your tests in the same format as the example. Test within PyCharm while developing/debugging.

Execute your tests within your jupyter notebook once everything works.

Submission

Like Homework 1, you must submit Homework 2 in two places.

Submit a PDF of your jupyter notebook under the Homework 2: Database Engine (PDF) tab on Gradescope. Make sure that the test output stays formatted correctly when you export. You may have to figure out the best way for you to export. Any export method (print preview, download as PDF, HTML, etc) is fine, it just matters that your PDF is legible.

Submit a ZIP file under the Homework 2: Database Engine (ZIP) tab on Gradescope. The Zip file must contain ONLY the following files:

W4111_HW2_Programming Folder

- CSVCatalog.py
- CSVTable.py
- HW3 README
- unit_test_catalog.py
- unit_test_csv_table.py
- W4111_HW2_Programming.ipynb