KeyDB

Members:

Benjamin Shen (bfs45)
Donal Lowsley-Williams (dml333)
Kevin Arias (ka383)

Status Meetings:

Tuesday: 2:30pm - 4:30pm
Thursday: 2:30pm - 4:30pm
Sunday: 1:00pm - 2:00pm

Proposal:

We'd like to implement a database management system that handles SQL queries through a REPL. A user will be able to:

1. interact with the DBMS through a toplevel
2. create a new table (file) or drop an existing table
3. insert, update, and remove rows from a particular table
4. select rows from a particular table with certain conditions
5. add, delete, and sum columns (if possible) from a particular table
6. see a log of actions and undo actions during a session

The user will be able to interact with the DBMS through the top level. Within the DBMS, the user should be able to create new tables and remove existing tables like so:

> **CREATE** table col1 col2 col3 … coln
>
> **DROP** table

Note that the keywords are bolded in this example. Creating a table will create a file named [table].csv with the columns [col1] through [coln]. Within a table, the user should have the ability to insert new rows, delete existing rows and update cells within a row. This will all be based on a certain key for the row to identify which record to either add, remove, or update. This key will be managed by the database system, so the user will not be able to alter key values themselves.

> **IN** table **INSERT** col1val col2val col3val … colnval
>
> **IN** table **REMOVE** keyval

[**IN** table] selects which table they are working with - the keyword [**IN**] designates which table, and [table] is the name of the table. [**INSERT**] will add a row to the table, with the trailing object

phrase representing the values for the various columns in the table. [**REMOVE**] will remove whatever row is associated with the key [keyval] from the table.

IN table **UPDATE** keyval col newval

[**UPDATE**] is the command to change the value in a specific cell. The command above will find the cell in table [table] that corresponds to column [col] and key [keyval] and change it to [newval].

From a table, the user will have various querying commands that display data from the table. They should be able to view things like the entire table, only certain columns, or records that match certain conditions. These commands will be similar to the MySQL notation:

IN table **SELECT** col1 … colx **WHERE** condition1 or condition2 … and conditionx

[**IN**] represents the same keyword as above. [**SELECT**] chooses which columns inside the table [table] they would like to pull data from. This could be replaced by [*] which would select all columns. [**WHERE**] is an optional keyword, if they want to select everything in the column then they could omit this. However, if they wanted to apply a condition (or multiple conditions) to the values in the column then they would add it here.

IN table **ADD** colx

IN table **DELETE** colx

The user will have the ability to modify the tables themselves and do things like add or remove columns. After designating which table to use, the keywords [**ADD**] and [**DELETE**] will either add or remove a column [colx] from a table.

IN table **SUM** colx

IN table **COUNT** colx

IN table **COUNT_NULL** colx

The user will also be able to manipulate data from a column with certain keywords. Above, [**SUM**] would add all the values in [colx], while [**COUNT**] would count all of the records. [**COUNT_NULL**] will count all the *null* values in [colx].

LOG

UNDO

These last functions are for the user to debug and solve their issues. With [**LOG**], the user will be able to see a log of all the actions they have done in the DBMS session. Calling [**UNDO**] will reverse the last action the user has made on the database, in case they made a mistake.

Roadmap:

MS1:

Satisfactory:

1. Create unit test file for test-driven development
2. Implement create_table and drop_table

Good:

1. Implement insert_row, update_cell, and remove_row
2. Allow user to use functions through toplevel

Excellent:

1. Implement select_all
2. Allow user to use select_all through toplevel with [in _ select *]

MS2:

Satisfactory:

1. Implement select_all and get_column
2. Allow user to use select_all and get_column through toplevel with [in _ select _]

Good:

1. Implement select
2. Allow user to use select through toplevel with [in _ select _ where _]

Excellent:

1. Implement add_column and remove_column

MS3:

Satisfactory:

1. Implement add_column, remove_column, sum_column, count, count_null
2. Allow user to use functions through toplevel

Good:

1. Implement make_log, get_log
2. Allow user to use get_log through toplevel with [log]

Excellent:

1. Implement undo
2. Allow user to use undo through toplevel with [undo]

Design Sketch:

Modules

- Table with table functions (create_table, drop_table, insert_row, remove_row, select_all, get_column, select, add_column, remove_colum) and row functor
- Row with row functions (update_cell)
- Log with log functions (make_log, get_log, undo)
- Command with types (phrase, command), exceptions (Empty, Malformed), and function (parse)
- Main with REPL interface functions (start_dbms, dbms)

Data

We will store csv files in a databases directory to imitate a database system. Each database table will be represented by a hash table, hashing keys to rows. Each row will be represented by an association list, mapping column names to values.

Libraries

We won't use any third-party libraries, but we will use the Hashtbl and List.Assoc built-in modules.

Testing

Before writing any functions, we'll create the unit test file. For each function, we'll write unit tests. For most of the functions, there will also be a way to test them through the REPL. As a team, we'll play the specification game and let the teammates that didn't write the function create the tests.