MelvinLecoy / gitcode Private

Code Issues Pull requests Actions Projects Security Insights Settings

master

gitcode / p3-database / silly.cpp

Kwan Ting Lau gitcode History

0 contributors

378 lines (343 sloc) | 9.9 KB

```cpp
// Project Identifier: C0F4DFE8B340D81183C208F70F9D2D797908754D
#include "Table.h"
#include "TableEntry.h"
#include "getopt.h"  //
#include <algorithm> //
#include <cstddef>
#include <cstring>   //
#include <exception> //
#include <getopt.h>
#include <ios>
#include <iostream> //
#include <iterator>
#include <map> //
#include <string>
#include <unordered_map>
#include <utility> //
#include <vector>  //

using namespace std;

class Silly {

public:
  bool quiet = false;
  unordered_map<string, Table> map;

  void QUIT() { // 地毯
    cout << "Thanks for being silly!"
         << "\n";
  }

  void CREATE() { //地毯
    string tablename;
    cin >> tablename;
    if (map.count(tablename)) {
```

```
      cout << "Error during CREATE: Cannot create already existing table "
           << tablename << "\n";
      getline(cin, tablename);
    } else {
      int col_num;
      cin >> col_num;
      Table temp;
      temp.table_init(col_num);
      map.emplace(tablename, temp);
      cout << "New table " << tablename << " with column(s) ";
      for (int i = 0; i < col_num; i++) {
        auto ans = map[tablename].columns[i].column_name;
        cout << ans << " ";
      }
      cout << "created"
           << "\n";
    }
  }

  void REMOVE() { // 地毯
    string name;
    cin >> name;
    if (!map.count(name)) {
      cout << "Error during REMOVE: " << name
           << " does not name a table in the database "
           << "\n";

      getline(cin, name);
    } else {
      map.erase(name);
      cout << "Table " << name << " deleted"
           << "\n";
    }
  }

  void PRINT() { // 地毯
    string junk;
    string tablename;
    cin >> junk >> tablename;
    if (map.count(tablename)) {
      int num;
      cin >> num;
      num = map[tablename].print_all(num, tablename, quiet);
      if (num == -1) {
        getline(cin, tablename);
      } else {
        cout << "Printed " << num << " matching rows from " << tablename
             << "\n";
      }
    } else {
      cout << "Error during PRINT: " << tablename
           << " does not name a table in the database "
           << "\n";

```

```
      getline(cin, tablename);
    }
  }

  void JOIN() { // 检查一下cin 还有最后的分类是不是正确
    string junk;
    string table_1;
    string table_2;
    string colname_1;
    string colname_2;

    int N;
    int value_1;
    int value_2;
    int table_value;
    int col_value;

    cin >> table_1 >> junk >> table_2 >> junk;

    if (map.count(table_1) && map.count(table_2)) {
      cin >> colname_1 >> junk >> colname_2 >> junk >> junk >> N;

      value_1 = map[table_1].find_column(colname_1);
      value_2 = map[table_2].find_column(colname_2);
      if (value_1 == -1) {
        cout << "Error during JOIN: " << colname_1
             << " does noy name a column in " << table_1 << "\n";
        getline(cin, junk);
        return;
      }
      if (value_2 == -1) {
        cout << "Error during JOIN: " << colname_2
             << " does noy name a column in " << table_2 << "\n";
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        return;
      }

      vector<string> up;
      vector<pair<int, int>> down;

      for (int i = 0; i < N; i++) {
        cin >> junk >> col_value;
        if (col_value == 1) {
          table_value = map[table_1].find_column(junk);
        } else {
          table_value = map[table_2].find_column(junk);
        }
        if (table_value != -1) {
          down.emplace_back(make_pair(col_value, table_value));
          up.emplace_back(junk);
        } else {
          if (col_value == 1) {
            cout << "Error during JOIN: " << junk
                 << " does not name a column in " << table_1 << "\n";
```

```
            getline(cin, junk);
            return;
          } else {
            cout << "Error during JOIN: " << junk
                 << " does not name a column in " << table_2 << "\n";
            getline(cin, junk);
            return;
          }
        }
      }
      int count_2 = 0;
      if (quiet) {
        unordered_map<TableEntry, vector<size_t>> middle;

        for (int row = 0; row < static_cast<int>(map[table_1].data.size());
             row++) {
          auto it = middle.find(map[table_1].data[row][value_1]);
          if (it == middle.end()) {
            vector<size_t> value_3(1, row);
            middle.emplace(map[table_1].data[row][value_1], value_3);
          } else {
            it->second.push_back(row);
          }
        }
        for (int row_2 = 0; row_2 < static_cast<int>(map[table_2].data.size());
             row_2++) {
          auto it = middle.find(map[table_2].data[row_2][value_2]);
          if (it != middle.end()) {
            count_2 += it->second.size();
          }
        }
      } else {
        for (int i = 0; i < static_cast<int>(down.size()); i++) {
          cout << up[i] << " ";
        }
        cout << "\n";
        unordered_map<TableEntry, vector<size_t>> middle_2;
        for (int row_2 = 0; row_2 < static_cast<int>(map[table_2].data.size());
             row_2++) {
          auto it = middle_2.find(map[table_2].data[row_2][value_2]);
          if (it == middle_2.end()) {
            vector<size_t> value_4(1, row_2);
            middle_2.emplace(map[table_2].data[row_2][value_2], value_4);
          } else {
            it->second.push_back(row_2);
          }
        }
        for (int row_3 = 0; row_3 < static_cast<int>(map[table_1].data.size());
             row_3++) {
          auto it = middle_2.find(map[table_1].data[row_3][value_1]);
          if (it != middle_2.end()) {
            count_2 += it->second.size();
            for (auto s : it->second) {
              for (pair<int, int> p : down) {
```

```
                if (p.first == 1) {
                  cout << (map[table_1].data[row_3])[p.second] << " ";
                } else {
                  cout << (map[table_2].data[s])[p.second] << " ";
                }
              }
              cout << "\n";
            }
          }
        }
      }
      cout << "Printed " << count_2 << " rows from joining " << table_1
           << " to " << table_2 << "\n";
    } else {
      if (!map.count(table_1)) {
        cout << "Error during JOIN: " << table_1
             << " does not name a table in the database "
             << "\n";
        getline(cin, junk);
      }
      if (!map.count(table_2)) {
        cout << "Error during JOIN: " << table_2
             << " does not name a table in the database "
             << "\n";
        getline(cin, junk);
      }
    }
  }

  void INSERT() { //地毯
    string INTO;
    string tablename;
    int N;
    string junk;
    cin >> INTO >> tablename >> N;
    if (map.count(tablename)) {
      cin >> junk;
      int end = map[tablename].insert(N);
      int start = end - N;
      int final = end - 1;
      cout << "Added " << N << " rows to " << tablename << " from position "
           << start << " to " << final << "\n";
    } else {
      cout << "Error during INSERT: " << tablename
           << " does not name a table in the database "
           << "\n";
      getline(cin, junk);
    }
  }

  void DELETE() { // 地毯
    string FROM;
    string tablename;
    cin >> FROM >> tablename;
```

```

    auto it = map.find(tablename);
    if (it == map.end()) {
      cout << "Error udring DELETE: " << tablename
           << " does not name a table in the database "
           << "\n";
      getline(cin, tablename);
    } else {
      int N = it->second.delete_row(tablename);
      if (N == -1) {
        getline(cin, tablename);
      } else {
        cout << "Deleted " << N << " rows from " << tablename << "\n";
      }
    }
  }

  void GENERATE() { // 地毯
    string FOR;
    string tablename;
    string indextype;
    string col_name;
    cin >> FOR >> tablename;
    auto it = map.find(tablename);
    if (it == map.end()) {
      cout << "Error during GENERATE: " << tablename
           << " does not name a table in the database "
           << "\n";
      getline(cin, tablename);
    } else {
      cin >> indextype;
      if (indextype == "hash") {
        cin >> col_name >> col_name;
        col_name = map[tablename].generate_hash(tablename);

      } else {
        cin >> col_name >> col_name;
        col_name = map[tablename].generate_bst(tablename);
      }
      if (col_name != "ERROR") {
        cout << "Created " << indextype << " index for table " << tablename
             << " on column " << col_name << "\n";
      }
    }
  }

  void shell() { // 地毯
    string cmd;

    cout << "%"
         << " ";
    while (cin >> cmd) {
      if (cmd[0] == '#') {
        getline(cin, cmd);
```

```
      } else if (cmd == "QUIT") {
        QUIT();
        break;
      } else if (cmd == "CREATE") {
        CREATE();
      } else if (cmd == "REMOVE") {
        REMOVE();
      } else if (cmd == "INSERT") {
        INSERT();

      } else if (cmd == "PRINT") {
        PRINT();
      } else if (cmd == "DELETE") {
        DELETE();
      } else if (cmd == "JOIN") {
        JOIN();
      } else if (cmd == "GENERATE") {
        GENERATE();
      } else {
        cout << "Error: unrecognized command\n";
        getline(cin, cmd);
      }

      cout << "%"
           << " ";
    }
  }

  void getMode(int argc, char *argv[]) { //地毯

    string mode;
    opterr = false;
    int choice;
    int option_index = 0;

    option long_options[] = {
        {"quiet", no_argument, nullptr, 'q'},
        {"help", no_argument, nullptr, 'h'},
        {nullptr, 0, nullptr, '\0'},
    };

    while ((choice = getopt_long(argc, argv, "qh", long_options,
                                 &option_index)) != -1) {
      switch (choice) {
      case 'h':
        cout << "somthing"
             << "\n";
        exit(1);
        break;

      case 'q':
        quiet = true;
        break;

```

```
      default:
        cerr << "Unknown command line option" << '\n';
        exit(1);
      }
    }
    shell();
  }
};

int main(int argc, char **argv) { //地毯
  Silly s_1;
  ios_base::sync_with_stdio(false);
  cin >> boolalpha;
  cout << boolalpha;

  s_1.getMode(argc, argv);

  return 0;
}
```

Give feedback