API Specification

See database.h, csce-315-p1.lib, and API Specification in folder.

How to Write API calls

Your API calls should be contained in a try/catch block to catch any errors you have in your API calls, as seen below.

Exceptions Thrown

Whenever there is an error with an API call, the API throws the DatabaseError class. This thrown object contains a specific error message, available with its what() function. The errors should be caught using a try/catch block as shown:

```
main () {
    try {
        // Add API Calls Here
    }
    catch (const DatabaseError& e) {
        cerr << "DatabaseError: " << e.what() << endl;
    }
}
```

The API call syntax is explained in sections Database Operations, Table Operations, Record Operations, and Query Operations.

Database Operations

`Database db;` constructs empty database.

`db.addTable(Table& t, const string& name);` adds Table t with name into the Database db.

`db.dropTable(const string& tableName);` drops table with table name from

the Database db; returns error message if table doesn't exist.

`vector<string> vs = db.listTables();` returns a list of all Table names vs in the Database db.

`vector<string> vs = db.listTables();` returns a vector of strings vs containing the names of all the tables in the Database db.

`vector<Table> vt = db.getTables();` returns a vector of all the Tables vt in the Database db.

`Table t = db.query("select", "from", "where");` runs a query on the Database db and returns a Table t. See Query Operations for usage information.

`Table t = db[i];` takes an integer i and copies the i-th table in Database db to Table t.

`db[i] = t;` takes an integer i and assigns a Database t to the i-th table in Database db.

`cout << db;` prints the contents of Database db.

Table Operations

`Table t;` constructs empty table.

`t.addAttribute(string attribute);` adds a single attribute to the Table t and its column.

`t.deleteAttribute(string attribute);` removes a single attribute from the Table t and its column.

`t.add(Record r);` adds a Record r to the Table t.

`t.getAttributes();` returns a list of the attributes within the Table t.

`int n = v.size();` returns the number of records n in the Table t.

`vector<Record>::iterator it = t.getIteratorIndex(int i);` gets iterator to iterate through records in Table t, starting at record i.

`t.specifyKey(string key);` sets a string name to a key only once, for Table t.

`Table t2 = t.crossJoin(const Table& t1);` takes 2 tables as an input and produces 1 as output.

`Table t2 = t.naturalJoin(const Table &t1);` takes 2 tables as an input and one table is produced as the output.

`t.count(string attribute);` returns a count for an attribute in Table t.

`t.min(string attribute);` returns the minimum value for a specified attribute in Table t.

`t.max(string attribute);` returns the maximum value for a specified attribute in Table t.

`Record r = t[i];` takes an integer i and copies the i-th record in Table t to Record r.

`t[i] = r;` takes an integer i and assigns a Record r to the i-th record in Table t.

`cout << t;` prints the contents of Table t.

- Restriction: Attribute names cannot contain spaces, for more info see Query Operations Restrictions.

Record Operations

`Record r;` creates a Record r with no entries.

`Record r(i);` takes an integer i and creates a Record r with i entries initialized to NULL.

`Record r(v);` or `Record r({"string1", "string2", "string3"});` takes an vector of strings v and creates a Record r with the specified attribute values. Cannot initialize a value to NULL with this method.

`r = v;` or `r = {"string1", "string2", "string3"};` assigns the specified attribute values in the vector of strings v to a Record r. Cannot initialize a value to NULL with this method.

`int n = r.size();` outputs an integer n for the number of entries in the Record r.

`r[i] = "string";` takes an integer i and sets the i-th entry in the Record r to the specified string.

- Restriction: Attribute values cannot contain the character `'`, for more info see Query Operations Restrictions.

`string s = r[i];` takes an integer i and sets a string to the i-th entry in the Record r.

`Record r1 = r;` copies a Record r to r1.

`r1 = r;` assigns a Record r to r1.

`cout << r;` prints entries in a Record r, tab separated.

`r[i] = NULL;` or `r[i] = nullptr;` takes an integer i and sets the i-th entry in the Record r to NULL and deallocates any previous memory.

Query Operations

`Table t = db.query("select", "from", "where");` runs a query on the Database db and returns a Table t.

Argument inputs:

1. `"select"` string contains a space-separated list of attribute names to keep in the outputted Table.
    - Format: `"attribute_name_1 attribute_name_2 attribute_name_3"`
    - Example: `"isbn title author"`
2. `"from"` string contains a single Table name to select from.
    - Format: `"table_name"`
    - Example: `"books"`
3. `"where"` string contains string comparisons between attribute names and attribute values.
    - Comparisons: The available attribute name and attribute value comparison operators are =, <>, >, <, >=, <=.
        - Format: `"attribute_name operator 'attribute value'"`
        - Example: `"isbn = '1928223578'"`
    - Combining Comparisons: The operators AND, OR, and NOT may be used along with nested parentheses ().
        - Example Format: `"attribute_name_1 <> 'attribute value 1' OR (attribute_name_2 >= 'attribute value 2' AND attribute_name_3 >= 'attribute value 3')"`

- Restrictions: Attribute names cannot contain spaces, and attribute values cannot contain the character `'` because of the format of the Database query function WHERE query comparison: `"attribute_name = 'attribute value'"`.