CS 211 Assignment 1
SimpleDB

Jay Kania
Fall 2023

Introduction

The SimpleDB project aims to create a rudimentary Database Management System (DBMS) implemented in the C programming language. This project provides a command-line interface (CLI) that allows users to interact with a basic database, performing essential operations such as setup, insert, delete, modify, get, and exit. The primary goal of SimpleDB is to provide a foundation for understanding the fundamental principles of data management in C.

Your program will be required to store three different types of data: Programming Languages, Operating Systems, and Databases. Each table will store various data types and differing values. The user will be able to declare the table size, insert data, delete data, modify existing records, and get all records via a set of commands. Your software should respond to these commands, perform the intended action, and provide the proper output back to the user.

Logistics

Your program must not utilize functions from any external C library. In other words, you must code all of the logic as per the standards set forth below. The libraries included in the skeleton code should be all that is required to complete a successful implementation.

You are able to create helper functions in order to complete the necessary implementations. However, ensure that the provided function signatures are the functions used for execution.

The skeleton code will provide you with an interactive shell and a means to parse the input from the user.

Tables

Your SimpleDB should be able to store three tables of data: Programing Languages, Operating Systems, and Databases. Each category should have a unique typedef struct that identifies its schema. When we reference "table," we are referencing the array of structs declared in memory. Each table should be defined as a single array in your C code stored in memory. When we reference "record," we are referencing the individual struct that is stored inside the array.

programmingLanguages

id: int
language: dynamically allocated string
year: int
creator: dynamically allocated string
paradigm: dynamically allocated string
popularityIndex: double

operatingSystems

id: int
name: dynamically allocated string
year: int
developer: dynamically allocated string
kernelType: dynamically allocated string

databases

id: int
name: dynamically allocated string
year: int
type: dynamically allocated string
developer: dynamically allocated string

There will be one additional field that is maintained internally (not shown to user), that is discussed in the DELETE passage

Table Metadata

In order for your code to properly function, you must maintain a separate structure that keeps track of various fields pertinent to each table. At a minimum, you should maintain the following fields:

- count: int
    - This integer will store the current count of records in a particular table. This is useful as when you traverse the array (table) of structs (records), you must know how many elements are currently in the array (table).
    - This should be initialized in initializeMetadata().

    - This should be updated as records are inserted.
- nextIndex: int
    - This integer will store the index of the next available slot in the array (table). This is important because as you insert, you must know what index you can insert into.
    - This should be initialized in initializeMetadata()
    - This should be updated as records are inserted.
- maxCount: int
    - This integer will store the maximum number of records (instances of structs) that can be stored in a table (array). This is important because as you insert, your program should know if there is enough space to insert.
    - This should be initialized during setup.

Commands

The user will have access to the following commands to interact with SimpleDB. The table names correspond to the camel-cased table names in section "Table."

- Setup
    - **setup {table} {numRows}**
    - Setup will allocate enough memory for {numRows} in {table}.
    - This command is to be executed before performing any other operations.
- Insert
    - **insert {table} {data}**
    - Insert will allow you to add {data} to {table} following {table}'s schema.
    - The order of the data must match the order of attributes as defined in "Tables"
        - E.g., for table databases: {id} {name} {year} {type} {developer}
    - The user does not provide any data for internally maintained fields (see "Delete")
- Delete
    - **delete {table} {id}**
    - Delete will allow you to soft delete **all** records that match {id} in {table}
- Modify
    - **modify {table} {id} {data}**
    - Modify will allow you to modify **all** records that match {id} in {table}.
    - It will overwrite all of the data points stored in the record
    - The data input will be the same as insert. See "Insert" and the examples below.
- Get
    - **get {table}**
    - Get will allow you to retrieve all non-deleted records from "table"
    - It will print in CSV format, where values that contain spaces are wrapped in quotes. See examples below.

        - E.g., Dennis Ritchie -> “Dennis Ritchie”
- Exit
    - **exit**
    - Exit will free all allocated memory and gracefully terminate the program.

void setup(char* table, int numRows)

```
CS211> setup {table}
```

The setup function is responsible for allocating enough memory to store *numRows* structs of data in a particular array. The function takes in the table name, and the number of rows to allocate as a parameter. You should use the name of the table to determine which table to allocate this space for. Additionally, you should update the table’s metadata accordingly.

You can assume that a setup command for each table will be executed before any other commands are executed (e.g., you will always have allocated memory before the user attempts to access/modify the database).

void insert(char** args)

```
CS211> insert {table} {data}
```

The insert function is responsible for inserting a row (instances of struct) into a particular table (array). As input, it will take the user’s command as an array of strings. It should first use the table name to determine which array the data is to be inserted. It should then check if the array has enough capacity to store this new entry (metadata!). If there is insufficient space, print “cannot insert due to insufficient capacity.\n”

Finally, it should use relevant information in the metadata to determine where to put the user’s data. Your insert function should insert to the next available index – the order of the records should be oldest to newest. To wrap up, you should also update the corresponding metadata fields.

You can assume that all of the data elements required for a particular table will be present in the command and will be in the correct order.

void delete(char* table, int id)

```
CS211> delete {table} {id}
```

The delete function is responsible for marking rows as “deleted.” This is known as a “soft-delete,” wherein the data remains stored, but has an associated flag to control how to handle soft-deleted records. **Your table structures should contain an internal field to the effect of “*isDeleted*.”** This field should not be displayed anywhere, and should only be maintained by the DBMS to control what records are displayed when the user retrieves all of the records. **You should not deallocate the data, but instead update the *isDeleted* field: this is known as a “soft-delete.”**

You should “delete” all records that match the ID passed in.

void modify(char** args)

```
CS211> modify {table} {id} {data}
```

The modify function is responsible for modifying existing row(s) (instances of structs) of a particular table (array). As input, it will take the user’s command as an array of strings. It should first use the table name to determine which array the data is to be inserted. It should then do a full replacement of the data, and update the internal fields.

NOTE: a “char*” is a pointer to dynamically allocated memory. Thus, the variable stores a memory location. Before you overwrite the pointer, you should take proper steps to deallocate the previous pointer before you allocate a new pointer.

You can assume that all of the data elements required for a particular table will be present in the command and will be in the correct order.

You should modify all records that match the ID passed in.

void get(char* table)

```
CS211> get {table}
```

The get function is responsible for fetching all of the **non-deleted** records in a particular table. As input, it will take the name of the table as a string. The function should then return a comma-separated list of the headers (exactly as depicted in the Tables section, case-sensitive), followed by the comma-separated list of the values. The headers and records should all be separated by new lines (see examples below).

NOTE: To print a float, you must use “%lf”

void exitProgram()

```
CS211> exit
```

The exit function is responsible for freeing all of the memory allocated by the program and terminating the process. It is important to note that you need to free any memory allocated inside of the struct before you free the struct pointer.

Getting Started

You should first download and expand the provided files:

```
tar xvf a1-provided.tar
```

You can also download these locally and then copy them to ilab using scp: **scp a1-provided.tar yourNetID@kill.cs.rutgers.edu:~/cs211** (assuming you have a cs211 directory in your root directory). The SCP command must be initiated on your local terminal/command prompt.

NOTE: ALL COMPILATION, TESTING, AND EXECUTION MUST BE PERFORMED ON THE ILAB. This is compulsory to ensure a standardized testing and grading environment for all students.

To compile the program, you should use gcc:

```
gcc a1.c -fsanitize=address -o a1
```

-fsanitize=address enables AddressSanitizer, a memory error detector that helps identify and diagnose memory-related issues like buffer overflows, and use-after-free errors. It is good practice to compile your code with this.

And to run it you should:

```
./a1
```

Submission

ONLY SUBMIT a1.c to Canvas. It is important that you only upload the completed C file. We will not accept a tar, zip, etc.

BE SURE THAT YOUR OUTPUTS MATCH EXACTLY AS THE EXECUTION SAMPLES.

Execution Samples

```
CS211> setup programmingLanguages 10
setup complete
CS211> insert programmingLanguages 1 C 1972 "Dennis Ritchie"
Procedural 8.2
insert complete
CS211> get programmingLanguages
id,language,year,creator,paradigm,popularityIndex
1,C,1972,"Dennis Ritchie",Procedural,8.200000
CS211> insert programmingLanguages 2 Python 1991 "Guido van Rossum"
"Multi-paradigm" 9.5
insert complete
CS211> insert programmingLanguages 3 Java 1995 "James Gosling"
"Object-Oriented" 7.9
insert complete
CS211> insert programmingLanguages 4 JavaScript 1995 "Brendan Eich"
"Multi-paradigm" 8.7
insert complete
CS211> insert programmingLanguages 5 Ruby 1995 "Yukihiro Matsumoto"
"Multi-paradigm" 7.1
insert complete
CS211> get programmingLanguages
id,language,year,creator,paradigm,popularityIndex
1,C,1972,"Dennis Ritchie",Procedural,8.200000
2,Python,1991,"Guido van Rossum","Multi-paradigm",9.500000
3,Java,1995,"James Gosling","Object-Oriented",7.900000
4,JavaScript,1995,"Brendan Eich","Multi-paradigm",8.700000
5,Ruby,1995,"Yukihiro Matsumoto","Multi-paradigm",7.100000
CS211> modify programmingLanguages 3 3 Java 1995 "AJ DiLeo"
"Object-Oriented" 8.7
modify complete
CS211> get programmingLanguages
id,language,year,creator,paradigm,popularityIndex
```

```
1,C,1972,"Dennis Ritchie",Procedural,8.200000
2,Python,1991,"Guido van Rossum","Multi-paradigm",9.500000
3,Java,1995,"AJ DiLeo","Object-Oriented",8.700000
4,JavaScript,1995,"Brendan Eich","Multi-paradigm",8.700000
5,Ruby,1995,"Yukihiro Matsumoto","Multi-paradigm",7.100000
```

```
CS211> setup operatingSystems 5
setup complete
CS211> insert operatingSystems 1 Linux 1991 "Linus Torvalds"
Monolithic
insert complete
CS211> insert operatingSystems 2 Windows 1985 "Microsoft" Hybrid
insert complete
CS211> insert operatingSystems 3 macOS 2001 "Apple" Microkernel
insert complete
CS211> insert operatingSystems 4 FreeBSD 1993 "Various" Monolithic
insert complete
CS211> insert operatingSystems 5 Android 2008 "Google" Monolithic
insert complete
CS211> insert operatingSystems 6 "Windows Server 2019" 2018
"Microsoft" "Hybrid"
cannot insert due to insufficient capacity.
```

```
CS211> setup operatingSystems 5
setup complete
CS211> insert operatingSystems 1 Linux 1991 "Linus Torvalds"
Monolithic
insert complete
CS211> insert operatingSystems 2 Windows 1985 "Microsoft" Hybrid
insert complete
CS211> insert operatingSystems 3 macOS 2001 "Apple" Microkernel
insert complete
CS211> insert operatingSystems 4 FreeBSD 1993 "Various" Monolithic
insert complete
CS211> insert operatingSystems 5 Android 2008 "Google" Monolithic
insert complete
CS211> get operatingSystems
id,name,year,developer,kernelType
1,Linux,1991,"Linus Torvalds",Monolithic
2,Windows,1985,"Microsoft",Hybrid
3,macOS,2001,"Apple",Microkernel
4,FreeBSD,1993,"Various",Monolithic
5,Android,2008,"Google",Monolithic
CS211> delete operatingSystems 3
delete complete
CS211> get operatingSystems
id,name,year,developer,kernelType
1,Linux,1991,"Linus Torvalds",Monolithic
2,Windows,1985,"Microsoft",Hybrid
4,FreeBSD,1993,"Various",Monolithic
5,Android,2008,"Google",Monolithic
CS211> delete operatingSystems 1
delete complete
CS211> delete operatingSystems 2
delete complete
CS211> delete operatingSystems 4
delete complete
CS211> delete operatingSystems 5
delete complete
CS211> get operatingSystems
id,name,year,developer,kernelType
```

```
CS211> setup databases 5
setup complete
CS211> insert databases 1 MySQL 1995 Relational "Oracle Corporation"
insert complete
CS211> insert databases 2 MongoDB 2009 NoSQL "MongoDB, Inc."
insert complete
CS211> insert databases 2 PostgreSQL 1989 Relational "PostgreSQL
Global Development Group"
insert complete
CS211> insert databases 4 SQLite 2000 Embedded "SQLite Development
Team"
insert complete
CS211> insert databases 5 Cassandra 2008 NoSQL "Apache Software
Foundation"
insert complete
CS211> get databases
id,name,year,type,developer
1,MySQL,1995,Relational,"Oracle Corporation"
2,MongoDB,2009,NoSQL,"MongoDB, Inc."
2,PostgreSQL,1989,Relational,"PostgreSQL Global Development Group"
4,SQLite,2000,Embedded,"SQLite Development Team"
5,Cassandra,2008,NoSQL,"Apache Software Foundation"
CS211> delete databases 2
delete complete
CS211> get databases
id,name,year,type,developer
1,MySQL,1995,Relational,"Oracle Corporation"
4,SQLite,2000,Embedded,"SQLite Development Team"
5,Cassandra,2008,NoSQL,"Apache Software Foundation"
CS211> insert databases 2 MongoDB 2009 NoSQL "MongoDB, Inc."
cannot insert due to insufficient capacity.
```

```
CS211> setup databases 5
setup complete
CS211> insert databases 1 MySQL 1995 Relational "Oracle Corporation"
insert complete
CS211> insert databases 2 MongoDB 2009 NoSQL "MongoDB, Inc."
insert complete
CS211> insert databases 2 PostgreSQL 1989 Relational "PostgreSQL
Global Development Group"
insert complete
CS211> insert databases 4 SQLite 2000 Embedded "SQLite Development
Team"
insert complete
CS211> insert databases 5 Cassandra 2008 NoSQL "Apache Software
Foundation"
insert complete
CS211> modify databases 2 3 SimpleDB 2023 Custom "Rutgers CS211"
modify complete
CS211> get databases
id,name,year,type,developer
1,MySQL,1995,Relational,"Oracle Corporation"
3,SimpleDB,2023,Custom,"Rutgers CS211"
3,SimpleDB,2023,Custom,"Rutgers CS211"
4,SQLite,2000,Embedded,"SQLite Development Team"
5,Cassandra,2008,NoSQL,"Apache Software Foundation"
CS211> exit
```