Maksym Turkot
11/10/22

Database Engine with Internal Schema Focus

Introduction

This project implements a simple database engine that focuses on low-level data storage. Specifically, it follows the internal schema architecture of a typical database by creating, indexing, and modifying memory blocks on the disk. The system maintains a primary data block that resembles the conceptual schema of the database: it holds attribute names and data types, as well as pointers to memory blocks (tables) with the data. The program also generates a system-dump, showing the conceptual schema, data tables, as well as memory locations of the data and table pointers.

This report includes the following sections:

Software Description

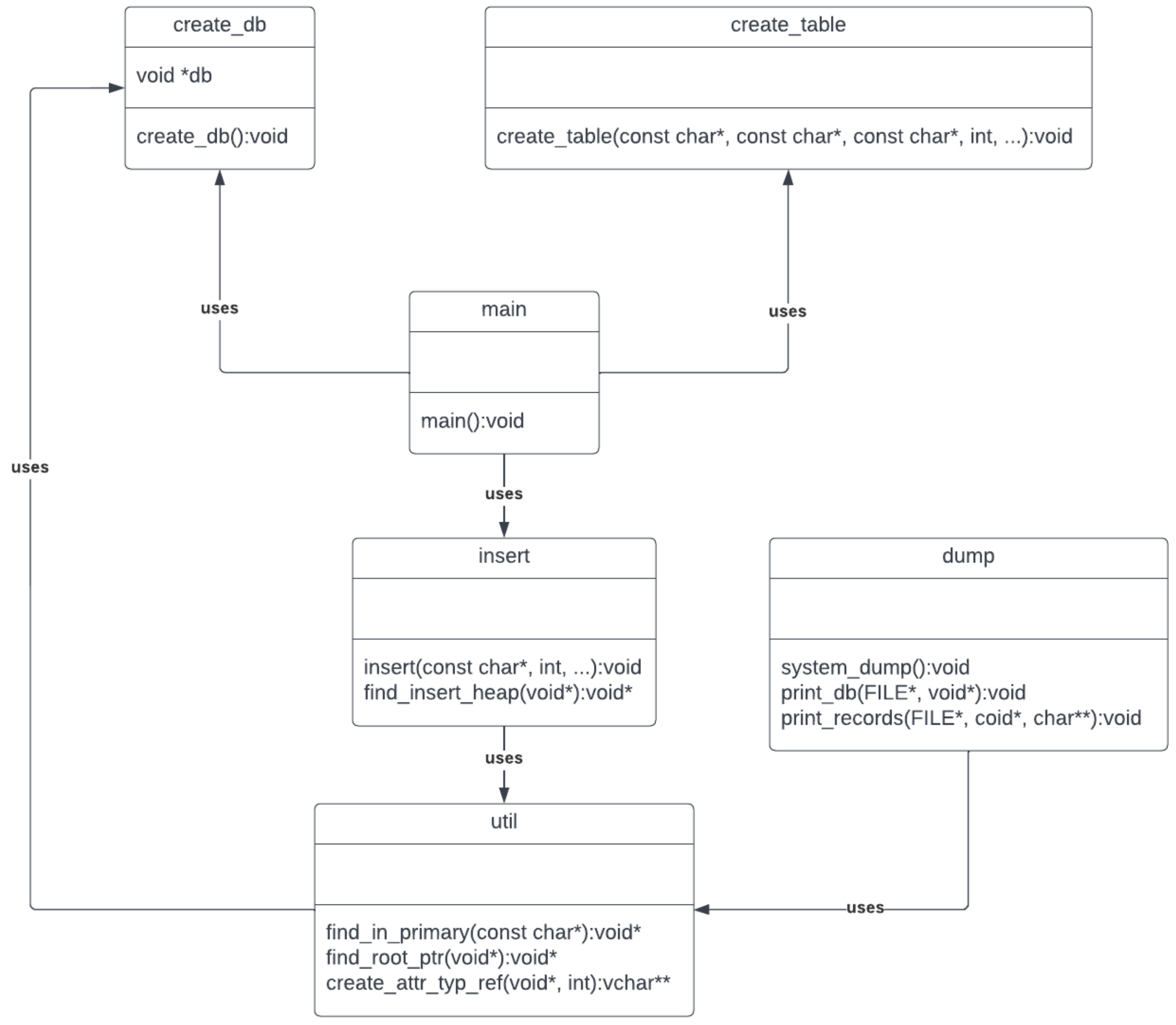
main

Runs a simple routine of the program.

main()
Creates a memory block, creates two tables, inserts a record. The complete functionality is executed during testing.

create_db

Creates a database.


create_db
void *db
create_db():void
create_table
create_table(const char*, const char*, const char*, int, ...):void
uses
uses
main
main():void
uses
uses
insert
insert(const char*, int, ...):void
find_insert_heap(void*):void*
dump
system_dump():void
print_db(FILE*, void*):void
print_records(FILE*, coid*, char**):void
uses
util
find_in_primary(const char*):void*
find_root_ptr(void*):void*
create_attr_typ_ref(void*, int):vchar**
uses

File Organization Diagram

create_db()
Allocates a memory block on the heap and returns a pointer to it that is stored in the global memory for reference throughout the program.

---

create_table

Creates a table and places its schema into the primary database table.

create_table()
Creates an entry in the db_primary table that stores the schema of the database. Attribute names are stored to the db_primary memory block, and a new table is created through allocation of memory on the heap. Pointer to that table is stored at the end of the schema table record.

insert

Inserts a record into the database table.

insert()
Finds the table that matches the passed name and inserts the passed values into it. Checks against the schema what type the next value should be.

find_insert_heap()
Looks for a location in the heap (unordered) organization to insert a record.

util

Provides utility functions commonly used by other files for data manipulation.

find_in_primary()
Finds the schema for the desired relation by matching the table name. Returns the pointer to the schema in the db_primary.

find_root_ptr()
Finds the pointer to the root block for a relation within the schema record.
create_attr_typ_ref()
Creates a string array holding types of the attributes in a table using the schema. Used for insertion of records by the insert() function.

dump

Generates a system dump. Prints schema information for each table, along with its contents and provides memory addresses for each schema and record.

system_dump()
Dumps the information stored in the database to a file.

print_db()
Prints the information about the database contained within the schema.

print_records()
Prints each record; outputs values for each attribute within a table for all records.

User Manual

Building the program

The program was implemented using C programming language and is built using CMake. To build the program, the user should enter the `build` folder and type:

```
cmake -DCMAKE_BUILD_TYPE=Debug ..
```

Running the program

The user may then use the makefile in the root folder with the following commands:

make all: compiles the program

make run: runs the main program, provides no output.

make test: runs the testing suite and performs a system dump to the file in the data folder.

make debug: runs the lldb debugger, was used during development.

make debug-tst: runs the debugger on google tests.

make clean: removes executable files.

Algorithms and Implementation

Create Table

This algorithm works with the pointer to db_primary which stores schemas for all tables. It creates a schema entry that holds information about the table, its organization, attributes, as well as a pointer to the table itself.

Db_primary is always terminated with a '#' character indicating the end of a file. Create Table looks for this character and sets its pointer at its position to insert a new schema record. It then stores table name, type of organization ("heap", "ordered", or "hash"), primary key, and pairs of attribute names and their types that were passed to the function. Once done, it inserts a terminating character '&' indicating the end of attributes passed by the user. It then allocates a memory block to store information, therefore creating a new table. This pointer is converted to an unsigned long and stored to the schema. This conversion is an extra step but it makes insertion and retrieval straightforward.

Schema record is terminated with a '$', and file is terminated with the '#'.

t	e	s	t	\0	i	d	\0	i	d	\0	i	n	t	\0	&	\x	\x	\x	\x	\x	\x	\x	\x	$	#

Organization of a schema entry in db_primary. \x indicates byte with address.

Insert

Algorithm for insert is implemented to store tables with unordered (heap) organization. First an appropriate schema is found in the db_primary with the matching name using the find_db_primary() function. If such a schema wasn't found, an error message is displayed.

Then, find_root_ptr() function finds the pointer to the root of the table within the schema record. This is done by finding the '&' terminator and converting the address of the root from the stored unsigned long to the original void pointer.

Using the schema, an array of strings is constructed storing the types of attributes within the table. This is done by reading attributes from the schema record while skipping attribute names, until the attribute-terminating character ‘&’ is reached. This array is used for reference when inserting new records to determine the appropriate type for each attribute value. Supported types include variable length string without limit, integer, smallint, and a real value.

At this point a location to store the new record is located using the appropriate function depending on the table organization. Find_insert_heap() does this for the heap organization by locating the terminating character of the file ‘#’ and appending a new record to the end of the memory block.

Now the values for a new record are ready to be inserted. Function iterates through the passed attributes, and for each one references the attr_typ_ref string array to determine what type is the next value for proper insertion. The function determines what pointer type to use for the attribute, casts to it, and makes the insertion. This ensures that the proper amount of space is used, and that the value can later be easily retrieved using this pointer type.

Once all values have been inserted, the record is terminated with a ‘$’, and the file is terminated with the ‘#’ terminating character.

System Dump

This algorithm allows the information from the entire database to be retrieved and stored to a file.

The db_primary is iterated and each schema information is printed out. Algorithm retrieves each strings and stores it to a file until the attribute-terminating character ‘&’ is reached. A string array is then constructed holding information about attribute types stored within the table. At this point it outputs the address of the root block, and proceeds to print the information stored within the table.

To do this, print_records() function iterates through the attributes using the string array holding data type information, and retrieves values by casting the void pointer to the appropriate data type.

This process is repeated for each schema, table, and record. For each table or record, memory address of that entity is printed.

Verification Plan

The backbone of the verification strategy are the unit tests implemented with Googletest suite. All tests are stored within the tests.cpp file in the **tst** folder, and are compiled separately from the main program.

These unit tests check all functions in the program to make sure it functions as expected. Tests create new tables, make some insertions and directly retrieve information from the computer’s memory. They then check if the relevant memory location stores correct piece of information, whether it’s a character, a number, or a terminating symbol.

They include:

TEST(create_table_test, create_table): Tests the create_table function of the create_table file. Creates tables and checks that selected characters are present in correct memory locations.

TEST(utils_test, find_in_primary): Tests the find_in_primary function of the utils file. Creates tables in db_primary and checks if pointers to the schema entries for a given table name are returned.

TEST(utils_test, create_attr_typ_ref): Tests the create_attr_typ_ref function of the utils file. Creates tables in db_primary and checks if attribute reference array is created with correct datatype names.

TEST(insert_test, insert): Tests the insert function of the insert file. Creates tables in db_primary, inserts an entry into the correct table and checks if its values are present in that memory block.

TEST(utils_test, system_dump): Dumps the system contents to a file. Creates tables in db_primary, inserts an entry into the tables and dumps the system to a file.

To check if the system functions and stores reasonable amounts of information, one test performs a complete system dump. It creates multiple tables, stores entries to them, and then dumps information to a file called system_dump.txt, stored in the **data** folder. This file shows the schema for each table along with all of the records. Memory locations are also provided which serves as an additional method to check if program runs correctly.