School of Computer Science

COMP30640

Project
Database management system

Teaching Assistant:	Thomas Laurent
Coordinator:	Anthony Ventresque
Date:	Friday 2nd November, 2018
Total Number of Pages:	8

General Instructions.

- You are encouraged to collaborate with your peers on this project, but all written work must be your own. In particular we expect you to be able to explain every aspect of your solution if asked.
- We ask you to hand in an archive (zip or tar.gz) of your solution: code/scripts, README.txt file describing how to run your programs, a 5-10 page pdf report of your work (no need to include code in it).
- The report should include the following sections:
  1. a short introduction
  2. a requirement section that answers the question *what* is the system supposed to do
  3. an architecture/design section that answers the question *how* your solution has been designed to address the requirements described in the previous section
  4. a series of sections that describe the different challenges you faced and your solutions. For instance, take one of the scripts, describe the difficulty you faced and your solution. These sections can be short – the objective here is to show how you crafted the solutions with the tools you have learned so far.
  5. a short conclusion
- The project is worth 30% of the total grade for this module. The breakdown of marks for the project will be as follows:
  - Commands: 35%
  - Server: 15%
  - Client(s): 25%
  - Report: 25%
- Due date: 03/12/2018

1 Introduction

In this project you will implement a rudimentary Database Management System (DBMS) server and clients who interact with this DBMS. A DBMS is a piece of software that helps with the definition, management and usage of a database, i.e. a collection of related data. Example real like DBMS are mysql or postgreSQL.

We will now describe the system you will have to build and its different features.

2 The Basic Commands of your Server

The server manages a set of databases. Each database DB, which name is $name, is represent by a directory $name. Examples of $name are "university" or "company". Each database contains tables. A table named $table is represented by a comma separated file $table. Examples of $table are "student" or "employee".

Suppose our DBMS manages two databases "university" and "company". Suppose the "university" database has 2 tables: "student" and "professor", and the "company" database has three tables: "employee", "department" and "manager", with the employee table recording the id and name of employees in the company. In the root directory of the DBMS we would see the following:

```
$> ls
company university
$> ls company
department employee manager
$> less company/employee
id,firstname,surname
1,Armand,Trueman
2,Hippolyte,Kurtzman
3,Nadege,Pruneau
...
$> ls university
professor student
```

The server needs to implement the following operations:

Create database Create a new folder to store table file

Create table Create a new file in a database folder and write the header of the file (the columns of the table)

Insert data Write a tuple in a table file

Select data Display data from a table specified by a query

2.1 Create database

First write a script `create_database.sh` that takes 1 parameter, $database, and creates the directory for the database $database. Your script should differentiate the following cases and print a corresponding message (make sure the exit message is the only thing printed in all your scripts, this will matter later in the project):

- No parameter was provided
- The database already existed
- Everything went well

```
$> ./create_database.sh
Error: no parameter
$> ./create_database.sh company
Error: DB already exists
$> ./create_database.sh newDB
OK: database created
```

2.2 Create table

Now write a script `create_table` that takes 3 parameters, $database $table $columns, and creates the file for the table $table with header $columns in the directory $database. Your script should differentiate the following cases and print a corresponding message:

- Too few or many parameters
- The database does not exist
- The table already exists
- Everything went well

```
$> ./create_table.sh company
Error: parameters problem
$> ./create_table.sh companydfghjk newTable column1,column2,column3
Error: DB does not exist
$> ./create_table.sh company employee id,firstname,surname
Error: table already exists
$> ./create_table.sh newDB newTable olumn1,column2,column3,column4
OK: table created
```

2.3 Insert data

You will now insert data in the databases and tables you can create. This will be done with the `insert.sh $database $table tuple` script. You will first need to check that the database and table exist. If they exist you then need to make sure that the tuple you are trying to insert has the right number of columns (compared to the table's file's header). If all those conditions are met, you append the tuple to the table's file. For each check, print a message if the check failed, and print a message to say all went well if the insertion is performed.

```
$> ./insert.sh company
Error: parameters problem
$> ./insert.sh companydfghjk newTable value1,value2,value3
Error: DB does not exist
$> ./insert.sh company newTable value1,value2,value3
Error: table does not exist
```

```
$> ./insert.sh company employee 42,thomas,laurent,bestTA
Error: number of columns in tuple does not match schema
$> ./insert.sh company employee 42,thomas,laurent
OK: tuple inserted
```

2.4 Select data

Databases are not useful if you can not query the data saved in them, so your DBMS will need a querying system. This will be handled by the `query.sh $database $table columns` script. Your script should check that the database, table, and columns specified exist and if so print "start_result", followed by the specified columns for each tuple, followed by "end_result". If no columns are specified, all columns are printed. As before print a message for each possible error.

```
$> ./select.sh company
Error: parameters problem
$> ./select.sh companydfghjk newTable 1,3
Error: DB does not exist
$> ./select.sh company newTable 1,3
Error: table does not exist
$> ./select.sh company employee 0,3
Error: column does not exist
$> ./select.sh company employee 2,3
start_result
firstname,surname
Armand,Trueman
Hippolyte,Kurtzman
Nadege,Pruneau
... #there should be tuples here not literally an ellipsis
end_result
```

3 The server

Now you will set up the server of your application. As a first step, your server will read commands from the prompt and will execute them. Write a script `server.sh`. This script is composed of an endless loop. Every time the script enters the loop it reads a new command from the prompt. Every request follows the structure `req id [args]`. There are five different types of requests, corresponding to the four scripts you've written so far:

- `create_database $database`: which creates database $database
- `create_table $database $table`: which creates table $table
- `insert $database $table tuple`: which inserts the tuple into table $table of database $database
- `select $database $table tuple`: which displays the columns from table $table of database $database
- `shutdown`: the server exits with a return code of 0

If the request does not have the right format, your script prints an error message: `Error: bad request`. A good programming structure for your script could be the `case` one. Your script will look like that:

```
while true; do
case "$1" in
        create_database)
            # do something
            ;;
        create_table)
            # do something
            ;;
        insert)
            # do something
            ;;
        select)
            # do something
            ;;
        shutdown)
        # do something
        ;;
        *)
            echo "Error: bad request"
            exit 1
done
```

At this point you should probably test that your server works well by sending requests to it and checking that the files are created and modified accordingly. For example run the following scenario (assuming no DB exists) and make sure things looks right after each step:

```
$>./ server
create_database company
OK: database created
create_database company
Error: DB already exists
create_table company employee id,firstname,surname
OK: table created
create_table companyyyyy employee id,firstname,surname
Error: DB does not exist
insert company employee value1,value2,value3
OK: tuple inserted
insert company employee value1,value2,value3,value4
Error: number of columns in tuple does not match schema
select company employee 1,2
start_result
value1,value2
end_result
```

Eventually your server will be used by multiple processes concurrently (the various clients accessing the server). You then need to execute the commands concurrently and in the background.

The problem is that with concurrent execution comes risks of inconsistencies; for instance when two commands accessing the same file runs concurrently. You then need to update all your scripts to avoid those potential inconsistencies. An option is to use a lock: $database.lock, whenever your server's scripts try to access database $database's folder or files. Modify the server script `server.sh` to start the commands in the background and update the four scripts to run concurrently.

4 The Clients

The last component of your system is the client application, which will send requests to the server. You will write a script `client.sh`, which gets only one parameter $id representing a user.

First your script has to check if a parameter has been given and if yes it enters an endless loop. Every time it enters the loop, the script reads a request from the user (requests have the form `req args`). If the request is well formed, then the script prints `req $id args` with $id the $id given as parameter of the script `client.sh`.

Now your system will connect the client(s) and server with named pipes. The server will create a named pipe called `server.pipe` and each client will create a named pipe `$id.pipe` with $id the id given as parameter of the client.sh script.

Modify the scripts `server.sh` and `client.sh` to create those named pipes.

It is likely that so far you've been using control + c to stop the scripts. now we want you to be able to delete the named pipes whenever you want to quit a script – so you need to trap the command control + c. Use the following idea in your programs to do so: trap control + c and then delete the named pipe and then exit with the exit code 0.

```
#!/bin/bash

# trap ctrl-c and call ctrl_c()
trap ctrl_c INT

function ctrl_c() {
        #do something when control c is trapped
}

#rest of the script
```

Also add a more graceful way to quit. Make the client accept the `exit` command to quit, i.e `client.sh` exits with code 0 when receiving the `exit` command. Also make sure the client sends the `shutdown` request to the server correctly, i.e. that the `server.sh` exits with code 0 when the user types "shutdown" into the client. Do not forget to delete your pipes in these cases too.

Now, clients need to send their requests on the server's pipe and, likewise, the server needs to read the requests on its named pipe. Modify both scripts accordingly. We recommend you to open two terminals, one for the client and one for the server in order to test your scripts.

The server now needs to send the replies to the clients' named pipes and not on the terminal. The client script needs to receive what's been written on the client's named pipe. There are two options here:

- the first word is `start_result` then what has to be printed on the terminal is everything until the keyword `end_result`
- the first word is `OK:` or `Error:`, which means that the command executed correctly and you can just print a message such as `command successfully executed`

Figure 1 shows a simple architecture diagram of the communication between processes using named pipes.

Conclusion

This is a complex project – yet every component is made of simple scripts/commands that you are/will be familiar with. As for every large project, do not be scared by the complexity but try to understand the overall picture and start coding small parts that you understand. For instance Section 2 should be ok so start with these scripts and focus only on them – later on you can integrate small pieces together, they should make more sense then.

Good luck!

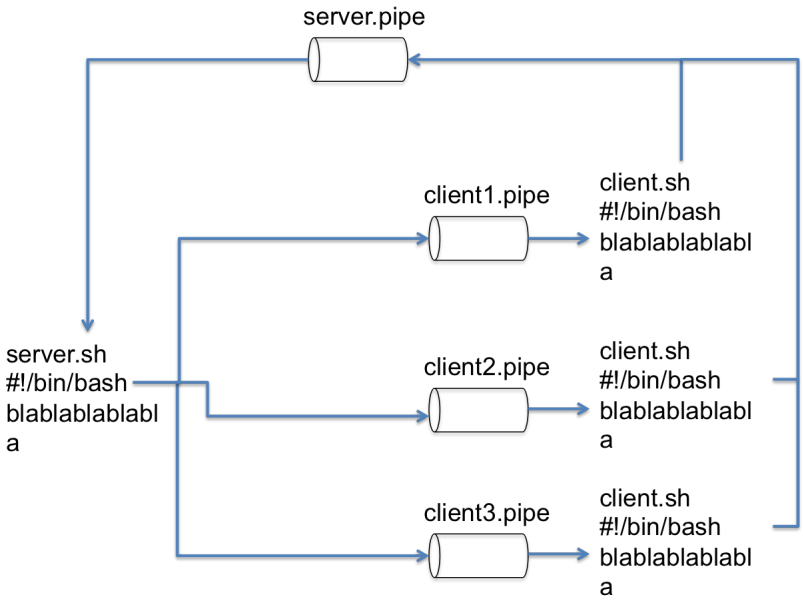

Figure 1: This diagram details the communication architecture of your application.

Bonus

If you finish the project early (or if you got addicted to writing bash scripts ;)), there are up to 10% bonus points available.

To get bonus points, implement an extra feature for our DBMS system and document it in your report: what it does, how to use it, etc. The feature should have some substance to it, printing "Welcome to the client" when starting the client is not a new feature worth extra points.

Examples of features you could implement are:

- Selecting the columns by name instead of index, and outputting them in the same order as the query (e.g. select company employee id,firstname or select company employee firstname,id ; which would have different outputs).
- Adding a "WHERE clause" to the select (e.g. select company employee 1,3 3 Trueman or whatever syntax you think best).