

Project 4: Design & Implementation of a Simple DBMS

CMPE 321, Introduction to Database Systems, Spring 2024

Due: 27.05.2024 23:59 o'clock

1 Introduction

Years after the final battle for the Iron Throne, the Seven Kingdoms are slowly restoring from the war's aftermath. The land of Westeros is beginning to breathe again under the new king's rule. However, the King realizes that years of war have caused much of the kingdom's history, tales of valor, notorious characters, and lineage records to scatter, and in some cases, lost forever.

To prevent further loss and to preserve the realm's history, the King has decided to establish an information system called "Westeros Archive." This archive will store all the crucial details about the Seven Kingdoms – the noble houses, their members, their bannermen, the castles, and more. The King has entrusted this task to the scholars of the Citadel, who will work in pairs to build this important system.

Your task, as the last scholars of the Citadel, is to design and implement the Westeros Archive system that will serve as a crucial tool for preserving the history of Westeros. Remember, winter is coming, and with it, the need to ensure the knowledge of the Seven Kingdoms is safe and accessible for generations to come.

2 First Phase: Design

In this phase, scholars are expected to design the Westeros Archive. The design must include data storage units (files, pages, and records) for storing the data. The system design must support the following operations:

Westeros Definition Language Operations

- Create a type (i.e., relation)

Westeros Manipulation Language Operations

- Create a record
- Delete a record
- Search for a record (by primary key)

The design should include but is not limited to:

- Define page size (i.e., decide the number of records in each page)
- Define file size (i.e., decide the number of pages in each file)
- Define what information to store in your page headers and record headers
- Define the maximum number of fields a type can have (this should not be less than 6)
- Define the maximum length of a type name (Your design should support maximum length of at least 12)
- Define the maximum length of a field name (Your design should support maximum length of at least 20)

Assumptions

- All fields, type, and field names are alphanumeric.
- The user always enters alphanumeric characters inside the test cases.
- You can assume that only int and string field-types will be used.
- The Westeros Archive will have a limit of 10 records per page.
- Note that type refers to a relation.

You can improvise by making further assumptions, as long as they **do not conflict with the base assumptions** and they are explained in the report CLEARLY.

Constraints

- Each type (i.e., relation) should be stored in a separate file. Each file must have multiple pages.
- The data must be organized in pages and pages must contain records. So, you must clearly explain your page and record structure in your report.
- Each page of a file is allowed to contain at most 10 records.
- You will use primary keys of types for the search-keys.
- With every create record and delete record operation, the corresponding page and file should be updated.
- Although a file contains multiple pages, it must read page by page when it is needed (for example: searching). Loading the whole file to RAM is not allowed.

3 Second Phase: Implementation

In this phase, you are expected to implement the Westeros Archive software that you have designed in Phase One. The implementation must support the Westeros Definition Language Operations and the Westeros Manipulation Language Operations.

4 Input/Output

Submissions will be graded automatically for various test cases. You will write your code in Python3. The input file's full path will be given as an argument to the executable program. The command that will be executed after unzipping your submission at the same location, is as follows:

```
python3 2019400XXX/archive.py fullinputfilepath
```

Input file contains a list of operations. Each line corresponds to an operation. The format for each operation is as follows:

4.1 Westeros Definition Language Operations

Table 1: Input and Output Formats of Create Type Operation

Operation	Input Format	Output Format
Create	create type <type-name><number-of-fields><primary-key-order><field1-name><field1-type><field2-name>	.None

4.2 Westeros Manipulation Language Operations

Table 2: Input and Output Formats of Create, Delete and Search Operations For Records

Operation	Input Format	Output Format
Create	create record <type-name><field1-value><field2-value>...	None
Delete	delete record <type-name><primary-key>	None
Search	search record <type-name><primary-key>	<field1-value><field2-value>...

4.3 Sample Input Output and Log Files

```
create type human 6 1 name str origin str title str age int weapon str skill str
create record human RamsayBolton Dreadfort Lord 21 Dagger Strategy
create type dragon 5 1 name str age int color str owner str skill str
create record dragon Viserion 5 White NightKing IceBreathing
create record human Bronn Stokeworth Knight 32 Crossbow Swordfighting
delete record human NedStark
search record human RamsayBolton
search record dragon Viserion
```

Listing 1: Sample Input (input.txt)

RamsayBolton Dreadfort Lord 21 Dagger Strategy
Viserion 5 White NightKing IceBreathing

Listing 2: Sample Output (output.txt)

1635018009,	create	type	human	6	1	name	str	origin	str	title	str	age	int	weapon	str	skill	str,	success
1635018014,	create	record	human	RamsayBolton	Dreadfort	Lord	21	Dagger	Strategy,									success
1635018016,	create	type	dragon	5	1	name	str	age	int	color	str	owner	str	skill	str,			success
1635018018,	create	record	dragon	Viserion	5	White	NightKing	IceBreathing,										success
1635018020,	create	record	human	Bronn	Stokeworth	Knight	32	Crossbow	Swordfighting,									success
1635018022,	delete	record	human	NedStark,														failure
1635018025,	search	record	human	RamsayBolton,														success
1635018027,	search	record	dragon	Viserion,														success

Listing 3: Sample Log File (log.txt)

5 Implementation Instructions

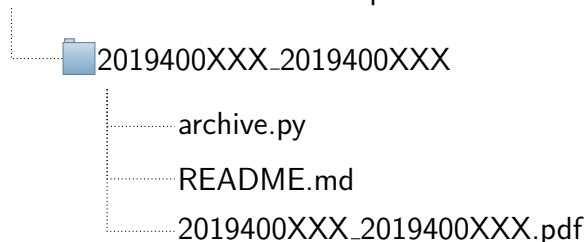
- Any field of a type can be the **primary key** and it is defined with *primary-key-order* in the create type operation. (Look for Table 1)
- Search and deletion of records shall always be done by primary key.
- Test cases are not necessarily independent of each other. So, if the type *castle* is created in test case 1, it should be accessible in test case N.
- The system should log all definition and manipulation operations into a CSV file (namely log.csv), which includes the UNIX time of occurrence, the operation string, and the status of the operation (either success or failure). The file should NOT be present in the submission. It WILL NOT be present at the first runtime.
 - Log file must be persistent and never deleted when the Westeros software is either stopped or restarted.
 - The UNIX times in the sample test cases are set deliberately for you to understand the time flow between instructions and which instruction comes before which. In real life, a user may have different time intervals between instructions since they write the instruction string into the terminal, which also takes time. Your system may also print the same time output for some consecutive fast instructions, which is fine and there's no error in that. As long as you read the input file line by line sequentially, your output should be correct.
 - You can use `import time` and `int(time.time())` to generate UNIX times.
- The system should correctly handle the create type, create record, delete record, and search record operations, and log success or failure for each operation.
- The status of an operation is decided based on whether or not the result of the operation is empty. For the searching operations, if the output of the operation is empty (i.e., if such a record is not found), it must be considered as failure and logged accordingly. The invalid operations for which the status should be failure are listed as follows: (No other case will be asked)

- Creating a type with a name of an existing type (Suppose that type *castle* exists and you want to create the type *castle* again.)
- Creating a record with a primary key of an existing record (Suppose that type *castle* has a record with a primary key *Dragonstone*, and you want to create another record of type *castle* with primary key *Dragonstone*.)
- Deleting a type which does not exist in the database (Suppose that type *castle* does not exist and you want to delete type *castle*.)
- Deleting a record with a primary key which does not belong to any record of that type. (Suppose that type *castle* has no record with a primary key *Dragonstone* and you want to delete the record with the primary key *Dragonstone*.)
- Searching a non-existing record. (Suppose that type *castle* has no record with a primary key *Dragonstone* and you want to search the record with the primary key *Dragonstone*.)
- All of the outputs should be printed to output.txt file NOT TO THE SCREEN (standard output) and NOT TO THE LOG FILE.

6 Submission

- This project can be implemented either individually or as a team of two people.
- The submission must include your report (report.pdf), source code (archive.py), and a README (README.txt or README.md) file that describes how to run your code.
- You **MUST** include all your design decisions and detail your approach in the pdf report.
- Include comments in your code file as it will be graded.
- Your directory must have the structure as shown below:

2019400XXX_2019400XXX.zip



- Name the folder as **StudentID1_StudentID2** if you are working as a group.
- Name the folder as **StudentID1** if you are working as an individual.
- Zip the folder for submission and name the .zip file with the same name given to the folder.
- Submit .zip file through **Moodle** until the deadline.

- Each group should submit **one** .zip file.
- Do not include any other files in your .zip file.
- **Any other submission method is not allowed.**
- 20 points will be deducted in case of non-compliance to **ANY** of the naming and folder conventions explained above.
- Note that your submissions will be inspected for plagiarism with previous years' submissions as well as this year's. Any sign of plagiarism will be penalized.

7 Late Submission Policy

We will accept late submissions, however;

- One day late (even one minute will be considered a day late) would mean -30 points penalty.
- Two days late (even one minute and a day will be considered two days late) would mean -60 points penalty.
- Moodle will close after two days. No other submission method will be accepted.

8 Academic Honesty

Please read carefully the academic honesty part of the syllabus as we give utmost importance to academic honesty.