# Project 4: DBMS Design

**Note:** This project is optional. If you submit it, we will use your top three (out of four) project scores for grading.

**Due**: 09.06.2023 23:59 o'clock - The deadline is sharp! Late submissions will not be graded.

## 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 number of fields a type can have (Your design should support at least 6 fields)
- Define the length of a type name (Your design should support strings of length at least 12)
- Define the length of a field name (Your design should support strings of length 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 60 records per type.
- 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 file must have multiple pages. You are not allowed to store all pages in the same file.
- 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.
- Your design should support 60 records per relation.
- You will use primary keys of types for the search-keys.
- With every create record and delete record operation, the corresponding 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

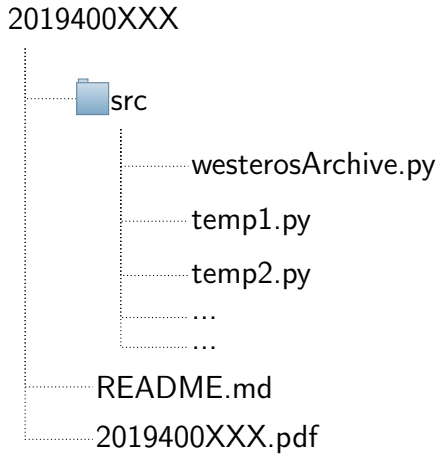
In this phase, you are expected to implement the Westeros Archive software that you have designed in Phase One. The implementation must support the Westeros Definition Language Operations and the Westeros Manipulation Language Operations.

# 4 Input/Output

Submissions will be graded automatically for various test cases. You will write your code in Python3. The input and output file names will be given as arguments to the executable program. The command that will be executed is as follows:

```
python3 2019400XXX/src/westerosArchive.py inputFilePath outputFilePath
```

Your directory must have the structure as shown below and the name of your main file must be westerosArchive.py.




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

**Westeros Definition Language Operations**

Table 1: Input and Output Formats of Operations

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

**Westeros Manipulation Language Operations**

Table 2: Input and Output Formats of Operations

| 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>... |

**Sample Inputs and Outputs**

```
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

```
RamsayBolton Dreadfort Lord 21 Dagger Strategy
Viserion 5 White NightKing IceBreathing
```

Listing 2: Sample Output

```
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

# 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.
- 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 westerosLog.csv), which includes the UNIX time of occurrence, the operation string, and the status of the operation (either success or failure).
  - 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.
- Status of an operation is decided whether or not the result of the operation is empty. For the searching operations, if the output of the operation is empty, it must be considered as failure and logged accordingly. Some invalid operations that their status should be failure can be listed as follows:
  - 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*.)

# 6 Submission

This project can be implemented either individually or as a team of two people. The submission must include your report, source code, .tex file, and a README file that describes how to run your code. If your file size is over the Moodle upload limit, submit a link (Drive, Dropbox, etc.) for downloading your project. Place all the required files into a folder named with the student IDs of the team members separated by an underscore (e.g. 2017400200 2018700120). Zip the folder for submission and name the .zip file with the same name. Submit the .zip file through Moodle until the deadline. Any other submission method and late submissions are not allowed. Each group should submit one .zip file. 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.