CMPE321
INTRODUCTION TO DATABASE SYSTEMS
Spring'22
Project 4: Project Horadrim

Altay Acar - Engin Oğuzhan Şenol
2018400084 - 2020400324

May 27, 2022

Contents

1 Introduction

In this project, we are asked to design a system catalog for storing the metadata, and creating files, pages, records for the actual data in Horadrim. Horadrim includes types such as angel and evil. For each type there are record fields and values for each field. We use the B+ Tree for indexing. By using the B+ Tree, we can reach the files of a type, its pages, and records in the actual data. There are some operations that we defined for reaching the actual data. We can divide them into two categories as Horadrim Definition Language Operations and Horadrim Manipulation Language Operations. While Horadrim Definition Language Operations include "Create a type", "Delete a type", and "List all types", Horadrim Manipulation Language Operations include "Create a record", "Delete a record", "Search for a record", "Update a record", "List all records of a type", and "Filter records".

With creating the actual data, there are some restrictions that we have to obey such as file size, page size, the number of pages in a file, the number of records in a page, length of the type names, length of the field names, and the length of their values. We also have headers for each data storage units. All the terms mentioned above are defined in the report.

Our code is written in Python3, and the main file named as horadrimSoftware.py To execute the HoradrimSoftware.py, one must execute the following command:

python3 2018400XXX/src/horadrimSoftware.py inputFile.txt outputFile.txt

The directory should contain an input file named inputFile.txt with lines of commands, and an output file named outputFile.txt to write the outputs for certain Language Operations. We are also creating a log file named horadrimLog.csv for logging every command in the input file. For each input command, the log file creates a line. Those lines are in the structure of:

timestamp,input command,status

timestamp represents the time that command executed, input command represents the command given in the input file, and status represents is the command executed successfully or not. Below there are examples of both success and failure cases:

1653607472,list type,failure

1653607473,create type angel 3 1 name str alias str affiliation str,success

Since we determined our storage structures for our files at first, it is sensible to begin clarifying our storage structures at first.

2 Storage Structures

The data consists of types and each type refers to a relation. Each type is stored in a file and those files are organized in pages that contain records as the lowest level in the hierarchical data storage structure of the project. So, each entry to the database is a record in an appropriate page of the appropriate file of the given type

2.1 File Format

Each file is designed as a txt file that is named after their respective types. Meaning that for each type there will be at least one txt file that will store each record for that type among their pages. For example when a type named angel is created, its file will be named angel-1.txt. In this format, each file will store file header, pages with page headers, and records as separated values with record headers. In the file header, there will be six pivotal fields with separate lines: First line represents file_id in int for storing the file ID (FID) of the file that is used to locate a file, second line represents is_full in bool for showing that does the file have a space for new record, third line represents type-name in str, fourth line represents the primary_key in str for showing the primary key of the type, fifth line represents the field_names as separated values for names of the types, and sixth line represents the field_types as separated values for the types of the fields in the same order as field_names.For example when a type named angel is created and three fields is specified as name, alias, and affiliation, each of them in str, then the corresponding txt file angel-1.txt will have six lines in its file header that can be seen below:

01
0
angel
name
name alias affiliation
str str str

Each file contains every page and their respective records in separate lines and those pages and records are separated and identified by their respective ids. page_id in int for storing the Page ID (PID) of the page that is used to locate a page, record_id in int for storing the record ID (RID) of the record that is used to locate a record, and type in string to identify the type of the recorded data. Meaning each record's record_id will be unique for its page, but each record in the same page will have the same page_id alongside their file_id. Because they reside in the same file. Similarly, each page's page_id will be unique for its file, and when the page_id field in the table changes we can understand that we are on the next page. But each page will have the same file_id, because they are in the same file. So, each entry in the table will have the same file_id. Below there is an example txt file of type angel with four pages, four records in the

first three page, and one record in the last page

```
01
0
angel
name
name                alias               affiliation
str                 str                 str

1
1
1
1
Michael             Manager             DunderMifflin
2
1
Jim                 Sales               DunderMifflin
3
1
Pam                 Sales               DunderMifflin
4
1
Dwight              Sales               DunderMifflin

2
1
1
1
Ryan                Temp                DunderMifflin
2
1
Kevin               Accounting          DunderMifflin
3
1
Kelly               CustomerService     DunderMifflin
4
1
Toby                HumanResources      DunderMifflin

3
1
1
1
Ted                 Architect           HIMYM
2
1
```

Robin NewsPresenter HIMYM
3
1
Barney Please HIMYM
4
1
Marshall Lawyer HIMYM

4
0
1
1
Lily Teacher HIMYM
2
0

3
0

4
0

Since its certain the number of pages in a file and the number of records in pages, in the creation of a file, there will be empty spaces for each record, and their headers will be written. As you can see in the last example, even if the last page is not full, record spaces is created with 20 empty characters for the incoming record, since the allocated memory space is 20 byte for field-vale.

To this extent, each page has a unique identifier of the <file_id>. This identifier will be used for accessing a file throughout the project. For clarification, from the example given above, the composite identifier of the file angel-1.txt would be 01.

Each txt file in the database has a unique <file_id> and this identifier is also mentioned in the file name upon the creation of the file. For example when the first file of the type angel is full, new records would be stored in a new file and its name would be angel-2.txt. Also, when a new type and its txt file is created, its identifier will have the next index that comes after the latest created file. For example if angel-1.txt, angel-2.txt, and angel-3.txt files exist, and a new type of evil is created, its initial file would be named evil-4.txt and so on. Thus, each file can be identified just by looking at their names. It would not require to have a record in the file to identify it. If the identifier of the file was not mentioned in the file name, a file would require at least one entry in it to fill its file_id field and access it by reading.

2.2 Page Format

As explained in detail in the previous section, each page is stored in a txt file and separated by their page_id fields from each other. So, the data of the txt file is parted into pages. Thus, the page_id field in the header of the file acts as the page header and is used to identify and locate a page of a specific file.

In this format, each file will store page header and records as separated values with record headers. In the page header, there will be two pivotal fields with seperate lines: First line represents page_id in int for storing the page ID (PID) of the page that is used to locate the page of its file, and second line represents is_full in bool for showing that does the page have a space for new record.For example when a record with the type angel is created then in the corresponding txt file angel-1.txt will have two lines in its first page header that can be seen below:

1
0

2.3 Record Format

As explained in detail in the File Format section, each record of a page is stored in a txt file and separated by their record_id fields from each other. Thus, the record_id field in the header of the file acts as the record header and is used to identify and locate a record of a specific page of a specific file. In the record header, there will be two pivotal fields with separate lines: First line represents record_id in int for storing the record ID (RID) of the page that is used to locate the record of its page and file, and second line represents is_full in bool for showing that is there a record in that record_id.

3 Assumptions & Constraints

In this project, B+ Tree indexing is used to handle all Horadrim manipulation language operations. For each type that is created, also a B+ Tree for that type would be created, then we crate a file in json format for this B+ Tree named type-tree.json. When we make operations with using the B+ Tree, we are updating our B+ Tree. At the end of creation of B+ Tree , we serialize this B+ Tree object and turn into a dictionary. Then, we write this dictionary to the type-tree.json file. Every execution of the code, B+ Tree reads the json file. Then again makes changes and then again write to finalized version to json file.

For this project, a fixed-length record approach is chosen. Each field has a fixed length and the number of fields is also fixed. As the length of the longest field given in the description, the maximum length of a field value is 20 and the maximum length of a field name is also chosen as 20. A record can have a maximum of 12 fields. Those definitions is listed in a tabular format below:

Feature	Abbreviation	Value
Maximum number of fields	F	12
Maximum number of field name	FN	20
Maximum number of field value	FL	120

Table 1: Field Definitions

3.1 Page Format

Each page contains a collection of records in the above defined format in a txt file. Each page has a page header and 8 records with record headers. Every record place takes up space even there is no record added, each page has the size of fixed. Also, since we are using new line character for a more clean view, we need to take account them either. Related calculations and information is given in the tabular format below:

Feature	Abbreviation	Value
Record header	RH	4
Page header	PH	4
Page size	PS	1964
Maximum records in a page	R	8

Table 2: Page Definitions

PS = PH + R * (RH + F * FN + 1)
PS = 4 + 8 * (4 + 12 * 20 + 1)
PS = 1964
(1 stands for the new line character.)

3.2 File Format

Each file also contains a collection of pages in the above defined format in a txt file. Each file has a file header and 8 pages with page headers. Since every page place takes up space even there is no record added, each page has the size of fixed. Also, since we are using new line character for a more clean view, we need to take account them either. Related calculations and information is given in the tabular format below:

Feature	Abbreviation	Value
File header	FH	529
File size	FS	16250
Maximum pages in a file	P	8

Table 3: File Definitions

FS = FH + P * (PS + 1) FS = 529 + 8 * (1964 + 1) + 1 FS = 16250
(1s stands for the new line characters.)

3.3 Index File

As the structure design of the project directs each type is stored in a file that contains pages of records for the given type. To ease the insertion, deletion, and search operations for the data in our database, each type, namely file, has its index file alongside its data of pages and records. To optimize the query for equality and range searches, also known as filters, only the RID and the primary key of each record are stored in the index files. This index file is stored as a balanced B+ tree.

In the leaf nodes of the B+ tree, the RID and the primary key of the given type is stored.

4 Operations

Language Operations of the Horadrim is divided into two subcategories in terms of its functions and tasks. Each operation has its own duty and affects the real data differently.

To read the operations, we execute the command line argument that includes the input file. Then we are reading the input line one by one and store each line in commands list. After storing all the commands, we are getting the commands one by one and storing each command in a list as tokenized format. By taking the first element of tokenized list, we get the type of the operations such as create, delete, list, update, search, and filter. The second element of the tokenized list represents that which data we will going to operate such as type and record. After getting these elements, the necessary operation runs in our horadrimSoftware.py. After the execution of each command line, we also append a line in the log file by using logger function that we have created.

4.1 Horadrim Definition Language Operations

These operations allow us to create the data structure of a type, delete the data structure of a type, and list the data types. There are three types of Definition Language Operations in Horadrim.

4.1.1 Create a type

This operation provides us to create a new type in our HoradrimSoftware. To use this operation, we must execute an input command template as in the example below.

create type <type-name> <number-of-fields> <primary-key-order> <field1-name> <field1-type> <field2-name> <field2-type>...

As you can see above, after the "create type", the first stated variable is the name of the type, second stated variable is the number of fields that type has, and the third stated variables is the order of primary key in the fields of that

type. The following variables are the name and values of the fields. There would be two times of the <number-of-fields> variable in the following variables since there are two features for each field. Below there is an example for create type operation.

create type evil 4 1 name str type str alias str spell str

In this operation, we are checking the length of the tokenized list to check is there an error in the input command. Then, we get the type-name, number-of-fields, and primary-key-order from the tokenized list. Then, we are creating two empty list for field-name(s) and field-type(s). Then in an if statement, we check that is the number of given arguments is equal to the number of elements in tokenized list. After, we read the types.json file, which we store the name of the types has already created, to check if the type is created before. After all the checks, we are creating a global B+ Tree for that type, and add the newly created B+ tree object to the dict using the type name as key and B+ tree as value. After creation of the B+ Tree, we append the names and types of the fields inside the field_name and field_time lists, respectively.

We had created a file named file_index.json which stores the information for indexing the file name in our directory. By using the index, we create the data file by using the create_type_file function that we created. After that, we store the type-name and number-of-fields of that type in types.json file. Then, we store the primary key order of the type and name of the type in prim_keys.json file. At the end, we increment the index in the file_index.json for indexing the next types that will be created.

4.1.2 Delete a type

This operation provides us to delete an existing type in our HoradrimSoftware. To use this operation, we must execute an input command template as in the example below.

delete type <type-name>

As you can see above after "delete type" there is only one variable, which is the name of type Below there is an example for delete type operation:

delete type evil

In this operation, we are checking the length of the tokenized list to check is there an error in the input command. Then, we get the type-name from the tokenized list. After, we read the types.json file, which we store the name of the types has already created, to check if the type is created before. Then, we create a list named files to get the names of all created files in the directory, and we are deleting all the files that exists by using the list. After deleting the files, we are also updating the types.json file by removing the type-name and

number-of-fields of that type. Then, we delete the primary key order of the type and name of the type that exists in prim_keys.json file. At the end, we are removing the B+ Tree of the type from global B+ Tree dict, then delete its file from the directory.

4.1.3 List types

This operation provides us to list all the existence types in our Horadrim-Software. To use this operation, we must execute an input command as in the example below.

list type

As you can see above after "list type" there is no variable. In this operation, we are checking the length of the tokenized list to check is there an error in the input command. After, we read the types.json file, which we store the name of the types has already created, to store the all the names of types that are created in a list named types. If the list is not empty, we write the names of the types to output file line by line.

4.2 Horadrim Manipulation Language Operations

These operations allow us to create a record, delete a record by using its primary key, update a record by using its primary key, search a record by using its primary key, list all the records of a type, filter certain records of a type. There are six types of Definition Language Operations in Horadrim.

4.2.1 Create a record

This operation provides us to create a new record of a certain type in our HoradrimSoftware. To use this operation, we must execute an input command template as in the example below.

create record <type-name><field1-value><field2-value>...

As you can see above, after the "create record", there are following variables for values of the each field for that record. Below there is an example for create record operation:

create record angel Tyrael ArchangelOfJustice HighHeavens

4.2.2 Delete a record

This operation provides us to delete a record of a certain type in our Horadrim-Software.To use this operation, we must execute an input command template as in the example below.

delete record <type-name><primary-key>

As you can see above, after the "create record", the first stated variable is the type-name for given record. The second stated variable is the primary key of the record that should be deleted.Below there is an example for delete record operation:

delete record

4.2.3 Search for a record

This operation provides us to search a record of a certain type by using its primary key in our HoradrimSoftware.To use this operation, we must execute an input command template as in the example below.

update record <type-name><primary-key><field1-value><field2-value>...

4.2.4 Update a record

This operation provides us to update a record of a certain type by using its primary key in our HoradrimSoftware.To use this operation, we must execute an input command template as in the example below.

search record

<

type-name><primary-key>

4.2.5 List all records of a type

This operation provides us to list all the records of a certain type in our HoradrimSoftware.To use this operation, we must execute an input command template as in the example below.

list record <type-name>

4.2.6 Filter records

This operation provides us to filter records of a certain type by using its primary key in our HoradrimSoftware.To use this operation, we must execute an input command template as in the example below.

filter record <type-name><condition>

5 Conclusions & Assessment

In this project, we implement a metadata and actual data for storing. We used B+ tree for indexing and reach the data that we store and manage it with using Python3.