CMPE 321 Project 4

Project Horadrim

Harun ERKURT

Taha Baturhan AKBULUT

May 28, 2022

Computer Engineering
Bogazici University
Turkey

Table of Contents

1. Introduction 1

2. Assumptions & Constraints 1

2.1. Assumptions 1

2.2. Constraints 1

3. Storage Structures 2

3.1. System Catalogue 2

3.2. Record 3

3.3. Page 3

3.4. File 4

4. Operations 4

4.1. Horadrim Definition Language Operations 4

4.2. Horadrim Manipulation Language Operations 5

5. How B-Plus Tree Works 5

6. Conclusion & Assessment 6

1. Introduction

In this project we designed a database storage management system. This system has records, pages, files and headers for these structures. The reason we created headers is to be able to quickly insert, search and delete operations. Pages, files and records structures help us to efficiently handle storage and search operations. This structure supports the creation, deletion, inheritance, and listing of type operations; create, delete, search, update, list and filter operations for records.

2. Assumptions & Constraints

2.1 Assumptions

- Each field has 20-byte space
- Each page has 2667-byte space.

- A file has 100 pages in it.
- All fields are alphanumeric.
- User always enters valid input
- There is no null value for any field.
- The inputs should be given in correct form.
- The input lines should be given line by line.

2.2 Constraints

- The data must be organized in pages and pages must contain records.
- It is must to use B+-Tree for indexing.
- Primary keys of types should be used for the search-keys in trees.
- There should be separate B+-Trees for each type, and these trees must be stored in file(s). So, when a type is created, its B+-Tree will also be created, and its structure should be stored.
- With every create record and delete record operations, the corresponding tree should be updated.
- B+-Trees should not be generated from scratch on the fly using records for DML operations. It should be utilized the tree structures that you stored in files. (We get the B-Plus Tree code from https://github.com/Nicolas LM/bplustree/tree/master/bplustree. and we changed some parts to make it compatible with our own code.)
- The data must be organized in pages and pages must contain records.
- It is not allowed to store all pages in the same file and a file must contain multiple pages. This means that your system must be able to create new files as Horadrim grows. Moreover, when a file becomes free due to deletions, that file must be deleted.
- Although a file contains multiple pages, it must read page by page when it is needed. Loading the whole file to RAM is not allowed.

3. Storage Structures

3.1 System Catalogue

The system catalog consists of tables and views that describe the structure of the database. System catalogue gives us metadata of our database storage system. We used system catalogue files frequently. The system catalog has made our work very comfortable. It has become very easy to do some checks. We

controlled type names to create a new type for if there exists before. Attribute Catalog holds:

- Relation Name
- Attributes
- Attribute Types
- Positions

Attribute Catalog	Attributes	Attribute Types	Positions
Relation Name 1	field	string	1
Relation Name 2	field	string	2

Relations Catalog holds:

- Relation
- File Names

Relation Catalog	Field
Relation Name 1	File Names
Relation Name 2	File Names

Primary Key Catalog holds:

- Relation Name
- Primary Key

Primary Key Catalog	Field
Relation Name 1	Primary Key
Relation Name 2	Primary Key

Index Catalog holds:

- Primary Key
- Index

Index Catalog	Field
Primary Key	Index 1
Primary Key	Index 2

Relation Index Catalog holds:

- Relation Name
- Index

Relation Index Catalog	Field
Relation Name 1	Index 1
Relation Name 2	Index 2

3.2 Record

For each type, we should make calculations. We are assuming each field is 252 bytes. Total size of a record is 263 bytes. Record header holds an integer for checking if there is a record, rid, and fields.

Record Header	Field	Record Id	Field
Record	0	0	Null
Record	0	1	Null
Record	0	2	Null
Record	0	3	Null
Record	0	4	Null
Record	0	5	Null
Record	1	6	angel Tyrael ArchangelOfJustice HighHeavens

3.3 Page

Page is a piece of data that is stored to disk as a unit. It is must for B-Plus Tree to store the number of records in a single node. Simply page holds records. Each page has 2667 bytes storage space for records. Every page has one type of records. Record per page is equal to 10. Page header is 37 bytes, so we have 2630 byte for records. Page header holds Page name, available record count and page id. Page id is required for indexing when using B-Plus Tree. When scanning file page by page, we compare the type that we searched with the type in the page header. And, we add the types of all the pages in the file to the files header to scan efficiently.

Page Header:	Page Name	Aver. Record Count	Page id
Record Header:	0	0	Null
Record Header:	0	1	Null
Record Header:	0	2	Null
Record Header:	0	3	Null
Record Header:	0	4	Null
Record Header:	0	5	Null
Record Header:	0	6	Null

Page Header:	Page Name	Aver. Record Count	Page id
Record Header:	0	7	Null
Record Header:	1	*8*	angel Tyrael ArchangelOfJustice HighHeavens
Record Header:	0	9	Null

3.4 File

Files have 100 pages. File header holds types of pages, Total pages in that file, and page id. We know page. File organization is the advanced method of an indexed sequential access method. It uses a tree-like structure to store the records in File. We can search every record using file. We are pulling types from file and pid.

File Header:	Types of Pages	Total Pages	Page Id
Page:	Angel	10	0
Record Header:	0	0	Null
Record Header:	0	1	Null
Record Header:	0	2	Null
Record Header:	0	3	Null
Record Header:	0	4	Null
Record Header:	1	5	angel Tyrael

4. Operations

Definition Language Operations:

- Create Type
- Delete Type
- List Type

Manipulation Language Operations:

- Create Record
- Delete Record
- Search for a record (by primary key)
- Update a record (by primary key)
- List all records of a type
- Filter records (by primary key)

4.1 Horadrim Definition Language Operations

Create Type

- Get input from user, which contains “create type” and field names.
- Looks for all files that if it contains current type page.
- Splits attributes
- Stores attributes
- Creates new page with give instructions
- Takes file with given type.
- If type already exists, return error.
- Adds page to taken file.
- Then adds the metadata to the relevant catalogs

Delete Type

- Get input from user, which contains “delete type” and field names.
- Then searches file name ends with .txt if given type name exists.
- If it exists, deletes the given type from the file containing the type.
- Prints log to log file.

List Type

- Get input from user, which contains “list type”.
- Searches files for given type name
- And lists if the type exists.

4.2 Horadrim Manipulation Language Operations

Create

- Get input from user, which contains “create record” operation and type name and field values.
- Checks if the same record name is given.
- Starting from the last file, checks file header for insertion.
- Checks if given record id exists.
- After passing all the tests, scans the file to see if there is a suitable page to place the record.

- If it cannot find the appropriate file, it searches for the appropriate file and creates a new page there. If it doesn't exist, it opens a new file and puts the record there. Prints log.

Delete

- Get input from user, which contains "delete record" operation and type name and primary key.
- Checks if the record name exists.
- Starting from the last file, checks file header for deletion.
- Checks if given record id exists.
- After passing all the tests, scans the file to find record.
- Last, deletes the record from the file and move all records below the deleted record up one line. Prints log.

Update

- Get input from user, which contains "update record" operation and type name, primary key and field values.
- Checks if the same record name is given or not from catalog.
- Starting from the last file, checks file header for updating.
- After passing all the tests, scans the file to find the key.
- If it does not exist print failure to log.
- If it exists update the record. Print success to log.

Search

- Get input from user, which contains "search record" operation and type name and record id.
- Checks catalog for given type name.
- If the name does not exists in the catalog, print failure to log.
- After passing all the tests, scans the file to find record.
- If the record exists prints the record. Prints success to log.

List

- Get input from user, which contains "list record" operation and type name.
- Reads catalog from given file name.
- If given type does not exists in the catalog prints failure.
- After passing all the tests, scans the file to find page.
- Prints list. Prints success to log.

Filter

- Get input from user, which contains “filter record” operation and type name and condition
- Reads catalog from given type name.
- Starting from last created file checks all record and if given condition is there, prints it to output file. Prints success to log.
- if given condition is not there, prints failure to log.

5. How B-Plus Tree Works

While defining the B-Plus tree object, we only gave file name and string serializer. We use close function to close the B-Plus Tree. For insertion there is an insert function that gets key and value. We use get function to search a value using primary key. In searching, firstly compares key with the root node. If key is bigger than current, goes to right child, else goes to left child. Lastly finds the right index. In insertion B Plus Tree finds a location at the leaf nodes. If the leaf node's key is less than m-1 inserts a node in increasing order. If node keys equal to m-1, then inserts a new item inn increasing number of items. Takes medians and splits into two nodes and pushes median to its parent node. If parent node key equals to m-1, repeats the procedure. Lastly inserts the element. In deletion, if node is bigger than m/2, deletes the given key from the node. Else if the left child contains m/2 elements, pushes the bigger element to the parent and deletes the key. If right child contains m/2 elements, pushes smallest element to parent.

6. Conclusion & Assessment

We tried to generate the B-Plus Tree from scratch and then we were told that we don't need to do it that way. Then we get the B-Plus Tree code from https://github.com/NicolasLM/bplustree/tree/master/bplustree. And we changed some parts to make it compatible with our own code. We implemented system catalog and type systems for metadata. We kept all the catalogs in separate files, but actually we could have kept them in a single file called the system catalog