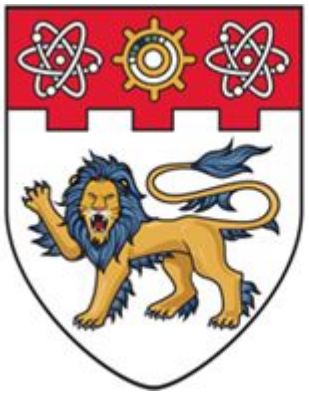


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CZ4031 Database System Principles
Project 1 Report
Group 24

1. Introduction

This project is to design and implement the following two components of a database management system, storage and indexing.

2. Design of Storage Component

1.1 Overview

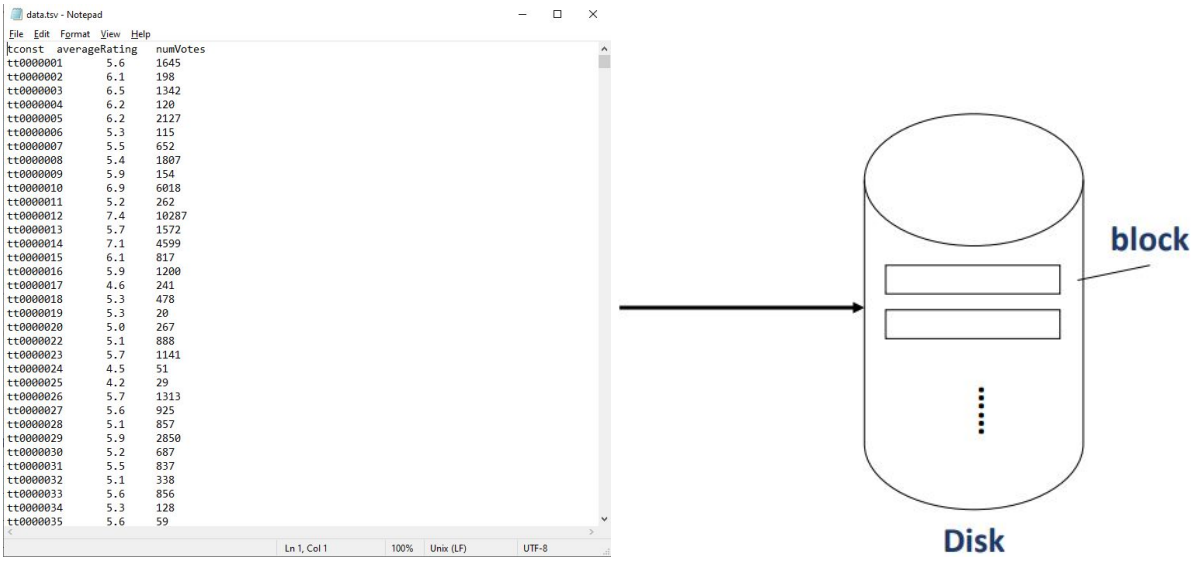


Figure 1: Overview of Storage Design

From the data.txt given, we first design a database to store all the data and then index it with B+ Tree Indexing.

1.2 How each data Item is stored as a field

```
void readFile(string filepath, int limit) {
    string delim = "\t";
    ifstream file(filepath);
    if (file.is_open()) {
        cout << "File opened" << endl;
        string line;
        int i = 0;
        getline(file, line);
        while (getline(file, line) && ((i < limit) || limit == 0)) {
            // cout << line << endl;
            vector<string>tokens = split(line, delim);

            string tc = tokens[0];
            double avg = stod(tokens[1]);
            int votes = stoi(tokens[2]);

            Record r(tc, avg, votes);
            addRecord(r);
            i++;
        }
        file.close();
    }
    else {
        exit(EXIT_FAILURE);
    }
}
```

Figure 2: Store data Item as a field in C++

Vectors are dynamic arrays that provide flexibility and are effective in handling dynamic elements. Thus, over instead of arrays which are static in nature, we use vectors to store the variable-length fields.

1.2 How fields are packed into a record

```
Record(string tc, double avg, int votes) {
    tconst = tc;
    average = avg;
    numVotes = votes;
}
```

Figure 3: Constructing record in C++

Tconst is declared as a string. AverageRatings is declared as a double. Lastly, numVotes is declared as an integer. Using the three fields, we construct a record.

```
void readFile(string filepath, int limit) {
    string delim = "\t";
    ifstream file(filepath);
    if (file.is_open()) {
        cout << "File opened" << endl;
        string line;
        int i = 0;
        getline(file, line);
        while (getline(file, line) && ((i < limit) || limit == 0)) {
            // cout << line << endl;
            vector<string>tokens = split(line, delim);

            string tc = tokens[0];
            double avg = stod(tokens[1]);
            int votes = stoi(tokens[2]);

            Record r(tc, avg, votes);
            addRecord(r);
            i++;
        }
        file.close();
    }
    else {
        exit(EXIT_FAILURE);
    }
}
```

Figure 4: Instantiating record in C++

The record is then instantiated as an object in the readFile function, and then passed to the addRecord function, which stores it in another vector, recordVector, within the block.

tconst	averageRatings	Numvotes
tt0000001	5.6	1645
tt0000002	6.1	198
...	...	...

Figure 5: How the fields are packed into a record

1.3 How records are packed into a block

```
void addRecord(Record &r) {
    if (blockVector.empty()) { // empty DB
        Block b(++current_size);
        b.addRecord(r);
        blockVector.push_back(b);
        ;
    }
    else {
        // extract the last block
        Block *lastBlock;
        lastBlock = &blockVector[current_size-1];

        // check for availabilty: 0 (can add), 1 (full)
        int avalability = lastBlock->checkBlockAvailabilty();
        if (avalability == 1) {
            Block b(++current_size);
            b.addRecord(r);
            blockVector.push_back(b);
        }
        else if (avalability == 0) {
            lastBlock->addRecord(r);
        }
    }
}
```

Figure 6: Adding records into blocks in C++

```
tt0000002 6.1 198 1
tt0000003 6.5 1342 1
tt0000004 6.2 120 2
tt0000005 6.2 2127 2
tt0000006 5.3 115 3
tt0000007 5.5 652 3
tt0000008 5.4 1807 4
tt0000009 5.9 154 4
tt0000010 6.9 6018 5
tt0000011 5.2 262 5
tt0000012 7.4 10287 6
tt0000013 5.7 1572 6
```

Figure 7: Records packed in blocks

A block is constructed with a block_id and a vector of records. For a block size of 100B, it can hold 2 records while a block size of 500B can hold 10 records. In Figure 5, it shows how we add records. In Figure 6, it is a simplified overview of how the records are packed in blocks.

3. Design of B+ Tree Component

3.1 Data Structure of a node

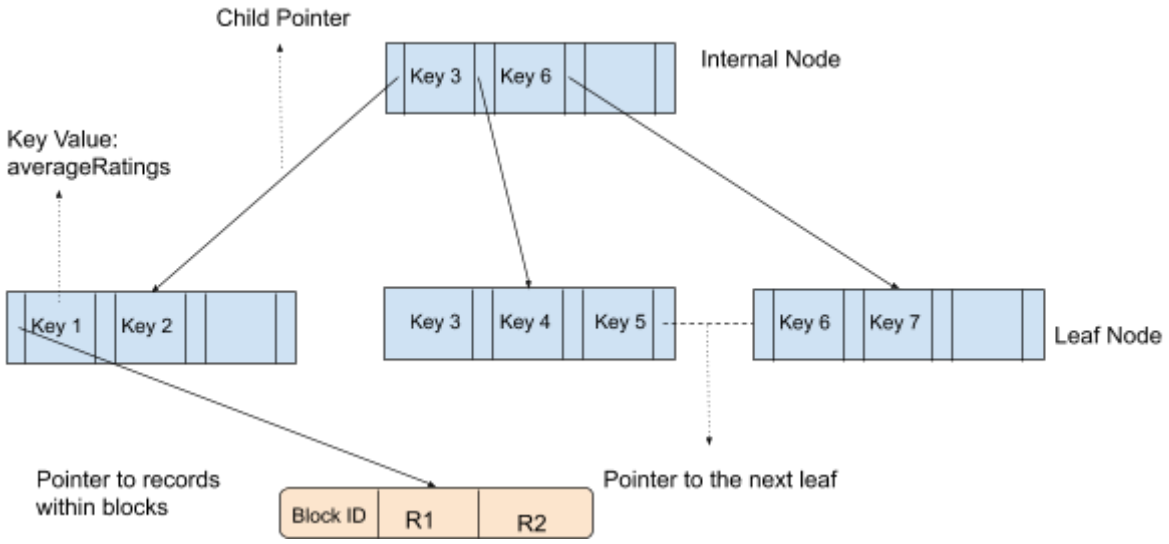


Figure 8: B+ tree design

The **maximum number** of keys a node maintains is 3. In this case, the data attribute averageRatings the search key value. If the leaf doesn't have required space, the node is split into 2 nodes, and the middle node is then copied to the next index node. If the index node doesn't have required space,the node is split and the middle element is copied to the next index page.

Each pointer points to records within the blocks (R1 and R2), and each key next points to the next leaf node in the B+ tree (shown in Figure 8).

4. Results of Experiment

3.1 Experiment 1

Output

Number of blocks and the size of database

Block size = 100B

```
Number of blocks of DB: 535159
Size of Database (B): 53515900
```

Block size = 500B

```
Number of blocks of DB: 107032
Size of Database (B): 53515888
```

Block Size	Block Capacity/records	Number of blocks	Size of Database
100B	2	535 159	53,515,900B
500B	10	107 032	53,516,000B

3.2 Experiment 2

Output

Height and the number of nodes
Block size = 500B

```
Height of B+Tree
Height = 12
```

```
Number of nodes in B+Tree: 324000
```

Block size = 100B

Actual content of the root node and its child nodes
Block size = 500B

```
Display B+Tree
0x265dcc0 8.6
0x4121f60 8.1 0x26a29c0 8.2
0x2f3a130 6.1 0x2654450 7 0x28e0d10 7.3
0x276a170 4.8 0x4a1f870 5.8
0x2ac1770 3.5 0x29b8c90 4.2 0x2676c70 4.4
```

Block size = 100B

```
Display B+Tree
tt0052954 8 tt0000361 8.1
tt0000003 6.5 tt0015739 7.6
tt0020604 5.3 tt0024194 5.5 tt3180704 6.3
tt0111532 4.3 tt0058062 4.4 tt0347102 4.7
tt0198086 3 tt0001444 3.3 tt1256584 4.1
```

Block Size	Block Capacity/records	Parameter n	No. of nodes	Height
100MB	2	3	1157753	13
500MB	10	3	324000	12

3.3 Experiment 3

Block Size	Number of Index nodes accessed	Number of data blocks accessed
100B	13	32369
500B	12	5189

Output

Content of index nodes access

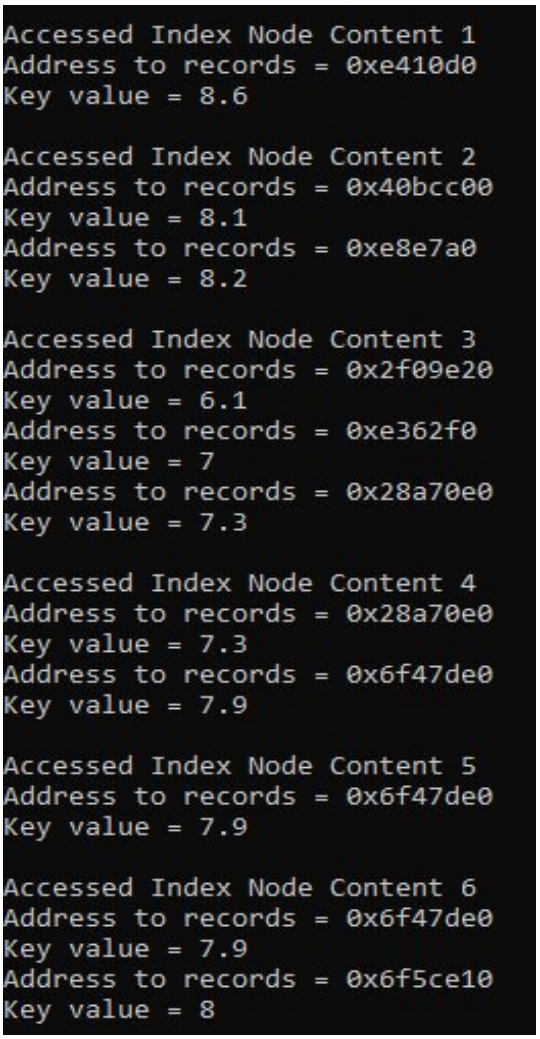
Block size = 100B

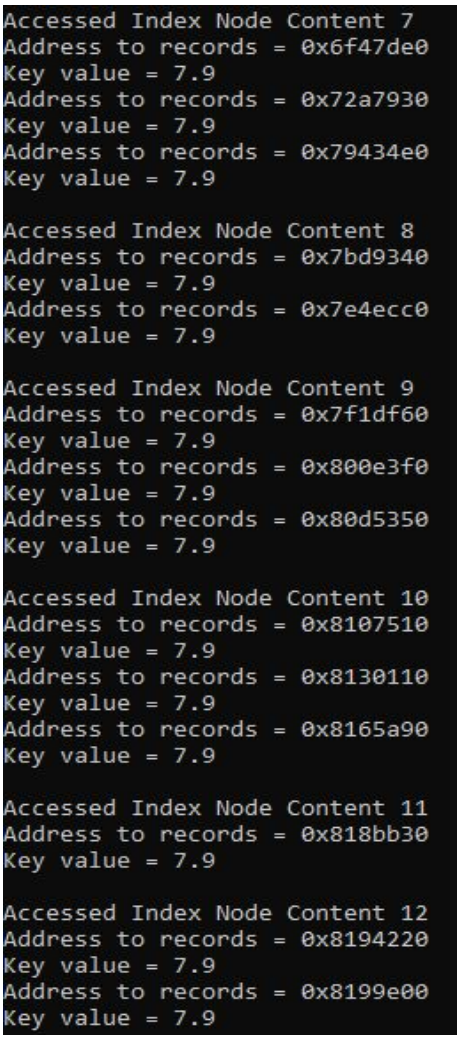
```
Number of indexes accessed=  13
```

```
Accessed Index Node Content 1
Address to records = 0x10cc470
Key value = 8
Address to records = 0xef1890
Key value = 8.1

Accessed Index Node Content 2
Address to records = 0x71a408
Key value = 6.5
Address to records = 0xf23b00
Key value = 7.6

Accessed Index Node Content 3
Address to records = 0xf23b00
Key value = 7.6
Address to records = 0x2fdf5a0
Key value = 7.9

Accessed Index Node Content 4
Address to records = 0x2fdf5a0
Key value = 7.9

Accessed Index Node Content 5
Address to records = 0x2a7dea8
Key value = 7.9

Accessed Index Node Content 6
Address to records = 0x2a7dea8
Key value = 7.9

Accessed Index Node Content 7
Address to records = 0x2a7dea8
Key value = 7.9
Address to records = 0x2cb0fc0
Key value = 7.9
Address to records = 0x4c0aa18
```

```
Accessed Index Node Content 8
Address to records = 0x4ce3e60
Key value = 7.9
Address to records = 0x5dce938
Key value = 7.9

Accessed Index Node Content 9
Address to records = 0x5e191e0
Key value = 7.9

Accessed Index Node Content 10
Address to records = 0x5e27dd8
Key value = 7.9
Address to records = 0x5e3c790
Key value = 7.9

Accessed Index Node Content 11
Address to records = 0x5e416a0
Key value = 7.9
Address to records = 0x5e46c90
Key value = 7.9
Address to records = 0x5e541a8
Key value = 7.9

Accessed Index Node Content 12
Address to records = 0x5e56b70
Key value = 7.9

Accessed Index Node Content 13
Address to records = 0x5e57670
Key value = 7.9
```

Block size = 500B





Content of data blocks accessed and output of attribute “tconst”
***Attached in separate txt file: averageRatings8 ***

3.4 Experiment 4

Block Size	Number of Index nodes accessed	Number of data blocks accessed
100B	13	375491
500B	4	74001

Output

Content of Index nodes Access

Block Size = 100B

```
Number of indexes accessed=  13
```

```
Accessed Index Node Content 1
Address to records = 0x117b6b0
Key value = 8
Address to records = 0x6b1100
Key value = 8.1

Accessed Index Node Content 2
Address to records = 0x1fa408
Key value = 6.5
Address to records = 0x6e3d10
Key value = 7.6

Accessed Index Node Content 3
Address to records = 0x44f6f00
Key value = 7.2

Accessed Index Node Content 4
Address to records = 0x6f0a78
Key value = 6.7
Address to records = 0x762478
Key value = 6.8
```

```
Accessed Index Node Content 5
Address to records = 0x1a8e460
Key value = 6.9
Address to records = 0x1516fa0
Key value = 6.9
Address to records = 0x48f06e8
Key value = 7.1

Accessed Index Node Content 6
Address to records = 0x1516fa0
Key value = 6.9

Accessed Index Node Content 7
Address to records = 0x1516fa0
Key value = 6.9
Address to records = 0x1649338
Key value = 6.9
Address to records = 0x4c00b08
Key value = 6.9
```

```
Accessed Index Node Content 8
Address to records = 0x4c3a588
Key value = 7
Address to records = 0x5cb8430
Key value = 7
Address to records = 0x4c1a680
Key value = 7

Accessed Index Node Content 9
Address to records = 0x5cd32f8
Key value = 6.9
Address to records = 0x4c00530
Key value = 7

Accessed Index Node Content 10
Address to records = 0x5ced6e0
Key value = 6.9
Address to records = 0x5d03f50
Key value = 6.9
Address to records = 0x5d40eb8
Key value = 6.9
```

```
Accessed Index Node Content 11
Address to records = 0x5d47738
Key value = 6.9
Address to records = 0x5d50b00
Key value = 6.9

Accessed Index Node Content 12
Address to records = 0x5d519f8
Key value = 6.9
Address to records = 0x5d55800
Key value = 6.9

Accessed Index Node Content 13
Address to records = 0x5d55540
Key value = 6.9
Address to records = 0x5d55ee0
Key value = 6.9
Address to records = 0x5d57958
Key value = 6.9
```

Block Size = 500B

```
Accessed Index Node Content 1
Address to records = 0x2610490
Key value = 8.6

Accessed Index Node Content 2
Address to records = 0x409b9a0
Key value = 8.1
Address to records = 0x2654860
Key value = 8.2

Accessed Index Node Content 3
Address to records = 0x2ef6e60
Key value = 6.1
Address to records = 0x2604a70
Key value = 7
Address to records = 0x288ccc0
Key value = 7.3

Accessed Index Node Content 4
Address to records = 0x6164240
Key value = 6.7
```

```
Accessed Index Node Content 8
Address to records = 0x78ada60
Key value = 6.9
Address to records = 0x7b8b510
Key value = 6.9
Address to records = 0x7e9ee40
Key value = 6.9

Accessed Index Node Content 9
Address to records = 0x7f59cc0
Key value = 6.9
Address to records = 0x802cd10
Key value = 6.9
Address to records = 0x810a890
Key value = 6.9

Accessed Index Node Content 10
Address to records = 0x812a160
Key value = 6.9
```

```
Accessed Index Node Content 5
Address to records = 0x6164240
Key value = 6.7

Accessed Index Node Content 6
Address to records = 0x6164240
Key value = 6.7
Address to records = 0x73768a0
Key value = 6.8
Address to records = 0x6f3f8f0
Key value = 6.9

Accessed Index Node Content 7
Address to records = 0x6f3f8f0
Key value = 6.9
Address to records = 0x75d0310
Key value = 6.9
```

```
Accessed Index Node Content 11
Address to records = 0x813a5b0
Key value = 6.9
Address to records = 0x8147000
Key value = 6.9
Address to records = 0x8162830
Key value = 6.9

Accessed Index Node Content 12
Address to records = 0x81674c0
Key value = 6.9

Number of indexes accessed=  12
```

Content of data blocks accessed and output of attribute “tconst”
***Attached in separate txt file: averageRatings7-9”**

5. Contribution

Name	Matriculation No.	Contribution
Loh Yi Xuan Renice	U1822247D	- Experiments 2,3,4 - Report
Ryan Lau	U1720870H	- Experiments 2,3,4 - Report
Dat	U1820970A	- Design of the storage component - Design of B+ Tree - Testing and Debugging - Experiment 1
Rachel	-	-