DBMS Lab 6 & 7

Rahul Barodia (B20CS047)

Q1)

a)

Let's see the types of normal forms and how to identify them:

1 NF - A relation is in 1NF if it contains an atomic value

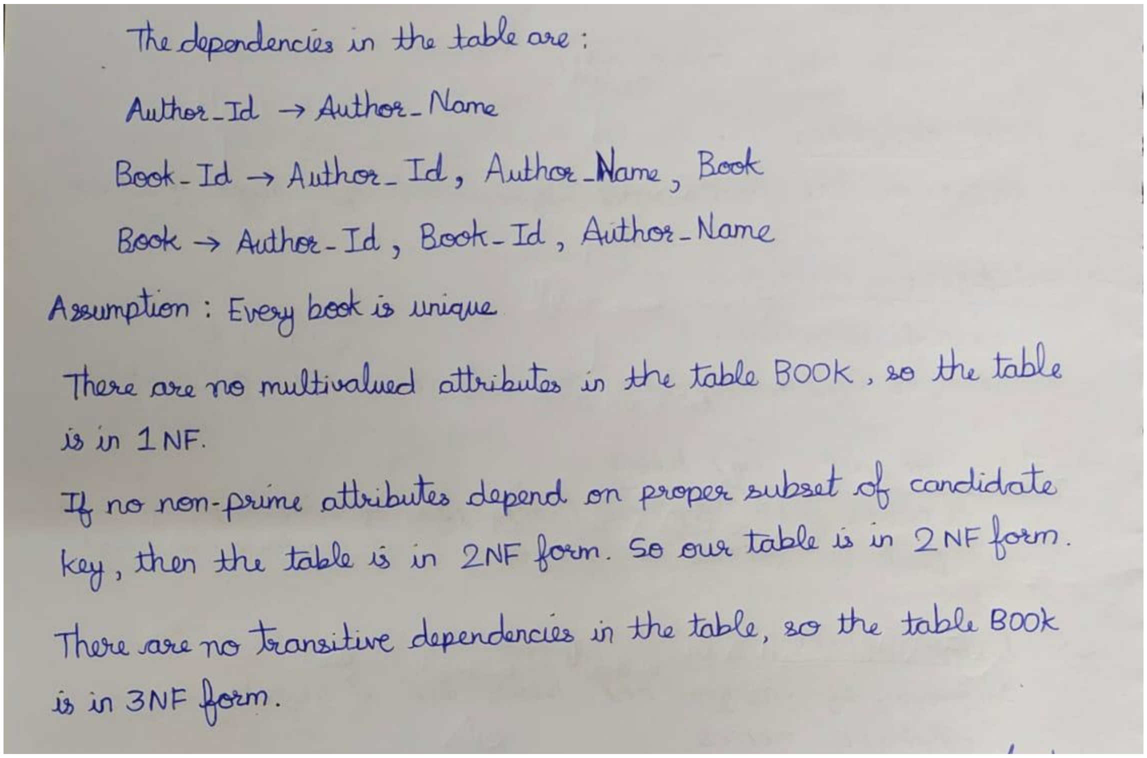
2 NF - A relation will be in 2NF if it is in 1NF and there is no partial dependency

3 NF - A relation will be in 3NF if it is in 2NF and no transition dependency

exists

In the given table BOOK, there are 4 entities, namely Author_ID, Book_ID, Author_Name and Book (Book name).

Book_ID is the Primary key.



The dependencies in the table are :

Author_Id → Author_Name

Book_Id → Author_Id, Author_Name, Book

Book → Author_Id, Book_Id, Author_Name

Assumption : Every book is unique

There are no multivalued attributes in the table BOOK, so the table is in 1NF.

If no non-prime attributes depend on proper subset of candidate key, then the table is in 2NF form. So our table is in 2NF form.

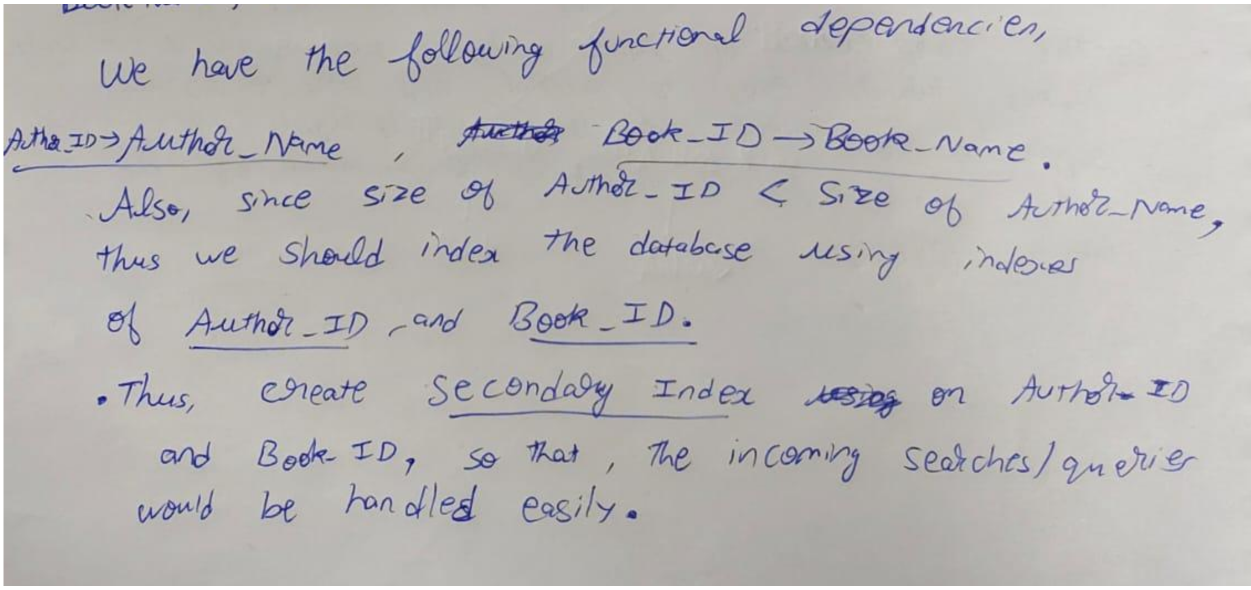
There are no transitive dependencies in the table, so the table BOOK is in 3NF form.

Since all attributes are dependent only on primary key, the table is in BCNF.

Therefore, the table BOOK is in BCNF form.

Since the redundancy in the database is also reduced to a lot extent in 3NF form, therefore for further hashing and indexing I would not normalize the table to BCNF.

b) Since majority of searches in the catalogue involve Author_Name and/or Book name, thus we should use those as indexes.



We have the following functional dependencies,

Autho_ID → Author_Name , ~~Author~~ Book_ID → Book_Name.

Also, since size of Author_ID < Size of Author_Name, thus we should index the database using indexes of Author_ID and Book_ID.

- Thus, create Secondary Index ~~using~~ on Author_ID and Book_ID, so that, the incoming searches/queries would be handled easily.

Q2 and Q3) Since the queries asked in Q2 and Q3 are the same I have represented Q 2 and 3 in the same as follows:

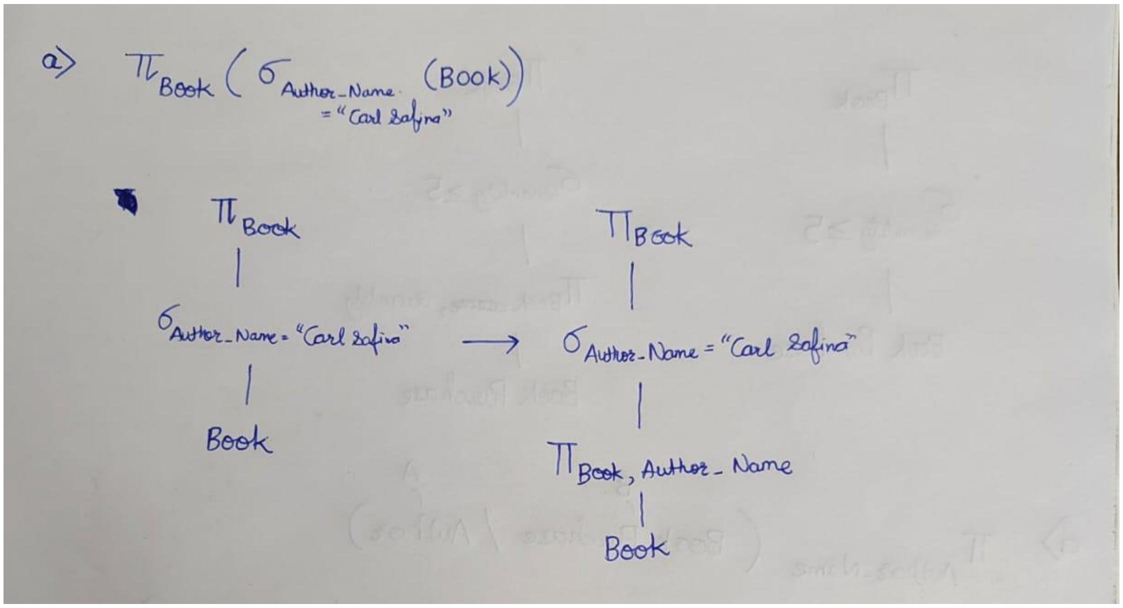
Have implemented the hash tables for extendible hashing and linear hashing according to the previous question. Have only made changes to the code to accommodate changes between the array data structure of the buckets to doubly linked lists.

The Book_ID, and Author_ID have been hashed according to the following function:

```
int hashFunction(string key) {  // Adding ASCII values of characters in
the book_id and then adding the last 4 characters (book_number) to it.

    int hashCode = 0;

    for (int i = 0; i < key.length()-4; i++) {

        hashCode += key[i];

    }

    return abs(hashCode+stoi(key.substr(key.size() - 4)));

}
```

Query Trees and Query Plans for the given Queries

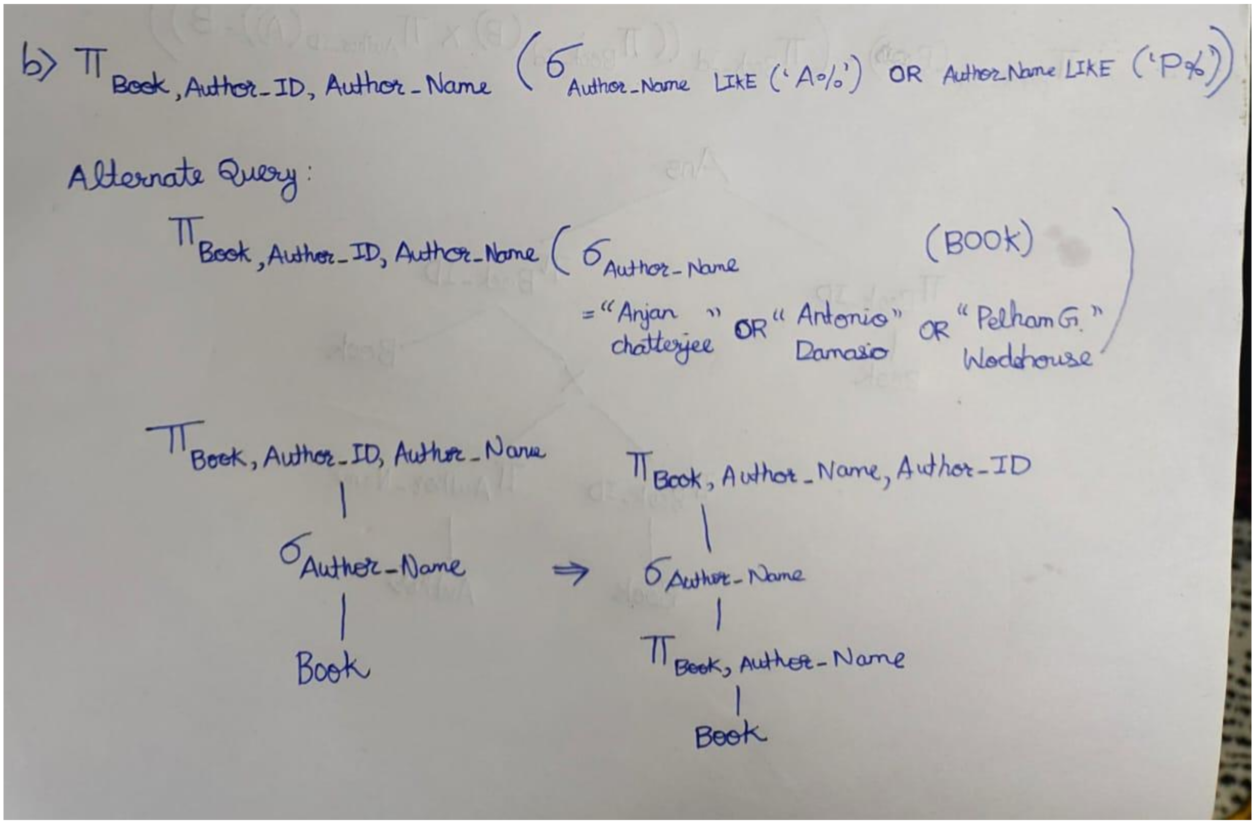
a)



a) $\pi_{Book}\left(\sigma_{Author\_Name = \text{"Carl Safina"}}(Book)\right)$

π_{Book}
$\sigma_{Author\_Name = \text{"Carl Safina"}}$
Book
$\longrightarrow$
π_{Book}
$\sigma_{Author\_Name = \text{"Carl Safina"}}$
$\pi_{Book,\ Author\_Name}$
Book

b)



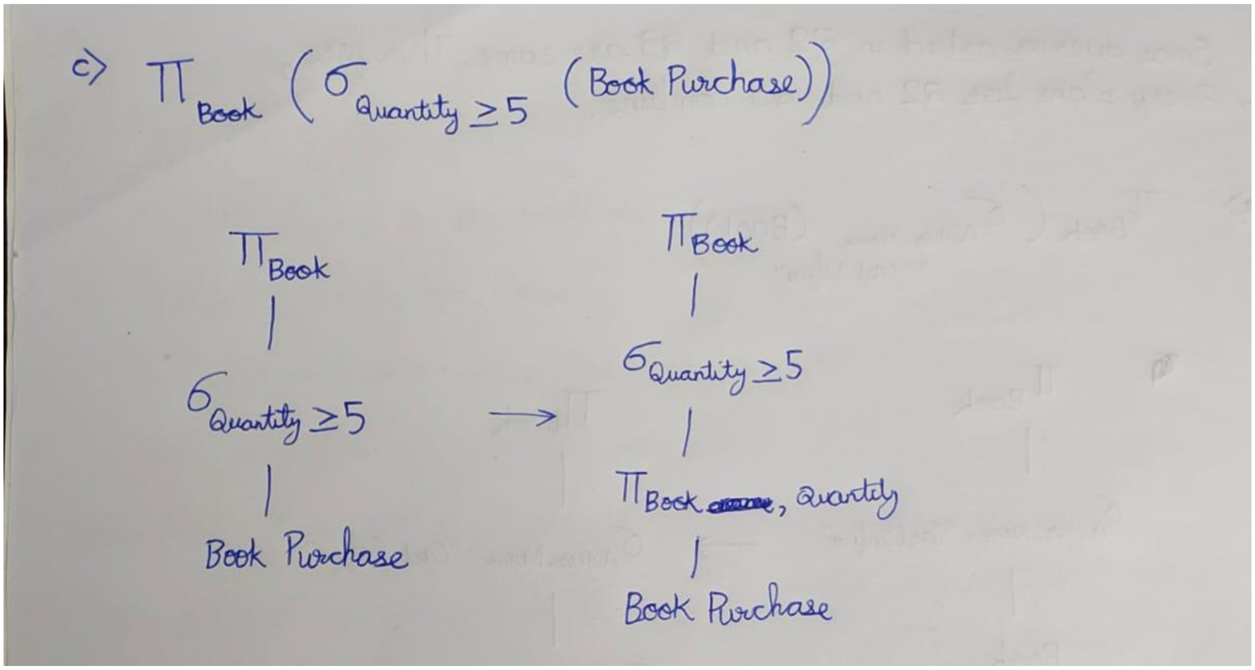
b) $\pi_{Book,\ Author\_ID,\ Author\_Name}\left(\sigma_{Author\_Name \text{ LIKE } (\text{'A\%'}) \text{ OR } Author\_Name \text{ LIKE } (\text{'P\%'})}\right)$

Alternate Query:

$\pi_{Book,\ Author\_ID,\ Author\_Name}\left(\sigma_{Author\_Name = \text{"Anjan chatterjee" OR "Antonio Damasio" OR "Pelham G. Wodehouse"}}(BOOK)\right)$

$\pi_{Book,\ Author\_ID,\ Author\_Name}$
σ_{Author_Name}
Book
$\Rightarrow$
$\pi_{Book,\ Author\_Name,\ Author\_ID}$
σ_{Author_Name}
$\pi_{Book,\ Author\_Name}$
Book

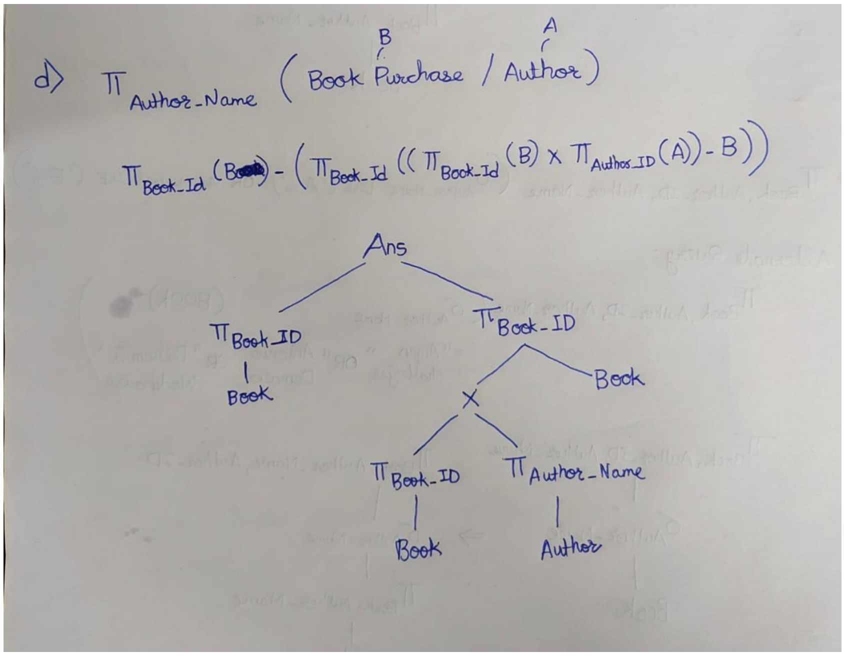
c)



c> $\Pi_{Book}(\sigma_{Quantity \geq 5}(Book\ Purchase))$

Π_{Book}
$\sigma_{Quantity \geq 5}$
Book Purchase
Π_{Book}
$\sigma_{Quantity \geq 5}$
$\Pi_{Book, quantity}$
Book Purchase

d)

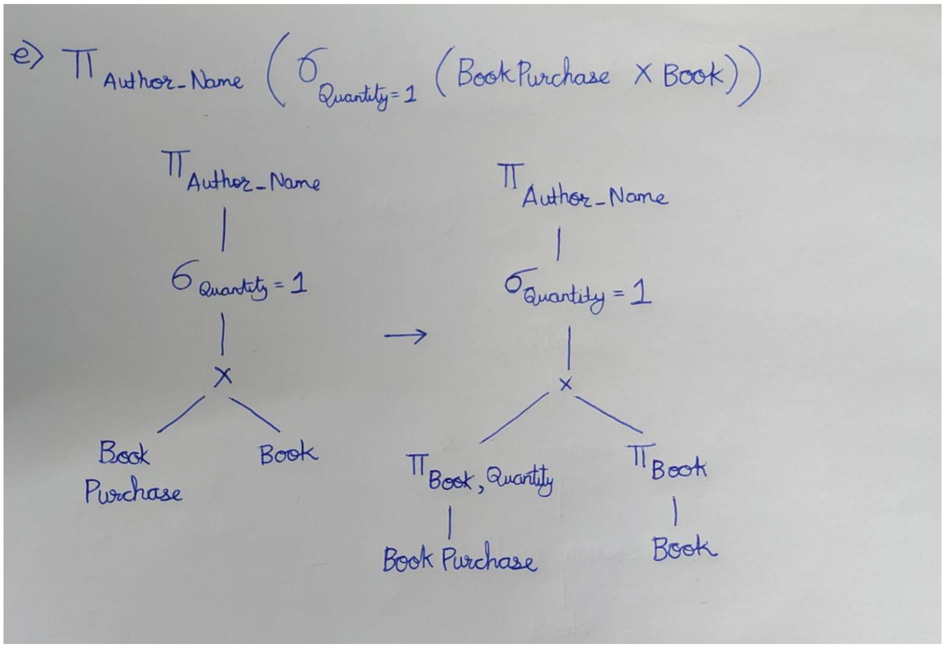


d> $\Pi_{Author\_Name}(\overset{B}{Book\ Purchase} / \overset{A}{Author})$

$\Pi_{Book\_Id}(B) - (\Pi_{Book\_Id}((\Pi_{Book\_Id}(B) \times \Pi_{Authos\_ID}(A)) - B))$

Ans
Π_{Book_ID}
Book
Π_{Book_ID}
Book
×
Π_{Book_ID}
Π_{Author_Name}
Book
Author

e)



e) $\pi_{Author\_Name}(\sigma_{Quantity=1}(BookPurchase \times Book))$

Q4) The comparison between the execution times are as follows:

<u>**Query 1**</u>

<u>Q2</u>

Ca_Sa_0319 Carl Safina Anim_CS_0319 What Animals Think

Time taken to search (in microseconds): 996

<u>Q3</u>

Searching new entry according to question

Ca_Sa_0319 Carl Safina Anim_CS_0319 What Animals Think

Time taken to search(in microseconds): 997

Query 2

Q2

An_Da_0104 Antonio Damasio Self Comes to Mind

An_Ch_0103 Anjan Chatterjee The Aesthetic Brain

Pe_Wo_1623 Pelham G. Wodehouse Wodehouse at the Wicket

Pe_Wo_1623 Pelham G. Wodehouse Aunts Aren't Gentlemen

Time taken to search (in microseconds): 853

Q3

An_Da_0104 Antonio Damasio Self Comes to Mind

An_Ch_0103 Anjan Chatterjee The Aesthetic Brain

Pe_Wo_1623 Pelham G. Wodehouse Wodehouse at the Wicket

Pe_Wo_1623 Pelham G. Wodehouse Aunts Aren't Gentlemen

Time taken to search (in microseconds): 1253

Query 3

Q2

Goblet of Fire_Harry Potter Gobl_JR_1018

Deathly Hallows_harry Potter Fant_JR_1018

Philosophers Stone_Harry Potter Phil_JR_1018

Prisoner of Azkaban_Harry Potter Pris_JR_1018

Time taken to search (in microseconds): 783

Q3

Goblet of Fire_Harry Potter Gobl_JR_1018

Deathly Hallows_harry Potter Fant_JR_1018

Philosophers Stone_Harry Potter Phil_JR_1018

Prisoner of Azkaban_Harry Potter Pris_JR_1018

Time taken to search (in microseconds): 1373

<u>Query 4</u>

<u>Q2</u>

Time taken to search (in microseconds): 935

<u>Q3</u>

Time taken to search (in microseconds): 1103

<u>Query 5</u>

<u>Q2</u>

Antonio Damasio

Anjan Chatterjee

Marvin Minsky

Marvin Minsky

Vilayanur Ramachandran

Time taken to search (in microseconds): 682

Q3

Antonio Damasio

Anjan Chatterjee

Marvin Minsky

Marvin Minsky

Vilayanur Ramachandran

Time taken to search (in microseconds): 872

Observation:

Upon observing the above results it can be said that on a general basis the time taken in extendible hashing is less than that of in linear hashing.

Also in terms of space complexity, extendible hashing is better than linear hashing.

Also upon observation, we can see that without optimization and after optimization, we can see about a 10% increase in performance after query optimization.

Bonus: I have implemented B++ trees. The code of the following is present in the zip file submitted.