Lab - 4

a)

1 NF [the table should not contain any multi valued attribute] -----> Yes

2NF [1NF + all the non prime attribute should be fully functional dependent on candidate key]

1st table : Book_ID → Book , Book_ID → Author_ID , Book -> Author_Id . So CK = Book_ID, Book and non prime attributes (Author_ID) is fully dependent on it.

2nd Table : Author_ID -> Author_name . So CK = Book_ID and non prime attributes (Author_name) is fully dependent on author_ID.

3rd Table : Book_ID → Copies , Book_ID → Purchase_Dt . So CK = Book_ID and non prime attributes (Copies , Purchase Dt) are fully dependent on it.

Assumed that book names can't be same.

3NF [2NF + There should be no transitive dependency in the table]

1st table : Book_ID → Book , Book_ID → Author_ID , Book -> Author_Id . No transitive dependency.

2nd table : Author_ID -> Author_name.No transitive dependency.

3rd table : Book_ID → Copies , Book_ID → Purchase_Dt. No transitive dependency.

BCNF [3NF + all the dependency on CK.]

True . In all the three tables all the dependency are on CK.

4NF [3NF + No multivalued dependency]

Multivalued dependency is when 2 attributes in a table are independent of each other and both depend on third attribute.

So here book_name and author id are independent of each other (Author id is encoded with the help of author’s name) and dependent on Book_Id .

So 4NF is True.

5NF [4NF + Lossless Decomposition]

Book Detail Table

Author_ID	Book_ID
Da_001	Da001_Sel
Mi_009	Mi009_Emo
Mi_009	Mi009_Soc
Ra_001	Ra001_Pha
Ro_015	Ro015_Fan
Ro_015	Ro015_Gob
Ro_015	Ro015_Phi
Ro_015	Ro015_Pri
Sa_001	Sa001_Voy
Sa_001	Sa001_Wha
To_015	To015_Fel
Wo_015	Wo015_Wod

Book_ID	Book
Da001_Sel	Self Comes to Mind
Mi009_Emo	Emotion Machine
Mi009_Soc	Society of Mind
Ra001_Pha	Phantoms in the Brain
Ro015_Fan	Fantastic Beasts and Where to Find Them
Ro015_Gob	Goblet of Fire_Harry Potter
Ro015_Phi	Philosopher's Stone_Harry Potter
Ro015_Pri	Prisoner of Azkaban_Harry Potter
Sa001_Voy	Voyage of the Turtle
Sa001_Wha	What Animals Think
To015_Fel	Fellowship of the Rings_Lord of the Rings
Wo015_Wod	Wodehouse at the Wicket

Author_ID	Book_ID	Book
Da_001	Da001_Sel	Self Comes to Mind
Mi_009	Mi009_Emo	Emotion Machine
Mi_009	Mi009_Soc	Society of Mind
Ra_001	Ra001_Pha	Phantoms in the Brain
Ro_015	Ro015_Fan	Fantastic Beasts and Where to Find Them
Ro_015	Ro015_Gob	Goblet of Fire_Harry Potter
Ro_015	Ro015_Phi	Philosopher's Stone_Harry Potter
Ro_015	Ro015_Pri	Prisoner of Azkaban_Harry Potter
Sa_001	Sa001_Voy	Voyage of the Turtle
Sa_001	Sa001_Wha	What Animals Think
To_015	To015_Fel	Fellowship of the Rings_Lord of the Rings
Wo_015	Wo015_Wod	Wodehouse at the Wicket

If common attribute is CK then there is lossless decomposition

Book Purchase Detail Table

Book_ID	Purchase_Dt
Da001_Sel	Sep 1, 2021
Mi009_Emo	Sep 2, 2021
Mi009_Soc	Sep 1, 2021
Ra001_Pha	Sep 2, 2021
Ro015_Fan	Sep 1, 2021
Ro015_Gob	Sep 1, 2021
Ro015_Phi	Sep 1, 2021
Ro015_Pri	Sep 1, 2021
Sa001_Voy	Sep 2, 2021
Sa001_Wha	Sep 2, 2021
To015_Fel	Sep 1, 2021
Wo015_Wod	Sep 5, 2021

Book_ID	Copies
Da001_Sel	1
Mi009_Emo	2
Mi009_Soc	2
Ra001_Pha	2
Ro015_Fan	3
Ro015_Gob	3
Ro015_Phi	3
Ro015_Pri	3
Sa001_Voy	2
Sa001_Wha	2
To015_Fel	3
Wo015_Wod	1

Book_Purchase_Details:

Book_ID	Purchase_Dt	Copies
Da001_Sel	Sep 1, 2021	1
Mi009_Emo	Sep 2, 2021	2
Mi009_Soc	Sep 1, 2021	2
Ra001_Pha	Sep 2, 2021	2
Ro015_Fan	Sep 1, 2021	3
Ro015_Gob	Sep 1, 2021	3
Ro015_Phi	Sep 1, 2021	3
Ro015_Pri	Sep 1, 2021	3
Sa001_Voy	Sep 2, 2021	2
Sa001_Wha	Sep 2, 2021	2
To015_Fel	Sep 1, 2021	3
Wo015_Wod	Sep 5, 2021	1

Here also common attribute is CK , so no lossless decomposition. So given database is in 5NF.

b) In 1st table there are 2 candidate keys Book's name and BOOk_ID. My choice of Index is Book_ID because it has been encoded by taking author_Id and first three letters of a book's name which makes it more exclusive. Though in given database book's name don't repeat but still in general the book's name is made up of many word and those words have high chances of being in similar.

On the other hand if we look at Book_id its less likely for one author to write 2 books with same initials (though it is possible).

Also the length of book_Id is way smaller than length of book name.

In 2nd table PK is Author_ID as many authors can have same name. So my choice of index is Author_ID.

In 3rd table PK is Book_ID as many books can have same Book_Dt and copies. So my choice of index is Book_ID

C++ code extendible hashing:

```
8
9   string hash_func(string s){
10      string hash_ouput;
11      fr(0,s.length()){
12          int a = int(s[i]);
13          string binaryNum = "";
14          while (a > 0) {
15              binaryNum += (a % 2==0) ? '0' : '1';
16              a = a / 2;
17          }
18          reverse(binaryNum.begin(),binaryNum.end());
19          hash_ouput += binaryNum;
20
21      }
22      return hash_ouput;
23
24  }
25  struct Bucket{
26      int ld=0;
27      list<string> a;
28  };
29  map<string,Bucket *> bmap;
30  int gd=1,bucket_capacity=4;
```

```
C:\Users\PRAGATI SINHA\Desktop\code\c++ code>g++ extendible_ha

C:\Users\PRAGATI SINHA\Desktop\code\c++ code>a.exe
00 Da001_Sel To015_Fel Wo015_Wod
0 Da001_Sel
1 Mi009_Emo Mi009_Soc Ra001_Pha
0 Da001_Sel Ro015_Fan
1 Mi009_Emo Mi009_Soc Ra001_Pha
0 Da001_Sel Ro015_Fan Ro015_Gob
1 Mi009_Emo Mi009_Soc Ra001_Pha
0 Da001_Sel Ro015_Fan Ro015_Gob
1 Mi009_Emo Mi009_Soc Ra001_Pha Ro015_Phi
00 Da001_Sel Ro015_Fan Ro015_Gob
01 Ra001_Pha Ro015_Phi Ro015_Pri
10 Da001_Sel Ro015_Fan Ro015_Gob
11 Mi009_Emo Mi009_Soc
00 Da001_Sel Ro015_Fan Ro015_Gob
01 Ra001_Pha Ro015_Phi Ro015_Pri Sa001_Voy
10 Da001_Sel Ro015_Fan Ro015_Gob
11 Mi009_Emo Mi009_Soc
00 Da001_Sel Ro015_Fan Ro015_Gob To015_Fel
01 Ra001_Pha Ro015_Phi Ro015_Pri Sa001_Voy
10 Da001_Sel Ro015_Fan Ro015_Gob To015_Fel
11 Mi009_Emo Mi009_Soc
00 Da001_Sel To015_Fel Wo015_Wod
01 Ra001_Pha Ro015_Phi Ro015_Pri Sa001_Voy
10 Ro015_Fan Ro015_Gob
11 Mi009_Emo Mi009_Soc
```

xtendible_hashing.cpp > main()

```
void split(string sp_Index,Bucket * sp_Bucket){
    Bucket * newB = new Bucket;
    list<string> temp;
    for(auto i : sp_Bucket->a){
        temp.pb(i);
    }
    sp_Bucket->a.clear();
    if(sp_Bucket->ld == gd){
        if(gd!=0){
            vector<string> abc;
            for(auto itr : bmap){
                if(itr.first!=sp_Index){
                    abc.push_back(itr.first);
                }
            }
            fr(0,abc.size()){
                Bucket* b = bmap[abc[i]];
                bmap.erase(abc[i]);
                bmap["0"+abc[i]] = b;
                bmap["1"+abc[i]] = b;
            }
        }
        bmap["1"+sp_Index] = newB;
        bmap["0"+sp_Index] = sp_Bucket;
        bmap.erase(sp_Index);
        gd++;
        sp_Bucket->ld++;
```

extendible_hashing.cpp > split(string, Bucket *)

```
58          newB->ld = sp_Bucket->ld;
59      }
60      else{
61          bmap[sp_Index]  = newB;
62          bmap[sp_Index]->ld = sp_Bucket->ld;
63          bmap[sp_Index]->ld++;
64          sp_Bucket->ld++;
65      }
66      for(auto itr : temp){
67          string h = hash_func(itr);
68          string key;
69          int count = gd;
70          for(int i=h.length()-1;i>=0;i--){
71              key += h[i];
72              count--;
73              if(count == 0){
74                  break;
75              }
76          }
77          reverse(key.begin(),key.end());
78          (bmap[key]->a).pb(itr);
79      }
80  }
81
82  void insert(string s){
83      string h = hash_func(s);
84      string key;
```

```
void insert(string s){
    string h = hash_func(s);
    string key;
    int count = gd;
    for(int i=h.length()-1;i>=0;i--){
        key += h[i];
        count--;
        if(count == 0){
            break;
        }
    }
    reverse(key.begin(),key.end());


    if((bmap[key]->a).size() < bucket_capacity){
        (bmap[key]->a).pb(s);
    }

    else{
        split(key,bmap[key]);
        insert(s);
    }
}
```

C++ code to linear hashing

```
hash_index for Da001_Sel : 1
hash_index for To015_Fel : 2
hash_index for Mi009_Emo : 2
hash_index for Mi009_Soc : 0
hash_index for Ra001_Pha : 1
hash_index for Ro015_Fan : 1
hash_index for Ro015_Gob : 1
hash_index for Ro015_Pri : 2
hash_index for Sa001_Wha : 0
hash_index for Wo015_Wod : 0
```

```
Mi009_Soc  :  3
Sa001_Wha  :  0
Wo015_Wod  :  3
Sa001_Voy  :  0
```

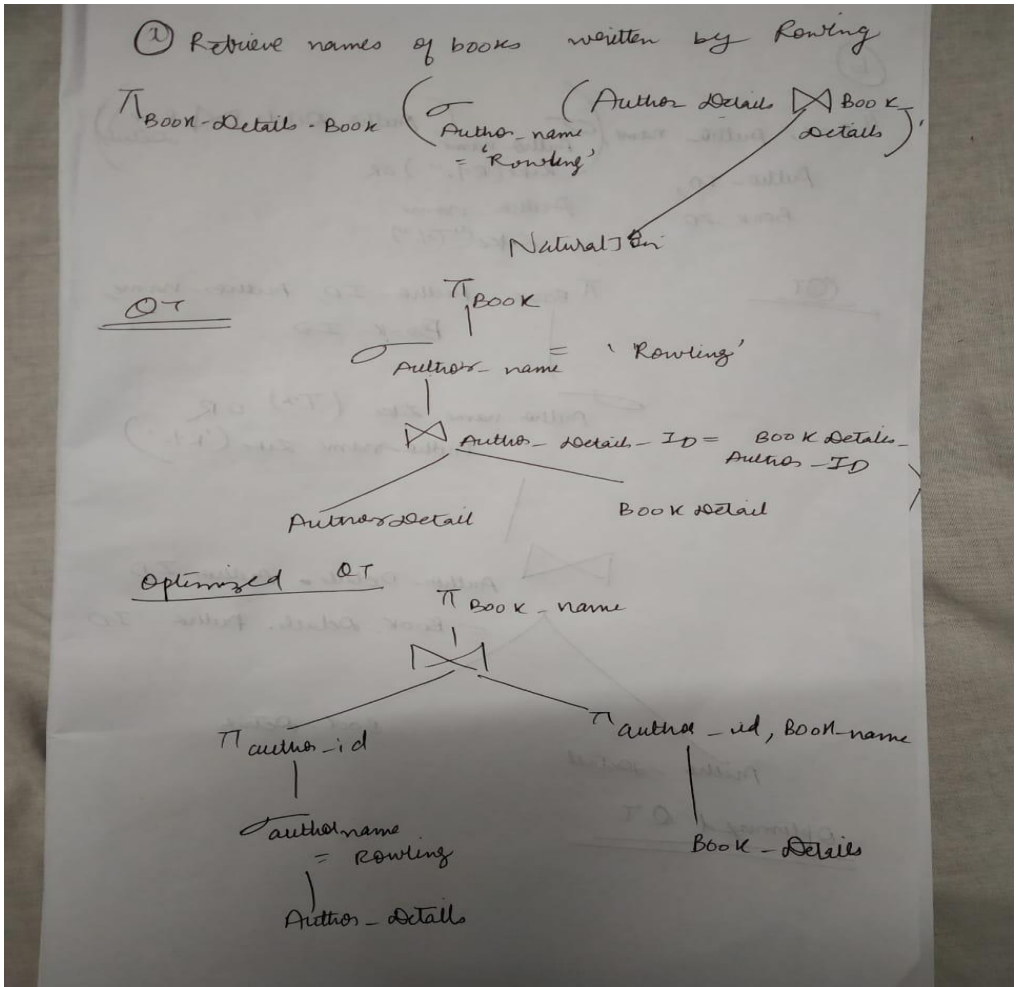
```
C:\Users\PRAGATI SINHA\Desktop\code\c++ code>a.exe
0
1 Da001_Sel
2
0
1 Da001_Sel
2 Mi009_Emo
0 Mi009_Soc
1 Da001_Sel
2 Mi009_Emo
1 Da001_Sel Ra001_Pha Ro015_Fan Ro015_Gob Ro015_Phi
2 Mi009_Emo Ro015_Pri
3 Mi009_Soc
0 Sa001_Voy Sa001_Wha
1 Da001_Sel Ra001_Pha Ro015_Fan Ro015_Gob Ro015_Phi
2 Mi009_Emo Ro015_Pri To015_Fel
3 Mi009_Soc
0 Sa001_Voy Sa001_Wha Wo015_Wod
1 Da001_Sel Ra001_Pha Ro015_Fan Ro015_Gob Ro015_Phi
2 Mi009_Emo Ro015_Pri To015_Fel
3 Mi009_Soc
```

```cpp
#include<bits/stdc++.h>
#define pb push_back
#define all(a) a.begin(),a.end()
#define cpresent(container,element) (find(all(container),element)!=container.end())
#define rep(i, a, b)    for(int i = a; i < b; ++i)
#define fr(a,b) for(int i = a; i < b; i++)
using namespace std;
struct Bucket{
    list<string> a;
};
map<int, Bucket *> bmap;
int gd=1,bucket_capacity=4;
int s = 0;
int size_of_map;

int hash_func(string s){
    int a = 0;
    fr(0,s.length()){
        a+= int(s[i]);
    }
    return a%3;
}
int hash_func2(string s){
    int a = 0;
    fr(0,s.length()){
        a+= int(s[i]);
    }
    return a%6;
}
```

```cpp
void split(){
    // split bucket s into s and size_of_map+1
    bmap[size_of_map+1] = new Bucket;
    size_of_map++;
    list<string> temp;
    for(auto itr : bmap[s]->a){
        temp.push_back(itr);
    }
    bmap[s]->a.clear();
    for(auto itr : temp){
        int x = hash_func2(itr);
        (bmap[x]->a).push_back(itr);
    }
    s++;
}

void insert(string s){
    int h = hash_func(s);
    if((bmap[h]->a).size() < bucket_capacity){
        (bmap[h]->a).pb(s);
    }
    else{
        (bmap[h]->a).pb(s);
        split();
    }
}
```

```
56
57  int main(){
58      vector<string> v = {"Da001_Sel","Mi009_Emo","Mi009_Soc","Ra001_Pha","Ro015_Fan","Ro015_Gob","Ro015_Phi","Ro015_Pri","Sa001_Voy",
59                          "Sa001_Wha","To015_Fel","Wo015_Wod"};
60
61      bmap[0] = new Bucket;
62      bmap[1] = new Bucket;
63      bmap[2] = new Bucket;
64      size_of_map = 2;
65      fr(0,v.size()){
66          insert(v[i]);
67          for(auto itr : bmap){
68              cout<<itr.first<<" ";
69              for(auto itr1 : itr.second->a){
70                  cout<<itr1<<" ";
71              }
72              cout<<"\n";
73          }
74      }
75
76  }
```

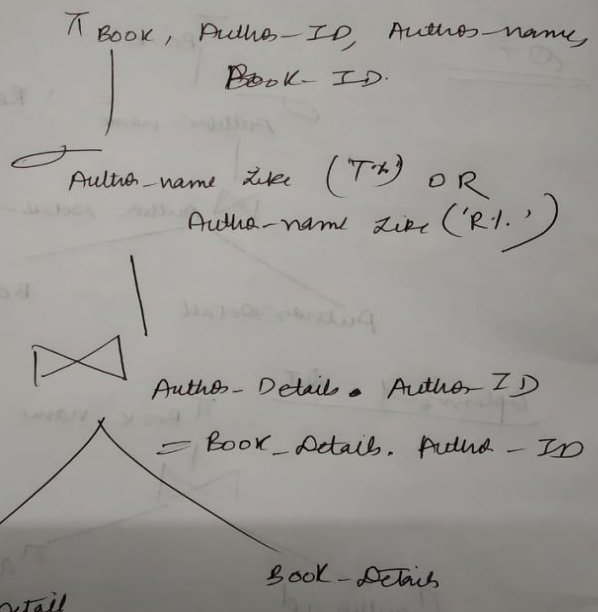
RA expression query tree and optimized query tree for all the Queries.



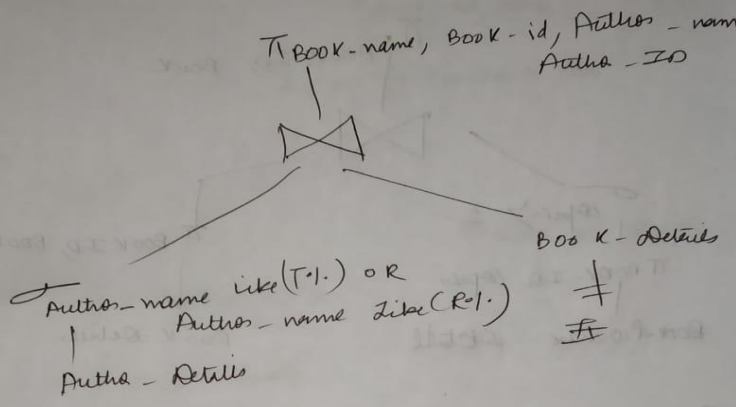
(b)

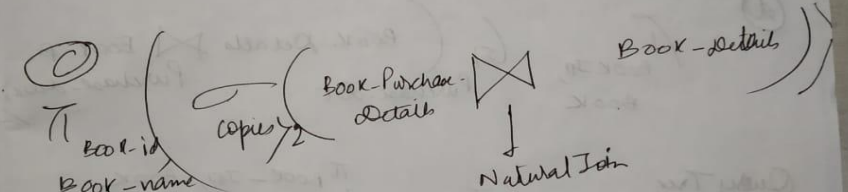
$\pi_{Book, Author\_name, Author\_ID, Book\_ID}\left(\sigma_{Author\_name\ like('R\%')\ OR\ Author\_name\ like('T\%')}\left(Author\ Details \bowtie Book\ Details\right)\right)$

QT

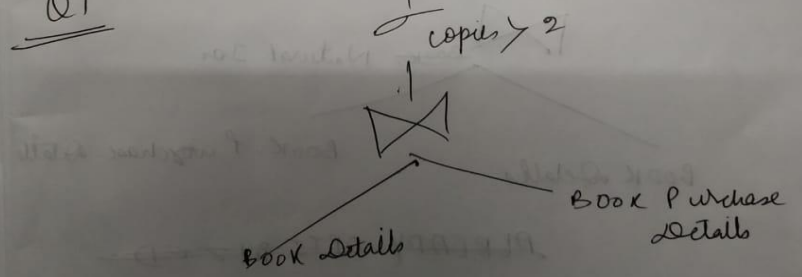


Optimized QT

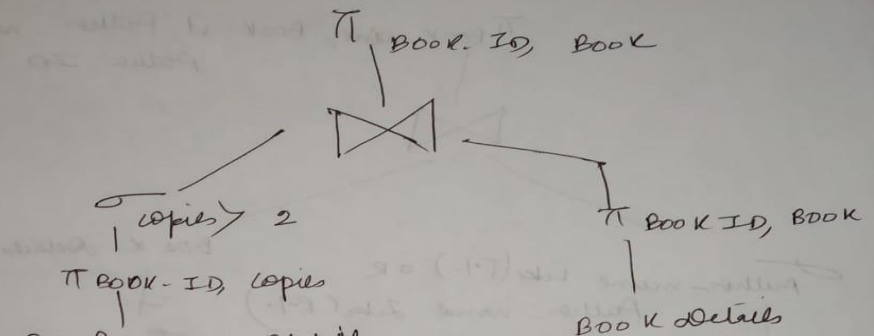




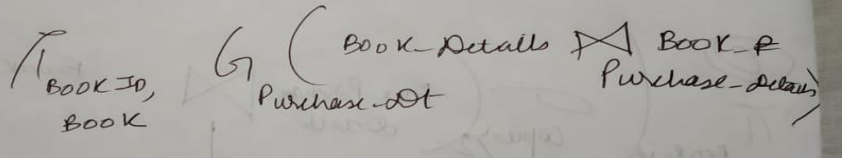
QT



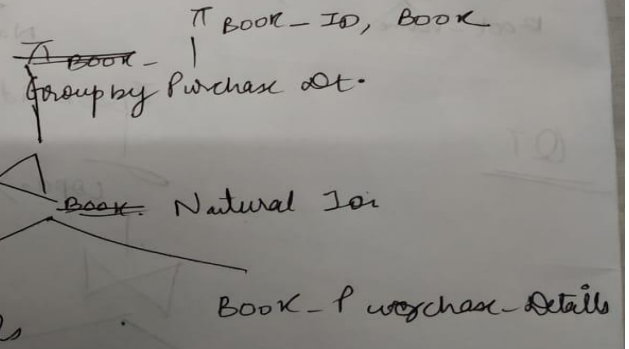
Optimized Q7



(a)

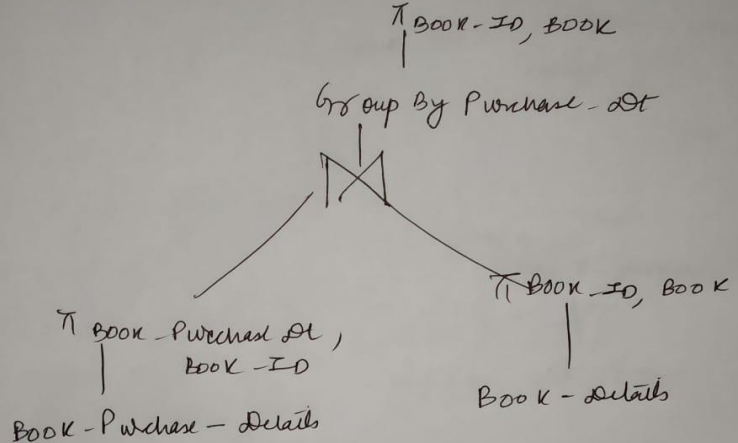


Query Tree



ALREADY OPTIMIZED

Optimized Q7



c,d,e

Query a:

First join operation (Natural Join) is performed then select operation is performed.

In optimized query tree as join a larger table takes a lot of amount of time we can first perform select as all other rows of the table will get removed then .

Then we can remove all the unnecessary attributes which won't be used further , here it is author name.

As from book _Detail table only book, book_id is req other unnecessary attribute can be removed.

After that join and projection would be much faster

In both the hashing technique first tables are joined. Then all the book_ids which contains the author name as Rowling can be searched and stored in array . Then search for find data can be done by :

Time complexity analysis (extendible hashing)

Here 1st Book_Id will be converted into binary (every character is given ASCII val, Ascii value is converted to binary then added to the string) , then LSBs according to Global depth are taken for keys. THis will take O(length of Book id time) . After that from the bucket using key that data can be found in O(bucket _ size) time.

Time complexity analysis (linear hashing)

Here 1st Book_Id will be converted into binary (every character is given ASCII val, Ascii value is converted to integer and then added to the final total) . THis final total modulo 3 is the key. THis will take O(length of Book id time) . After that from the bucket using key that data can be found in O(overflow bucket _ size) time.

Query b :

In 1st query Author Detail table and book detail table are natural join then selection according to condition then Projection of attribute.

In optimized query tree as join a larger table takes a lot of amount of time we can first perform select as all other rows of the table will get removed then .[Selection performed on author detail table]

After that join and projection would be much faster.

Time complexity analysis:

Similarly as query A just there will be slight difference in final condition i.e. instead of checking of name of author first letter with a ‘OR’ condition will be checked.

c) First book purchase detail and book detail is natural join then copies> 2 is selected then projection of attributes req is done.

In optimized query tree Book_id , copies attribute is selected from book purchase details and book id , book from book details followed by selection then join then projection.

Time complexity analysis

Hera also only slight changes from 1st 2 queries . First through linear search all the Book_ID with copies > 2 can be stored then final searching similar to described in query a.

d) First natural Join book purchase detail and book detail table then group by purchase date the project book_ID and book name.

As groups can only be made on final table here we can’t do first group then join. However in order to further optimize the query . first book purchase date and book id can be projected from Book purchase Detail table and simultaneously Book_id and book from Book_Details table.

Time complexity analysis

A unordered_map<date,linked_list> can be used for grouping it will take O(length of table) time. Then all The book ids can be searched similar to query a .