DBMS LAB 4

Ridhima Kohli
B19CSE071

1. Consider the following partial tables of an ordered library catalogue:

Book_Details:

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

Author_Details:

Author_ID	Author_Name
Da_001	Damasio
Mi_009	Minsky
Ra_001	Ramachandran
Ro_015	Rowling
Ru_021	Russel
Sa_001	Safina
Ta_001	Tagore
To_015	Tolkien
Wo_015	Wodehouse

Book_Purchase_Details:

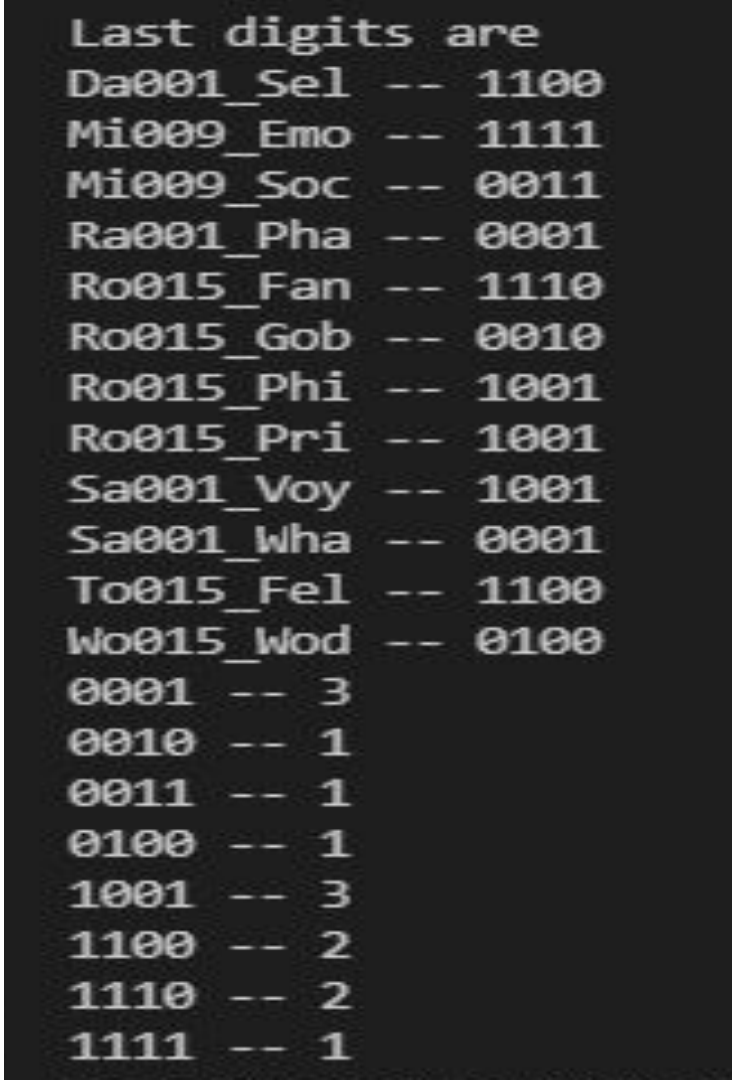
Book_ID	Purchase_Dt	Copies
Da001_Sel	Sep 1, 2021	1
Mi009_Emo	Sep 2, 2021	2
Mi009_Soc	Sep 1, 2021	2
Ra001_Pha	Sep 2, 2021	2
Ro015_Fan	Sep 1, 2021	3
Ro015_Gob	Sep 1, 2021	3
Ro015_Phi	Sep 1, 2021	3
Ro015_Pri	Sep 1, 2021	3
Sa001_Voy	Sep 2, 2021	2
Sa001_Wha	Sep 2, 2021	2
To015_Fel	Sep 1, 2021	3
Wo015_Wod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

Extendible Hashing

```
 We have entry : Da001_Sel

last 1 digit(s) : 0
0 - 1:Da001_Sel
1 -
 We have entry : Mi009_Emo

last 1 digit(s) : 1
0 - 1:Da001_Sel
1 - 2:Mi009_Emo
 We have entry : Mi009_Soc

last 1 digit(s) : 1
0 - 1:Da001_Sel
1 - 2:Mi009_Emo  3:Mi009_Soc
 We have entry : Ra001_Pha

last 1 digit(s) : 1
0 - 1:Da001_Sel
1 - 2:Mi009_Emo  3:Mi009_Soc  4:Ra001_Pha
 We have entry : Ro015_Fan

last 1 digit(s) : 0
0 - 1:Da001_Sel  5:Ro015_Fan
1 - 2:Mi009_Emo  3:Mi009_Soc  4:Ra001_Pha
 We have entry : Ra001_Pha

last 1 digit(s) : 1
0 - 1:Da001_Sel  5:Ro015_Fan
1 - 2:Mi009_Emo  3:Mi009_Soc  4:Ra001_Pha  6:Ra001_Pha
 We have entry : Ro015_Fan

last 1 digit(s) : 0
0 - 1:Da001_Sel  5:Ro015_Fan  7:Ro015_Fan
1 - 2:Mi009_Emo  3:Mi009_Soc  4:Ra001_Pha  6:Ra001_Pha
 We have entry : Ro015_Gob
```

```
0 - 1:Da001_Sel  5:Ro015_Fan  7:Ro015_Fan
1 - 2:Mi009_Emo  3:Mi009_Soc  4:Ra001_Pha  6:Ra001_Pha
 We have entry : Ro015_Gob

last 1 digit(s) : 0
0 - 1:Da001_Sel  5:Ro015_Fan  7:Ro015_Fan  8:Ro015_Gob
1 - 2:Mi009_Emo  3:Mi009_Soc  4:Ra001_Pha  6:Ra001_Pha
 We have entry : Ro015_Phi

last 1 digit(s) : 1
Bucket is full ! Extend global directory
Redistribute the values in slot 1 of global directory to new slots
 We have entry : Mi009_Emo

last 2 digit(s) : 11
0 - 1:Da001_Sel  5:Ro015_Fan  7:Ro015_Fan  8:Ro015_Gob
1 - 3:Mi009_Soc  4:Ra001_Pha
2 -
3 - 2:Mi009_Emo
 We have entry : Mi009_Soc

last 2 digit(s) : 11
0 - 1:Da001_Sel  5:Ro015_Fan  7:Ro015_Fan  8:Ro015_Gob
1 -
2 -
3 - 2:Mi009_Emo  3:Mi009_Soc
 We have entry : Ra001_Pha
```

Since bucket was full so slots were extended to higher bit



4 bits are needed for accomodation

The given output displays the number of Book_IDs which belong to respective LSBs

Extendible hashing code

```
#include <iostream>
#include <bits/stdc++.h>
#include <stdlib.h>
using namespace std;

struct globalDirectory{
    int index;
    int bucket[4];
    vector<string> Sbucket;
    int bfull;
};
struct globalDirectory *T1;
int gd=2;
int b=1;
```

```
int binToDec(string n)
{    string num = n;
    int dec_value = 0;
    // Initializing base value to 1, i.e 2^0
    int base = 1;
    int len = num.length();
    for (int i = len - 1; i >= 0; i--) {
        if (num[i] == '1')
            dec_value += base;
        base = base * 2;    }
    return dec_value;}
void printTable(struct globalDirectory* T){
for(int i=0;i<gd;i++){
    cout<<T[i].index<<" - ";
    for(int bu=0;bu<T1[i].bfull;bu++){
        if(T[i].bucket[bu]!=-1)
    cout<<T[i].bucket[bu]<<":"<<T[i].Sbucket[bu]<<"
";    }    cout<<endl;}}
```

```
void ConverToBinary(string s,int entrynum)
{       cout<<" We have entry : "<<s<<endl;
    int n = s.length();
int v = 0;
string bins="";
    for (int i = 0; i <= n; i++)
    { int val = int(s[i]);
        v+=val;
        string bin = "";
        while (val > 0)
        {              (val % 2)?
bin.push_back('1') :
                    bin.push_back('0');
            val /= 2;          }
        reverse(bin.begin(), bin.end());
        bins+=bin;     }
    reverse(bins.begin(), bins.end());
    cout<<"\n";
```

```
  string lsb = bins.substr(0 , b);
     reverse(lsb.begin(), lsb.end());
    cout<<"last "<<b<<" digit(s) : "<<lsb<<"\n";
int declsb = binToDec(lsb);
for(int i=0;i<gd;i++){     if(T1[i].index==declsb &&
T1[i].bfull<4){         T1[i].bucket[T1[i].bfull]=entrynum;
        T1[i].Sbucket.push_back(s);
             T1[i].bfull++;
             printTable(T1);
             break;     }
    else if(T1[i].index==declsb && T1[i].bfull==4){
        cout<<"Bucket is full ! Extend global directory\n";
        cout<<"Redistribute the values in slot "<<i<<" of
global directory to new slots\n";
            b++;
            gd+=gd+b-2;
            for(int t=0;t<4;t++){
                int ent = T1[i].bucket[t];
                string st = T1[i].Sbucket[t];
                T1[i].bucket[t] = -1;
                T1[i].Sbucket[t]="";
                T1[i].bfull--;
                 ConverToBinary(st,ent); }}}}
```

Recursive call with updated bits and global depth

```
int main(){
T1=(struct globalDirectory*)malloc(sizeof(struct
globalDirectory)*50);

for(int g=0;g<50;g++){
    T1[g].bfull=0;
    T1[g].index=g;}
T1[0].index=0;
T1[1].index=1;
string colour[] = {"Da001_Sel" , "Mi009_Emo",
"Mi009_Soc" , "Ra001_Pha" ,
"Ro015_Fan","Ra001_Pha","Ro015_Fan","Ro015_Gob","Ro
015_Phi" ,"Ro015_Pri" ,"Sa001_Voy" ,"Sa001_Wha
","To015_Fel","Wo015_Wod"};
for(int sn=0;sn<12;sn++){
    ConverToBinary(colour[sn],sn+1);
}
    return 0;
}
```

Similarly other tables are made

```
ASCII value of Da001_Sel is 108
Hash function : h1 = key % 3 gives hashKey = 0 key name : Da001_Selkey id :1
0 --> 1
1 -->
2 -->
ASCII value of To015_Fel is 108
Hash function : h1 = key % 3 gives hashKey = 0 key name : To015_Felkey id :2
0 --> 1 2
1 -->
2 -->
ASCII value of Mi009_Emo is 111
Hash function : h1 = key % 3 gives hashKey = 0 key name : Mi009_Emokey id :3
0 --> 1 2 3
1 -->
2 -->
ASCII value of Mi009_Soc is 99
Hash function : h1 = key % 3 gives hashKey = 0 key name : Mi009_Sockey id :4
0 --> 1 2 3 4
1 -->
2 -->
ASCII value of Ra001_Pha is 97
Hash function : h1 = key % 3 gives hashKey = 1 key name : Ra001_Phakey id :5
0 --> 1 2 3 4
1 --> 5
2 -->
ASCII value of Ro015_Fan is 110
Hash function : h1 = key % 3 gives hashKey = 2 key name : Ro015_Fankey id :6
0 --> 1 2 3 4
1 --> 5
2 --> 6
ASCII value of Ro015_Gob is 98
Hash function : h1 = key % 3 gives hashKey = 2 key name : Ro015_Gobkey id :7
0 --> 1 2 3 4
1 --> 5
2 --> 6 7
ASCII value of Ro015_Pri is 105
Overflow Alert : slot 0 's bucket is full
Overflow happens , Split slot 0 into slots 0 -- 3
New Hash function : h2 = key % 4 gives hashKey = 3for slot 0 and 3
0 --> 1 2
1 --> 5
2 --> 6 7
3 --> 3 4 8
ASCII value of Sa001_Wha is 97
```

Implementation of linear hashing

An array of structs was made where each element represents a slot and each slot has various entities like id (taken as 1 , 2 , 3 ..etc for simplicity of displaying) , bucket , overflow bucket , each of size 4 , etc as shown in the next slide

Overflow adjusted

Linear hashing code

```
#include <iostream>
#include <bits/stdc++.h>
#include <stdlib.h>
using namespace std;

struct Slot{
    int slotNo; //slot number
    int bucketFilled; // number of bucket components filled in original bucket
    int bucket[4]; //original bucket
    int asc[4]; //stores ascii value of original-bucket-inserted BookID which can be later
extracted while rehashing
    int overflowFill; //stores count of overflow bucket components
    int ovasc[4]; //stores ascii value of overflow-bucket-inserted BookID which can be later
extracted while rehashing
    int overflowBucket[4]; //overflow bucket
    int hashFunction; //stores hash function applicable for slot
};
struct Slot *Storage;
```

```
int toSplit = 0;
int extended=3;
int over=0;
void hash(string s , int x)
{   int n = s.length();
    int v = 0;
    for (int i = 8; i <= n; i++)
    {
        int val = int(s[i]);
        v+=val;
    }
    cout<<"ASCII value of "<<s<<" is "<<v<<endl;
    int m = v%3; //first hash function
    if(Storage[m].bucketFilled!=4){
        cout<<"Hash function : h1 = key % 3 gives hashKey = "<<m<<" key name : "<<s<<"key id
:"<<x<<endl;
        Storage[m].bucket[Storage[m].bucketFilled] = x;
        Storage[m].asc[Storage[m].bucketFilled] = v;
        Storage[m].bucketFilled++;}
```

```
  else{ //overflow
      over++;
      cout<<"Overflow Alert : slot " <<m<<" 's bucket is full \n";
        cout<<"Overflow happens , Split slot " <<toSplit<<" into slots "<<toSplit<<" -- "<<extended<<endl;
        int m2=v%6;
        cout<<"New Hash function : h2 = key % 4 gives hashKey = "  <<m2<< "for slot "<<toSplit<<" and "<<extended<<endl;
        vector<vector< int> > newMapping(2);
        Storage[toSplit].hashFunction=2;
        Storage[extended].hashFunction=2;
            //new storage location is m2
            //rehash toSplit into toSplit and extended
            for(int SplitBind=0;SplitBind<4;SplitBind++){
              int vasc = Storage[toSplit].asc[SplitBind];
              int newSlot = vasc%6;
              if(newSlot==toSplit)
                newMapping[0].push_back(Storage[toSplit].bucket[SplitBind]);
                else
                 newMapping[1].push_back(Storage[toSplit].bucket[SplitBind]);
           Storage[toSplit].bucket[SplitBind]=-1;
              Storage[toSplit].bucketFilled--;            }
if(m==toSplit){ //if overflow is in slot to be split take new hash function
               if(m2==toSplit)
                newMapping[0].push_back(x);
                else
                 newMapping[1].push_back(x);}
else{ //put in overflow bucket
    Storage[m].overflowBucket[Storage[m].overflowFill] = x;
        Storage[m].overflowFill++;}
```

```
        for(int r=0;r<newMapping[0].size();r++){
            Storage[toSplit].bucket[r] = newMapping[0][r];
        }

         for(int r=0;r<newMapping[1].size();r++){
            Storage[extended].bucket[r] = newMapping[1][r];
        }

        // rehash

        toSplit++;
        extended++;
    }
    for(int st=0;st<3+over;st++){
        cout<<st<<" --> ";
        for(int b=0;b<4;b++){
 if(Storage[st].bucket[b]!=-1)
    cout<<Storage[st].bucket[b]<<" ";
}cout<<endl;}}
```

```
int main(){
Storage = (struct
Slot*)malloc(sizeof(struct Slot)*6);
for(int i=0;i<6;i++){
    Storage[i].slotNo=i;
     Storage[i].bucketFilled=0;
      Storage[i].overflowFill=0;
       Storage[i].hashFunction=1;
       for(int j=0;j<4;j++){
Storage[i].bucket[j]=-1;

Storage[i].overflowBucket[j]=-1;
       }}
    string Book_ID_1 ="Da001_Sel";
    string Book_ID_2="To015_Fel";
    string Book_ID_3="Mi009_Emo"; //
    string Book_ID_4="Mi009_Soc";
    string Book_ID_5="Ra001_Pha";
    string Book_ID_6="Ro015_Fan";
```

```
string Book_ID_7="Ro015_Gob";
   string Book_ID_8="Ro015_Pri";
   string Book_ID_9="Sa001_Wha"; //
   string Book_ID_10="Wo015_Wod"; //
   string b = "Sa001_Voy"; //
   string c = "Ro015_Phi";
   hash(Book_ID_1,1);
   hash(Book_ID_2,2);
   hash(Book_ID_3,3);
   hash(Book_ID_4,4);
   hash(Book_ID_5,5);
   hash(Book_ID_6,6);
   hash(Book_ID_7,7);
   hash(Book_ID_8,8);
   hash(Book_ID_9,9);
   hash(Book_ID_10,10);
   hash(b,11);
    hash(c,12);
   return 0;}
```

Keys and Dependencies

For the given tables , the indexes taken are Book_ID , Author_ID and Book_ID respectively as they help in unique identification of entity

Table	Keys	Dependencies
Book_Details	CK : Book_ID , Book PA : Book_ID , Book NPA : Author_ID	Book_ID-> Book Book_ID->Author_ID Book-> Author_ID
Author_Details	CK: Author_ID PA : Author_ID NPA : Author_Name	Author_ID -> Author_Name
Book_Purchase_Details	CK: Book_ID PA : Book_ID NPA : Purchase_Date , Copies	Book_ID -> Copies Book_ID -> Purchase_Date

Normalization check

Normal form	1NF	2NF	3NF	BCNF	4NF	5NF
Is in normal form	Yes	Yes	Yes	Yes	Yes	Lets check on next slide
Reason	No multivalued attribute	No partial dependency	No transitive Dependency	LHS of all dependencies is Candidate Key	No multivalued dependency	

Book details

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

No lossless decomposition

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

Book_Details:

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

Book_Purchase_details

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

No lossless decomposition

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

Hence the database is in 5nf

1. Consider the following partial tables of an ordered library catalogue:

Book_Details:

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

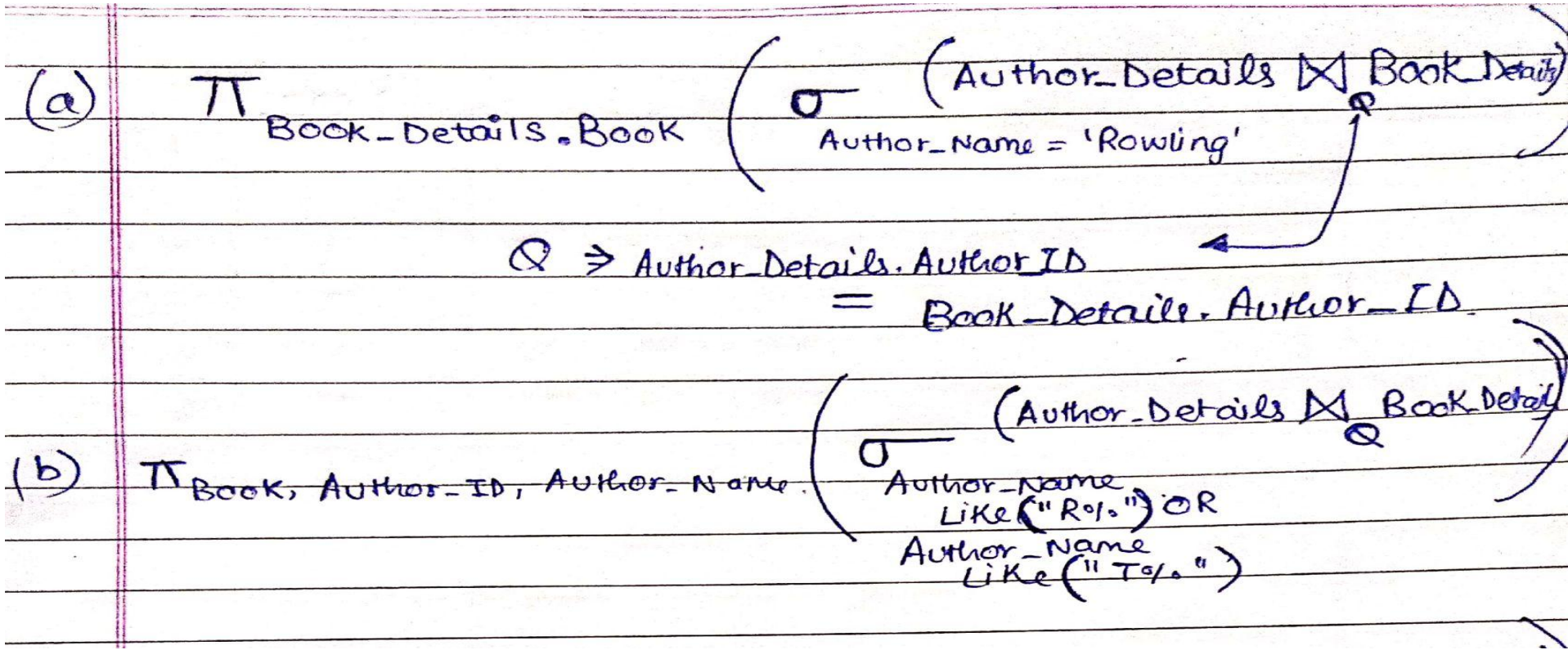
Author_Details:

Author_ID	Author_Name
Da_001	Damasio
Mi_009	Minsky
Ra_001	Ramachandran
Ro_015	Rowling
Ru_021	Russel
Sa_001	Safina
Ta_001	Tagore
To_015	Tolkien
Wo_015	Wodehouse

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

a. Retrieve names of books written by 'Rowling'
b. Retrieve book names and author details of all books written by authors with names beginning with 'R' or 'T'



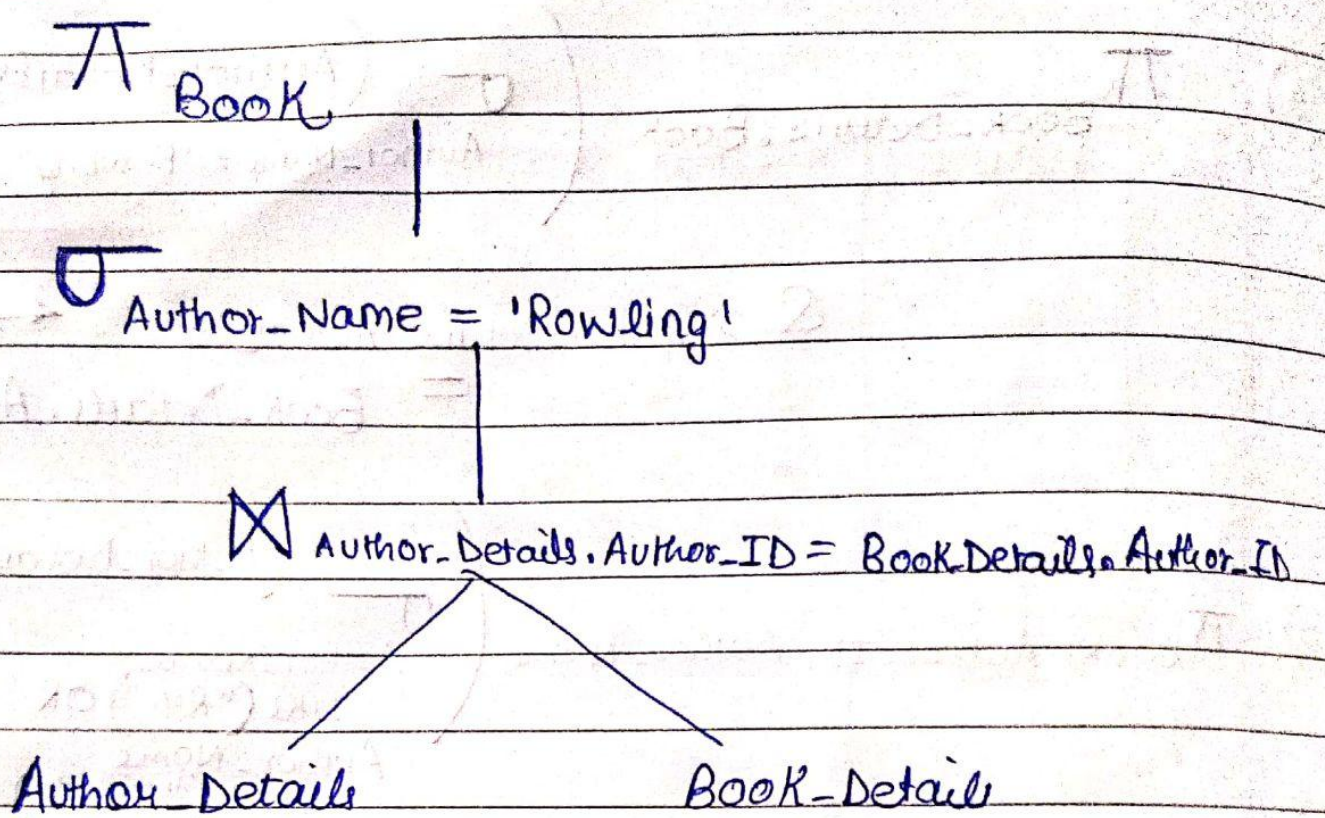
c. Retrieve all books with more than 2 copies
d. Retrieve book_ids and book_names of all books that were purchased on the same date

(c) $\pi_{Book}\left(\sigma_{Copies>2}\left(Book\_Purchase\_Details \bowtie_{c} Book\_Details\right)\right)$

$c \Rightarrow Book\_Purchase\_Details.Book\_ID = Book\_Details.Book\_ID$.

(d) $\pi_{Book\_ID,\ Book}\ G_{Purchase\_Dt}\left(Book\_Purchase\_Details \bowtie_{c} Book\_Details\right)$

a)


π Book
σ Author_Name = 'Rowling'
⋈ Author_Details.Author_ID = Book_Details.Author_ID
Author_Details
Book_Details

(b)

$\Pi_{Book, Author\_ID, Author\_Name}$

|

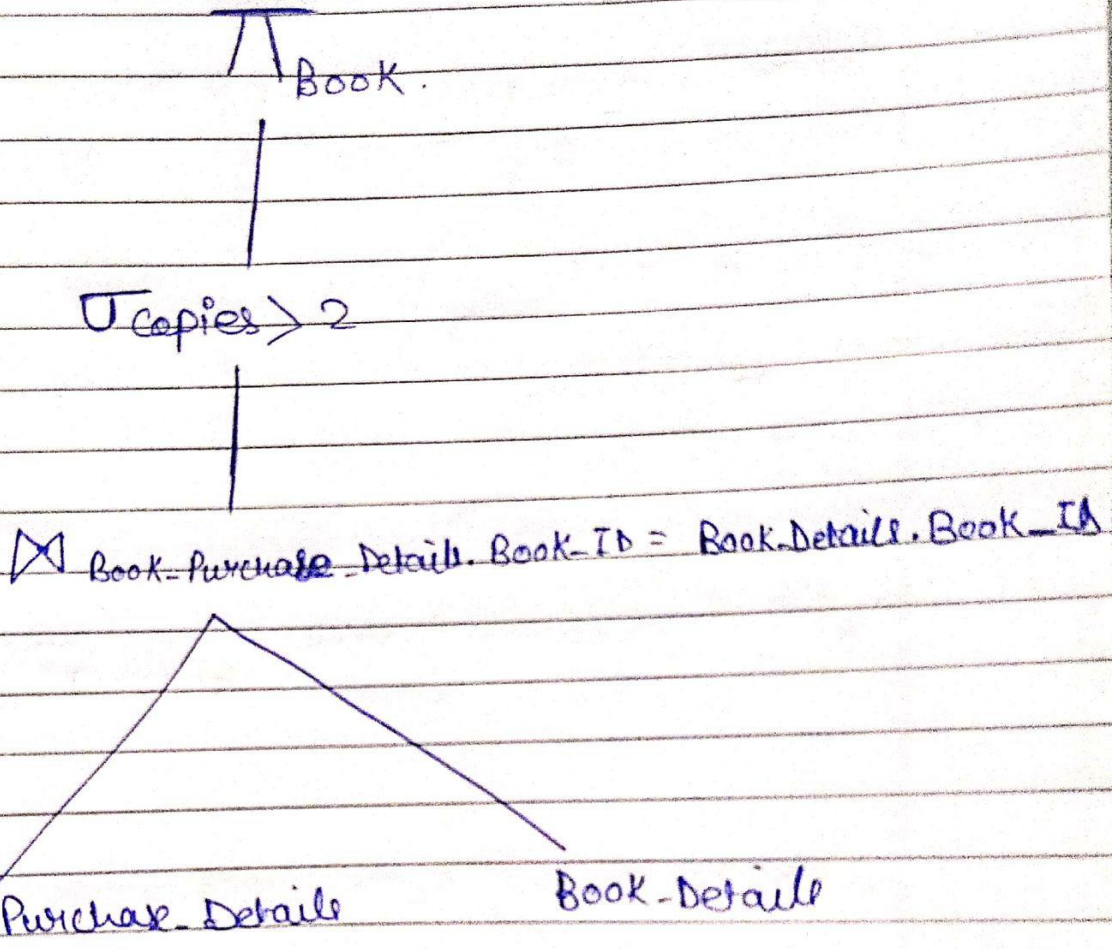
$\sigma_{Author\_Name\ Like("R\%")\ OR\ Author\_Name\ Like("T\%")}$

|

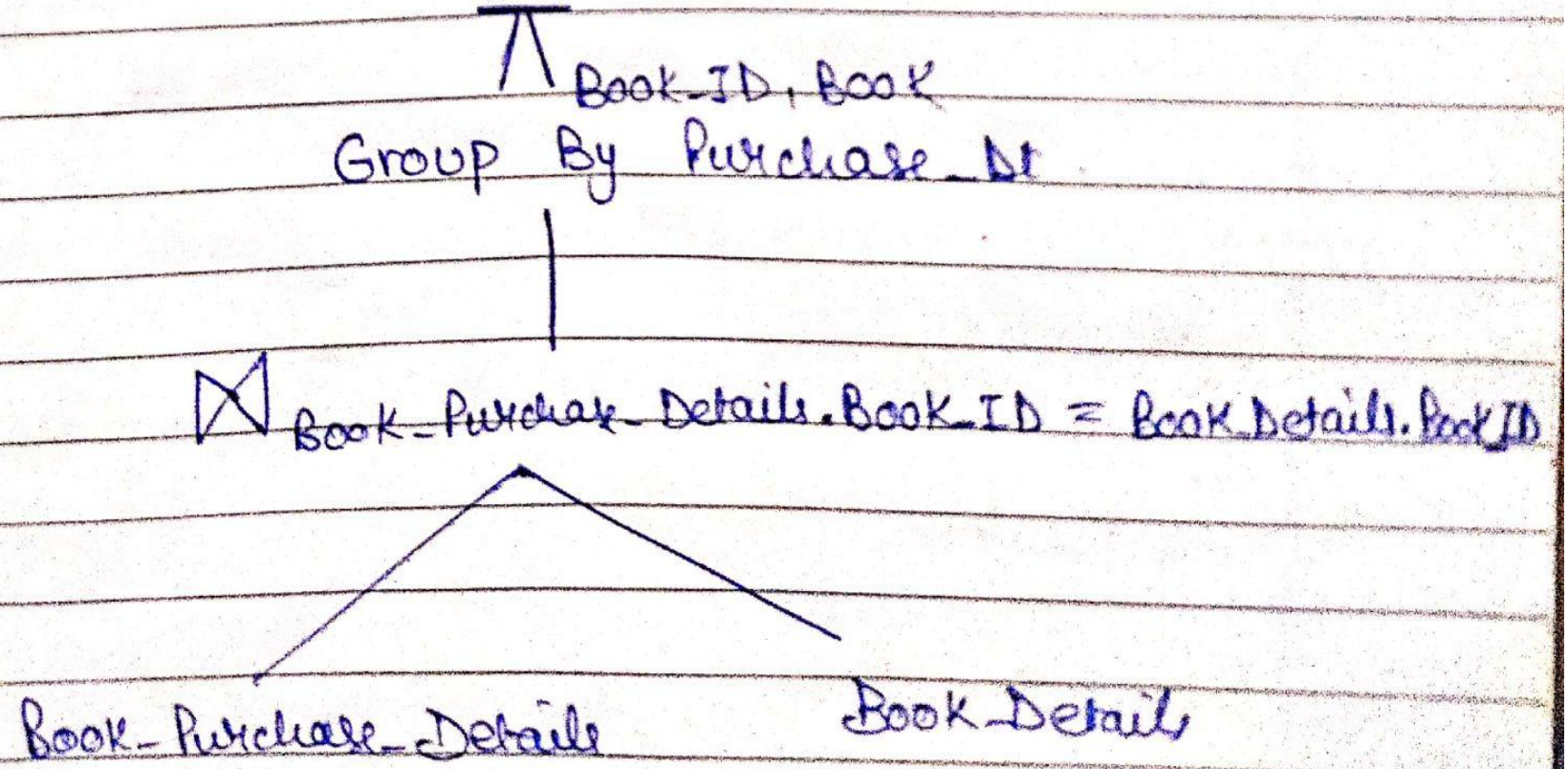
$\bowtie_{Author\_Details.Author\_ID = Book\_Details.Author\_ID}$

Author_Details Book_Details

(c)

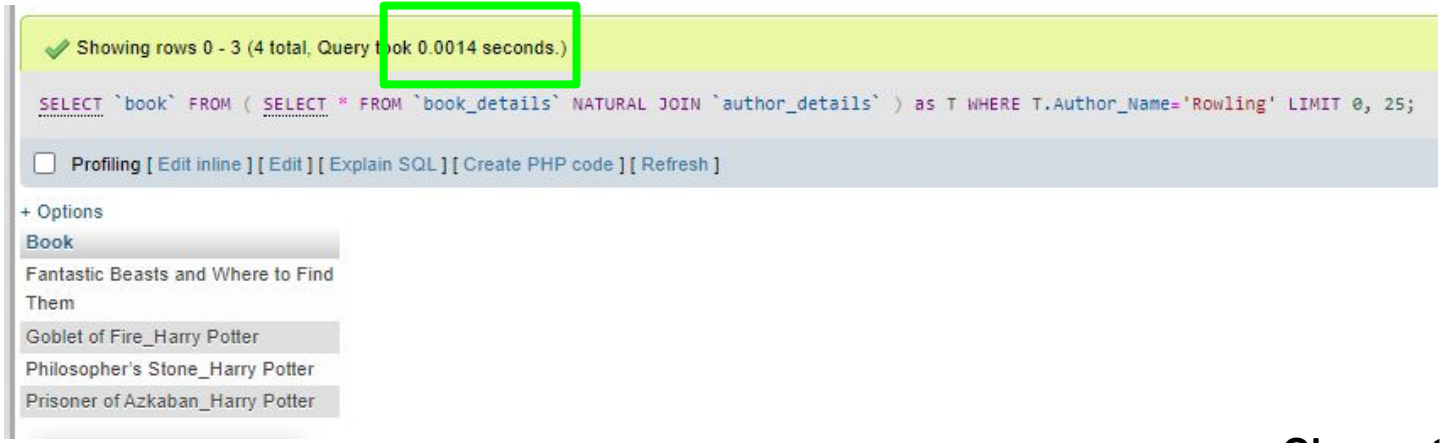

Book.
Copies > 2
Book_Purchase_Details.Book_ID = Book_Details.Book_ID
Book_Purchase_Details
Book_Details

(d)

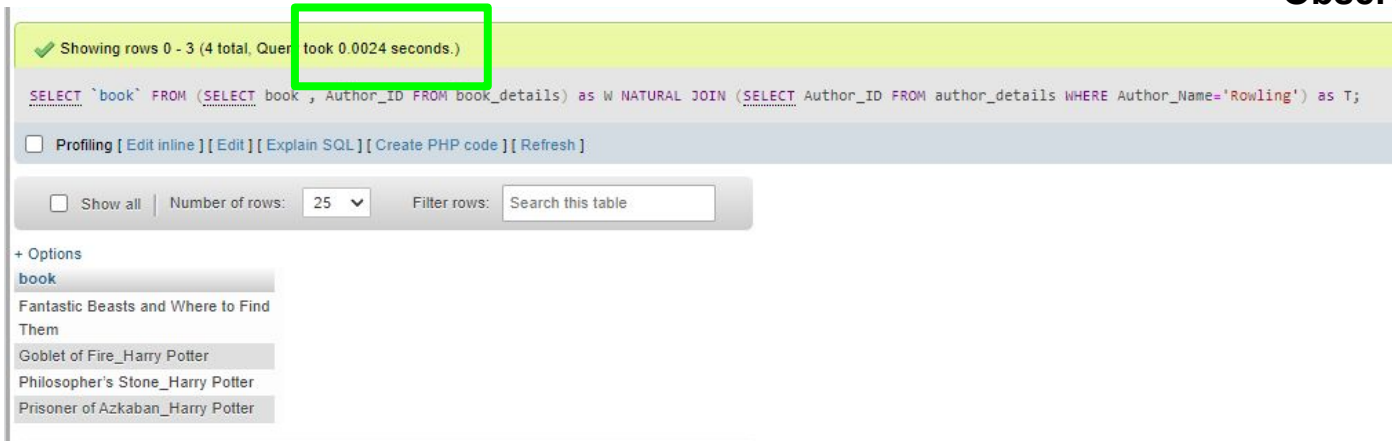

Book_ID, Book
Group By Purchase_Dt
Book_Purchase_Details.Book_ID = Book_Details.Book_ID
Book_Purchase_Details
Book_Details

Comparison of design per query & optimization

Query	Comments	Further optimization ?
Query a	The joint operation performs join on all attributes and author names and after it we select the author name.	To **save the unnecessary joining of other attributes and entities** , lets just take only those where author name is Rowling So we can try to have optimized query order : Select Author_IDs from Author_Details where Author_Name = “Rowling” Let its result be t1 And Select Author_ID and Book from Book_Details table Let it be t2 After this , Join entities of t1 and t2 where author_id is same Now , retrieve the names of all books in resultant table **But** again increasing the number of select statement increases the query execution time - as observed in the screenshots



Observation



Query	Comments	Further optimization ?
Query b	The joint operation performs join on all attributes and entities	Similar to query a , to **save the unnecessary joining of other attributes and entities** ,we can first select the corresponding author ids of authors whose names start with R or T and then join and project required details. Similar to query a , to **save the unnecessary joining of other attributes and entities** ,we can first select the corresponding book ids where copies > 2 and then join the required **However subqueries are slower than using join as shown in the previous slide , hence it would be better to use join**
Query c		
Query d	The query seems to be optimized	-

Time Complexity analysis for queries on both hashings

a. Retrieve names of books written by ‘Rowling’

Only those values where Author_IDs match in both tables are joined. After this , search for Rowling in Author Name attribute begins. The Book_ID which contains “Rowling” is then searched for. Since Book_ID is in hashed Book , the method of search will differ here.

Extendible Hashing

In extendible hashing Book_ID is first converted to binary and then its last b LSBs (b = number of bits taken on the directory) are compared. The search for these LSBs matches with one of the global directories in O(1) time and the corresponding Bookname is retrieved by a linear search in the bucket. Hence O(4) =O(1) time since bucket size is fixed

Linear Hashing

In linear hashing Book_ID is first converted to ascii and then hash function is applied to get its slot. If hash function for the resulting slot has been modified , the new hash result slot is chosen. This slot is then searched for the value of corresponding Book_ID and the book name is retreived. Taking O(n) time where n = size of bucket or overflow bucket which is 4 here too

b. Retrieve book names and author details of all books written by authors with names beginning with ‘R’ or ‘T’

Similar search as query a , just we need to search for R and T separately or we can also make an array and store the corresponding author_IDs in a single search by applying OR condition to the if statement used for query

c. Retrieve all books with more than 2 copies

Linear search for checking copies value which satisfy the given condition and store the corresponding Book_IDs in an array. Now search for the required values as described in query a

d. Retrieve book_ids and book_names of all books that were purchased on the same date
Here a multi-map can be used for storing corresponding Book-IDs mapped to date

Multimap : { {1/9/2002 : book_id1 , book_id5} , {2/9/2002 : book_id4 , book_id3} ….}

Iterate over the dates and for each date search for corresponding book name and display the query results grouped by dates.

Conclusion

- Even though joins are costly , creating subqueries by select increase the time of execution
  - As observed in query a , the time taken were
    - Subquery : 0.0024 seconds
    - Join : 0.0014 seconds

This difference will increase if the amount of data in tables is increased