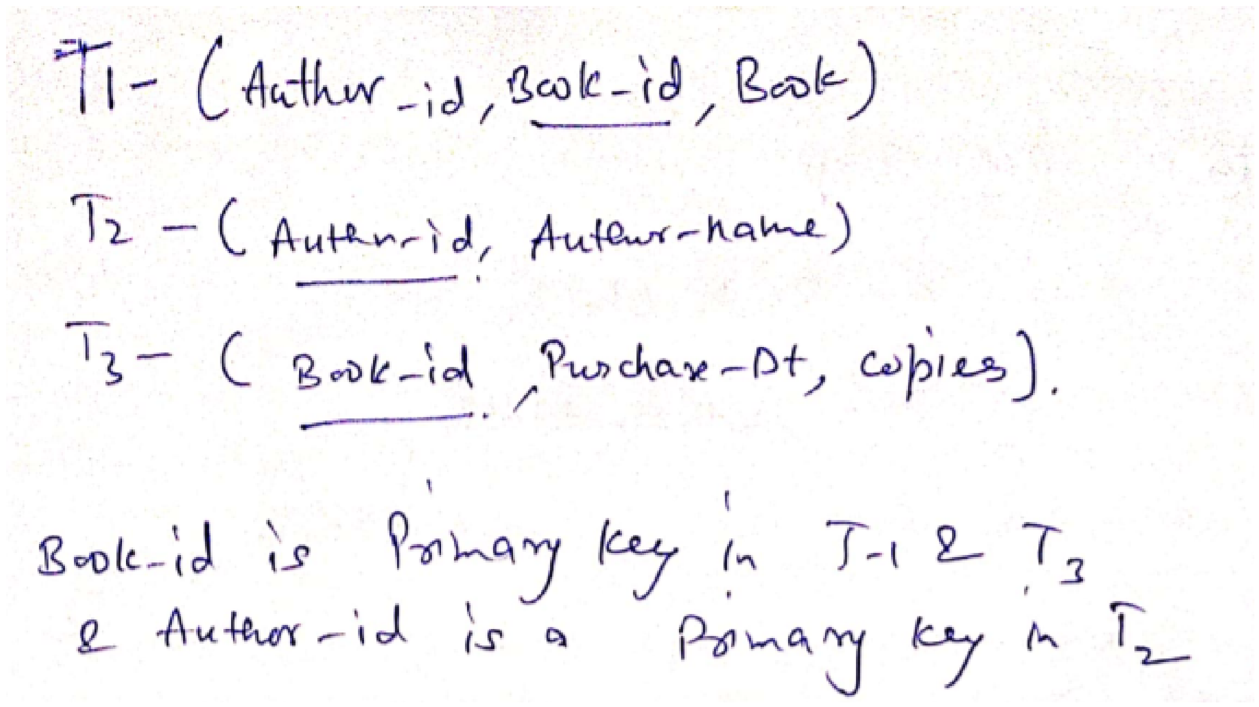
Nirbhay Sharma (B19CSE114) lab4

a)



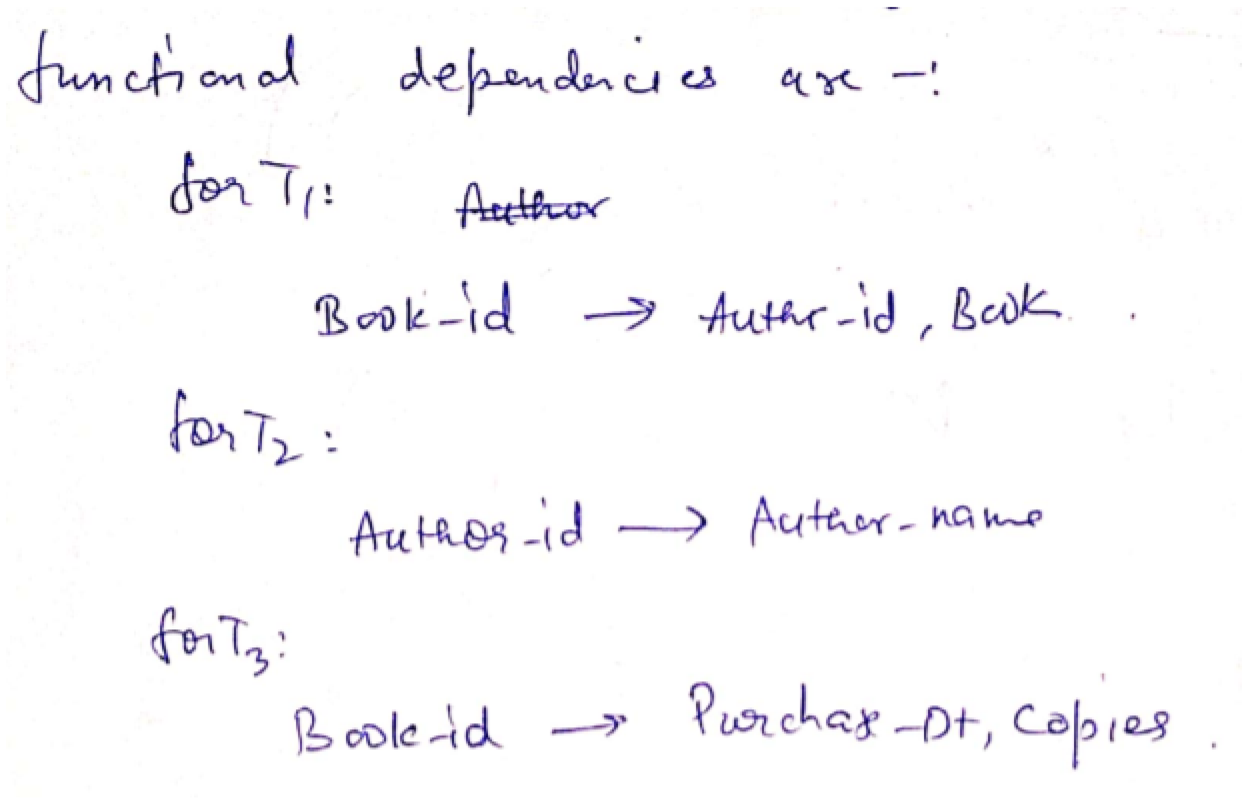
T1- (Author-id, Book-id, Book)

T2 - (Author-id, Author-name)

T3 - (Book-id, Purchase-Dt, copies).

Book-id is Primary key in T-1 & T_3
& Author-id is a Primary key in T_2

\
Functional dependency for the above tables are:



functional dependencies are -:

for T_1: ~~Author~~

Book-id $\rightarrow$ Author-id, Book.

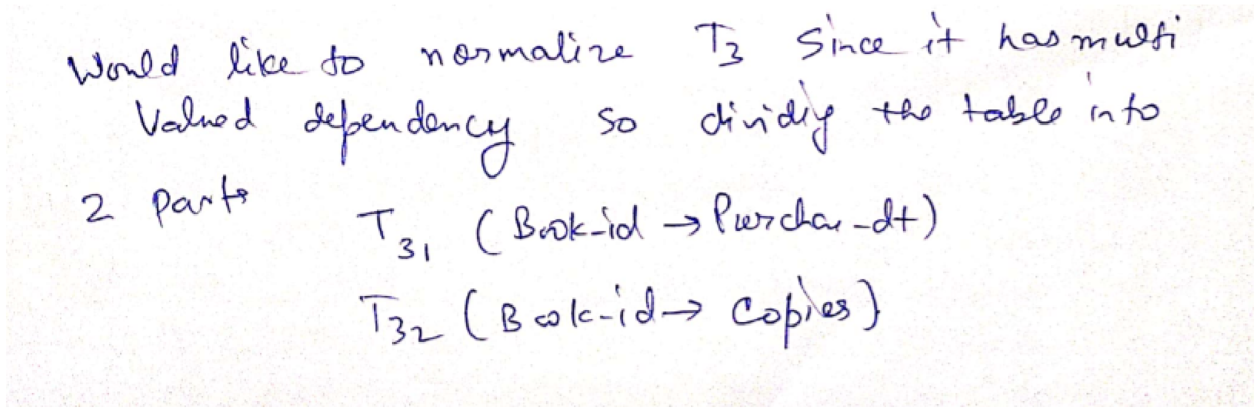
for T_2:

Author-id $\longrightarrow$ Author-name

for T_3:

Book-id $\rightarrow$ Purchase-Dt, Copies.

So the tables look normalized till BCNF Normal form since each table does not contain any kind of transitive or partial dependency and all the dependency on the Left Hand Side contains only the primary key.


Would like to normalize T_3 Since it has multi Valued dependency so dividing the table into 2 parts

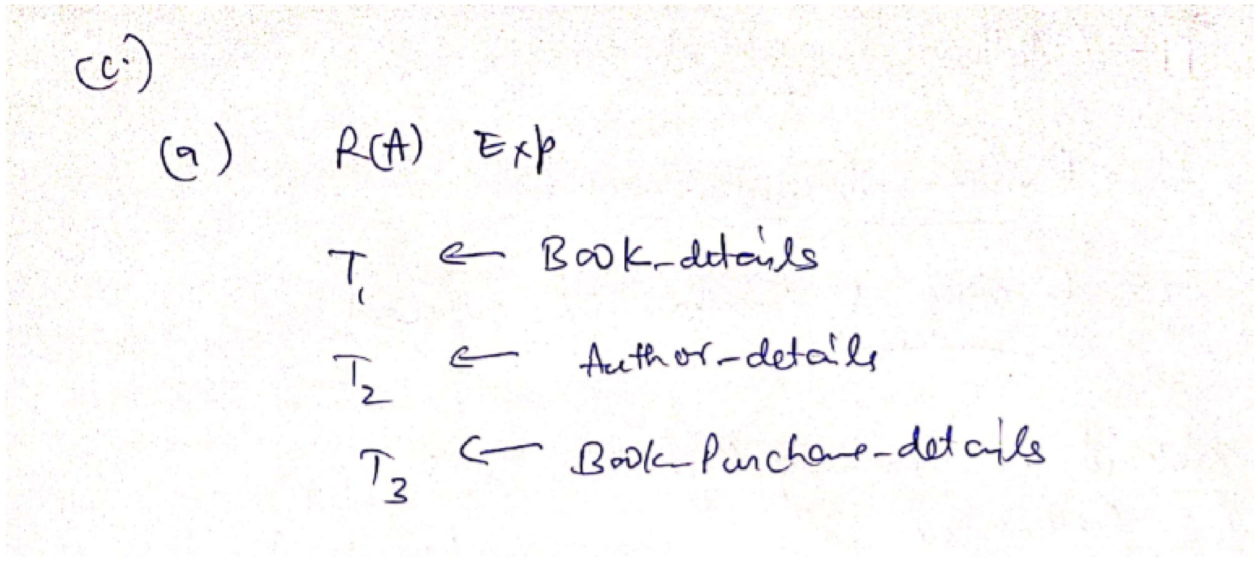
T_{31} (Book-id $\rightarrow$ Purchase-dt)

T_{32} (Book-id $\rightarrow$ Copies)

b)

For table 1 my index would be Book_id since it is primary key in the table so it would be making my searching operations faster
For table 2 my index would be author_id since it is also the primary key and can uniquely determine author_name so it would also make search operation faster
For table 3 my index would be book_id since it is primary key because it can uniquely determine the purchase date and the copies of the book so it would also be making my search operation bit faster

NOTE: For queries in all the below parts the following convention is used


(c)

(a) R(A) Exp

$T_1 \leftarrow$ Book_details

$T_2 \leftarrow$ Author-details

$T_3 \leftarrow$ Book-Purchase-details

c)

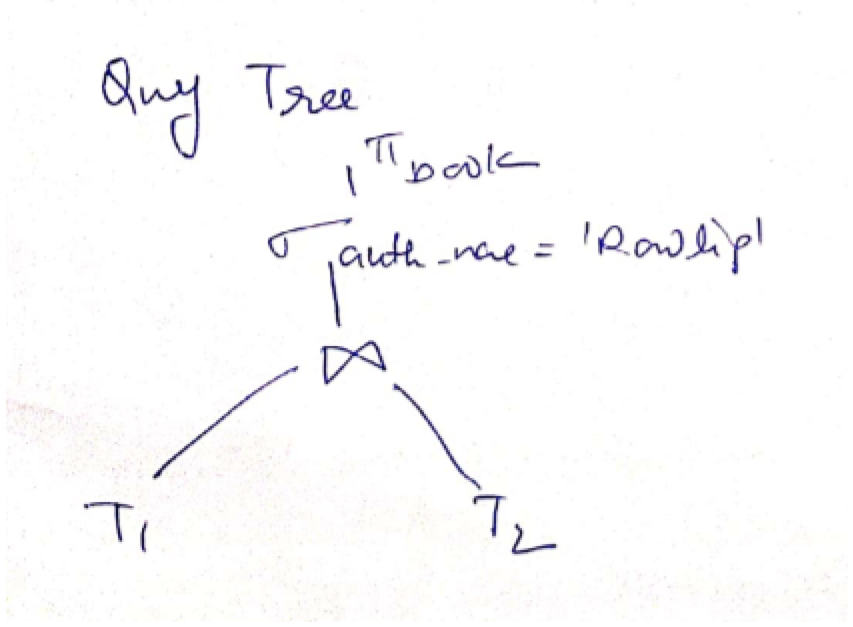
Hash function used is normal hash function which basically adds the ascii values of the char in the string and then takes the modulo with the size of the global directory and the since the size grows dynamically so our hash function will also change dynamically

c.a) for part a following are the queries and corresponding query tree

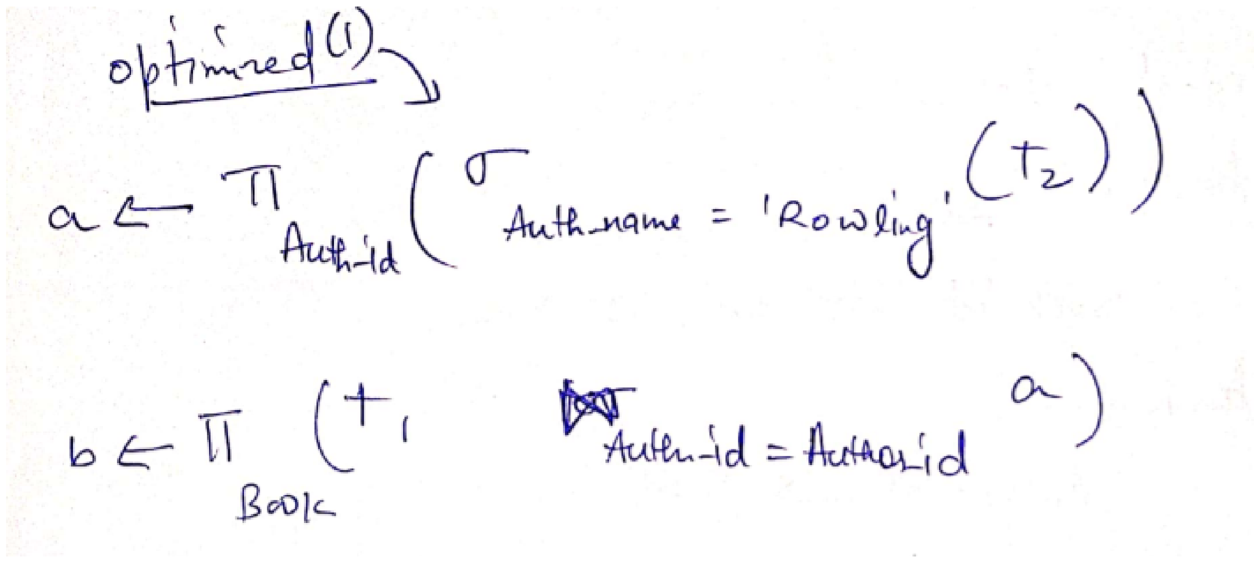
Query -1

$$\pi_{book}\left(\sigma_{author\ name = 'Rowling'}\left(T_1 \bowtie T_2\right)\right)$$

Corresponding query tree



Query -2 (optimised version -1)

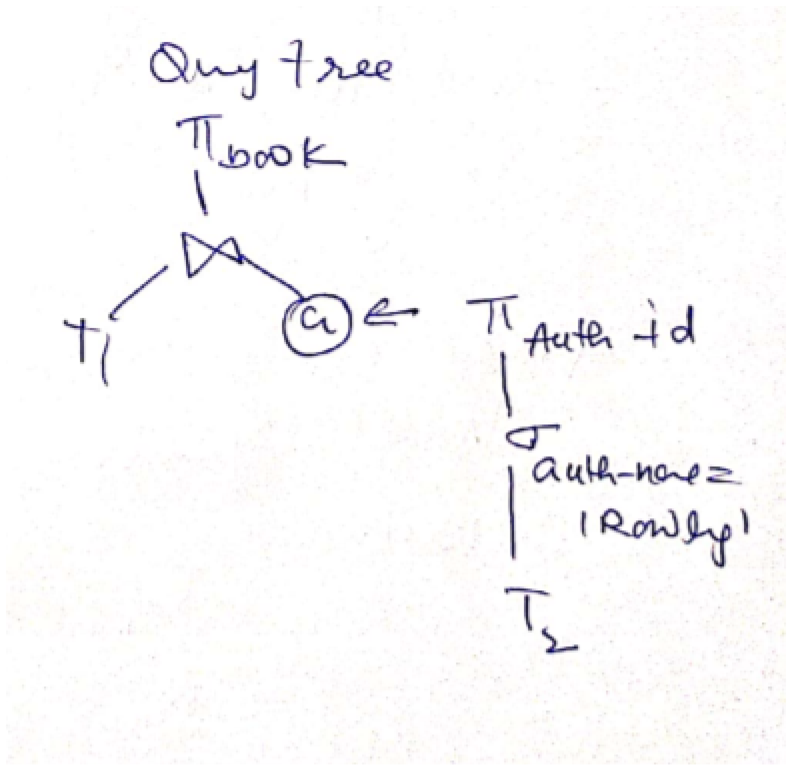


<u>optimized</u> (1)

$$a \leftarrow \pi_{Auth\_id}\left(\sigma_{Auth\_name = 'Rowling'}(T_2)\right)$$

$$b \leftarrow \pi_{Book}\left(T_1 \bowtie_{Auth\_id = Author\_id} a\right)$$

Query tree

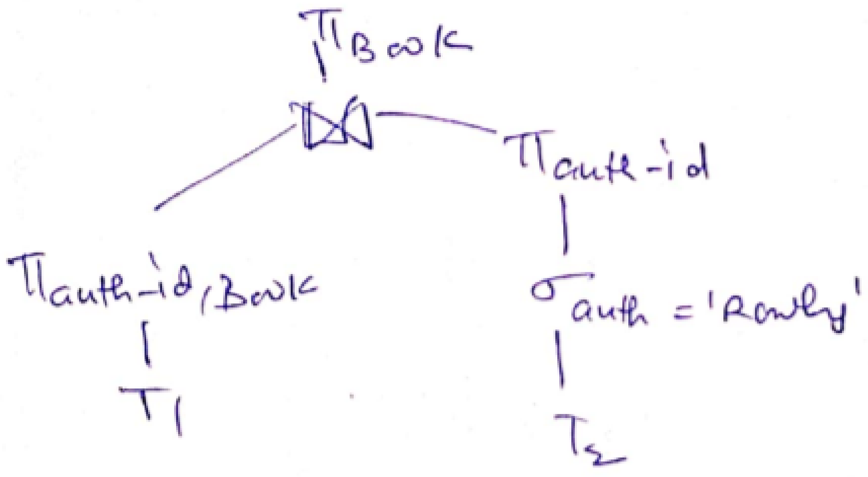


Final optimised version with query tree

$$a \leftarrow \Pi_{auth\text{-}id}\left(\sigma_{auth\text{-}name='Rowling'}(T_2)\right)$$

$$b \leftarrow \Pi_{auth\text{-}id,\ Book}(T_1)$$

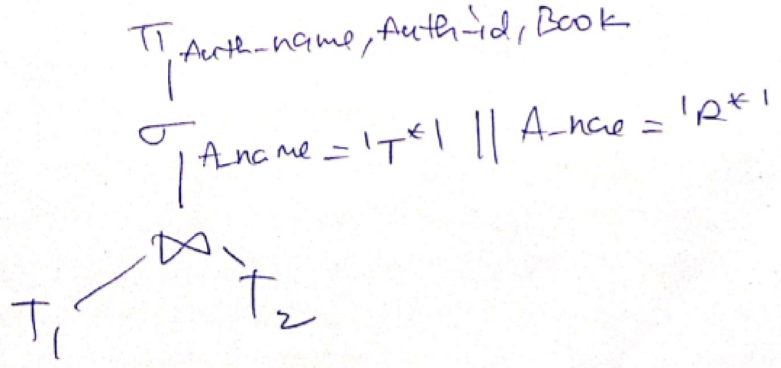
$$c \leftarrow \Pi_{Book}(a \bowtie b)$$



c.b)
Unoptimized query

(b)

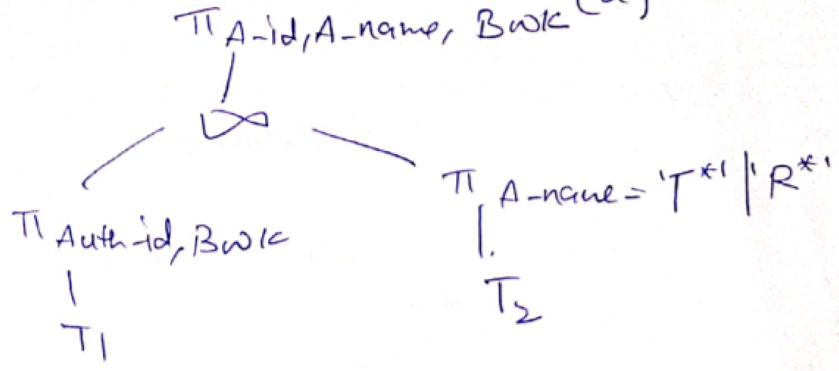
$$\Pi_{\text{Auth-id, Auth-name, Book}}\ \sigma_{\text{Auth-name = 'T*' || Auth-name = 'R*'}}\ (T_1 \bowtie T_2)$$



Optimised query with query tree

$$\left(\Pi_{\text{Auth-id, Book}}(T_1) \bowtie \Pi_{\text{A-name = 'T*' || 'R*'}}(T_2)\right) \rightarrow a$$

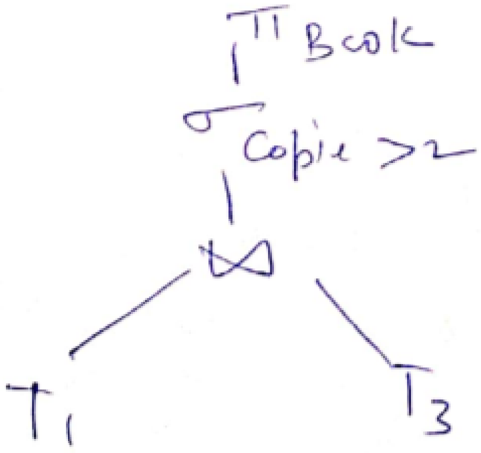
$$\Pi_{\text{A-id, A-name, Book}}(a)$$



c.c)
Unoptimized query

(c) [illegible]

$$\Pi_{Book}(\sigma_{copies>2}(T_1 \bowtie T_3))$$

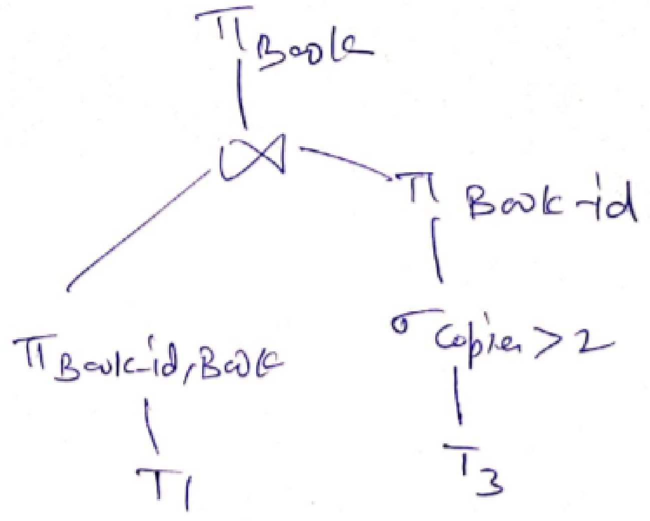


Optimized query with query tree

$$a \leftarrow \Pi_{Book\text{-}id}\left(\sigma_{copies>2}(T_3)\right)$$

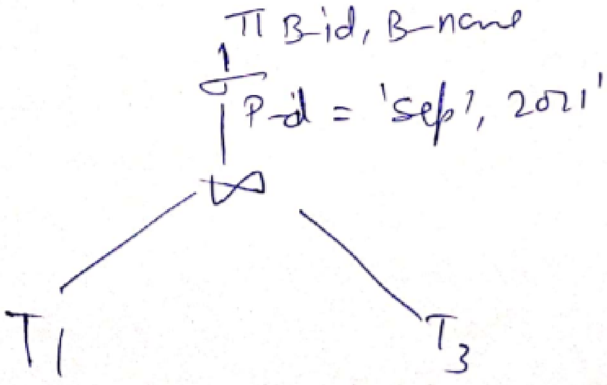
$$b \leftarrow \Pi_{Book\text{-}id,\,Book}(T_1)$$

$$c \leftarrow \Pi_{Book}(a \bowtie b)$$



c.d)
Unoptimized query (assuming date to be sept 1,2021

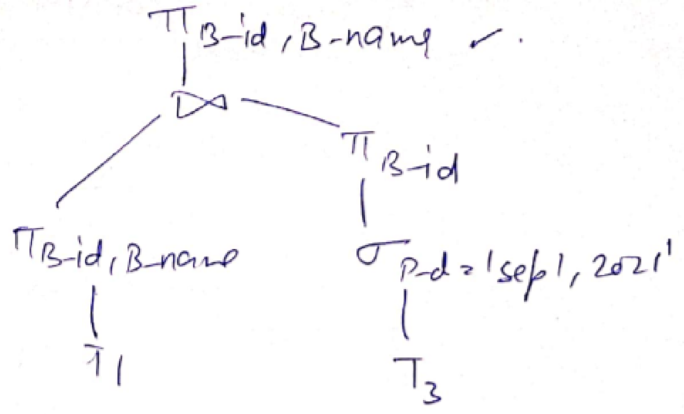
$$\pi_{B\text{-}id,\,B\text{-}name}\left(\sigma_{P\_d = 'sep 1, 2021'}\left(T_1 \bowtie T_3\right)\right)$$



Optimized query with query tree

$$\left(\pi_{B\text{-}id,\,B\text{-}name}(T_1)\right) \bowtie \pi_{B\text{-}id}\left(\sigma_{P\_d = 'sep 1, 2021'}(T_3)\right) \Rightarrow a$$

$$\pi_{B\text{-}id,\,B\text{-}name}(a)\ ✓$$



Comparison in design per query
c.a) there are three designs propose in this query first one is simple implementation as we are first joining T1 and T2 table and then selecting tuples with author_name rowling and then project the book_name -> this query has drawback that it has many unnecessary tuples in the first join of T1 and T2
Compared to it there is another query in which we are first selecting according to the author_name rowling from T2 table and then we are joining the resultant table with T1 and finally projecting the book_name from the final table
But in previous approach we have one issue that we need only book_name and we are joining according to the author_id so we are getting this book_id unnecessary so writing the final query like this we will extract author_id according to author_name as rowling this will reduce one column from my T2 table and also it will reduce the tuples
Then I will extract author_id and book from the T1 table since these two I really want so It will reduce the number of columns from T1 and then finally joining both the resultat tables and projecting book from there

c.b) first query is again written in most straightforward manner first join T1 and T2 and then select according to the author_name beginning with R and T and the finally get authro_id, author_name,book -> issue is again many unnecessary tuples will be there
Final optimized query is -> first select from T2, author_name with R and T and then from T1 get author_id and book since book_id we don't need so finally join them according to author_id and project author_id, author_name,book

c.c) first query is again written in most straightforward manner first join T1 and T3 and then select according to the copies > 2 and the finally get book -> issue is again many unnecessary tuples will be there
Final optimized query is -> first select from T3, copies > 2 and project book_id and then from T1 get book_id and book since author_id we don't need so finally join them according to book_id and project book -> this will reduce many unnecessary tuples and columns and return exactly whatever we want

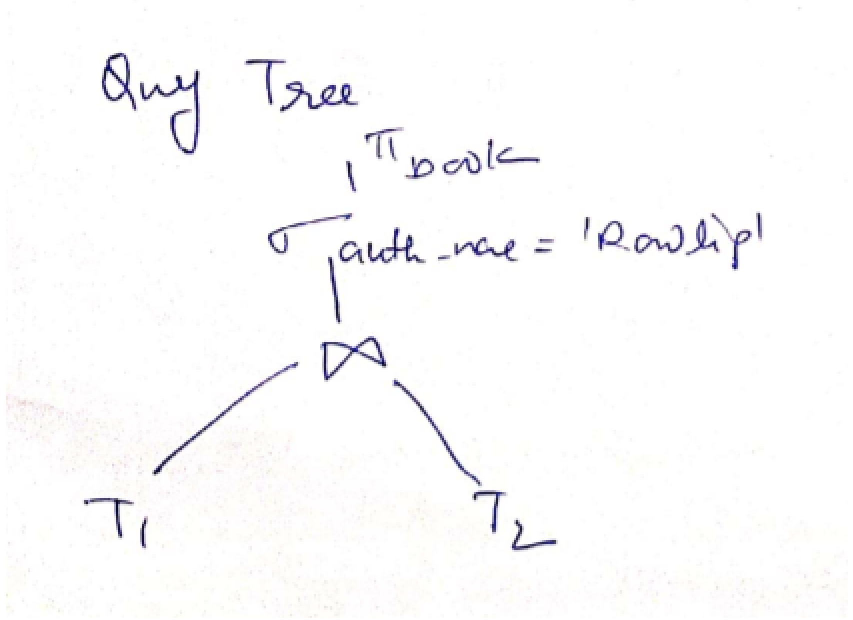
c.d) assumption -> fixing date to sept 1, 2021
first query is again written in most straightforward manner first join T1 and T3 and then select according to the pd = sept 1,2021 and the finally get book and book_id -> issue is again many unnecessary tuples will be there and also many unnecessary column will be there in after join so it will also make my join operation slow
Final optimized query is -> first select from T3, pd = 'sep 1,2021' and project book_id and then from T1 get book_id and book since author_id we don't need so finally join them according to book_id and project book,book_id -> this will reduce many unnecessary tuples and columns and return exactly whatever we want and hence making our join operation fast

d)
Hash function used is normal hash function which basically adds the ascii values of the char in the string and then takes the modulo with the size of the global directory and the since the size grows dynamically so our hash function will also change dynamically

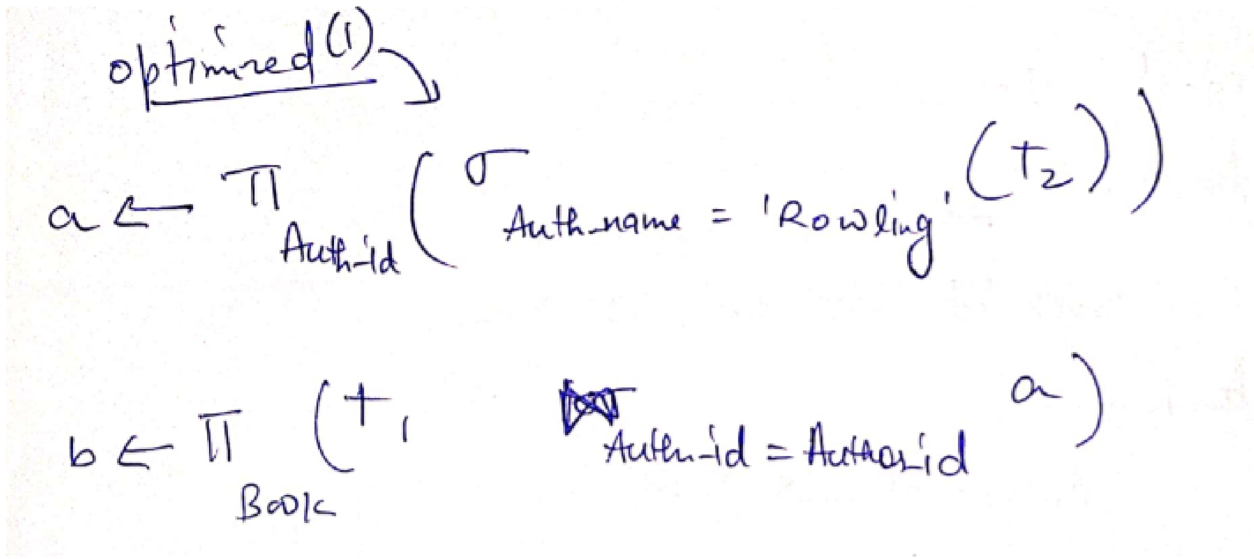
d.a) for part a following are the queries and corresponding query tree
Query -1

$$\pi_{book}\left(\sigma_{author\ name = 'Rowling'}\left(T_1 \bowtie T_2\right)\right)$$

Corresponding query tree



Query -2 (optimised version -1)

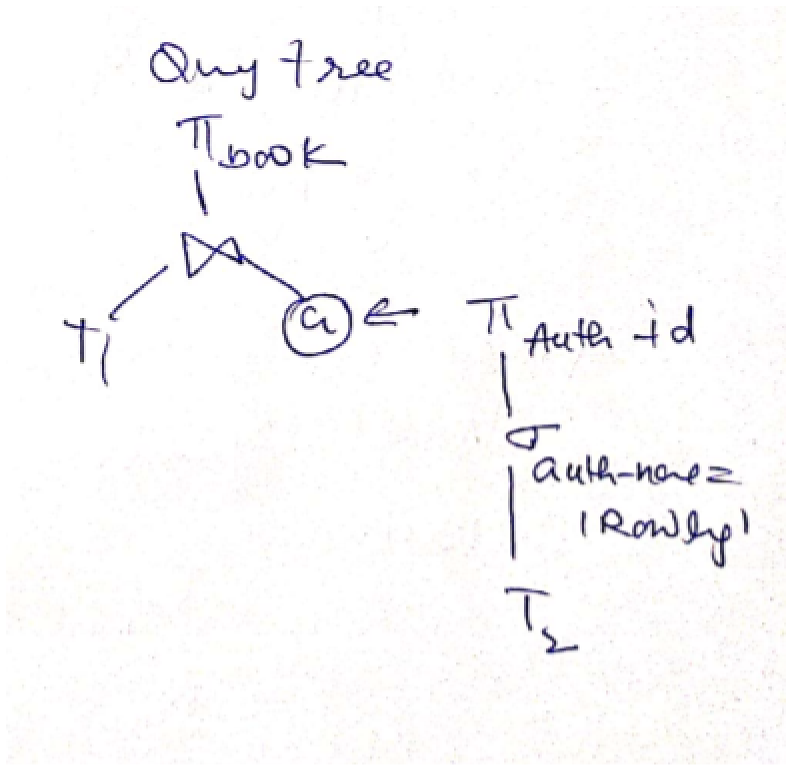


optimized(1)

$$a \leftarrow \pi_{Auth\_id}\left(\sigma_{Auth\_name = 'Rowling'}(T_2)\right)$$

$$b \leftarrow \pi_{Book}\left(T_1 \bowtie_{Auth\_id = Author\_id} a\right)$$

Query tree

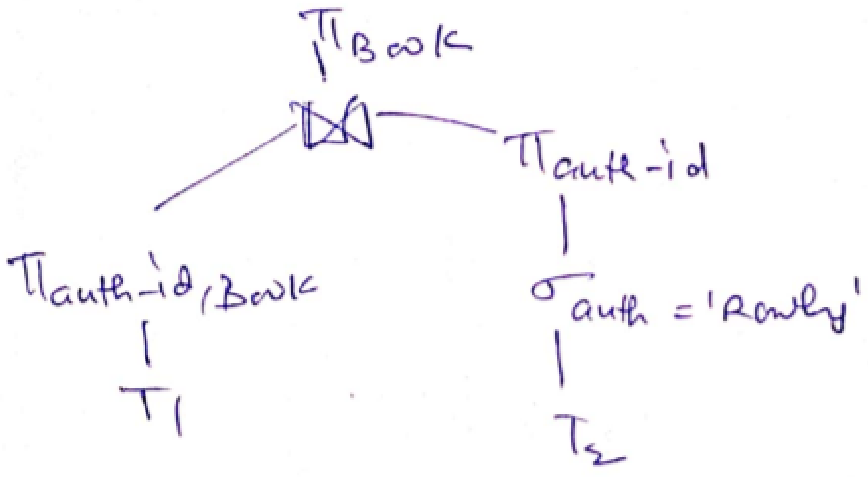


Final optimised version with query tree

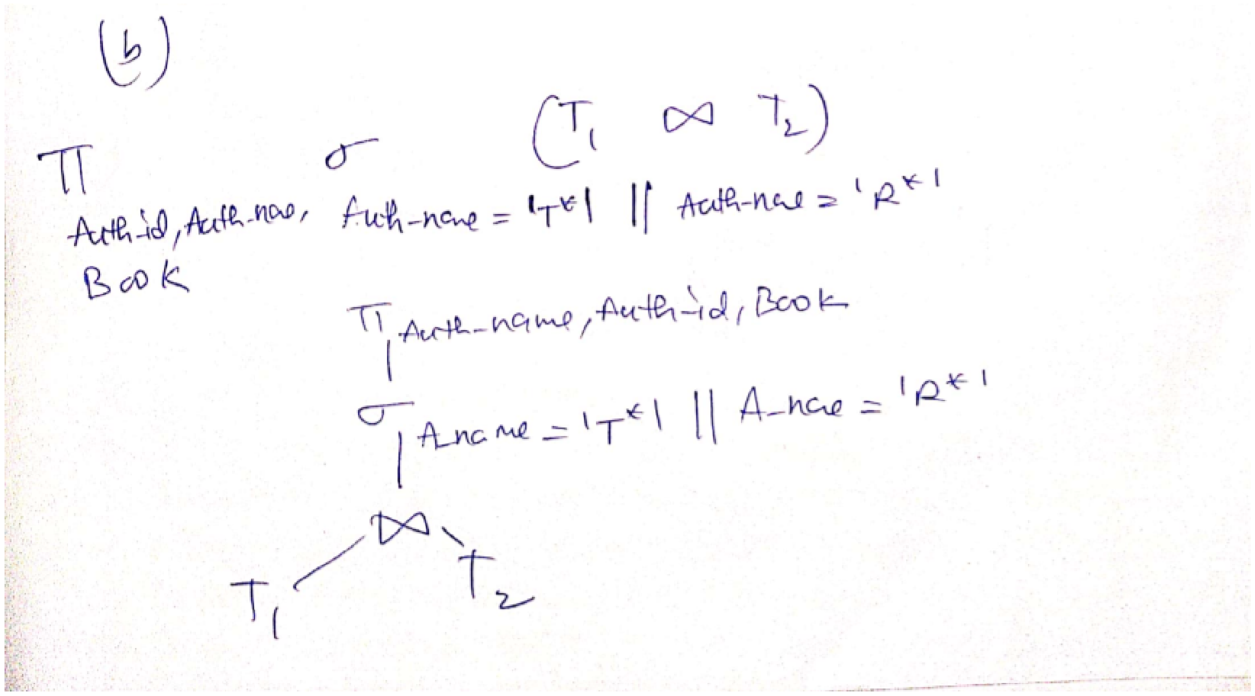
$$a \leftarrow \pi_{auth\text{-}id}\left(\sigma_{auth\text{-}name='Rowlig'}(T_2)\right)$$

$$b \leftarrow \pi_{auth\text{-}id,\ Book}(T_1)$$

$$c \leftarrow \pi_{Book}(a \bowtie b)$$



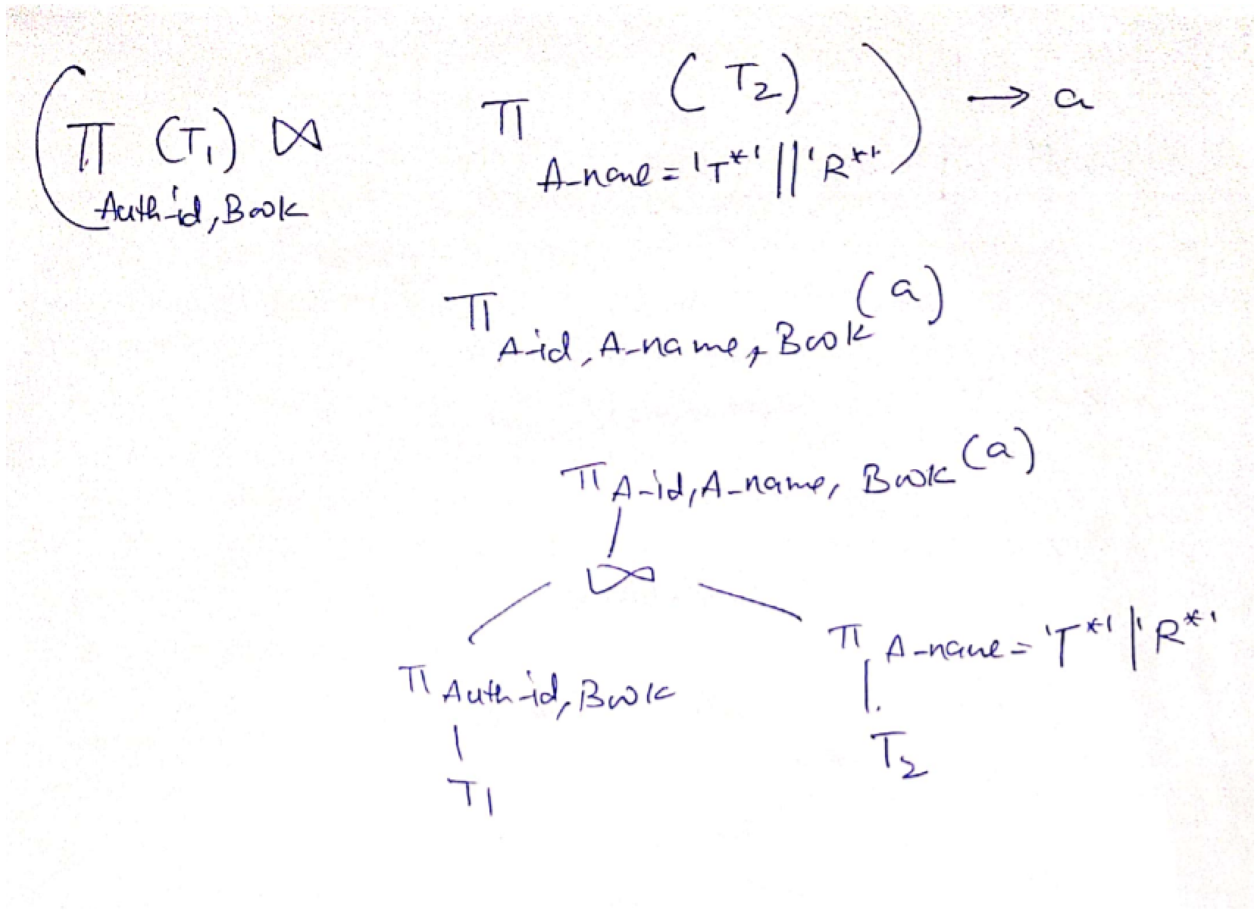
d.b)
Unoptimized query


(b)

$$\Pi_{\text{Auth-id, Auth-nae, Book}}\ \sigma_{\text{Auth-name = 'T*' || Auth-nae = 'R*'}}\ (T_1 \bowtie T_2)$$

$$\begin{tikzcd}
 & \Pi_{\text{Auth-name, Auth-id, Book}} \arrow[d, dash] & \\
 & \sigma_{\text{A-name = 'T*' || A-nae = 'R*'}} \arrow[d, dash] & \\
 & \bowtie \arrow[dl, dash] \arrow[dr, dash] & \\
T_1 & & T_2
\end{tikzcd}$$

Optimised query with query tree


$$\left(\Pi_{\text{Auth-id, Book}}(T_1) \bowtie \Pi_{\text{A-nae = 'T*' || 'R*'}}(T_2)\right) \rightarrow a$$

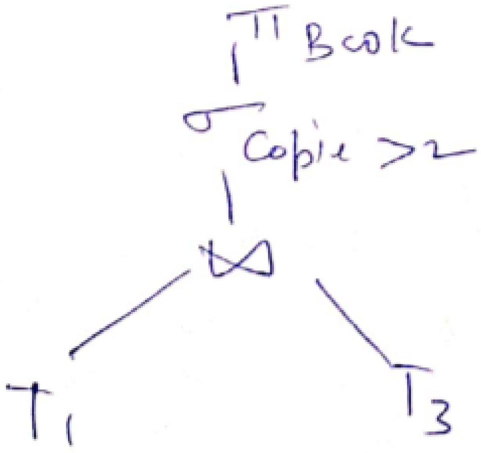
$$\Pi_{\text{A-id, A-name, Book}}(a)$$

$$\begin{tikzcd}
 & \Pi_{\text{A-id, A-name, Book}}(a) \arrow[d, dash] & \\
 & \bowtie \arrow[dl, dash] \arrow[dr, dash] & \\
\Pi_{\text{Auth-id, Book}} \arrow[d, dash] & & \Pi_{\text{A-nae = 'T*' | 'R*'}} \arrow[d, dash] \\
T_1 & & T_2
\end{tikzcd}$$

d.c)
Unoptimized query

(c) [illegible]

$$\pi_{Book}\left(\sigma_{copies>2}\left(T_1 \bowtie T_3\right)\right)$$

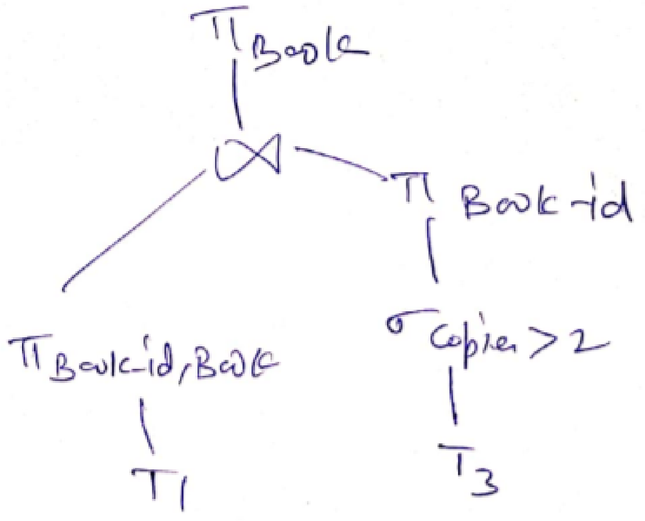


Optimized query with query tree

$$a \leftarrow \pi_{Book\text{-}id}(\sigma_{copies>2}(T_3))$$

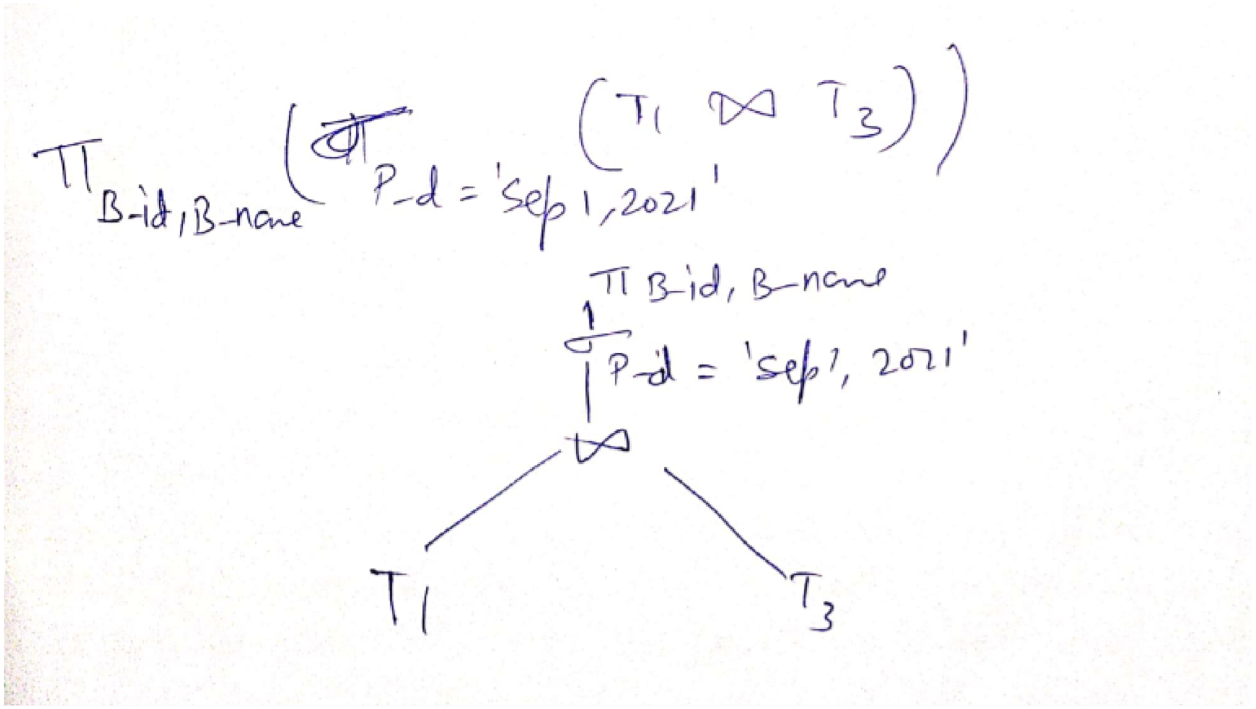
$$b \leftarrow \pi_{Book\text{-}id,\, Book}(T_1)$$

$$c \leftarrow \pi_{Book}(a \bowtie b)$$

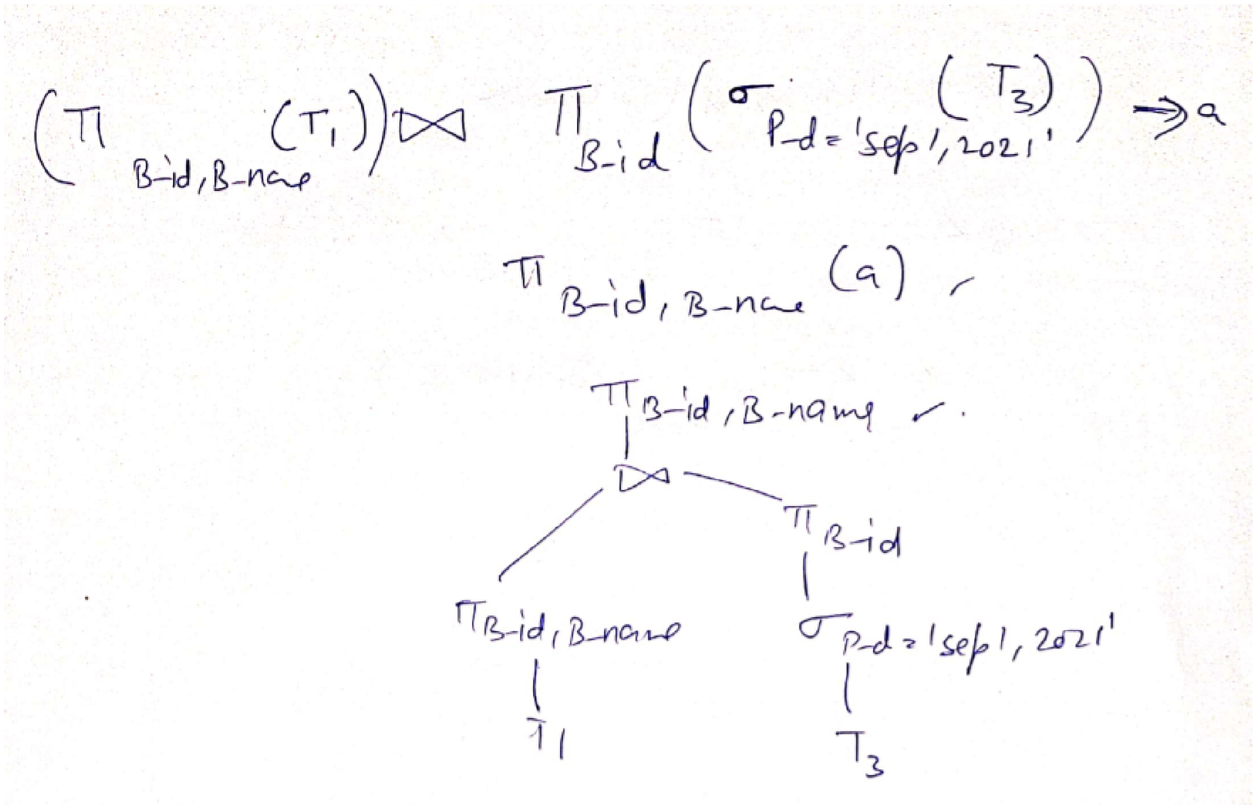


d.d)

Unoptimized query (assuming date to be sept 1,2021


$$\pi_{B\text{-}id,\,B\text{-}name}\left(\sigma_{P\text{-}d\,=\,'sep 1, 2021'}\left(T_1 \bowtie T_3\right)\right)$$

Optimized query with query tree


$$\left(\pi_{B\text{-}id,\,B\text{-}name}(T_1)\right) \bowtie \pi_{B\text{-}id}\left(\sigma_{P\text{-}d\,=\,'sep 1, 2021'}(T_3)\right) \Rightarrow a$$

$$\pi_{B\text{-}id,\,B\text{-}name}(a)$$ ✓

Comparison in design per query
d.a) there are three designs propose in this query first one is simple implementation as we are first joining T1 and T2 table and then selecting tuples with author_name rowling and then project the book_name -> this query has drawback that it has many unnecessary tuples in the first join of T1 and T2
Compared to it there is another query in which we are first selecting according to the author_name rowling from T2 table and then we are joining the resultant table with T1 and finally projecting the book_name from the final table
But in previous approach we have one issue that we need only book_name and we are joining according to the author_id so we are getting this book_id unnecessary so writing the final query like this we will extract author_id according to author_name as rowling this will reduce one column from my T2 table and also it will reduce the tuples
Then I will extract author_id and book from the T1 table since these two I really want so It will reduce the number of columns from T1 and then finally joining both the resultat tables and projecting book from there

d.b) first query is again written in most straightforward manner first join T1 and T2 and then select according to the author_name beginning with R and T and the finally get authro_id, author_name,book -> issue is again many unnecessary tuples will be there
Final optimized query is -> first select from T2, author_name with R and T and then from T1 get author_id and book since book_id we don’t need so finally join them according to author_id and project author_id, author_name,book

d.c) first query is again written in most straightforward manner first join T1 and T3 and then select according to the copies > 2 and the finally get book -> issue is again many unnecessary tuples will be there
Final optimized query is -> first select from T3, copies > 2 and project book_id and then from T1 get book_id and book since author_id we don’t need so finally join them according to book_id and project book -> this will reduce many unnecessary tuples and columns and return exactly whatever we want

d.d) assumption -> fixing date to sept 1, 2021
first query is again written in most straightforward manner first join T1 and T3 and then select according to the pd = sept 1,2021 and the finally get book and book_id -> issue is again many unnecessary tuples will be there and also many unnecessary column will be there in after join so it will also make my join operation slow
Final optimized query is -> first select from T3, pd = ‘sep 1,2021’ and project book_id and then from T1 get book_id and book since author_id we don’t need so finally join them according to book_id and project book,book_id -> this will reduce many unnecessary tuples and columns and return exactly whatever we want and hence making our join operation fast

e)since for c and d part the query are same so comparing the complexity of different query plans

A	B	C	D
For A the query 3 will be of very less complexity as it reduces the number of columns and number of tuples as well so join operation which is the most expensive operation in the whole query becomes faster hence reducing the complexity of the query	For B the query 2 is of less complexity as it reduces number of columns and rows also in and join becomes much faster than the queries in which join was there which has more columns and more rows	For C the query 2 is of less complexity as by using this query reduces the number of columns and rows hence reducing the execution time for that query	For D the query 2 is of less complexity as it reduces number of columns and rows also in and join becomes much faster than the queries in which join was there which has more columns and more rows

Bonus part:
Extendible hashing with B-Tree as local directory / core database system is implemented and code is attached in zip file