1

09/02/2022

SEM 5
END SEM EXAM
ADBMS

JUNAID . GIRKAR
60004190057
TE COMPS A4

Q1

ANS In order to implement dynamic multilevel indexing, B-tree and B+ trees are generally employed. The drawback of B-tree used for indexing, however is that it stores the data pointer corresponding to a particular key value, along with the ~~node~~ key value in the node of a B-tree. This technique greatly reduces the number of entries that can be packed into a node of a B-tree, thereby contributing to the increase in the number of levels in the B-tree, hence increasing the search time of a record.

B+ tree eliminates the above drawback by storing data pointers only at the leaf nodes of the tree. Thus, the structure of leaf nodes of a B+ tree is quite different from the structure of internal nodes of a B+ tree. It may be noted here that, since data pointers are present only at the leaf nodes, the leaf nodes must necessarily store all the key values along with their corresponding data pointers to the disk file block, in order to

access them. Moreover the leaf nodes are linked to provide ordered access to the records. The leaf nodes, therefore form the first level of index, with the internal nodes forming the other levels of a multilevel index. Some of the key values of the leaf nodes also appear in the internal nodes, to simply act as a medium to control the searching of a record.

From the above info, it is apparent that a B+ tree unlike a B- tree has two orders, "a" and "b", one for internal nodes and the other for the external nodes (leaf).

- The structure of the internal nodes of a B+ tree of order "a" is as follows:

1. Each internal node is of the form: $<P_1, K_1, P_2, K_2, \ldots, P_c>$ where $c <= a$ and each P_i is a tree pointer (i.e. points to another node of the tree) and each K_i is a key value

2. Each internal node has: $K_1 < K_2 \ldots < K_{c-1}$

3. For each search fields values 'x' in the sub-tree pointed at by P_i, the following condition holds:

$K_{i-1} < X <= K_i$ for $1 < i < c$ and

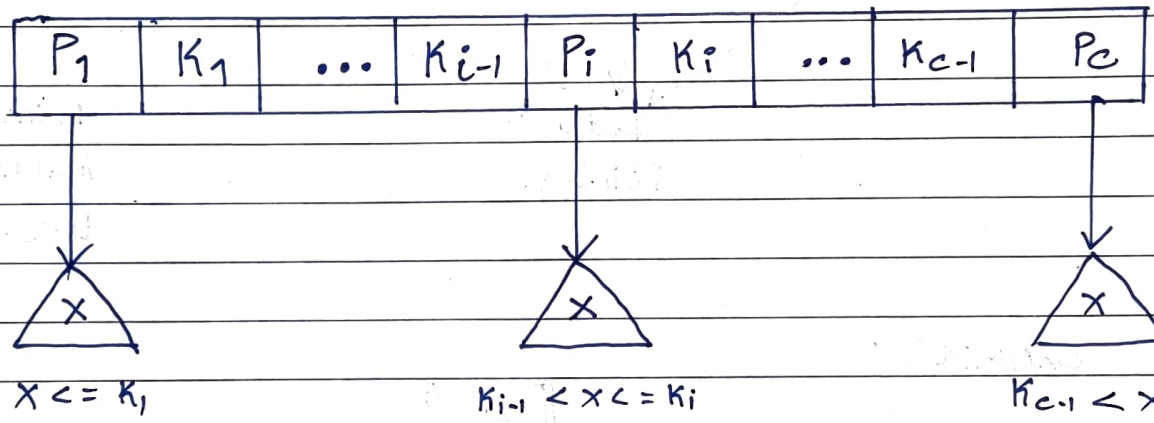
$K_{i-1} < X$ for $i = c$

8000419005J

4. Each internal node has at most 'a' tree pointers

5. The root node has, at least two tree pointers, while the other internal nodes have atleast \ciel (a/2) tree pointers each.

6. If every internal node has 'c' pointers, $c <= a$, then it has 'c - 1' key values.



II The structure of the leaf nodes of a B+ tree of order 'b' is as follows :

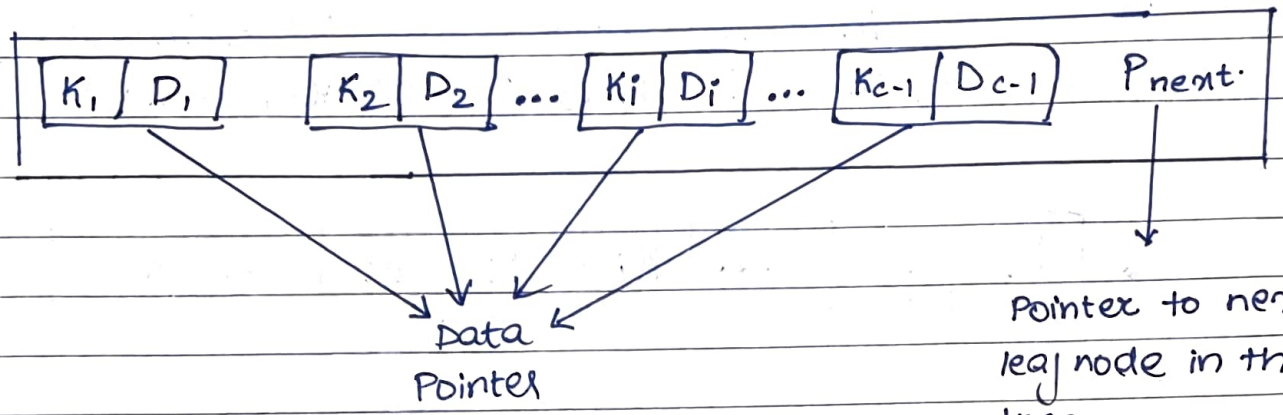
1. Each leaf node is of the form : $<<K_1, D_1>, <K_2, D_2>, \ldots <K_{c-1}, D_{c-1}>, P_{next}>$

where $c <= b$ and each D_i is also a data pointer and each K_i is a key value and P_{next} points to the next leaf node in the B+ tree.

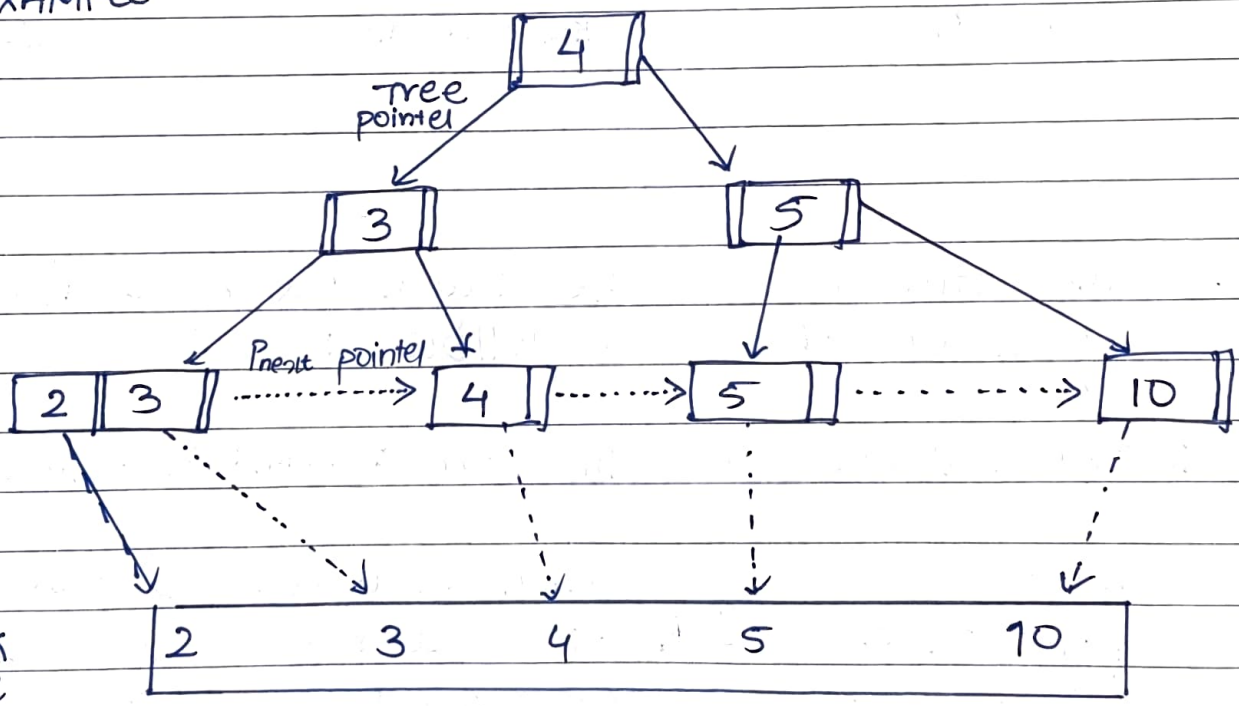
2. Every leaf node has : $K_1 < K_2 < \dots < K_{c-1}$, $c <= b$

3. Each leaf node has atleast ceil $(b/2)$ values

4. All leaf nodes are at same level.



EXAMPLE :



G0004190057

Q2 why is there a need of document oriented databases? Explain any one real world application

ANS Rather than storing data in the form of rows and columns, document databases store data as documents. A document can be defined as a self contained data entry containing everything needed to understand its meaning, similar to real-world documents. These are NoSQL databases with data usually stored in JSON format. other formats such as XML and YAML can also be used.

A real world application can be user profiles or contact card database.
consider an employee "Tom", his contact card ~~could~~ could be like:

```
{
    "_id" : "~~tom~~", "Voldemort",
    "firstName" : "Tom",
    "lastName" : "Riddle",
    "department" : "Dark side"
}
```

and another person "Harry" could have:

```
{ "_id" : "BWL",
    "firstName" : "Harry",
    "lastName" : "Potter",
    "department" : ["Light", "Ministry"]
}
```

the second document is a bit different from the first as it stores a list in the department field and therefore they have different schemas.

So we use document databases here as they don't need a defined schema and are dynamic and also allow nested documents

Q3

ANS SORT-MERGE JOIN

- It is a common algorithm in data base systems that use sorting
- the join predicate needs to be an equality join predicate
- The algorithm sorts both relations on the join attribute and then merges the sorted relations by scanning them sequentially and looking for qualifying tuples.
- The sorting step groups all tuples with the same value in the join attribute.
- such groups are sorted based on the value in the join attribute so that it becomes easy to locate groups from the two relations with the same attribute value

E.g:

USER

Name	UID
Tom	10
DICK	20
Harry	25

BILLS

BID	UID	AMOUNT
1	25	₹100
2	20	₹900
3	10	₹50
4	10	₹150

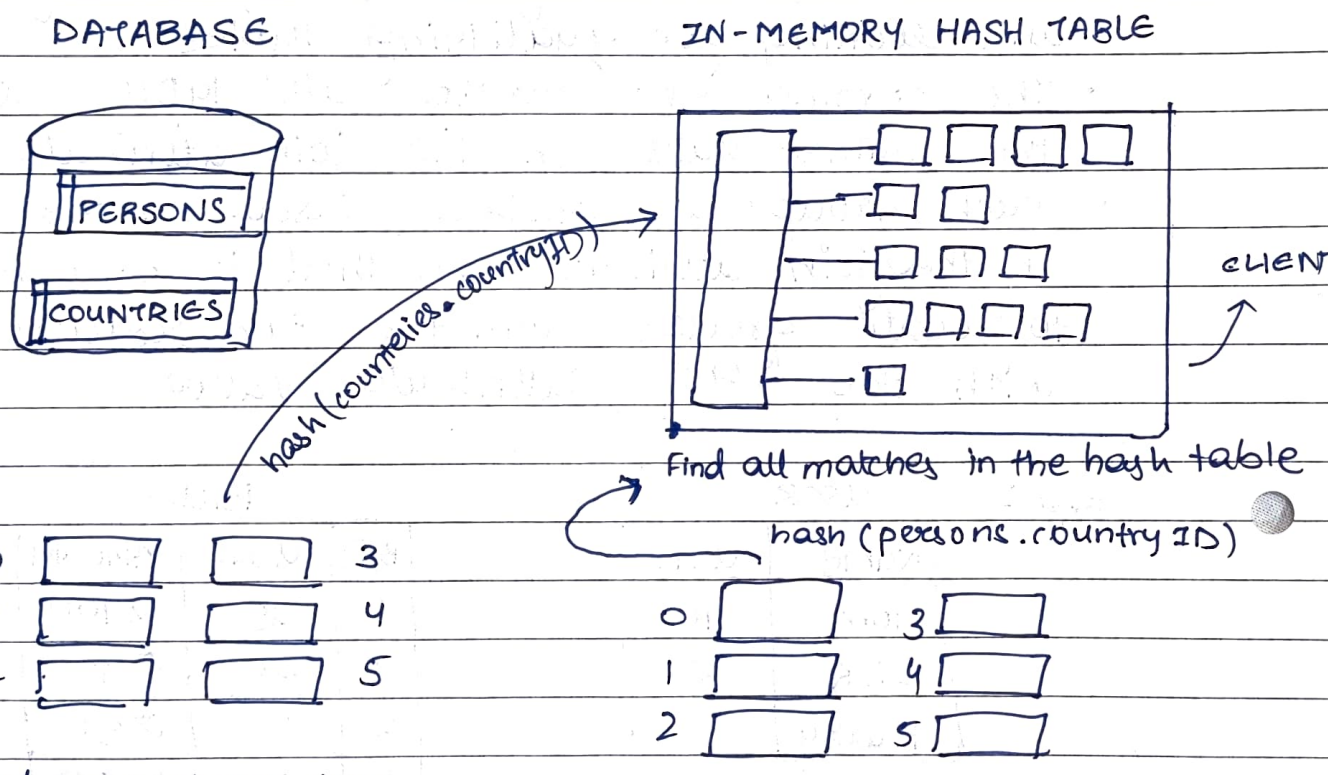
After sort-merge join

Name	UID	BID	Amount
DICK	20	1	₹900
Harry	25	2	₹100
Tom	10	3	₹50
Tom	10	4	₹150

II HASH JOIN

- Hash join is a way of executing a join where hash table is used to find matching rows between the two inputs (one or more tables)
- It typically works more efficiently than nested loop joins, especially if one of the inputs fit in memory.

Example:



Build input chunk files on disk

Probe input chunk files on disk

QUERY: SELECT given_name, country_name FROM PERSONS JOIN COUNTRIES ON PERSONS.COUNTRYID = COUNTRIES.COUNTRYID

S

G0004190057.

Q4

ANS Databases that store temporal data i.e. data that is time dependent are called temporal databases which store information about states of the real world across time rather than storing information about the current state only.

It stores information relating to past, present and future time.

Examples of temporal database applications:-

- Healthcare systems
- Banking systems
- Insurance systems
- Reservation systems

Example :-

- Tom was born on 15th December, 1930 in England
- He went to Hogwarts on 1st September, 1941
- He got a job in Borgin & Burks on 1st September, 1948
- He was first vanquished on 31st October, 1981
- He finally died on 2nd May, 1997.

6000119005 7

Temporal Aspects

- VALID TIME : time period when fact is true
- TRANSACTION TIME : Time period during which a fact stored in the database, based on transaction serialization order and is auto generated by system

Temporal Relations

- It is where each tuple has an associated time : either valid time or transaction time
- Uni-temporal relations - has one axis of time, either valid time or transaction time.
- Bi-temporal relations - has two axis of time, includes valid start time, valid end time, transaction start time and transaction end time.

ADVANTAGES

- Provides historical and roll-back information
- Historical information - valid time
- Rollback information - Transaction time

6

Q5 PROJ (PNO , PNAME , BUDJET)
PAY (TITLE , SALARY)
EMP (ENO , ENAME , TITLE)
ASG (ENO , PNO , RESPONSIBILITY , DURATION)

a] Horizontal fragmentation divides a relation horizontally into group of rows to create subsets of tuples specified by a condition on one or more attributes of relation.
The tuples that belong to the horizontal fragment is specified by some condition on one or more attributes of relation.

FRAGMENT 1 : All positions with salary >= 50,000.

a] Relations : PAY
b] Attributes : All (*) of positions
c] Guard condition : SALARY >= 50,000
d] QUERY

$$F_1 \leftarrow (\sigma_{SALARY >= 50,000} [PAY])$$

FRAGMENT 2 : All projects with budjet <150,000
a] Relations : PROJ
b] Attributes : All (*) of projects
c] Guard condition : BUDJET >< 150,000
d] QUERY : $F_1 \leftarrow (\sigma_{BUDJET < 150,000} (PROJ))$

b Derived Fragmentation.

FRAGMENT 1: All department names

a] Relations : ASG
b] Attributes : Employee Numbel, Project number & Duration
c] Guard condition : RESPONSIBILITY = "MANAGER"
d] Query

$F_1 \leftarrow \pi_{PNO, ENO, DURATION}((\sigma_{RESPONSIBILITY = "MANAGER"}(ASG))$

c) Vertical Fragmentation
- Vertical fragmentation divides a relation vertically into group of columns.
- When each side does not need all the attributes of a relation, vertical fragmentation is used to fragment relation vertically by columns.
- It is necessary to include primary key or some common candidate key in every vertical fragment to reach the original relation from the fragments

FRAGMENT 1 : All department names
a] Relations : PROJ
b] Attributes : Project Number and Budget
c] Guard Condition : NO
d] Query : $F_1 \leftarrow \pi_{PNO, PNAME}(PROJ)$