Advanced Database

Over View Storage and Indexing

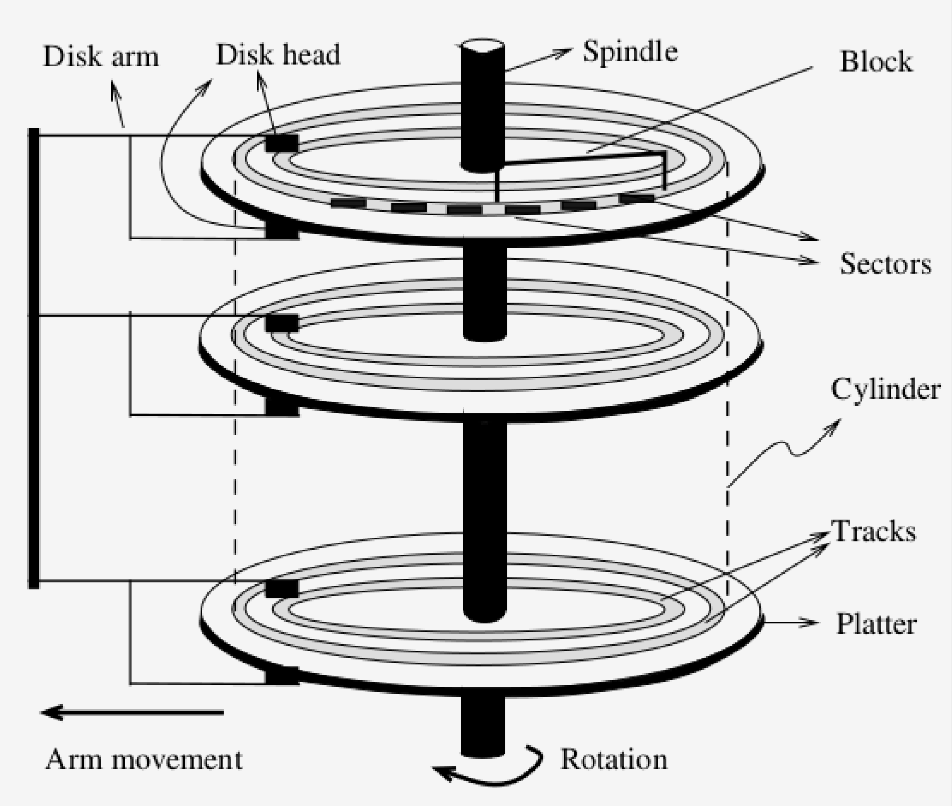
Adaptation of Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

Plan

- Secondary storage structure
- File storage
- File organization
- Introduction to index
- Introduction to Hash index
- Introduction to tree index
- Index selection guidelines

Secondary storage structure

- Sector: smallest unit on disk (512 B or 4096B)
- Block or page is sequence of sectors
  - OS and DBMS work with block
- Read or write time for one block is 1 I/O.



- Read or write sequence of close blocks is faster.

File storage

- File is a chain of blocks
  - Continuous blocks or not
  - Block has unique address

  - Why second case happen?
  - Which case can be read faster?
- MySQL block size or page size is 16k

File storage

- Data file of DBMS

Logical presentation

TID	Tname	DoB	Degree	Field	#Dname
1	Sok Dara	01/01/85	Master	Math	Math
2	Sam Sambath	01/02/80	PhD	Mechanic of fluid	Mechanic
3	Sao Piseth	05/08/70	PhD	Biology	Environment
4	Tao Pisey	14/07/65	Engineer	Electronic	Electronic
5	Van Dany	08/12/87	Engineer	hydropower	Environment
…..	….	…	…	..	..

Storage presentation

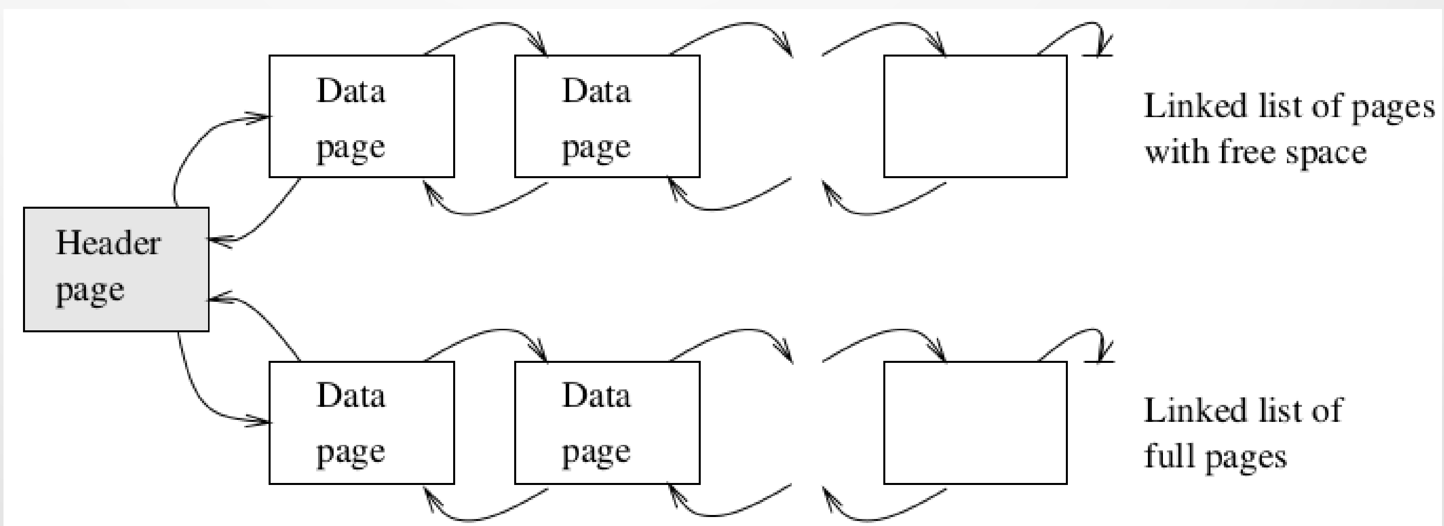
Page 1	Page 2	Page 3	….	Page N

rid	table's row
rid	table's row
rid	table's row
….	…..
free	space

- Record id (rid): unique record id
  - Contain information about where the row is located on disk

File organization

- Heap file: records are stored without any specific order.
  - First record will be stored in the first page
  - New record will be appended to the last inserted page
  - Or will be inserted into any page that has enough free space
- What will happen when some records are deleted?
- What will happen when some records are updated , and its size change?



File organization

- Sort file: records are stored with a specific order.
  - For example : order by primary key attribute.
  - Insert :
    - find the right page and right order
    - Move other records to make space for the new row
  - What will happen when some records are deleted?
  - What will happen when some records are updated , and its size change?
- Cluster index file: the order of records in data file is the same or similar to the order of records in index file.

Introduction to index

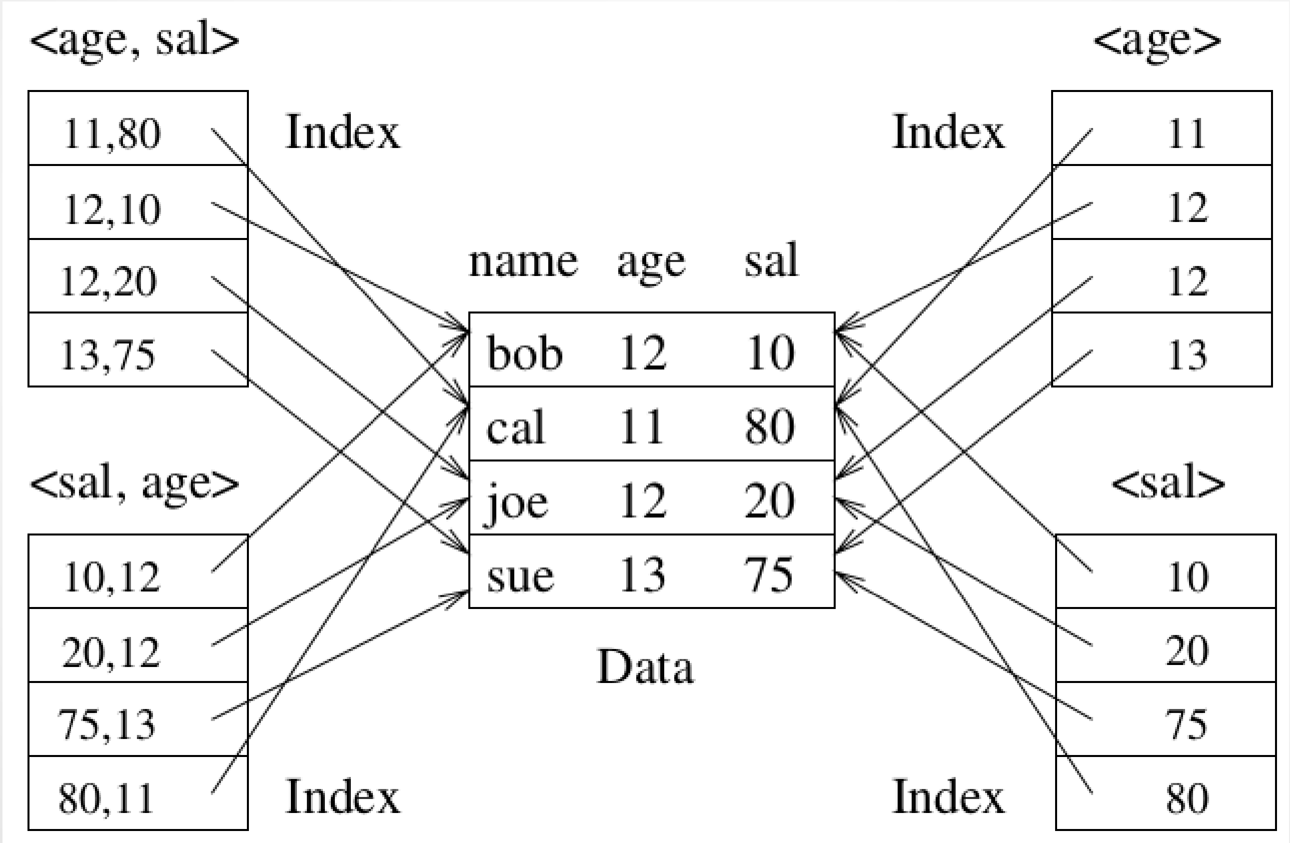
- Index: is an auxiliary structure designed to speed up query.
  - It provide efficient way to locate data with a given search key.
  - Similar to the index of a book

SUBJECT INDEX

AUTHOR INDEX

Introduction to index

- Example : Employees(name, age, sal)

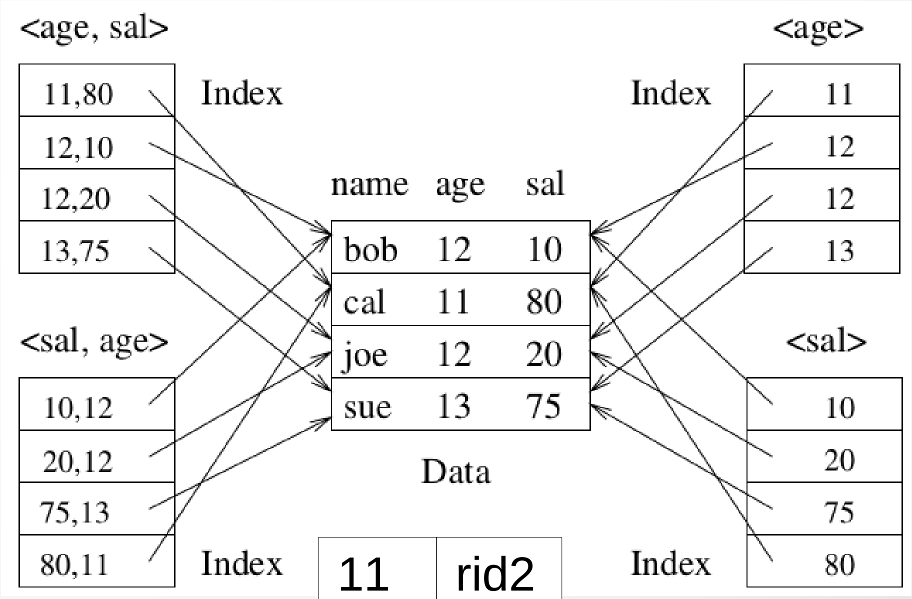


Introduction to index

- Index is stored in file just like data of table
  - Format of record in index file: <value, address>
    - <k,rid>
    - <k, list-rid>

      k: value of search key

      rid: rid of row that much k



Data file

rid1	Bob 12 10
rid2	Cal 11 80
rid3	Joe 12 20
rid4	Sue 13 75
….	…..

11	rid2
12	rid1
12	rid3
13	rid4
….	…..

Index file

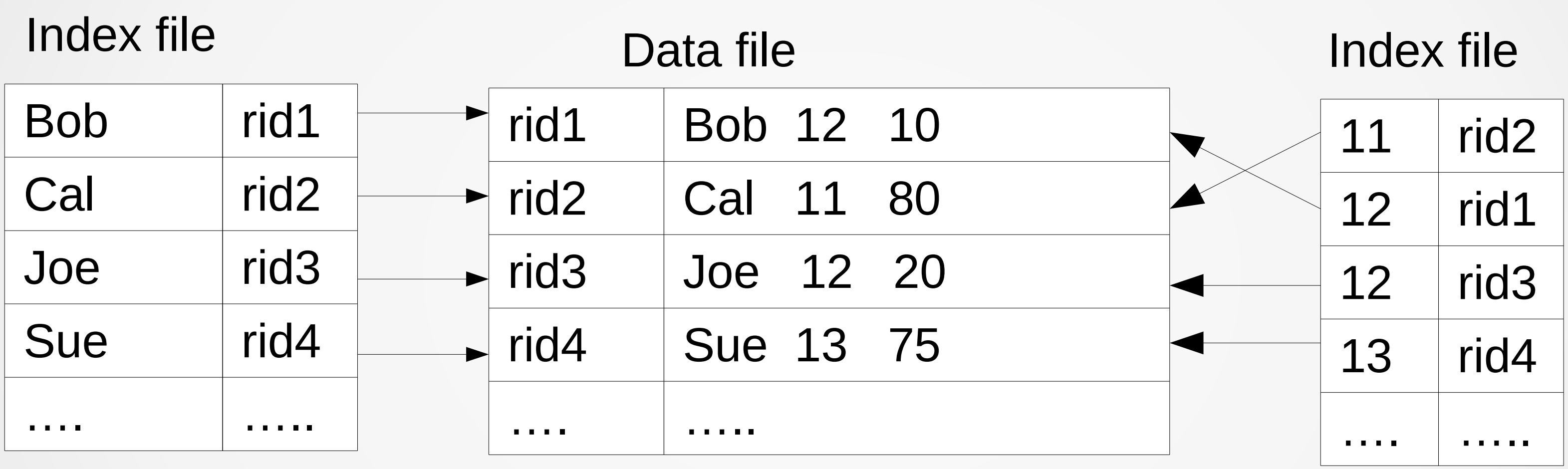
Introduction to index

Index classification

- Primary vs secondary: If search key contains primary key, then called primary index.
  - Unique index: Search key contains a candidate key.
- Clustered vs unclustered : If order of data records is the same as, or `close to', order of data entries (records in index file), then called clustered index.
  - A file can be clustered on at most one search key.
  - Cost of retrieving data records through index varies greatly based on whether index is clustered or not!

Introduction to index

- Cluster vs unclustered index



Introduction to index

- Questions:
  - How to build an index? Who is responsible for the task?
  - How to choose search key?
  - What type of index?

Introduction to index

- Practically the size of index is about 10% of the size of data.
  - Smaller than data, but still big for searching

    => Study on efficient structure of index
- Cost of Index
  - Space consumption
  - Cost in maintenance
    - When there is any update to the data, index has to be updated as well.

      => slow down the update of data

Hash index

- Hash function: Map an input of any size into a fixed size output
- Example: a hash function that sum ASCII code of all characters of input and modulo it with 6

Hash(‘sok dara’) = 5;

Hash(‘sok pisey’) = 1;

Hash(‘hash function’) = 2;

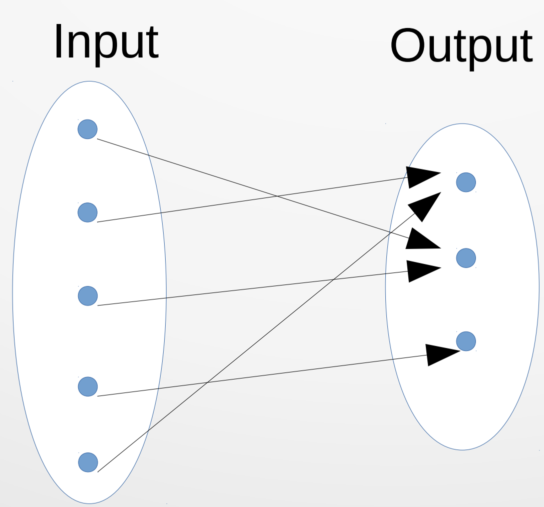
Hash(‘b’) = 2;

Hash(‘d’) = 4;

Hash(‘f’) = 0;

Hash index

- Hash function can be used in
  - Data encryption
  - Data storage and searching
- Some property
  - The same input always returns the output
  - Different inputs can return the same output (called collision)



Hash index

SELECT * FROM employees where eid = 12345;

- Heap file without index: scan table ; for the worst case

I/O cost = B (number of data pages)

- Heap file with index on eid

I/O cost = binary search on index file + 1I/O for retrieve data

- What can be improve?

I/O cost for searching on index file

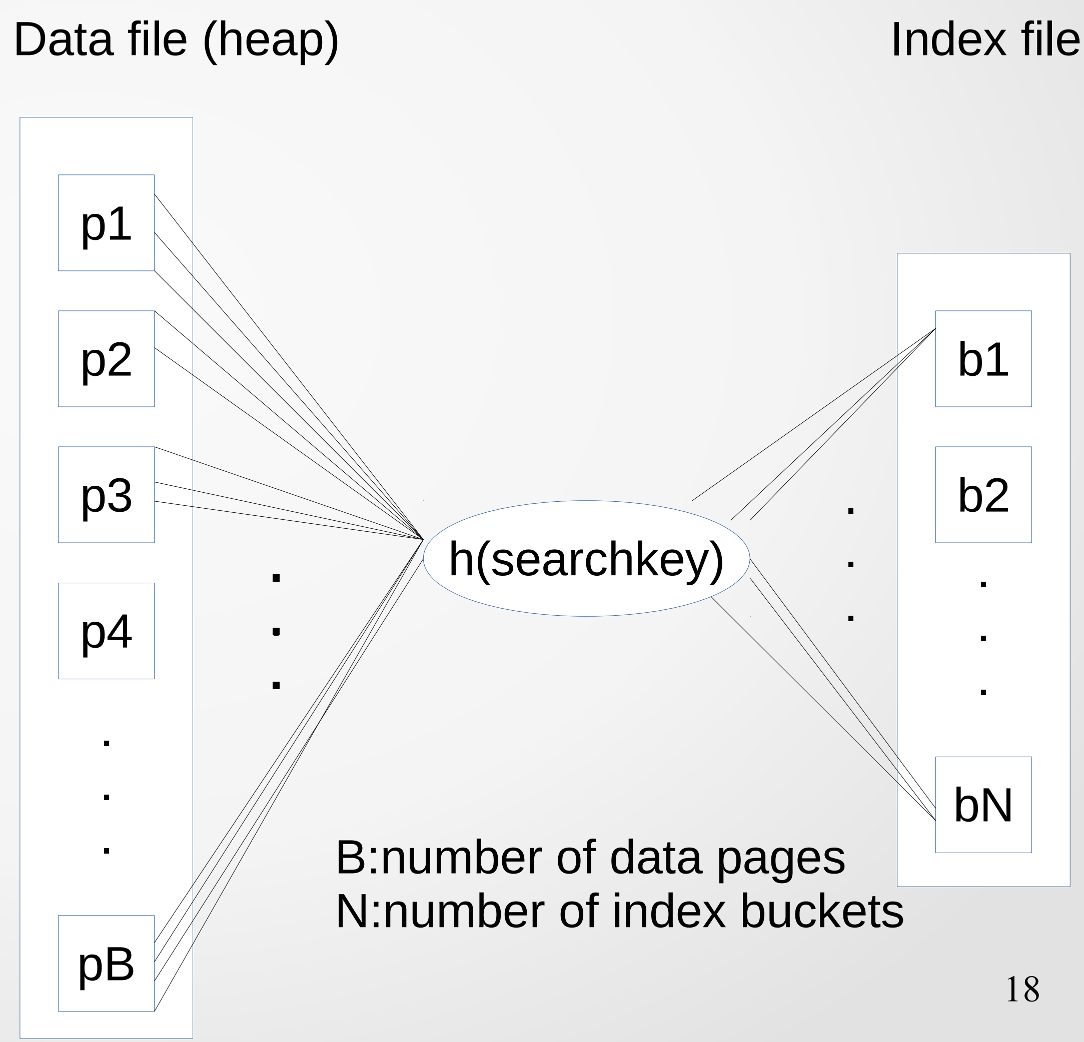
Hash index

- Hash index: is a collection of buckets.
- Build index
  - Scan data file

h(search key) = n (bucket nº)

Insert <key, rid> to bucket n

- Data retrieval
  - h(search key) = n
  - Read bucket n to get rid
  - Read data page pi to get data



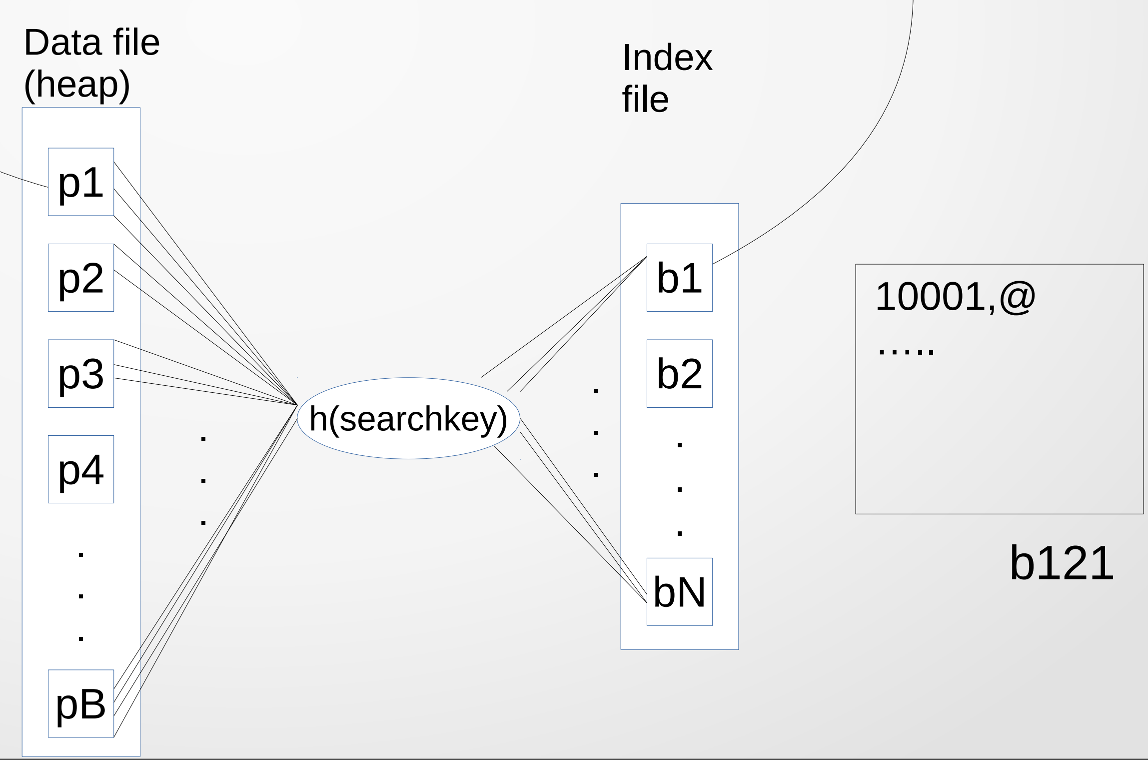
Hash index

- Closer look

@10001, Georgi Facello, 47, 88958, d005
@10002, Bezalel Simmel, 36, 72527, d007
….
@10122, Ohad Esposito, 35, 48464, d005
……

10122,@
10363, @
….

```
Int h(int skey){
  Return (skey mod 241) + 1;
}
```



- Why 241?
- Estimate I/O cost :

Select * from employees

Where eid = 12345;

Hash index

- Problem
  - Build index

if h(k) = n where bucket n is full, what is the solution?

  - Is hash index on age useful for this query?

Select * from employees where age = 35;

This type of query is called equality selection

  - How about this ?

Select * from employees where age > 35;

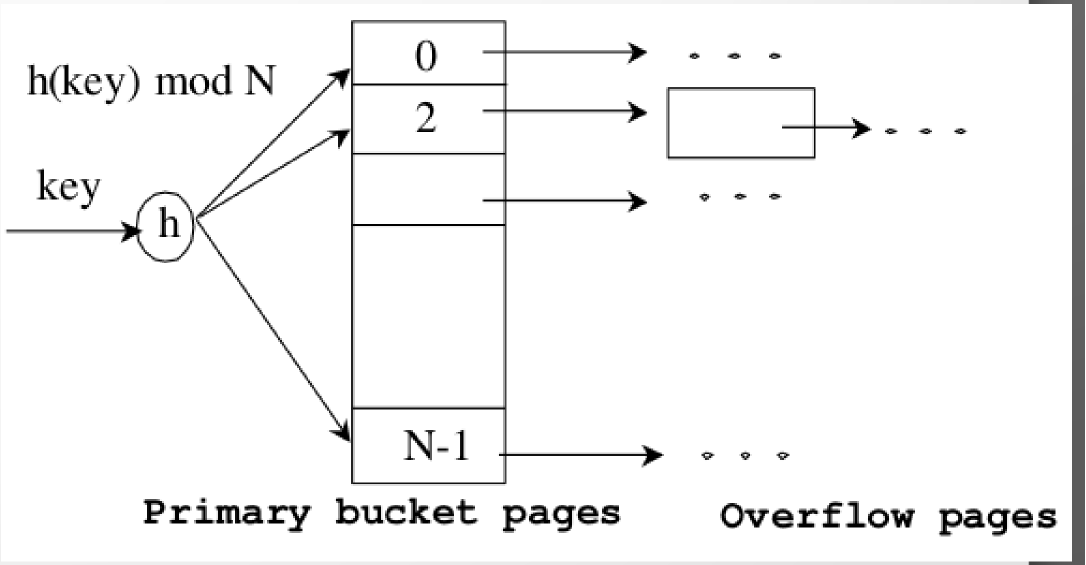
This type of query is called **range selection**

Hash index

- Deal with the first problem
  - Well defined hash function
  - Good decision on the number of buckets and its size
  - Overflow buckets
    - Fewer level of overflow page is better
- No overflow buckets.

80% page occupancy

=> index size = 12.5% data size



Tree index

Select * from employees where age > 35;

- Heap file without index: scan table ;

I/O cost = B (number of data pages)

- Heap file with index on age

I/O cost = binary search on index file

+read index pages with search key

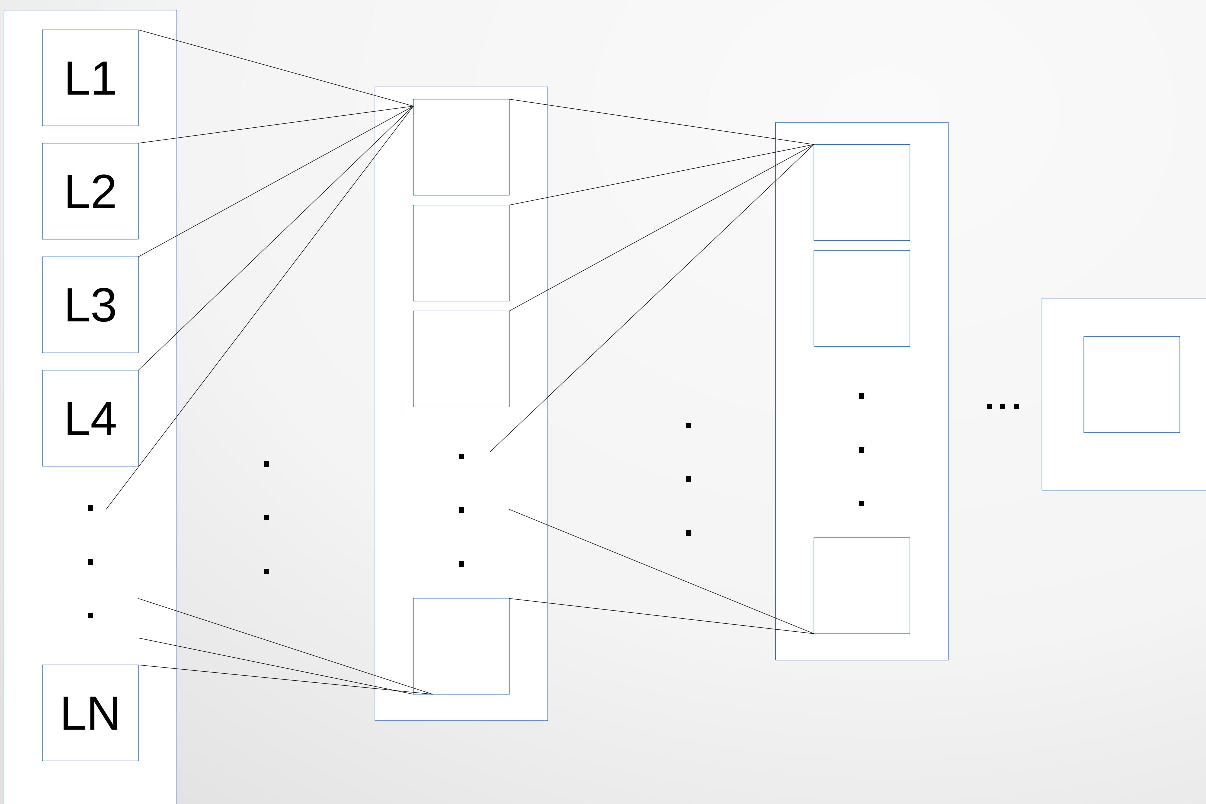
+ I/O for retrieve data (Worst case 1 I/O per row)

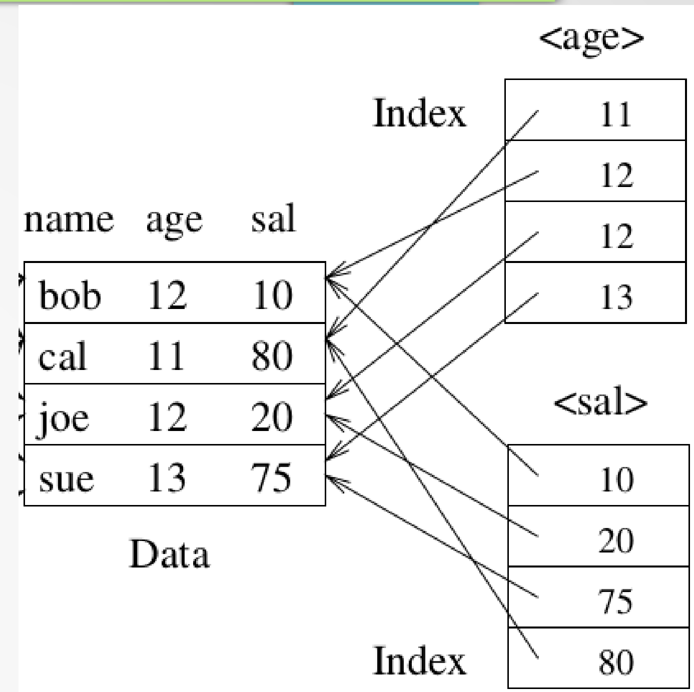
- If most of employees are older than 35, scan table is a lot more faster.

Tree index

- Improvement on index structure
  - Index over index file

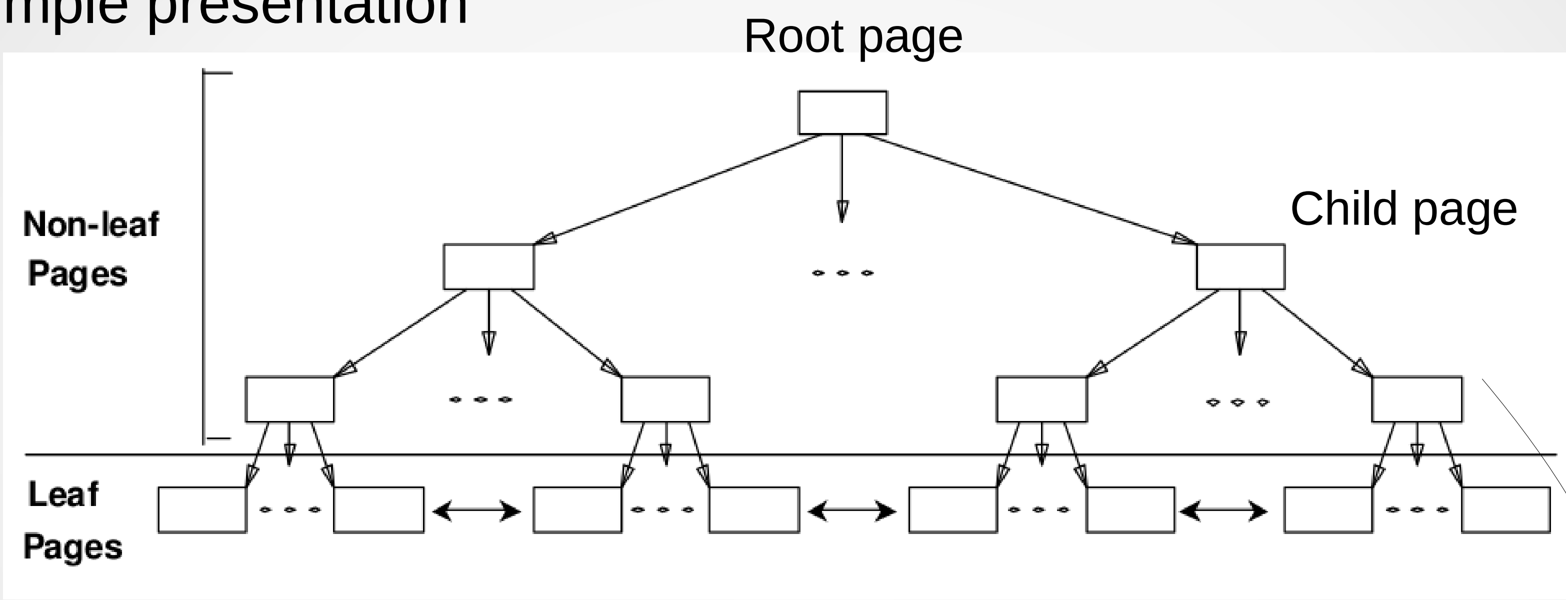
Index file (leaf pages)





Tree index

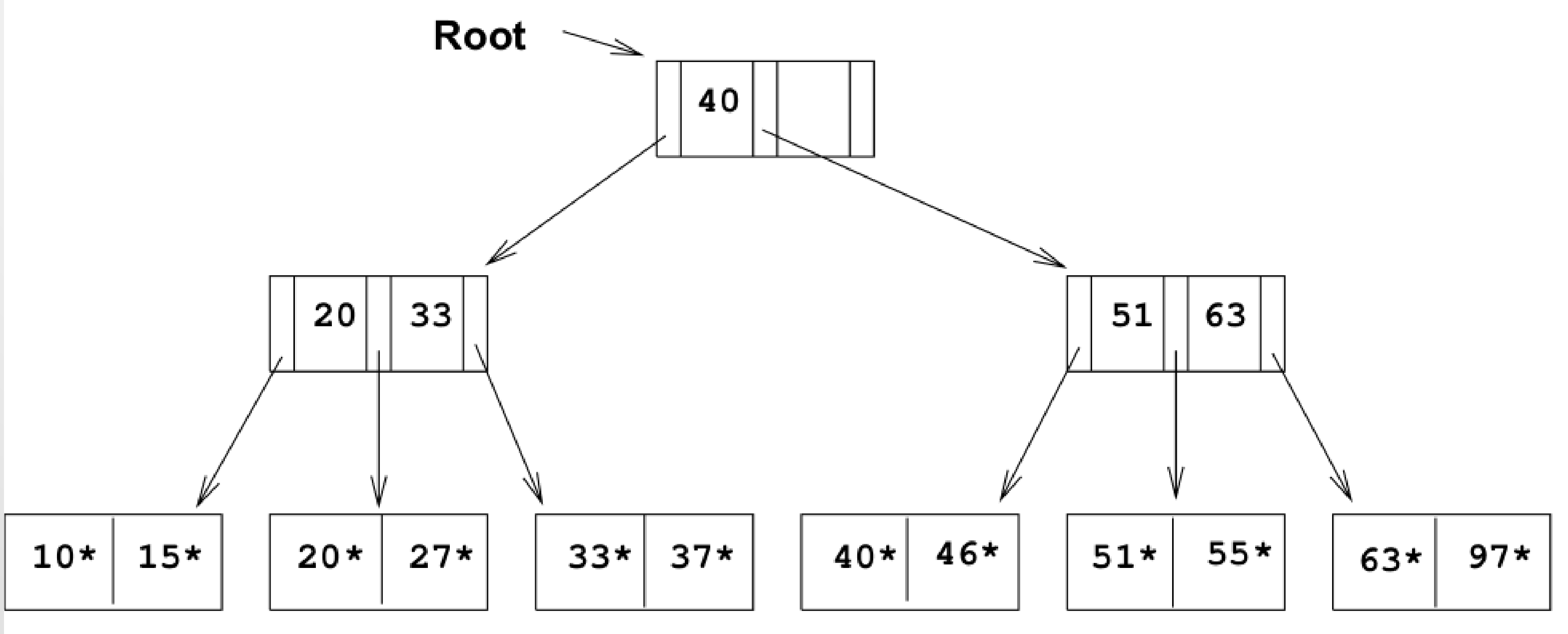
- Simple presentation



- Leaf pages contain data entries, and are chained (prev & next)
- Non-leaf pages contain index entries and direct searches:

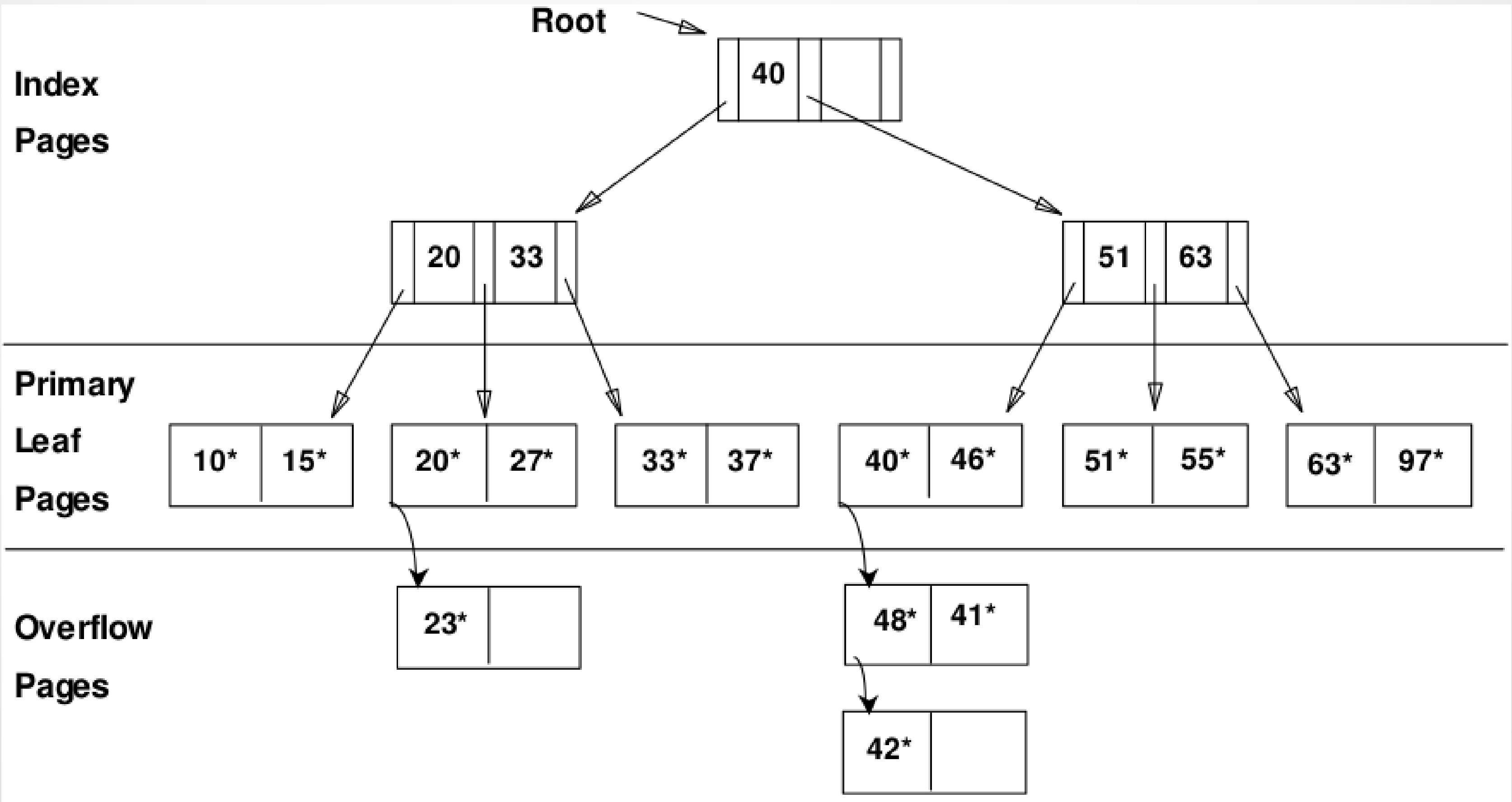
Tree index

- Static tree index (ISAM)
- Example: employees (name, age, sal) with tree index on age
  - Search for rows with k = 27?



Tree index

- Static tree index (ISAM)
  - Overflow pages needed for the new inserted row



Tree index

- Static tree index (ISAM)
  - Leaf pages are sequentially allocated at the creation of index
    - No link between leaf pages
  - Limit to only 80% occupation at the creation of index (for update of the table)
  - The structure of the tree is static. The update to the table only change the leaf and over flow pages.
  - Long list of over pages can hurt the searching performance.
    - Solution? Delete the index and recreate it again.

Tree index

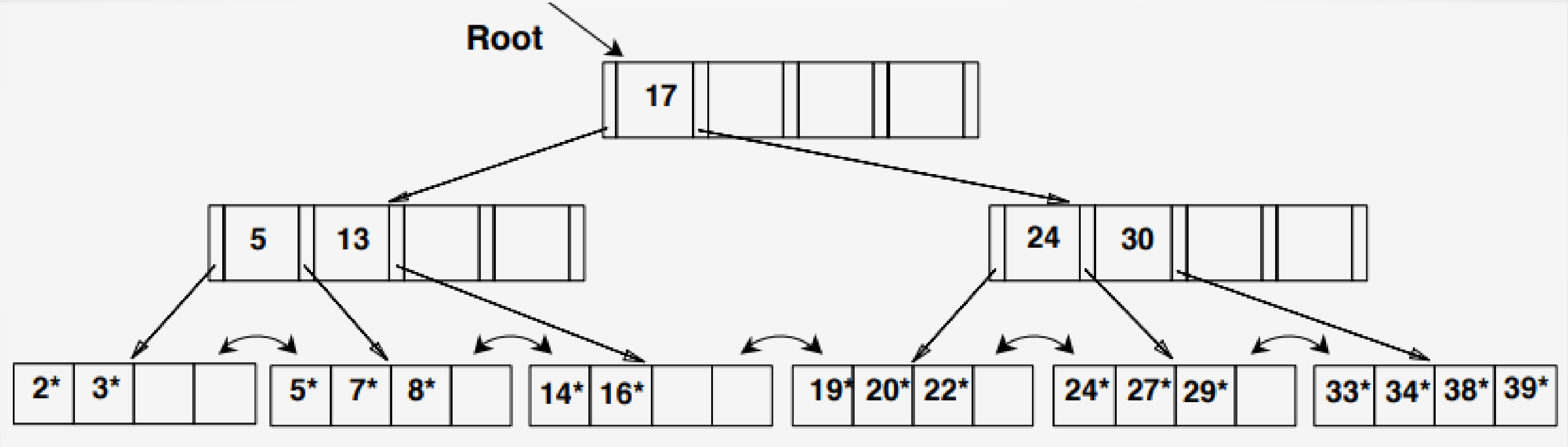
- Dynamic tree index : B tree
  - Insert/delete at $\log_F N$ cost; keep tree height-balanced.

    F = fanout (average child nodes) , N = # leaf pages
  - Minimum 50% occupancy (except for root).

    Each node contains d <= m <= 2d entries; d is called the order of the tree.
  - Supports equality and range-searches efficiently.
- In practice
  - Limit to 67% occupation of non-leaf node
  - D is around 100
  - F is around 133
  - Height : no more than 3 or 4

Tree index

- Example: B tree index with height H = 2
  - Search for rows with k = 5, k = 14, k = 15, k = 24



- I/O cost for search traverse the tree = H

Index selection guidelines

- For each query in the workload:
  - Which relations does it access?
  - Which attributes are retrieved?
  - Which attributes are involved in selection/join conditions?
  - How selective are these conditions likely to be?
- For each update in the workload:
  - Which attributes are involved in selection/join conditions?
  - How selective are these conditions likely to be?
  - The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

Index selection guidelines

- For each query in the workload:
  - Which relations does it access?
  - Which attributes are retrieved?
  - Which attributes are involved in selection/join conditions?
  - How selective are these conditions likely to be?
- For each update in the workload:
  - Which attributes are involved in selection/join conditions?
  - How selective are these conditions likely to be?
  - The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

Index selection guidelines

- What indexes should we create?
  - Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- For each index, what kind of an index should it be?
  - Clustered? Hash/tree?

Index selection guidelines

- One approach: Consider the most important queries . Consider the best plan using the current indexes, and see if a better plan is possible with an additional index. If so, create it.
  - Obviously, this implies that we must understand how a DBMS evaluates queries and creates query evaluation plans!
  - For now, we discuss simple 1-table queries.
- Before creating an index, must also consider the impact on updates in the workload!
  - Trade-off: Indexes can make queries go faster, updates slower. Require disk space, too.

Index selection guidelines

- Attributes in WHERE clause are candidates for index keys.
  - Exact match condition suggests hash index.
  - Range query suggests tree index.
    - Clustering is especially useful for range queries; can also help on equality queries if there are many duplicates.
- Multi-attribute search keys should be considered when a WHERE clause contains several conditions.
  - Order of attributes is important for range queries.
  - Such indexes can sometimes enable index-only strategies for important queries.
    - For index-only strategies, clustering is not important!

Index selection guidelines

- Try to choose indexes that benefit as many queries as possible.
- Since only one index can be clustered per relation, choose it based on important queries that would benefit the most from clustering.

Index selection guidelines

- Consider the following queries:
  - What does the query ask for?
  - How can the query be evaluated without any index or sorted file organization?
  - Suggest index or file organization to speed up the query.

```
SELECT E.dno
FROM Emp E
WHERE E.age>40
```

```
SELECT E.dno, COUNT (*)
FROM Emp E
WHERE E.age>10
GROUP BY E.dno
```

```
SELECT E.dno
FROM Emp E
WHERE
E.hobby=Stamps
```

Index selection guidelines

- B+ tree index on E.age can be used to get qualifying tuples.
  - How selective is the condition?
  - Is the index clustered?

```
SELECT E.dno
FROM Emp E
WHERE E.age>40
```

- Consider the GROUP BY query.
  - If many tuples have E.age > 10,

    using E.age index and sorting the retrieved tuples may be costly.
  - Clustered E.dno index may be better!

```
SELECT E.dno, COUNT (*)
FROM Emp E
WHERE E.age>10
GROUP BY E.dno
```

- Equality queries and duplicates:
  - Clustering on E.hobby helps!

```
SELECT E.dno
FROM Emp E
WHERE
E.hobby=Stamps
```

Index selection guidelines

- Which of the following index can help to speed up each query

a) B+ index on <age, sal>

b) B+ index on <sal, age>

c) Hash index on <sal>

d) B+ index on <sal>

e) Hash index on <age>

f) B+ index on <age>

```
SELECT *
FROM EMP
WHERE sal > 100;
```

```
SELECT *
FROM EMP
WHERE sal > 100 and age < 30;
```

```
SELECT *
FROM EMP
WHERE sal > 100 and age = 30;
```

Index selection guidelines

- Index only plan
- Which of the following index can help to speed up each query

a) Hash index on <age>

b) B+ index on <age>

```
SELECT Min(age)
FROM EMP;
```

```
SELECT Max(age)
FROM EMP;
```

```
SELECT avg(age)
FROM EMP;
```

Index selection guidelines

- Consider the following queries, suggest index that allow index only plan.

```
SELECT D.mgr
FROM Dept D, Emp E
WHERE D.dno=E.dno
```

```
SELECT D.mgr, E.eid
FROM Dept D, Emp E
WHERE D.dno=E.dno
```

```
SELECT E.dno, COUNT (*)
 FROM Emp E
GROUP BY E.dno
```

```
SELECT E.dno, MIN (E.sal)
FROM Emp E
GROUP BY E.dno
```

```
SELECT AVG (E.sal)
FROM Emp E
WHERE E.age=25 AND
E.sal BETWEEN 3000 AND 5000
```

Index selection guidelines

- Which one is better?

a) B+ tree <dno, age>

b) B+ tree <age, dno>

```
SELECT E.dno, COUNT (*)
FROM Emp E
WHERE E.age=30
GROUP BY E.dno
```

```
SELECT E.dno, COUNT (*)
FROM Emp E
WHERE E.age>30
GROUP BY E.dno
```

Index selection guidelines

- Consider slide Ch8_Storage_Indexing_Overview.pdf from page 19.