Course code : **CSE2007**
Course title : **Database Management System**
Module : **6**
Topic : **4**

Indexing and Single Level Ordered Index

Objectives

This session will give the knowledge about

- Indexing Structures for Files
- Single Level Ordered Indexes

Introduction

An index is an optional structure, associated with a table or table cluster, that can improve the data access.

It reduces the disk I/O

Indexes used to speed up record retrieval in response to certain search conditions

Index structures provide secondary access paths

Any field can be used to create an index

Multiple indexes can be constructed

Introduction

Most indexes based on ordered files

Tree data structures organize the index

Student file should be ordered using ordering key

Sid	Dept	Name	Mark
101	BCD	Tony	89
102	BCN	Roy	90
104	BCD	Mala	95

Sid	Dept	Name	Mark
111	BCI	Reddy	99
112	BCN	Rao	70
114	BCD	Harini	85

Sid	Dept	Name	Mark
151	BCD	Menon	93
152	BCI	Nair	96
154	BCD	Komal	91

Field	Pointer
101	T1S3B2
111	T1S3B4
151	T1S3B6

If the field is a Primary key of the record, then index is Primary Index

Types of Single-Level Ordered Indexes

Ordered index similar to index in a textbook

Indexing field (attribute)

- Index stores each value of the index field with list of pointers to all disk blocks that contain records with that field value

Values in index are ordered

Primary index

- Specified on the ordering key field of ordered file of records

Types of Single-Level Ordered Indexes

Clustering index

Used if numerous records can have the same value for the ordering field

Secondary index

Can be specified on any non-ordering field

Data file can have several secondary indexes

Primary Indexes

Ordered file with two fields

- Primary key, *K(i)*
- Pointer to a disk block, *P(i)*

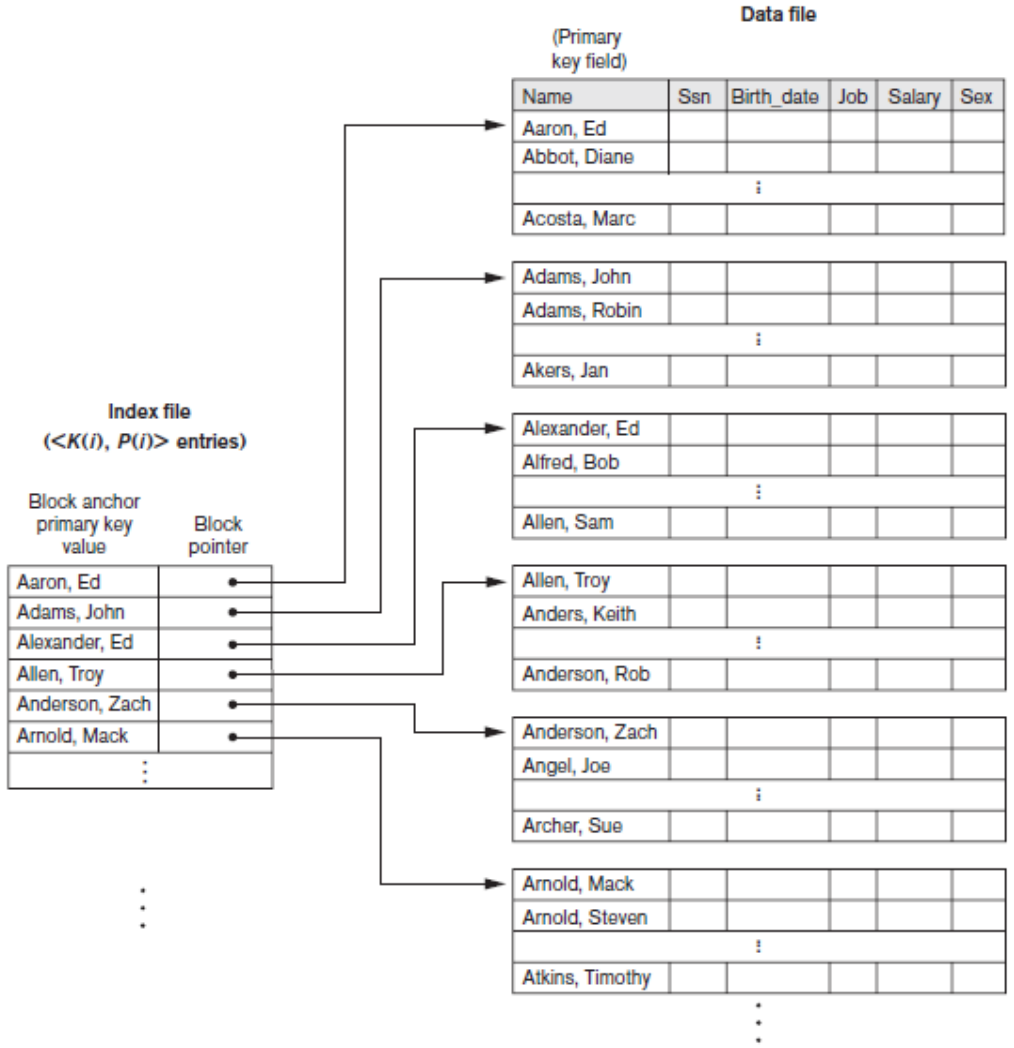
One index entry in the index file for each block in the data file

Indexes may be dense or sparse

- Dense index has an index entry for every search key value in the data file
- Sparse index has entries for only some search values

Primary Indexes

Primary index on the ordering key field of the file shown in Figure



Primary Indexes: Problem

Assume a file (b) consists of:

Total no of records (r): 30,000 Each record size (R): 100 bytes

Block size (B): 1024 bytes

How many records a block can hold (bfr)?

-> 1024/100 = 10 records/block

How many blocks will be required?

-> 30,000/10 = 3000 blocks

How many blocks should be accessed to perform binary search?

$Log_2(Blocks) = Log_2(3000) = 12$ blocks

Primary Indexes: Problem

Assume: (With Primary Index)

Total no of records: 30,000 Each record size: 100 bytes

Block size: 1024 bytes Index size: 15 bytes (9 for field, 6 for pointer)

How many records a block can hold? -> 1024/100 = 10 records/block

How many blocks will be required? -> 30,000/10 = 3000 blocks

How many index can hold in a block? -> 1024/15 = 68 index/block

How many blocks required to store 3000 indexed? -> 3000/68 = 45 blocks

How many blocks should be accessed to perform binary search?

$Log_2(Blocks) = Log_2(45) = 6$ blocks

Additional disk access required to access from index to file : 1 block

There fore the final disk access using Index is : 6+1 = 7 blocks

Primary Indexes

Search operation is faster with Primary Indexes

No of access required in primary index of b blocks: $Log_2(Blocks) + 1$

Major problem: insertion and deletion of records

Move records around and change index values

Solutions:

- Use unordered overflow file
- Use linked list of overflow records

Clustering Indexes

Clustering field

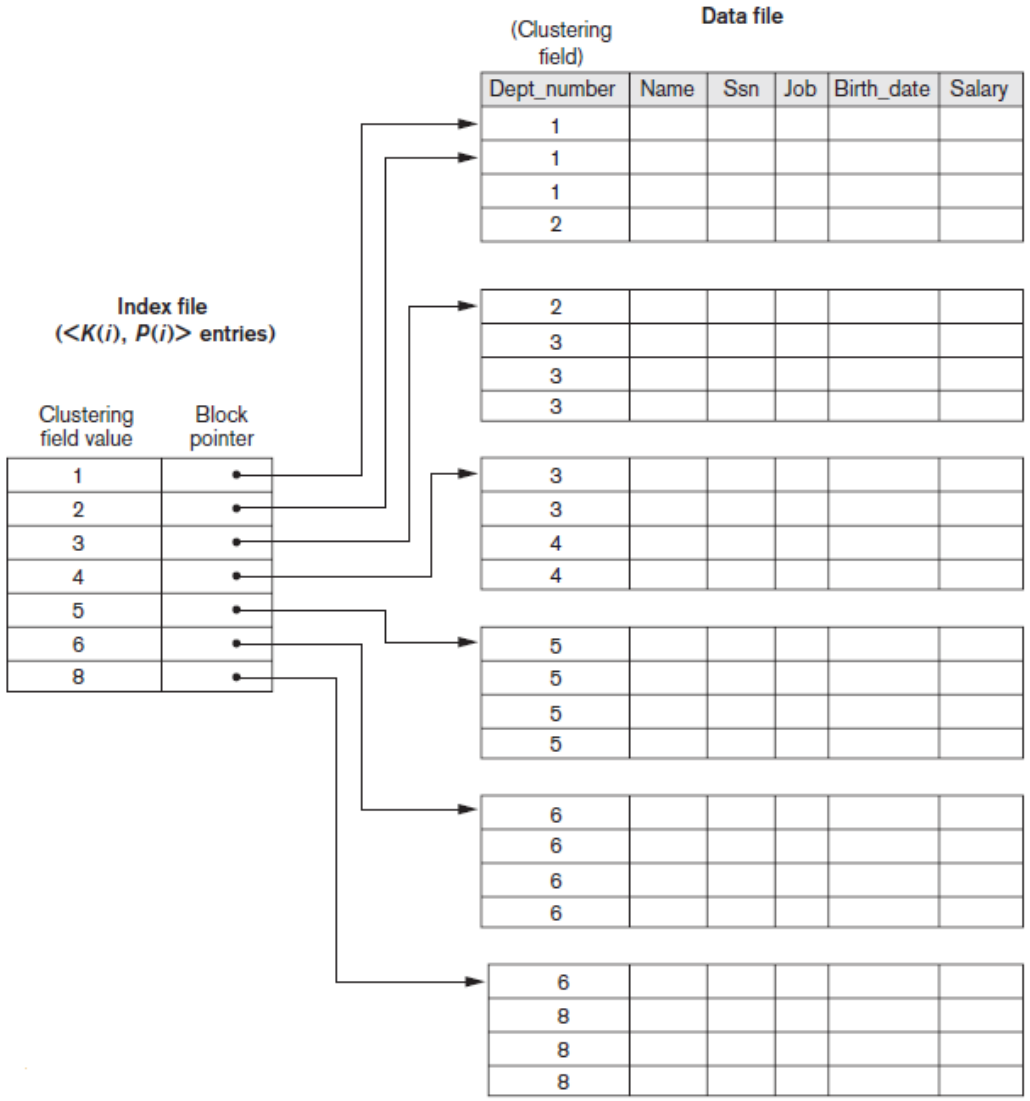
- File records are physically ordered on a non-key field without a distinct value for each record

Ordered file with two fields

- Same type as clustering field
- Disk block pointer

Clustering Indexes

clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file



Secondary Indexes

Provide secondary means of accessing a data file

- Some primary access exists

Ordered file with two fields

- Indexing field, *K(i)*
- Block pointer or record pointer, *P(i)*

Usually need more storage space and longer search time than primary index

- Improved search time for arbitrary record

Introduction

Most indexes based on ordered files

Tree data structures organize the index

Field	Pointer
70	
85	
89	
90	
91	
93	
95	
96	
99	

Field	Pointer
101	T1S3B2
111	T1S3B4
151	T1S3B6

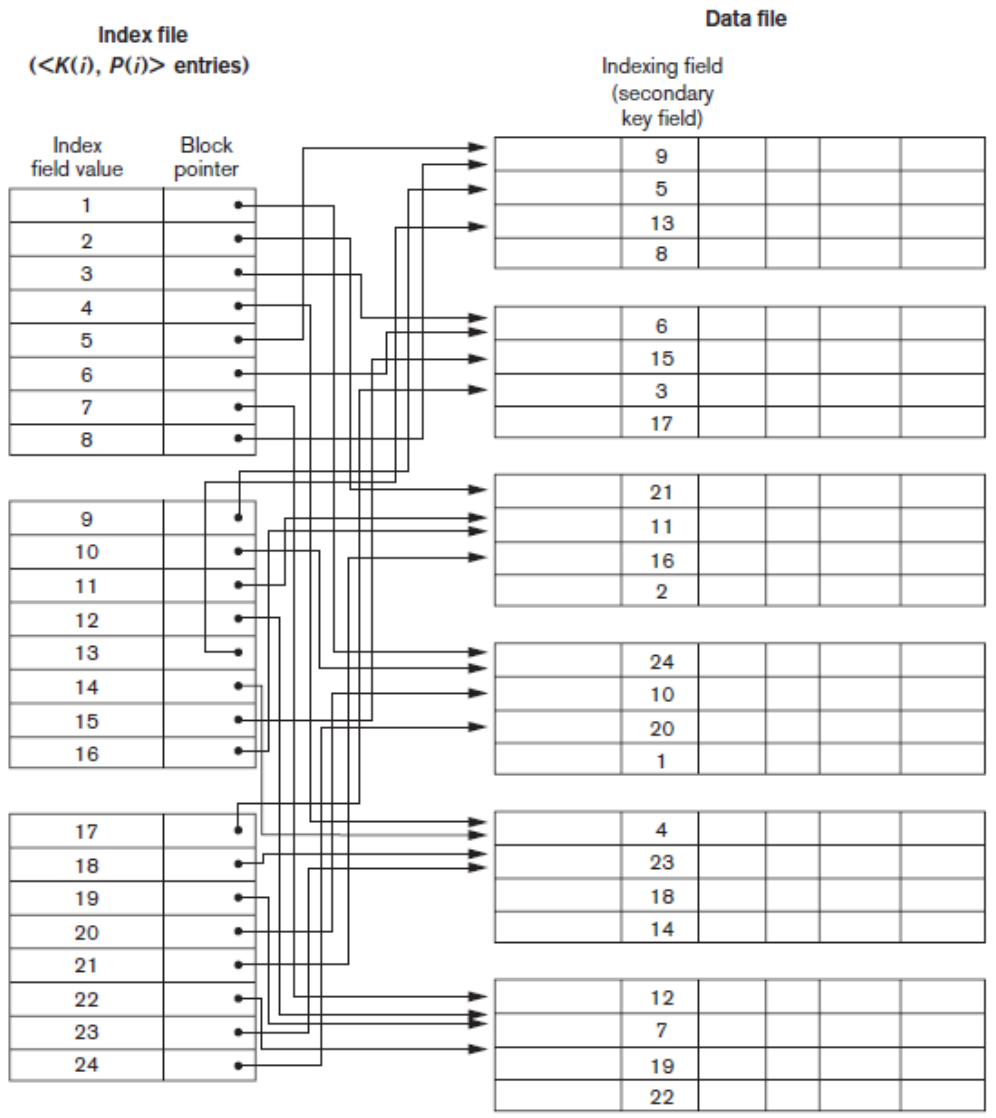
Sid	Dept	Name	Mark
101	BCD	Tony	89
102	BCN	Roy	90
104	BCD	Mala	95

Sid	Dept	Name	Mark
111	BCI	Reddy	99
112	BCN	Rao	70
114	BCD	Harini	85

Sid	Dept	Name	Mark
151	BCD	Menon	93
152	BCI	Nair	96
154	BCD	Komal	91

Secondary Indexes

Dense secondary index (with block pointers) on a non-ordering key field of a file.



Secondary Indexes: Problem

Assume: (With Secondary Index)

Total no of records: 30,000 Each record size: 100 bytes

Block size: 1024 bytes Index size: 15 bytes (9 for field, 6 for pointer)

How many records a block can hold? -> 1024/100 = 10 records/block

How many blocks will be required? -> 30,000/10 = 3000 blocks

How many index can hold in a block? -> 1024/15 = 68 index/block

How many blocks required to store 3000 indexed? -> 30000/68 = 442 blocks

How many blocks should be accessed to perform binary search?

$Log_2(Blocks) = Log_2(442) = 9$ blocks

Additional disk access required to access from index to file : 1 block

There fore the final disk access using Index is : 9+1 = 10 blocks

Secondary Indexes: Advantages

It enables binary search when primary index is not useful.

It is better than linear search but not than primary index.

Sparse Index:

If the index consists only few records, it is called sparse index.

Dense Index:

If you are creating an index for every record, it is called dense index.

Indexing Types

Types of indexes based on the properties of the indexing field

	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

Indexing Types

Properties of index types

Type of Index	Number of (First-Level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.
[b]For option 1.
[c]For options 2 and 3.

Exercises

Consider a disk with block size $B = 512$ bytes. A block pointer is $P = 6$ bytes long, and a record pointer is $P_R = 7$ bytes long. A file has $r = 30{,}000$ EMPLOYEE records of *fixed length*. Each record has the following fields: Name (30 bytes),Ssn (9 bytes), Department_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth_date (8 bytes), Sex (1 byte), Job_code (4 bytes), and Salary (4 bytes, real number). An additional byte is used as a deletion marker.

a. Calculate the record size R in bytes.

b. Calculate the blocking factor *bfr* and the number of file blocks b, assuming an unspanned organization.

Exercises

c. Suppose that the file is *ordered* by the key field Ssn and we want to construct a *primary index* on Ssn. Calculate (i) the index blocking factor bfr_i (which is also the index fan-out *fo*); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it into a multilevel index; (iv) the total number of blocks required by the multilevel index; and (v) the number of block accesses needed to search for and retrieve a record from the file—given its Ssn value—using the primary index.

d. Suppose that the file is *not ordered* by the key field Ssn and we want to construct a *secondary index* on Ssn. Repeat the previous exercise (part c) for the secondary index and compare with the primary index.

Summary

This session will give the knowledge about

- Indexing Structures for Files
- Single Level Ordered Indexes