Database Management Systems (DBMS)

Lec 20: Query processing and optimization (Contd.)

Ramesh K. Jallu
IIIT Raichur

Date: 09/04/21

Recap

- Introduction to query processing and query optimization
- Parameters to measure query costs
- Enhanced relational algebra operations
  - Generalized projection and aggregate functions
  - Outer Join of two relations R and S
    - **Left outer join**: All the tuples from the left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the right relation S, then the attributes of S in the resulting relation are made **NULL**
    - **Right outer join**
    - **Full outer join**

Today's overview

- One more example for outer join
- The OUTER UNION operation
- Introduction to indexing
- Algorithms for SELECTION operation

Example: Outer join

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course_id	*title*	*dept_name*	*credits*	*prere_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

The OUTER UNION Operation

- The OUTER UNION operation takes the union of tuples from two relations that have some common attributes, but are ***not*** union (type) compatible
- This operation will take the UNION of tuples in two relations ***R***(***X***, ***Y***) and ***S***(***X***, ***Z***), that are partially compatible, and produces a new relation ***T***(***X***, ***Y***, ***Z***)
- Two tuples t_1 in ***R*** and t_2 in ***S*** are said to match if $t_1[X] = t_2[X]$ These will be combined into a single tuple in ***t***
- Tuples in either relation that have nomatching tuple in the other relation are padded with **NULL** values

- STUDENT(Name, Ssn, Department, Advisor) and INSTRUCTOR(Name, Ssn, Department, Rank)
- STUDENT_OU_INSTRUCTOR(Name, Ssn, Department, Advisor, Rank)

Non-compatible Attributes

Name	SSN	Dept	Advisor
Ali	111	COE	Sami
Adel	222	EE	Khaled
Fahd	333	COE	Sami

U

Name	SSN	Dept	Rank
Sami	444	COE	FP
Khaled	555	EE	AP
Adel	222	EE	TA

$\downarrow$

Name	SSN	Dept	Advisor	Rank
Ali	111	COE	Sami	NULL
Adel	222	EE	Khaled	TA
Fahd	333	COE	Sami	NULL
Sami	444	COE	NULL	FP
Khaled	555	EE	NULL	AP

Indexing

- Many queries need only a small portion of a data record in a file
- Indexes are used to speed up the retrieval of records in response to certain search conditions
- Indexes are additional structures that are associated with files
- An attribute (or a set of attributes) used to look up records in a file is called a ***search key***
- We often want to have more than one index for a file: if there are several indices on a file, there are several search keys

Primary Index

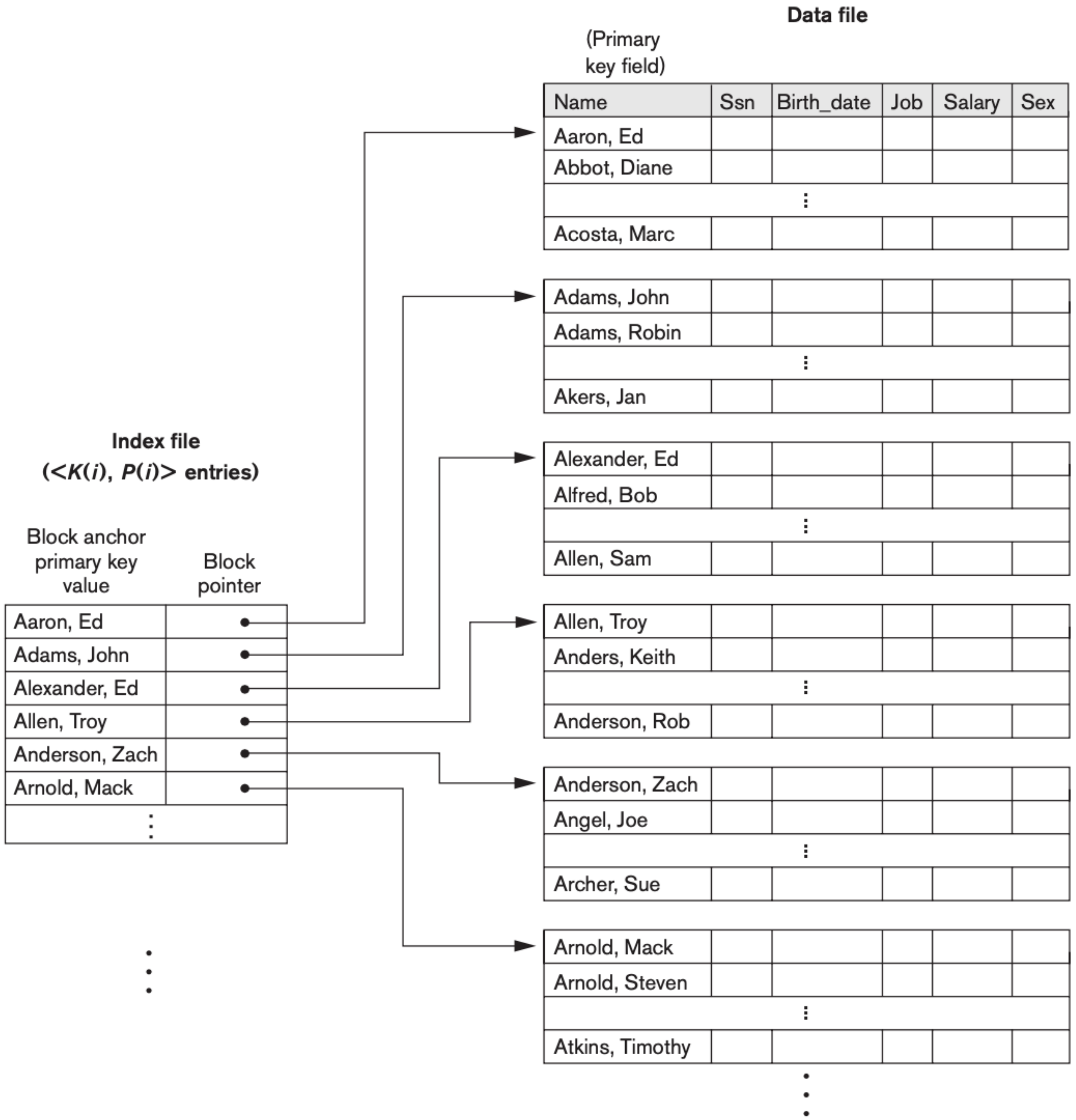
- Primary indexing is used on a ordered data file with search key as the primary key

- The structure contains two fields. The first field is same as the primary key and the second field contains a pointer to a block

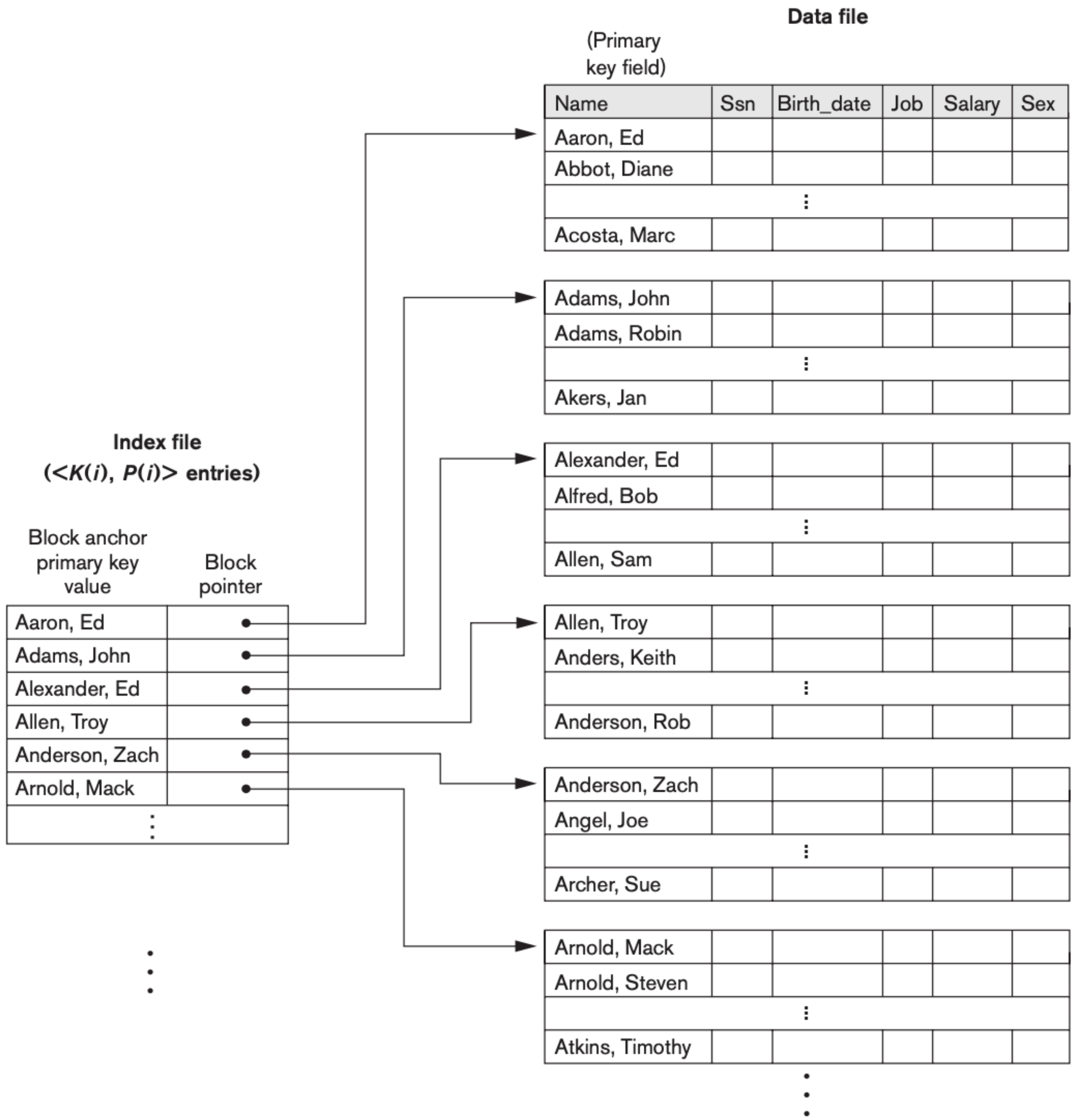
- There is one ***index entry*** (or ***index record***) in the index file for each *block* in the data file

Search key (***K(i)***)	Pointer to a block (***P(i)***)

- A ***record pointer*** uniquely identifies a record and provides the address of the record on disk



- Suppose that we have an ordered file with r = 300,000 records stored on a disk
- Block size B = 4,096 bytes and each record length R = 100 bytes
- Thus there are 40 records per block
- The number of blocks needed for the file are 7,500 blocks
- A binary search on the data file would need approximately 13 block accesses



- Now suppose that the ordering key field of the file is $V = 9$ bytes long, a block pointer is $P = 6$ bytes long

- The size of each index entry is 15 bytes, so 273 entries per block as each block is 4MB

- The total number of index entries is equal to the number of blocks in the data file, which is 7,500

- The number of index blocks is 28

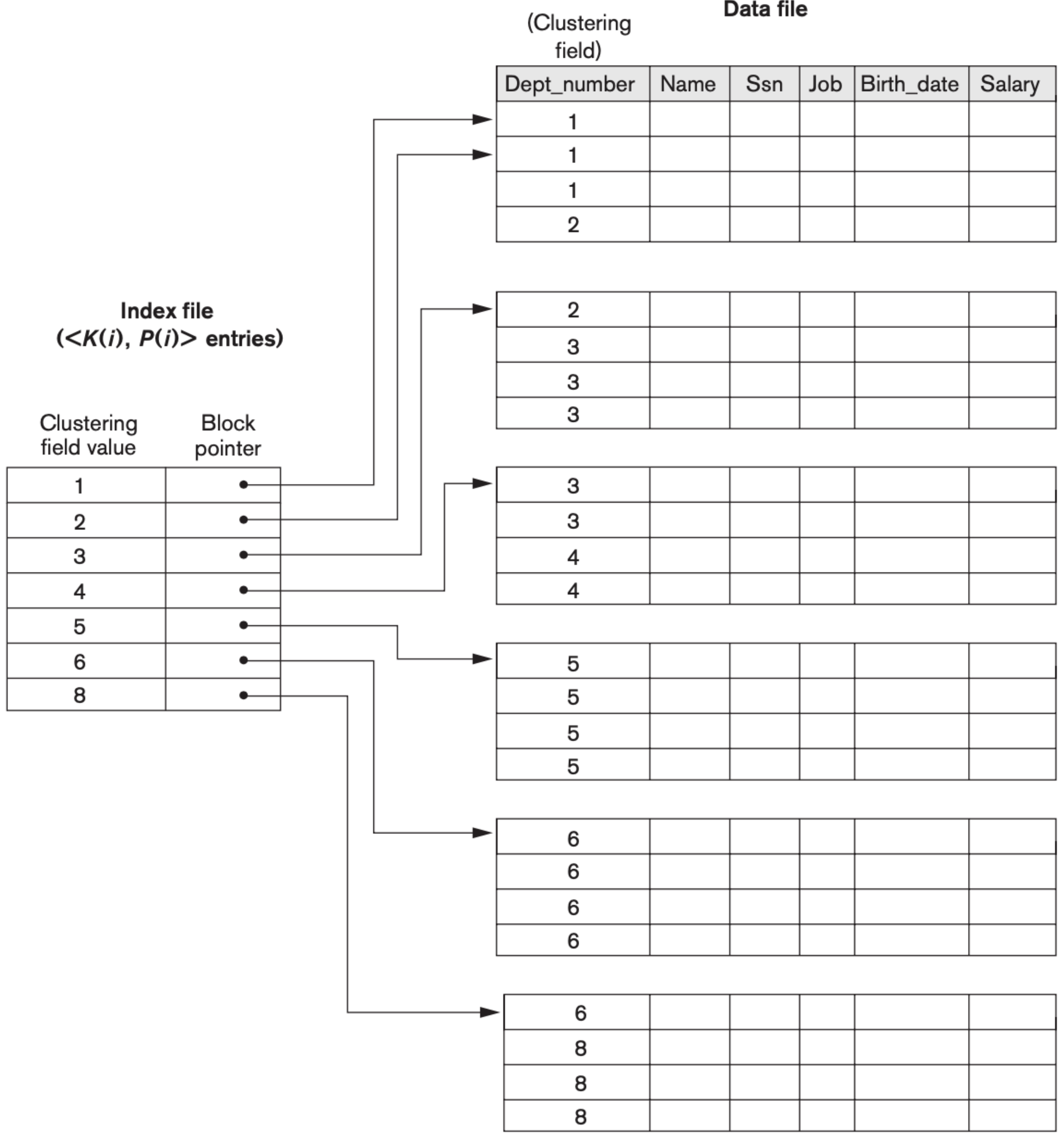
- To perform a binary search on the index file would need 5 block accesses

Clustered Index

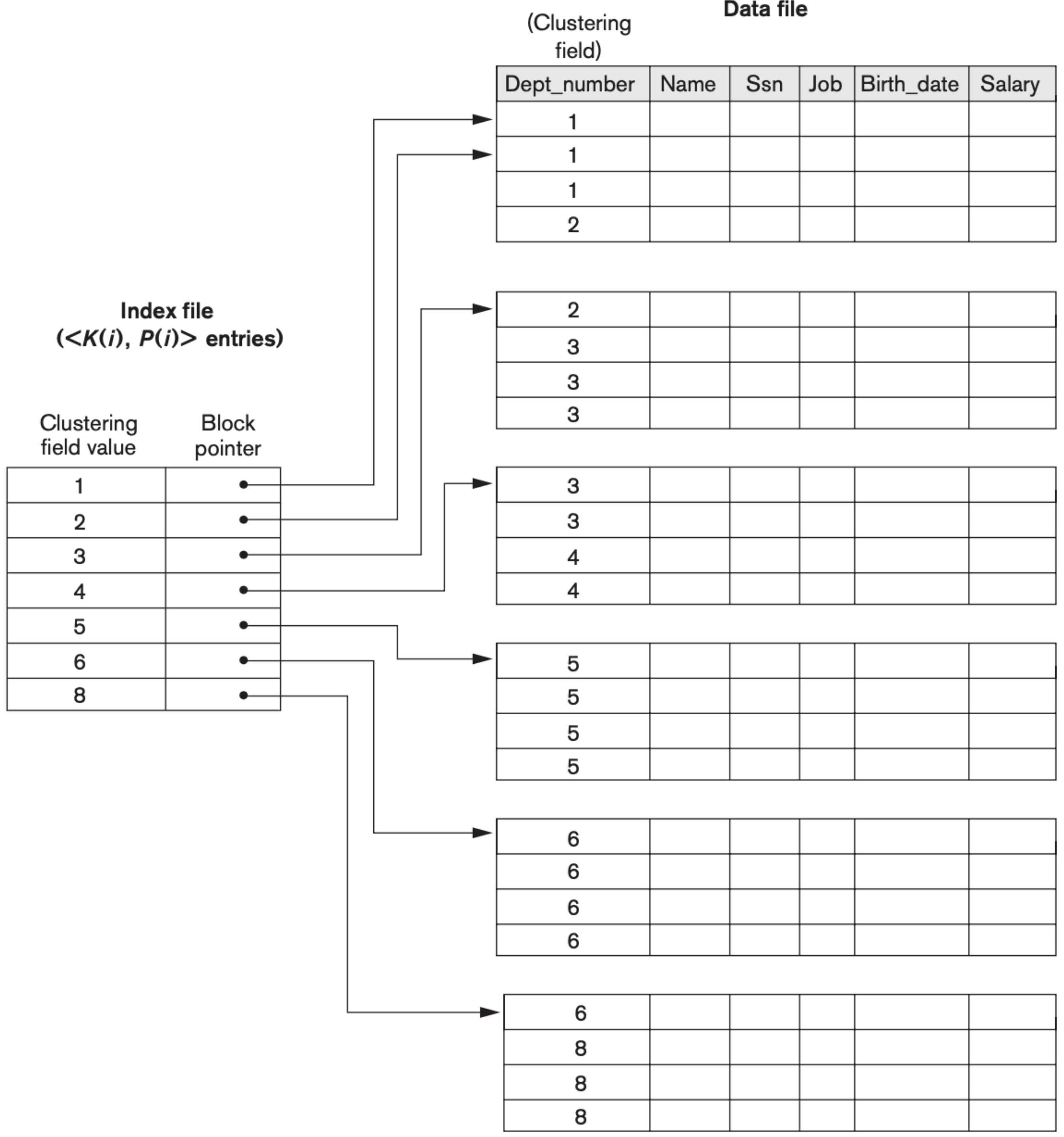
- If file records are ordered on a nonkey field that field is called the ***clustering field*** and the data file is called a ***clustered file***

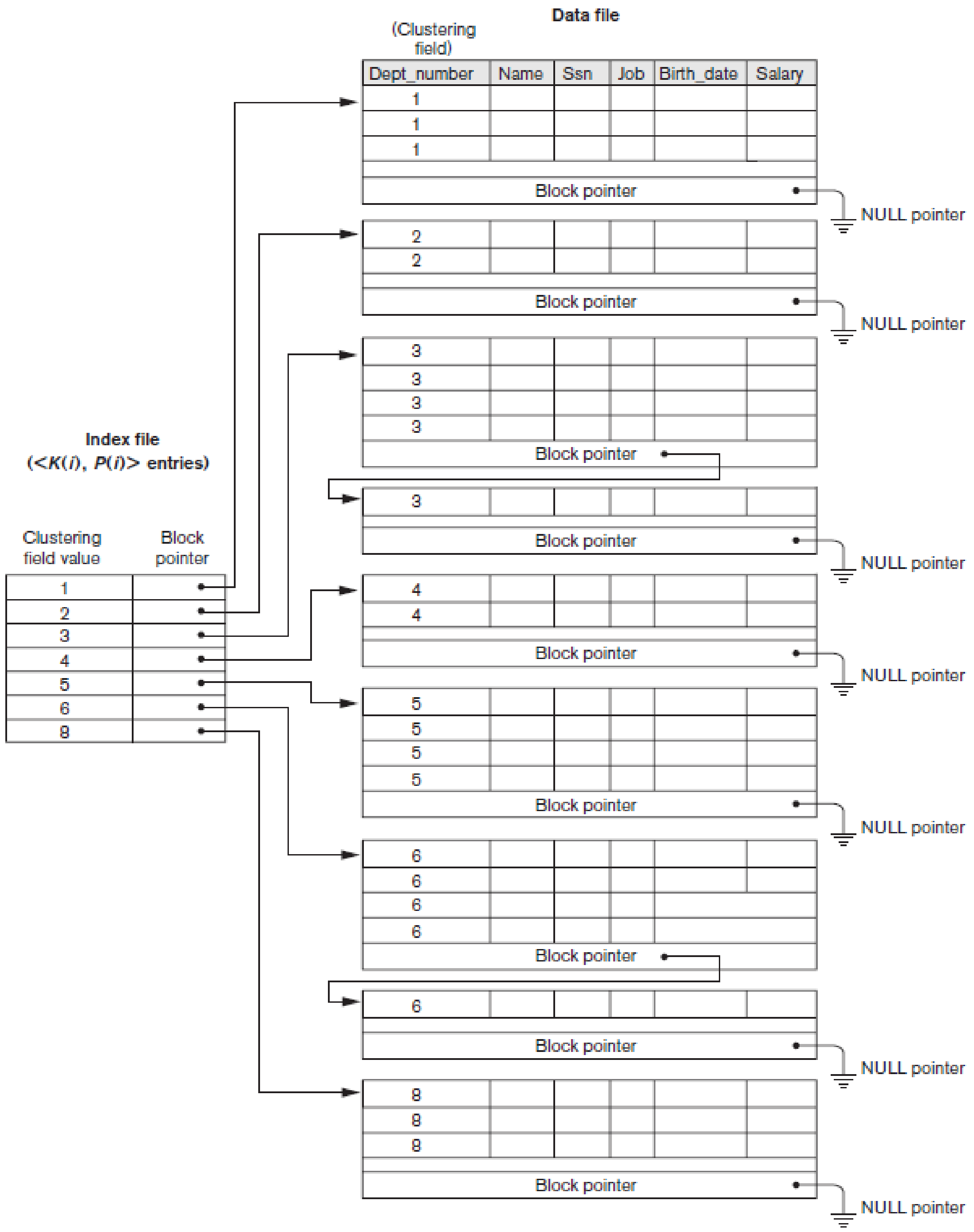
- The structure is same as the primary index: the first field contains the clustering field of the data file, and the second field is a disk block pointer

- There is one entry in the clustering index for each *distinct value* of the clustering field and a pointer to the *first block* in the data file that has a record with that value for its clustering field



- Suppose Dept_number has 1,000 distinct numbers in the file with an average 300 records per department number
- The size of each index entry is 15 bytes, so 273 entries per block as each block is 4MB
- The index in this case has 1,000 index entries, so the number of index blocks is 4
- To perform a binary search on the index file would need 2 block accesses


Data file
(Clustering field)
Dept_number
Name
Ssn
Job
Birth_date
Salary
Index file
(<K(i), P(i)> entries)
Clustering field value
Block pointer
1
2
3
4
5
6
8

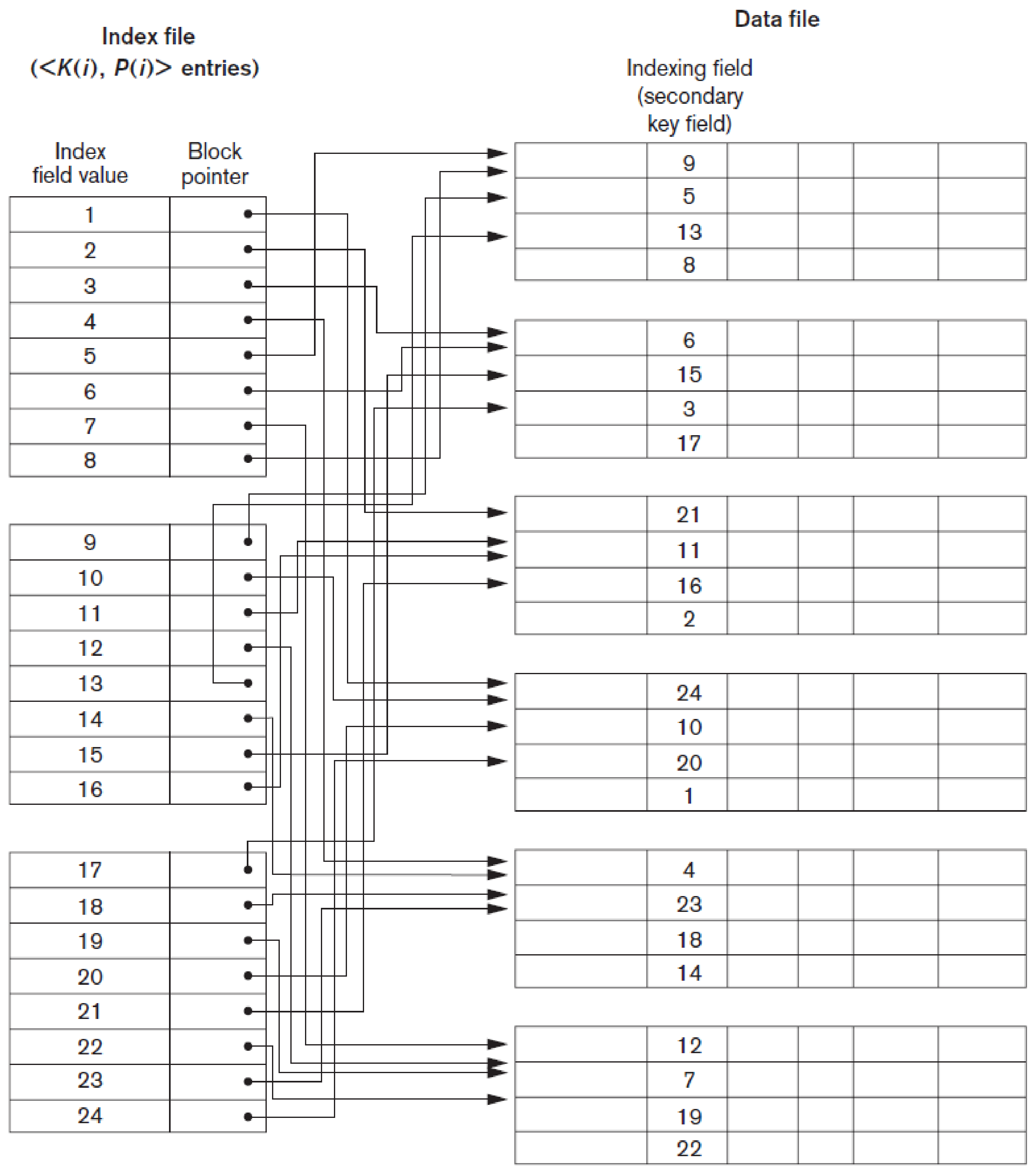

Data file
(Clustering field)
Dept_number
Name
Ssn
Job
Birth_date
Salary
Block pointer
NULL pointer
Index file
(<K(i), P(i)> entries)
Clustering field value
Block pointer
1
2
3
4
5
6
8

Secondary Indexes

- A secondary index provides a secondary means of accessing a data file for which some primary access already exists
- The data file records could be ordered or unordered
- The secondary index is created on a field that is a candidate key
- The index is an ordered file with two fields similar to primary and cluster indexing
  - The first field is of the same as the candidate key. The second field is either a *block pointer* or a *record pointer*

Secondary Indexes (Contd.)

- The entries are ordered by value of key because the records of the data file are not physically ordered by values of the secondary key field, we cannot use block pointers
- An index entry is created for each record in the data file, rather than for each block
- A secondary index usually needs more storage space and longer search time than does a primary index
- However, the improvement in search time for an arbitrary record is much greater for a secondary index than for a primary index



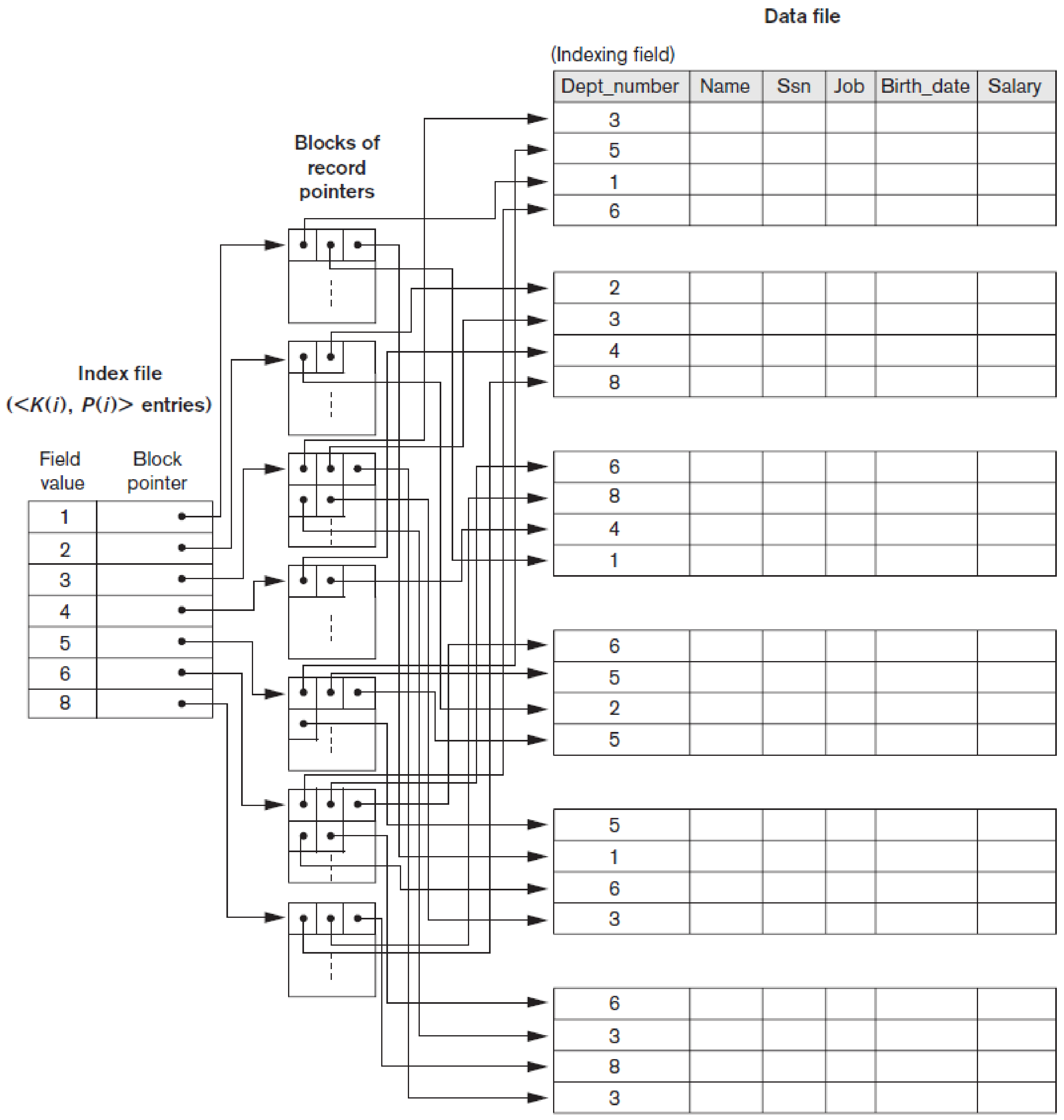
- Consider the data file with the same statastics as in Ex. of primary index
- Without the secondary index, to do a linear search on the file would require 3,750 block accesses on the average
- The total number of index entries is equal to the number of records in the data file
- The number of blocks needed for the index = $\lceil (300{,}000/273) \rceil$ = 1,099 blocks
- A binary search on this secondary index needs 11 block accesses

Secondary Indexes (Contd.)

- We can also create a secondary index on a nonkey, nonordering field of a file
  - Numerous records in the data file can have the same value for the indexing field

- There are several schems for implementing such an index

1. include duplicate index entries with the same *K(i)* value
2. have variable-length records for the index entries, with a repeating field for the pointer - one pointer to each block that contains a record whose indexing field value equals *K(i)*
3. create an extra level of indirection to handle the multiple pointers
   - In this scheme, the pointer *P(i)* in index entry <*K(i), P(i)*> points to a disk block, which contains a set of record pointers
   - each record pointer in that disk block points to one of the data file records with value *K(i)* for the indexing field


Data file
(Indexing field)
Dept_number
Name
Ssn
Job
Birth_date
Salary
Blocks of record pointers
Index file
(<K(i), P(i)> entries)
Field value
Block pointer
1
2
3
4
5
6
8

Algorithms for SELECT Operation

- An algorithm to select records from a disk file is known as ***file scan***
- ***Linear search:*** Retrieves every record in the file, and tests whether its attribute values satisfy the selection condition
  - An initial seek is required to access the first block of the file
  - In case blocks of the file are not stored contiguously, extra seeks may be required
- ***Binary search:*** If the selection condition involves an equality comparison on a key attribute on which the file is ordered

Algorithms for SELECT Operation (Contd.)

- ***Using a primary index***: If the selection condition involves an equality comparison on a key attribute with a primary index
- ***Using a hash key:*** If the selection condition involves an equality comparison on a key attribute with a hash key
- ***Using a primary index to retrieve multiple records:*** If the comparison condition is >, >=, <, or <= on a key field with a primary index then retrieve all subsequent records in the (ordered) file.

Thank you!