Indices in DBMS

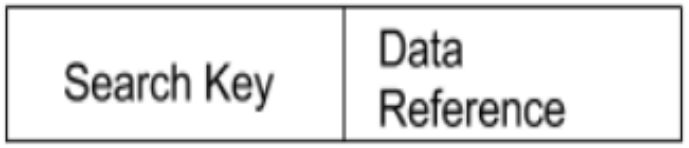
By :
Dr. Rinkle Rani
Associate Professor, CSED
TIET, Patiala

What is Indexing

- Indexing is a way to optimize performance of a database by minimizing the number of disk accesses required when a query is processed.
- An index or database index is a data structure which is used to quickly locate and access the data in a database table.

How indexes are created ?

- Indexes are created using some database columns.
- The first column is the Search key that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly (Note that the data may or may not be stored in sorted order).
- The second column is the Data Reference which contains a set of pointers holding the address of the disk block where that particular key value can be found.

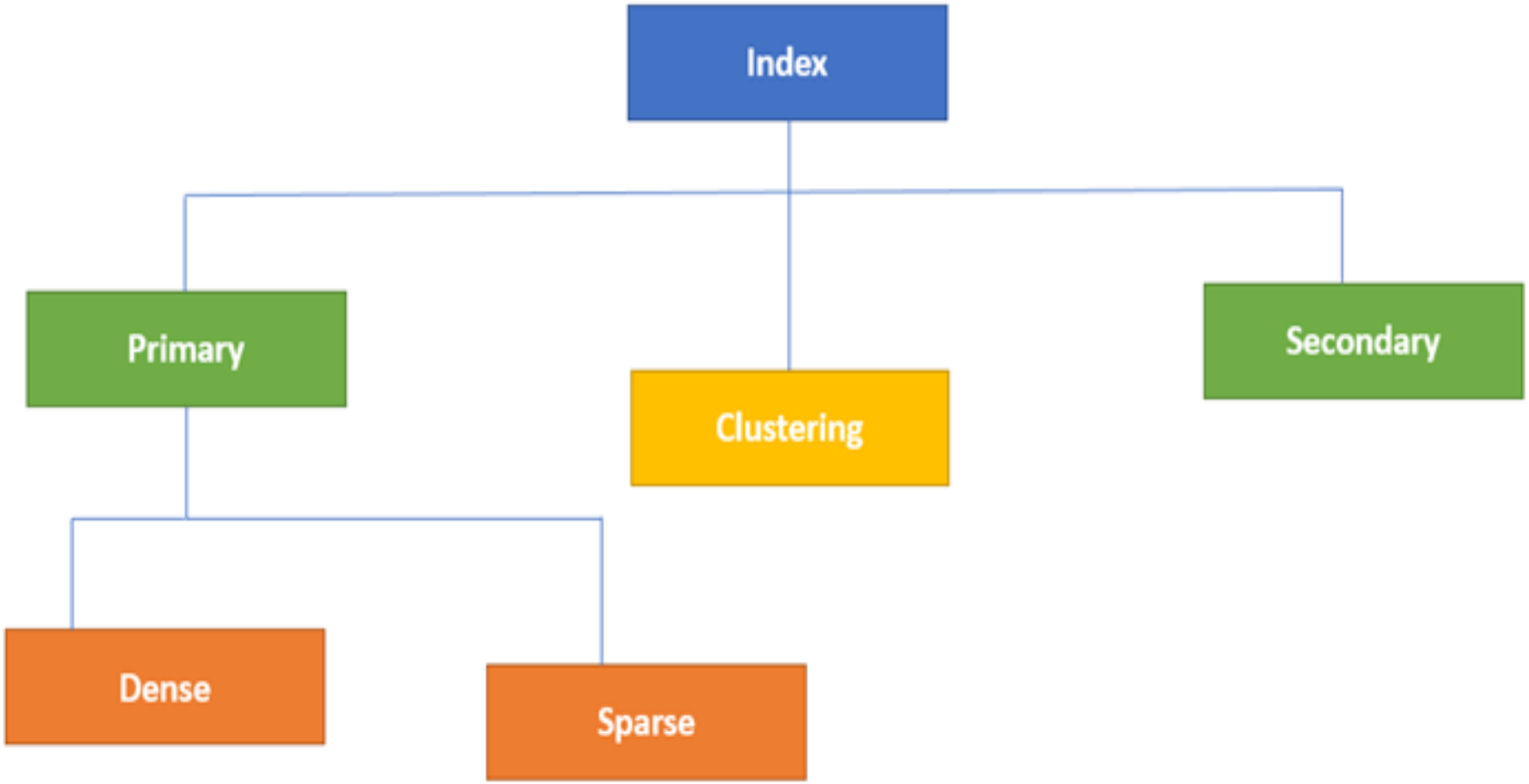


Structure of an index

Types of Indexes

- Types of Single-level Ordered Indexes
  - ï Primary Indexes
  - ï Clustering Indexes
  - ï Secondary Indexes
- Multilevel Indexes

Type of Indexes in Database

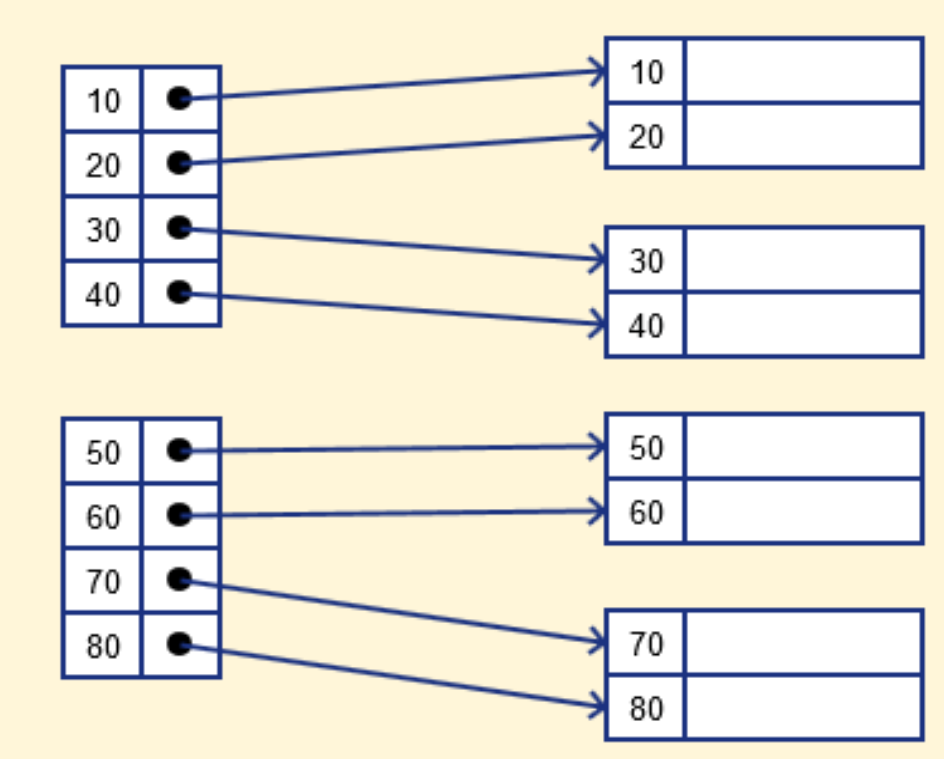


Primary Index

- Primary Index is an ordered file which is fixed length size with two fields. The first field is the same a primary key and second, filed is pointed to that specific data block.
- Since these primary keys are unique to each record and it has 1:1 relation between the records, it is much easier to fetch the record using it.
- The primary Indexing in DBMS is also further divided into two types.
    i. Dense Index
    ii. Sparse Index

Dense Index

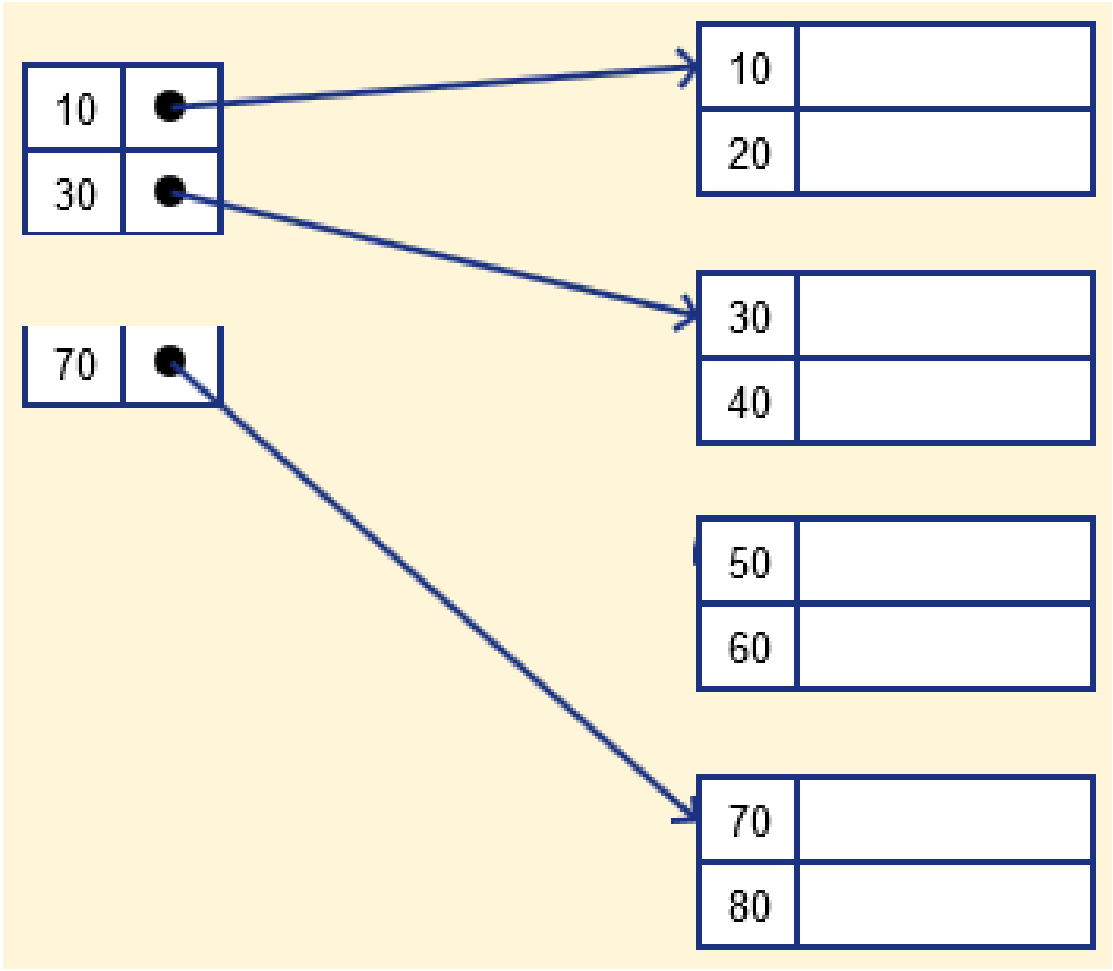
- In a dense index, a record is created for every search key valued in the database.
- This helps you to search faster but needs more space to store index records.
- In this Indexing, method records contain search key value and points to the real record on the disk.



Sparse Index

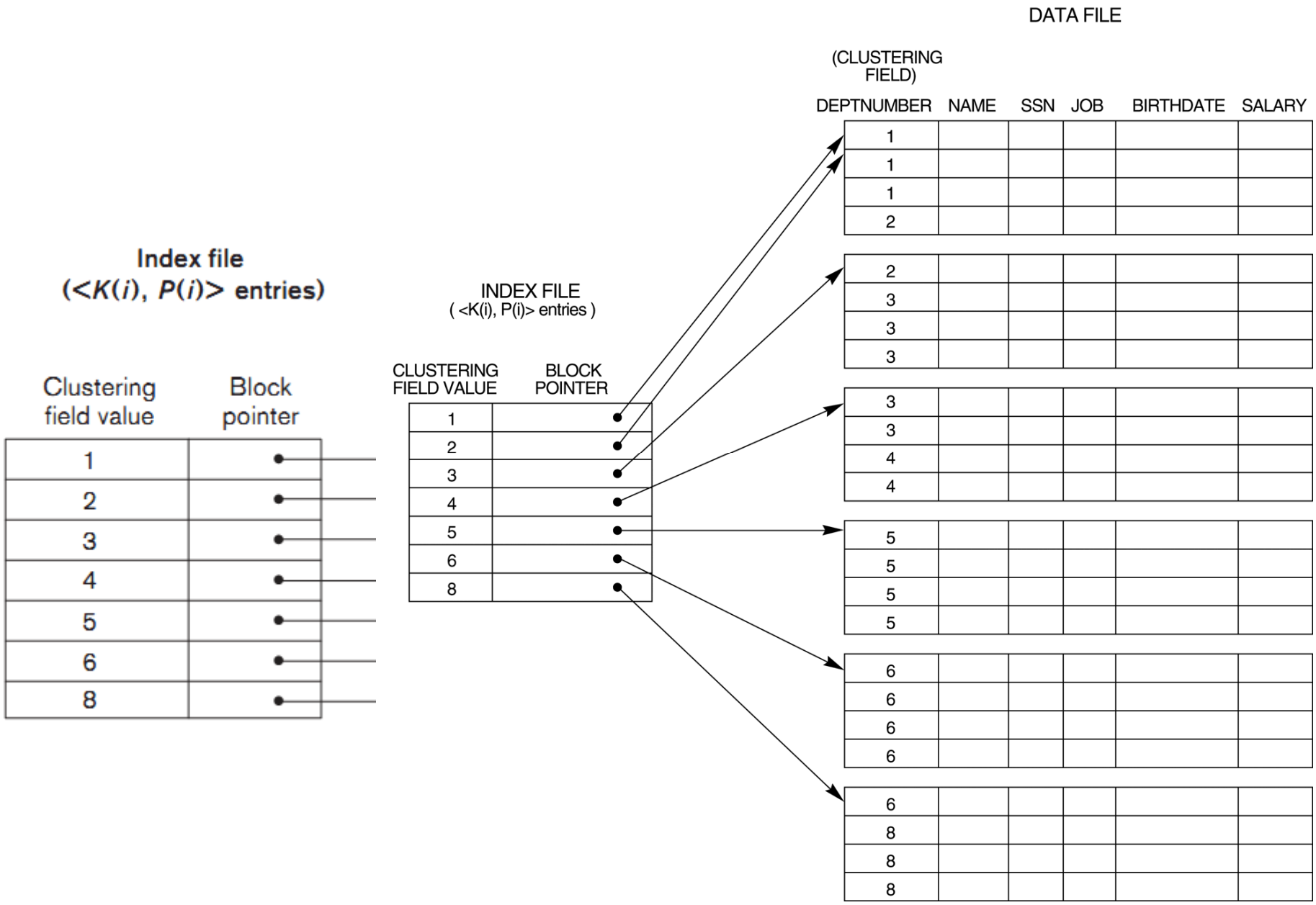
- It is an index record that appears for only some of the values in the file.
- Sparse Index helps you to resolve the issues of dense Indexing in DBMS.
- In this method of indexing technique, a range of index columns stores the same data block address, and when data needs to be retrieved, the block address will be fetched, then the system starts sequential search until the desired data is found.
- However, sparse Index stores index records for only some search-key values. It needs less space, less maintenance overhead for insertion, and deletions but It is slower compared to the dense Index for locating records.

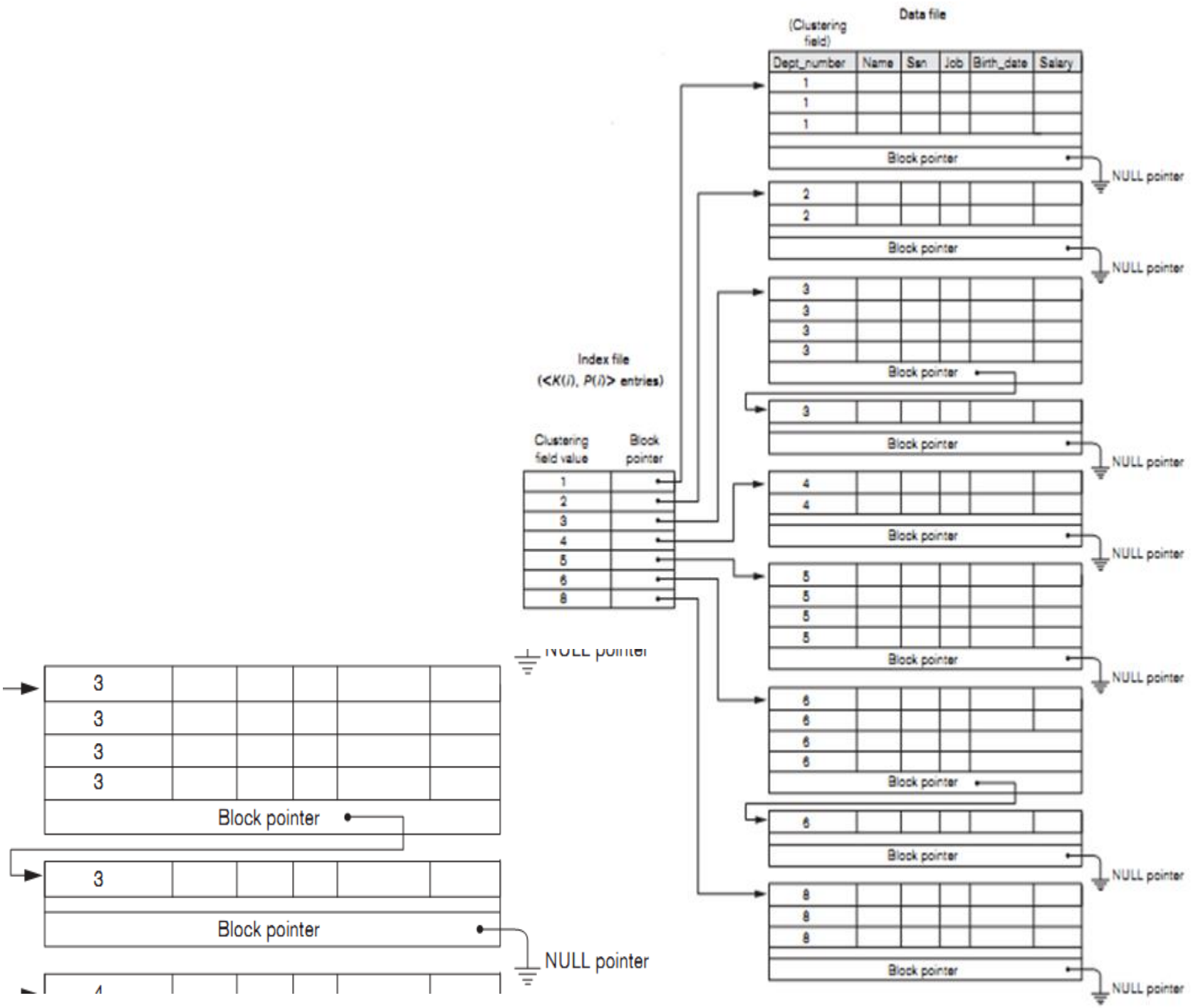
Below is an database index Example of Sparse Index



Clustering Index

- Defined on an **ordered** data file
- The data file is ordered on a *non-key field* unlike primary index, which requires that the ordering field of the data file have a distinct value for each record.
- Includes one index entry *for each distinct value* of the field; the index entry points to the first data block that contains records with that field value.
- It is another example of *nondense* index.

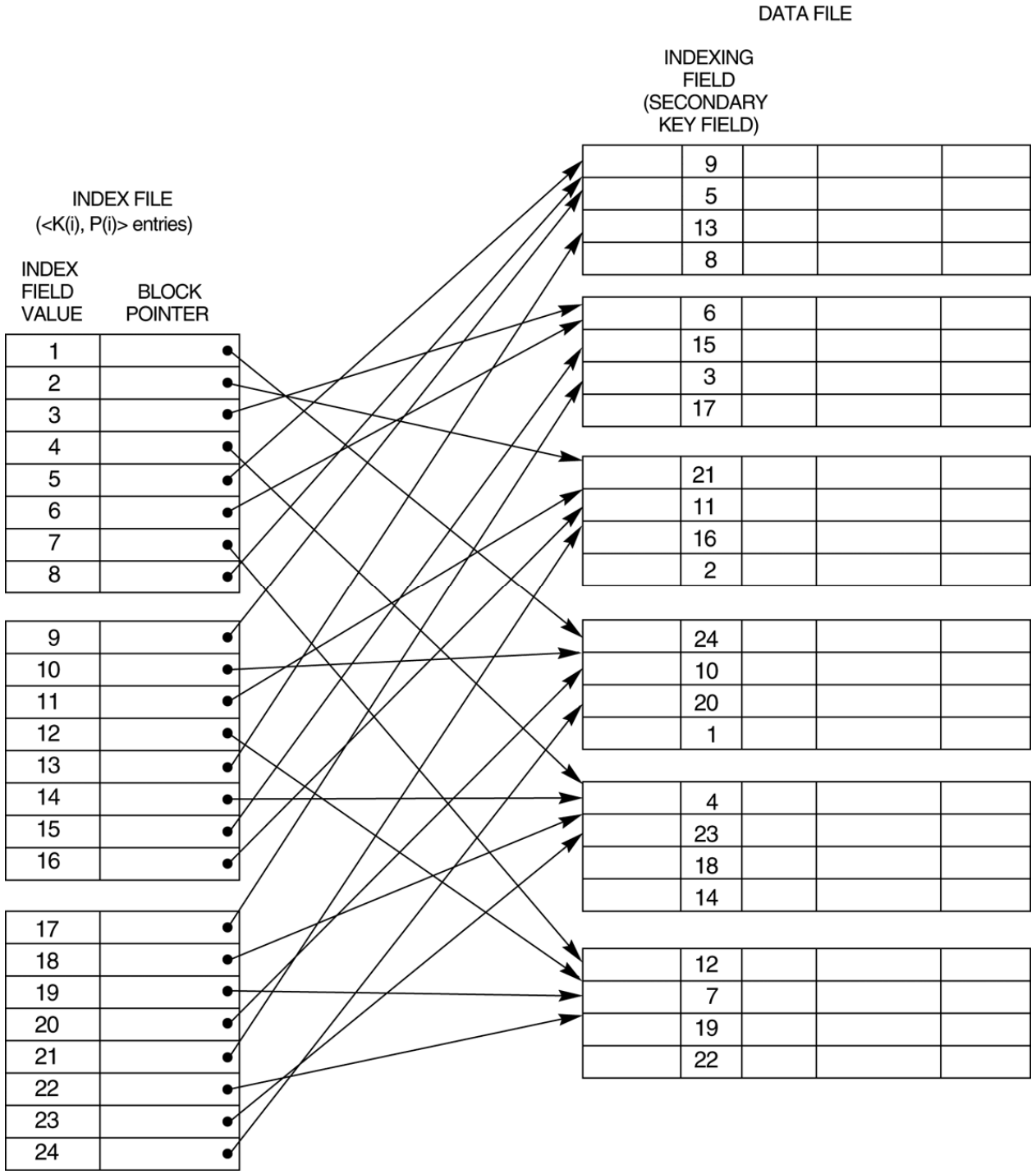

DATA FILE
(CLUSTERING FIELD)
DEPTNUMBER
NAME
SSN
JOB
BIRTHDATE
SALARY
Index file
(<K(i), P(i)> entries)
Clustering field value
Block pointer
INDEX FILE
(<K(i), P(i)> entries)
CLUSTERING FIELD VALUE
BLOCK POINTER


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
3
3
3
3
Block pointer
3
Block pointer
NULL pointer

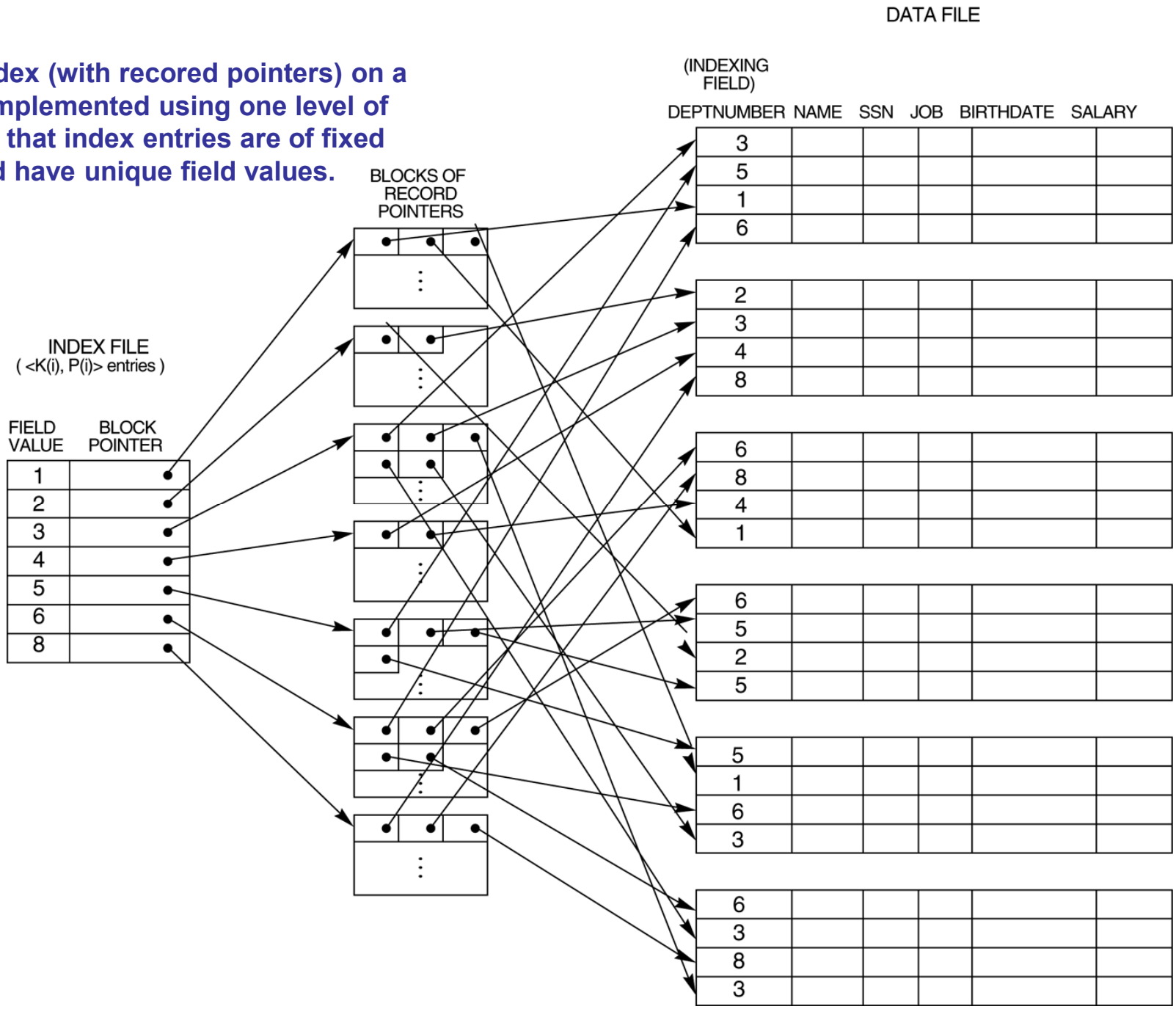
Secondary Index

- A secondary index provides a secondary means of accessing a file for which some primary access already exists.
- The secondary index may be on a field which is a candidate key and has a unique value in every record, or a nonkey with duplicate values.
- The index is an ordered file with two fields.
  - The first field is of the same data type as some *nonordering field* of the data file that is an *indexing field.*
  - The second field is either a *block* pointer or a *record* pointer. There can be *many* secondary indexes (and hence, indexing fields) for the same file.
- Includes one entry *for each record* in the data file; hence, it is a *dense index*

A dense secondary index (with block pointers) on a nonordering key field of a file.



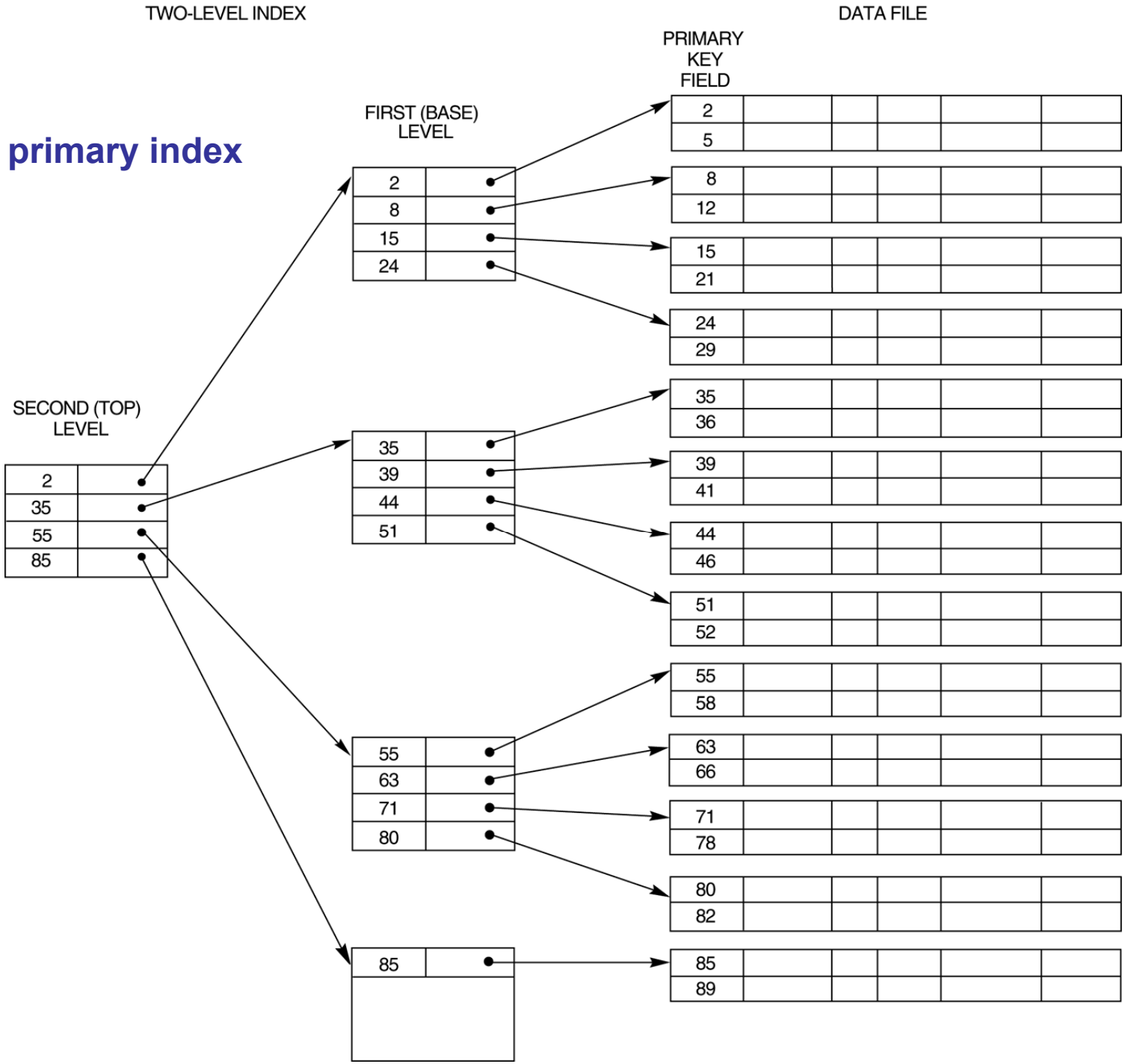
A secondary index (with recored pointers) on a nonkey field implemented using one level of indirection so that index entries are of fixed length and have unique field values.



Multi-Level Indexes

- Because a single-level index is an ordered file, we can create a primary index *to the index itself* ; in this case, the original index file is called the *first-level index* and the index to the index is called the *second-level index.*
- We can repeat the process, creating a third, fourth, ..., top level until all entries of the *top level* fit in one disk block
- A multi-level index can be created for any type of first-level index (primary, secondary, clustering) as long as the first-level index consists of *more than one* disk block

A two-level primary index



What is an Index?

An index:

- Is a schema object
- Is used by the Oracle server to speed up the retrieval of rows by using a pointer
- Can reduce disk I/O by using a rapid path access method to locate data quickly
- Is independent of the table it indexes
- Is used and maintained automatically by the Oracle server

How Are Indexes Created?

- Automatically: A unique index is created automatically when you define a `PRIMARY KEY` or `UNIQUE` constraint in a table definition.
- Manually: Users can create nonunique indexes on columns to speed up access to the rows.

Creating an Index

- Create an index on one or more columns.

```
CREATE INDEX index
ON table (column[, column]...);
```

- Improve the speed of query access to the LAST_NAME column in the EMPLOYEES table.

```
CREATE INDEX  emp_last_name_idx
ON            employees(last_name);
Index created.
```

When to Create an Index

You should create an index if:

- A column contains a wide range of values
- A column contains a large number of null values
- One or more columns are frequently used together in a `WHERE` clause or a join condition
- The table is large and most queries are expected to retrieve less than 2 to 4 percent of the rows

When Not to Create an Index

It is usually not worth creating an index if:

- The table is small
- The columns are not often used as a condition in the query
- Most queries are expected to retrieve more than 2 to 4 percent of the rows in the table
- The table is updated frequently
- The indexed columns are referenced as part of an expression

Function-Based Indexes

- A function-based index is an index based on expressions.
- The index expression is built from table columns, constants, SQL functions, and user-defined functions.

```
CREATE INDEX upper_dept_name_idx
ON departments(UPPER(department_name));

Index created.

SELECT *
FROM   departments
WHERE  UPPER(department_name) = 'SALES';
```

Removing an Index

- Remove an index from the data dictionary by using the DROP INDEX command.

```
DROP INDEX index;
```

- Remove the UPPER_LAST_NAME_IDX index from the data dictionary.

```
DROP INDEX upper_last_name_idx;
Index dropped.
```

- To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.