# Functional Dependency

- The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.
- A functional dependency is denoted by an arrow →.
- The functional dependency of X on Y is represented by X → Y
- X is called determinant set
- Y is called dependent attribute

ie Given the value of X if there is only one value of Y corresponding to it, then Y is said to be functionally depended on X

Eg: Itemcode ->Itemname

- Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

- Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

- Functional dependency can be written as:
- Emp_Id → Emp_Name
- We can say that Emp_Name is functionally dependent on Emp_Id.

# Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B
- **Example:**

Consider a table with two columns Employee_Id and Employee _Name.

{Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as

Employee_Id is a subset of {Employee_Id, Employee_Name}.

Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

# Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.
- **Example:**

ID → Name,

Name → DOB

# Transitive dependency

- A transitive dependency exists when there is an intermediate in functional dependency.

Notation

- A→ B, B→ C, and if A→ C then it can be stated that the transitive dependency exists.
- A→ B→C
- Eg: STAFF NUMBER −→ DESIGNATION
- DESIGNATION −→ SALARY
- STAFF NUMBER −→ SALARY
- There is a transitive dependency between STAFF NUMBER and SALARY

# Partial Functional Dependency

- **Partial Dependency** occurs when a non-prime attribute is **functionally** dependent on part of a candidate key.
- The 2nd Normal Form (2NF) eliminates the **Partial Dependency**.

- **A partial dependency means if the non-key attributes depend on the part of candidate key then it is said to be partial dependency.**

- **Full Functional Dependency :** In a relation , there exists Full Functional Dependency between any two attributes X and Y,  when X is functionally dependent on Y and is not functionally dependent on any proper subset of Y.

# Inference Rule (IR):

- The Armstrong's axioms are the basic inference rule.
- Armstrong's axioms are used to conclude functional dependencies on a relational database.
- The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
- Using the inference rule, we can derive additional functional dependency from the initial set.
- The Functional dependency has 6 types of inference rule:

# 1. Reflexive Rule ($IR_1$)

- In the reflexive rule, if Y is a subset of X, then X determines Y.
- If $X \supseteq Y$ then $X \rightarrow Y$
- **Example:**
- X = {a, b, c, d, e}
- Y = {a, b, c}

# 2. Augmentation Rule ($IR_2$)

- The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.
- If X → Y then XZ → YZ
- **Example:**
- For R(ABCD), **if** A → B then AC → BC

# 3. Transitive Rule ($IR_3$)

- In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.
- If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

# 4. Union Rule ($IR_4$)

- Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.
- If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$
- **Proof:**
- 1. $X \rightarrow Y$ (given)
  2. $X \rightarrow Z$ (given)
  3. $X \rightarrow XY$ (using $IR_2$ on 1 by augmentation with X. Where XX = X)
  4. $XY \rightarrow YZ$ (using $IR_2$ on 2 by augmentation with Y)
  5. $X \rightarrow YZ$ (using $IR_3$ on 3 and 4)

# 5. Decomposition Rule ($IR_5$)

- Decomposition rule is also known as project rule. It is the reverse of union rule.
- This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.
- If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$
- **Proof:**
- 1. $X \rightarrow YZ$ (given)
  2. $YZ \rightarrow Y$ (using $IR_1$ Rule)
  3. $X \rightarrow Y$ (using $IR_3$ on 1 and 2)

# 6. Pseudo transitive Rule ($IR_6$)

- In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.
- If X $\rightarrow$ Y and YZ $\rightarrow$ W then XZ $\rightarrow$ W
- **Proof:**
- 1. X $\rightarrow$ Y (given)
  2. WY $\rightarrow$ Z (given)
  3. WX $\rightarrow$ WY (using $IR_2$ on 1 by augmenting with W)
  4. WX $\rightarrow$ Z (using $IR_3$ on 3 and 2)

# Anomalies in a database

STUDENT-TB

| Name | Course | Pnone-no | Major | Prof | Grade | |
|---|---|---|---|---|---|---|
| John | 353 | 235-4539 | Computer-science | Smith | A | |
| Ng | 329 | 457-6534 | Chemistry | Taniya | B | |
| John | 328 | 235-4539 | Computer-science | Clark | B | |
| Martin | 456 | 456-876 | Physics | James | A | |
| Duke | 293 | 371-765 | Maths | Cook | C | |
| Duke | 356 | 371-765 | Maths | Bond | In prog | |
| John | 379 | 235-4539 | Computer-science | Cross | In prog | |

# Update anomalies

- Multiple copies of the same fact may lead update anomalies or inconsistencies when an update is made and only some of the multiple copies are updated.
- Thus a change in the phone number of John must be made for inconsistency in all tuples pertaining to the student Jone.
- If one of the three tuples in the given table changed to reflect the new phone number of John there will be an inconsistency in the data

# Insertion anomalies

- If this is the only relation in the database showing the association between a faculty member and the course he or she teaches, the fact that a given processor is teaching a given course cannot be entered in the database unless a student is registered in the course.
- Also if another relation also establish relationship between a course and a professor who teaches that course the information stored in this relation has to be consistent.

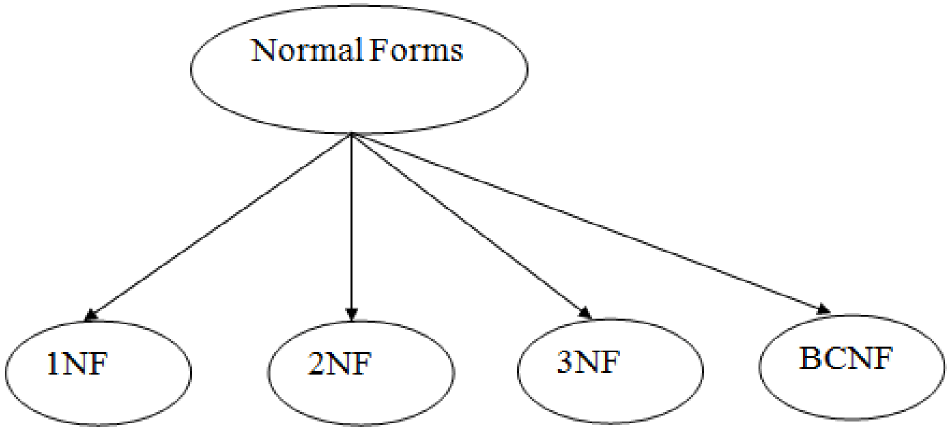
# Deletion anomalies

- If the only student registered in a given course discontinuous the course, the information as to which professor is offering the course will be lost, this is the only relation in the database showing the association between a faculty member and their course she or he teaches.
- If another relation in the database also established the relationship between a course and a professor who teaches that course the deletion of the last tuple in STUDENT-TB for a given course will not cause information about the course's teachers to be lost

# Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations.
- It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

# Types of Normal Forms



# First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

**EMPLOYEE table:**

| EMP_ID | EMP_NAME | EMP_PHONE | EMP_STATE |
|---|---|---|---|
| 14 | John | 7272826385, 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389, 8589830302 | Punjab |

| EMP_ID | EMP_NAME | EMP_PHONE | EMP_STATE |
| --- | --- | --- | --- |
| 14 | John | 7272826385 | UP |
| 14 | John | 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389 | Punjab |
| 12 | Sam | 8589830302 | Punjab |

# Second Normal Form (2NF)

- A table is said to be in 2NF if it is in 1NF

and non-key attributes are functionally dependents on the key attribute.

- Further, If the key has more than one attribute then **no non key** attribute should be functionally depends upon a part of the key attribute

# Second Normal Form

EMPLOYEE-TB

| Emp-no | Ename | Project-id | Projectname | Project-location | Hrs-worked |
|---|---|---|---|---|---|
| E01 | Jaine Bhatta | P01 | HMS | Kolkutta | 20 |
| E02 | Pinki Bose | P02 | SIS | Mumbi | 30 |
| E03 | Kushi Kosh | P01 | HMS | Kolkutta | 40 |
| E04 | Guthm Deny | P02 | SIS | Mumbi | 30 |
| E05 | Sourav Basker | P03 | RRS | Kolcutta | 25 |
| | | | | | |
| | | | | | |

EMP-TB

| Emp-no | Project-id | Hrs-worked |
|---|---|---|
| E01 | P01 | 20 |
| E02 | P02 | 30 |
| E03 | P01 | 40 |
| E04 | P02 | 30 |
| E05 | P03 | 25 |

EMP-DETAILS-TB

| Emp-no | Ename |
|---|---|
| E01 | Jaine Bhatta |
| E02 | Pinki Bose |
| E03 | Kushi Kosh |
| E04 | Guthm Deny |
| E05 | Sourav Basker |

PROJECT-TB

| Project-id | Projectname | Project-location |
|---|---|---|
| P01 | HMS | Kolkutta |
| P02 | SIS | Mumbi |
| P01 | HMS | Kolkutta |
| P02 | SIS | Mumbi |

- FD1:{Emp-no, Project-id} -> Hrs-worked----- Primary key
- FD2: Emp-no -> Ename  partial  dependency
- FD3: Project-id -> Projectname, Project-location

partial dependency.

# Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and no non-key attributes is functionally dependent on any other non key attribute.
- 

| EMP_ID | EMP_NAME | EMP_ZIP | EMP_STATE | EMP_CITY |
|---|---|---|---|---|
| 222 | Harry | 201010 | UP | Agra |
| 333 | Stephan | 02228 | US | Boston |
| 444 | Lan | 60007 | US | Chicago |
| 555 | Katharine | 06389 | UK | Norwich |
| 666 | John | 462007 | MP | Bhopal |

**EMPLOYEE table:**

| EMP_ID | EMP_NAME | EMP_ZIP |
| --- | --- | --- |
| 222 | Harry | 201010 |
| 333 | Stephan | 02228 |
| 444 | Lan | 60007 |
| 555 | Katharine | 06389 |
| 666 | John | 462007 |

**EMPLOYEE_ZIP table:**

| EMP_ZIP | EMP_STATE | EMP_CITY |
| --- | --- | --- |
| 201010 | UP | Noida |
| 02228 | US | Boston |
| 60007 | US | Chicago |
| 06389 | UK | Norwich |
| 462007 | MP | Bhopal |

# Third normal form

| EMP-NO | NAME | DEG | SALARY | DEPT-NO | DEPT-NAME |
|---|---|---|---|---|---|
| E01 | KEVIN | MANAGER | 25000 | D01 | ADMINISTRATION |
| E05 | PINKY | PROGRAMMER | 15000 | D03 | PROJECT |
| E06 | GOUTHM | FINANCE-OFFICER | 10000 | D02 | PERSONAL |
| E07 | RAJIB | SYSTEM ANALYSIST | 20000 | D03 | PROJECT |

| EMP-NO | NAME | DEG | SALARY | DEPT-NO | |
|---|---|---|---|---|---|
| E01 | kEVIN | MANAGER | 25000 | D01 | |
| E05 | PINKY | PROGRAMMER | 15000 | D03 | |
| E06 | GOUTHM | FINANCE-OFFICER | 10000 | D02 | |
| E07 | RAJIB | SYSTEM ANALYSIST | 20000 | D03 | |

| DEPT-NO | DEPT-NAME |
|---|---|
| D01 | ADMINISTRATION |
| D03 | PROJECT |
| D02 | PERSONAL |
| | |

# Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency $X \rightarrow Y$, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

**Example:** Let's assume there is a company where employees work in more than one department.

**EMPLOYEE table:**

| EMP_ID | EMP_COUNTRY | EMP_DEPT | DEPT_TYPE | EMP_DEPT_NO |
|---|---|---|---|---|
| 264 | India | Designing | D394 | 283 |
| 264 | India | Testing | D394 | 300 |
| 364 | UK | Stores | D283 | 232 |
| 364 | UK | Developing | D283 | 549 |

EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
**Candidate key: {EMP-ID, EMP-DEPT}**
The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

**EMP_COUNTRY table:**

| EMP_ID | EMP_COUNTRY |
|---|---|
| 264 | India |
| 264 | India |

**EMP_DEPT table:**

| EMP_DEPT | DEPT_TYPE | EMP_DEPT_NO |
|---|---|---|
| Designing | D394 | 283 |
| Testing | D394 | 300 |
| Stores | D283 | 232 |
| Developing | D283 | 549 |

**EMP_DEPT_MAPPING table:**

| EMP_ID | EMP_DEPT |
|---|---|
| D394 | 283 |
| D394 | 300 |
| D283 | 232 |
| D283 | 549 |

**Functional dependencies:**
EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
**Candidate keys:**
**For the first table:** EMP_ID
**For the second table:** EMP_DEPT
**For the third table:** {EMP_ID, EMP_DEPT}
Now, this is in BCNF because left side part of both the functional dependencies is a key.

## Eg: Converting a Relation to BCNF

- Consider a relation which is in 3NF but not in BCNF.
- The relation R has three attributes:
- R{PATIENT, DOCTOR, HOSPITAL}
  - {PATIENT, HOSPITAL} →DOCTOR
  - DOCTOR → HOSPITAL
- R is not in BCNF because DOCTOR is not the super key.
- To convert the relation R into BCNF, split the relation R into two relations R1 and R2.
  - R1{PATIENT, DOCTOR}
  - R2{DOCTOR, HOSPITAL}

# Indexing Structures for Files

# Indexes as Access Paths

- A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- The index is usually specified on one field of the file (although it could be specified on several fields)
- One form of an index is a file of entries <**field value, pointer to record>**, which is ordered by field value
- The index is called an access path on the field.

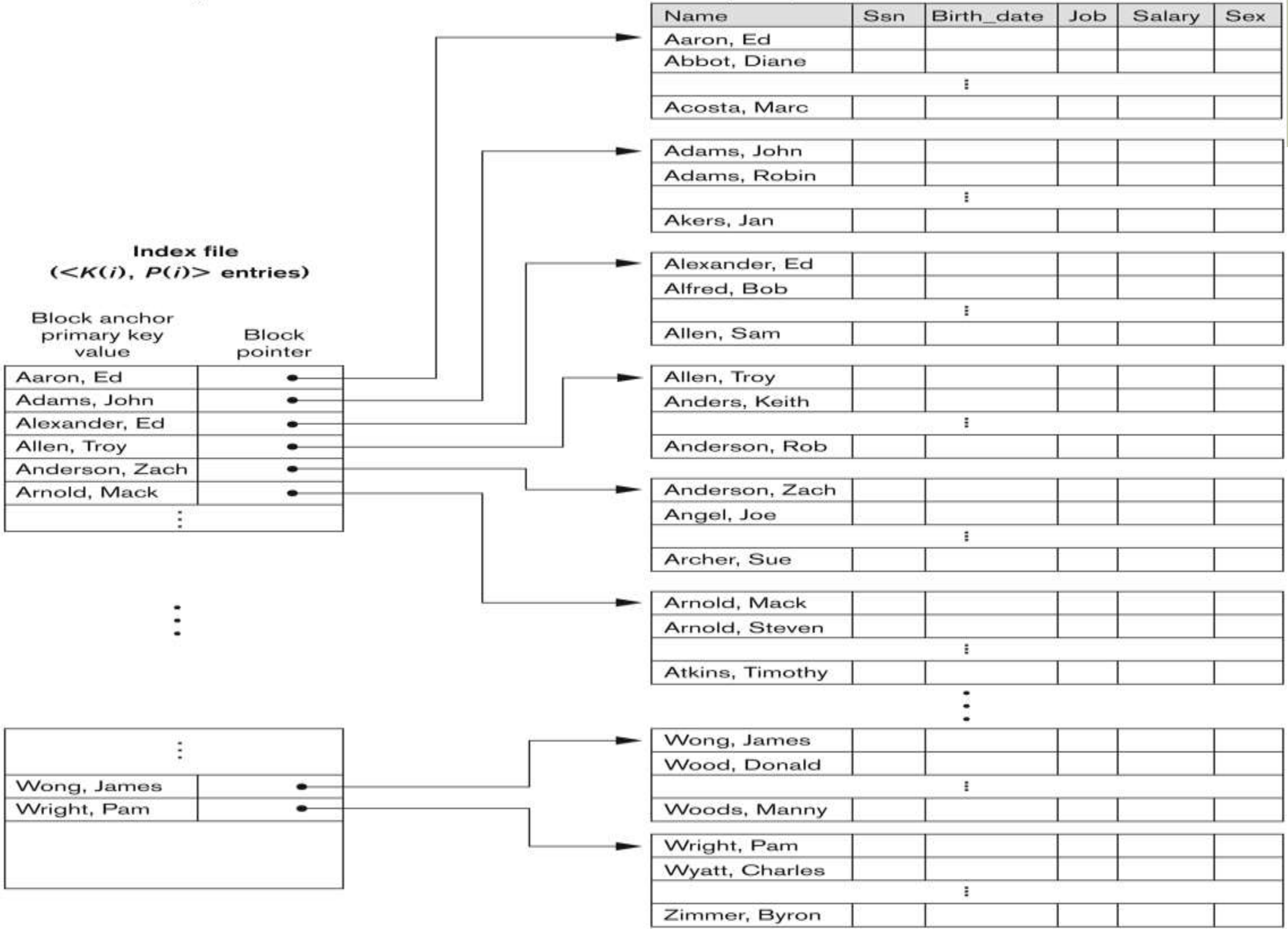
# Indexes as Access Paths (contd.)

- The index file usually occupies considerably less disk blocks than the data file because its entries are much smaller
- A binary search on the index yields a pointer to the file record
- Indexes can also be characterized as dense or sparse
  - A **dense index** has an index entry for every search key value (and hence every record) in the data file.
  - A **sparse (or nondense) index**, on the other hand, has index entries for only some of the search values

# Types of Single-Level Indexes

- Primary Index
  - Defined on an ordered data file whose records are of fixed length with 2 fields.
  - The data file is ordered on a **key field**
  - The first field is of the same data type as the ordering key field called primary key of the data file.
  - The second field is a pointer to a disk block(block address).
  - Includes one index entry *for each block* in the data file; the index entry has the key field value for the *first record* in the block, which is called the *block anchor and a pointer to that block.*
  - *<K(i),P(i)>*
  - A primary index is a nondense (sparse) index, since it includes an entry for each disk block of the data file and the keys of its anchor record rather than for every search value.

**Figure 14.1**
Primary index on the ordering key field of the file shown in Figure 13.7.



# Types of Single-Level Indexes

- Clustering Index
  - Defined on an ordered data file
  - The data file is ordered on a *non-key field* unlike primary index, which does not have a distinct value for each record – that field is called ***clustering field***.
  - We will create a different type of index called ***clustering index,*** to speed up retrieval of records that have the same value for the clustering field.
  - 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.
  - For insertion and deletion there are some problems.
  - It is another example of *nondense* index where Insertion and Deletion is relatively straightforward with a clustering index.

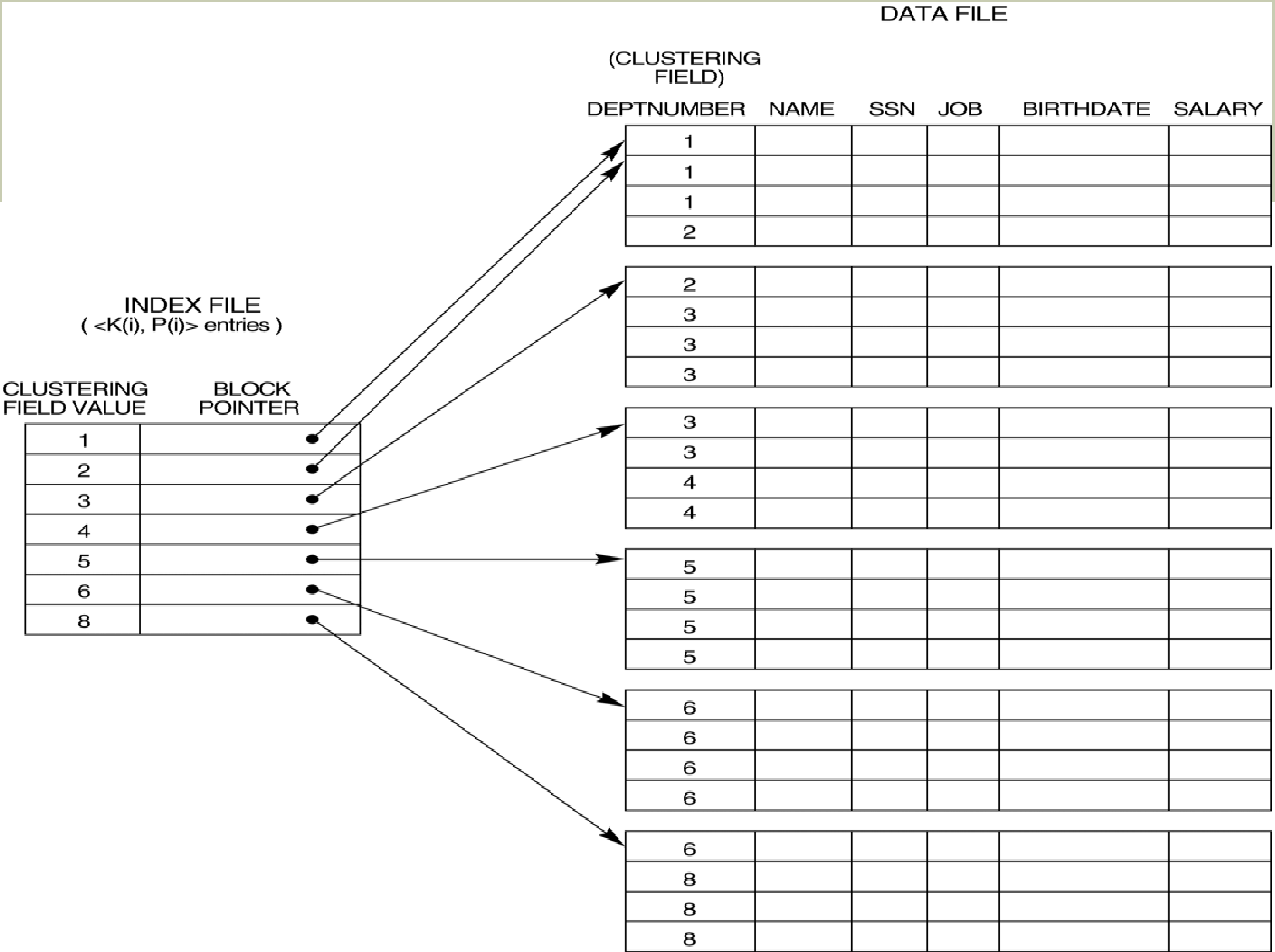
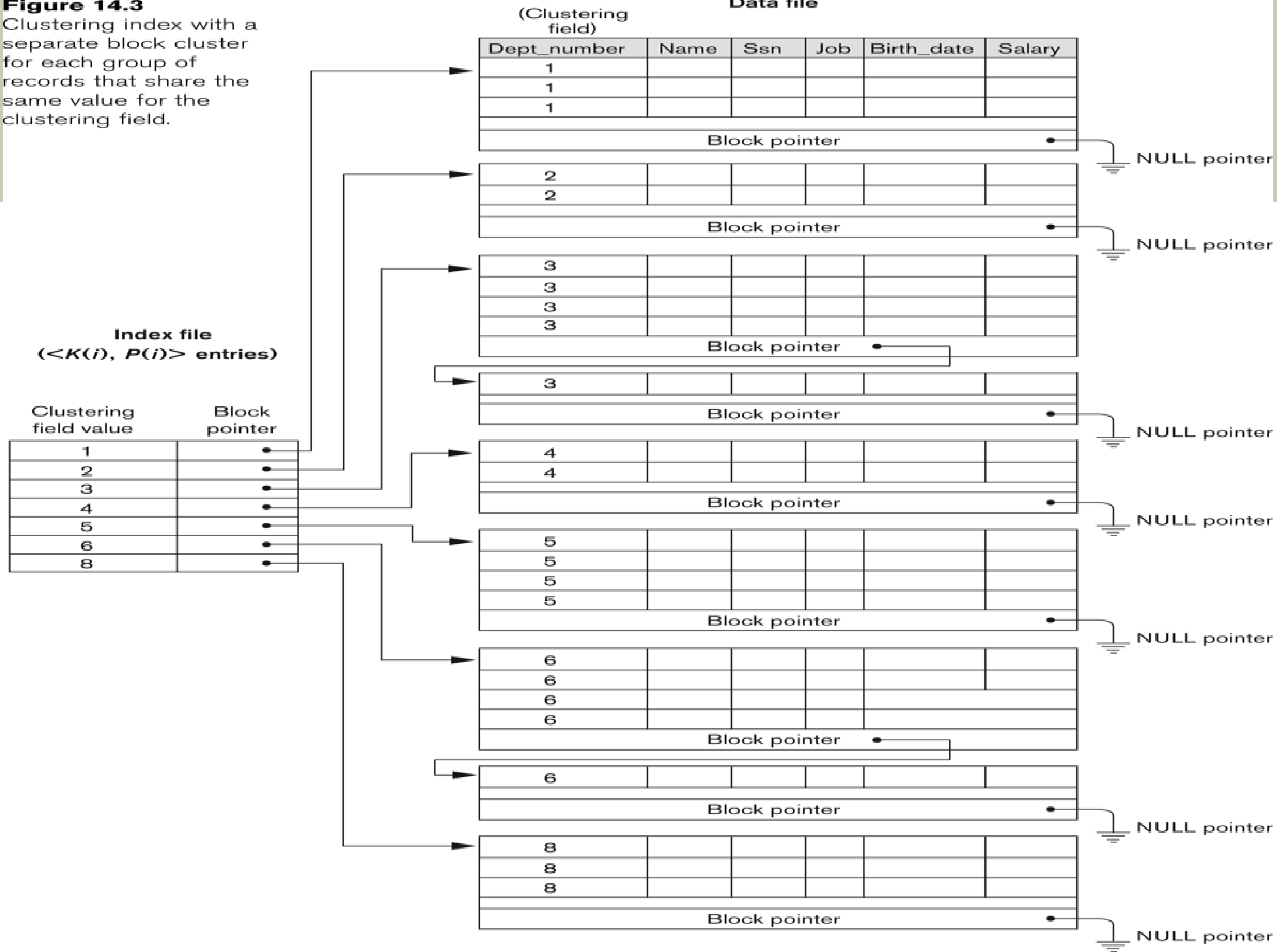

DATA FILE
(CLUSTERING FIELD)
DEPTNUMBER
NAME
SSN
JOB
BIRTHDATE
SALARY
1
1
1
2
2
3
3
3
3
3
4
4
5
5
5
5
6
6
6
6
6
8
8
8
INDEX FILE
( <K(i), P(i)> entries )
CLUSTERING FIELD VALUE
BLOCK POINTER
1
2
3
4
5
6
8

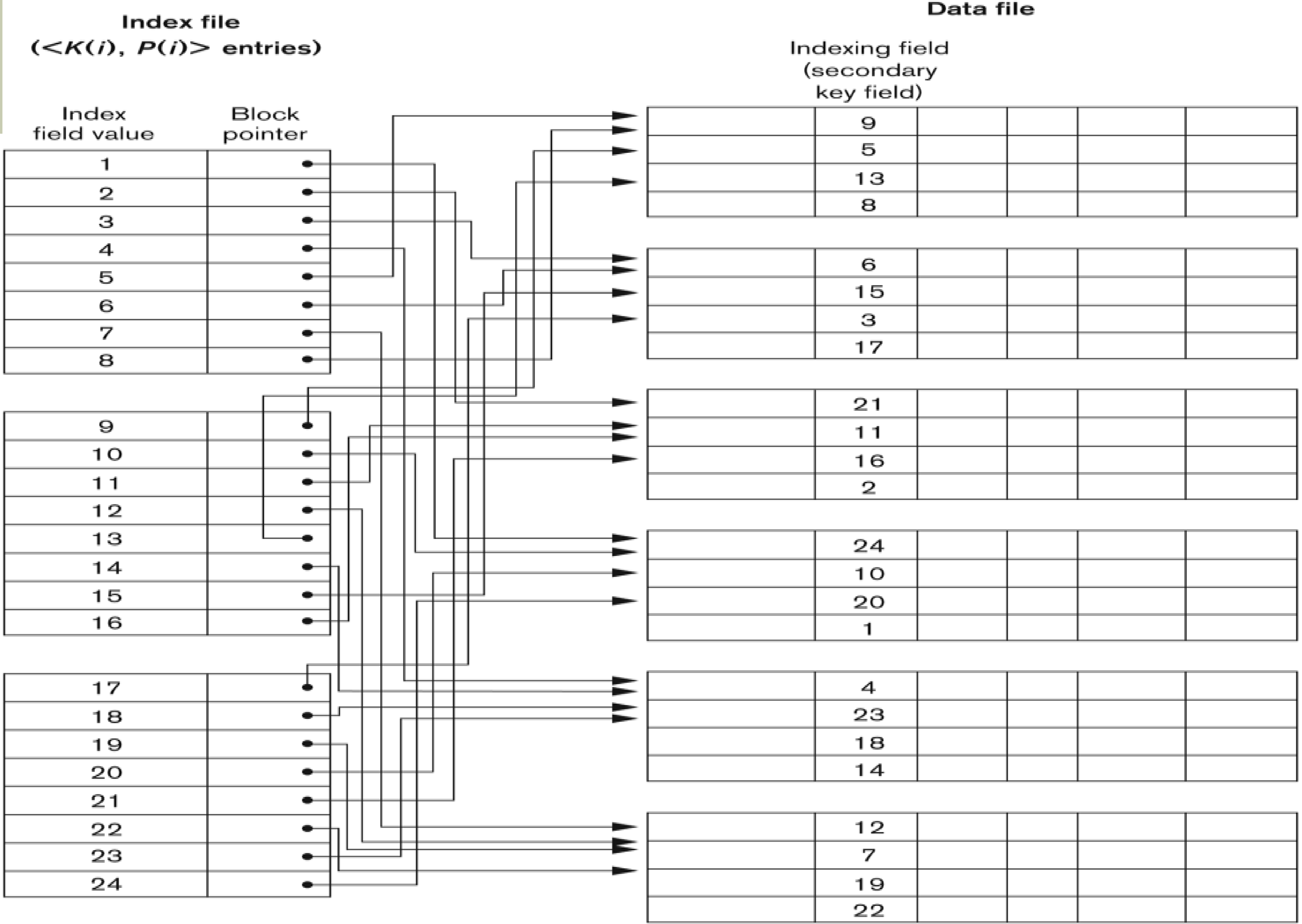
**Figure 14.3**
Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.



# Types of Single-Level Indexes

- 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 non-key with duplicate values.
  - The index is an ordered file with two fields.
    - The first field is of the same data type as some **non-ordering 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.*
  - Needs more storage space and longer search time than does a primary index.

Figure 14.4
A dense secondary index (with block pointers) on a nonordering key field of a file.



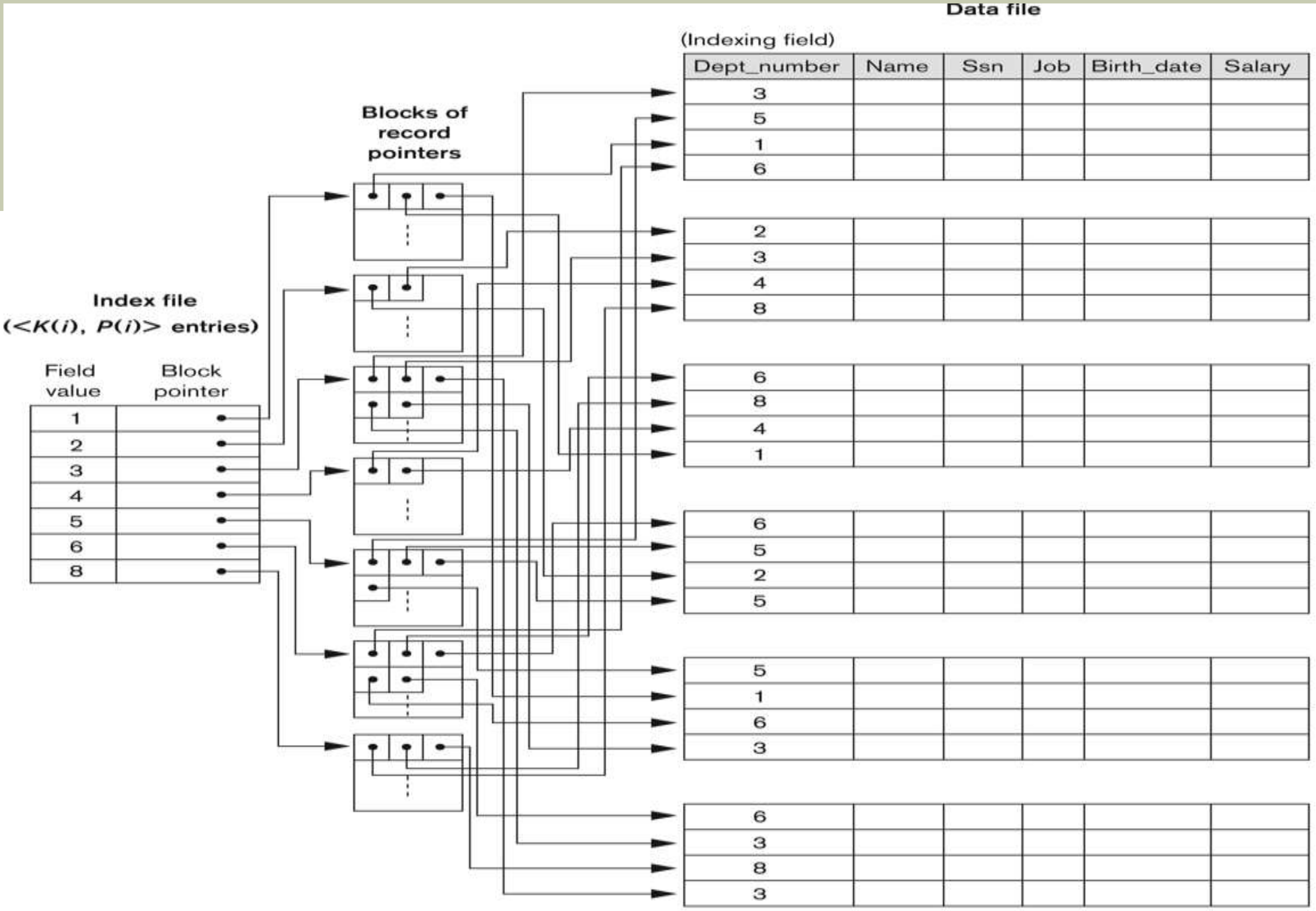



**Figure 14.5**
A secondary index (with record pointers) on a nonkey field implemented using one level of indirection so that index entries are of fixed length and have unique field values.

# Properties of Index Types

TABLE 14.2 PROPERTIES OF INDEX TYPES

| TYPE OF INDEX | NUMBER OF (FIRST-LEVEL) INDEX ENTRIES | DENSE OR NONDENSE | 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.