T.E. Computer

Database Management Systems

Unit-IV

By

Prof. Dr. K.P. Adhiya

SSBT's COET, Bambhori-Jalgaon

Acknowledgment

1. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, “Database System Concepts”, 6th Edition, McGraw-Hill Hill Education.
2. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, “Database System Concepts”, 4th Edition, McGraw-Hill Hill Education.
3. Ramez Elmasri and Shamkant B. Navathe “Fundamentals of Database Systems”, 5th Edition, Pearson.
4. Express Learning, “Database Management Systems”, ITL Education Solutions Limited.
5. Archana Verma, “Database Management Systems”, GenNext Publication.
6. Dr. Rajiv Chopra, “Database Management Systems (DBMS) – A Practical Approach”, 5th Edition, S. Chand Technical
7. Tanmay Kasbe, “Database Management System Concepts – A Practical Approach”, First Edition, Educreation Publishing.
8. Mahesh Mali, “Database Management Systems”, Edition 2019, TechKnowledge Publications.
9. Rajendra Prasad Mahapatra, Govind Verma, “Database Management System”, Khanna Publishing.
10. Malay K. Pakhira, “Database Management System”, Eastern Economy Edition, PHI.
11. Sarika Gupta, Gaurav Gupta, “Database Management System”, Khanna Book Publishing Edition.
12. Riktesh Srivastava, Rajita Srivastava, “Relational Database Management System”, New Age International Publishers.
13. Peter Rob, Carlos Coronel, “Database System Concepts’, Cenage Learning, India Edition
14. Bipin C. Desai, “An Introduction to Database Systems”, Galgotia Publications.
15. G.K. Gupta, “Database Management Systems”, McGraw Hill Education.
16. Shio Kumar Singh, “Database Systems – Concepts, Design and Applications”, 2nd Edition, PEARSON.
17. S.D.Joshi, “Database Management System”, Tech-Max Publication.
18. R. Ramkrishnan , J. Gehrke, "Database Management Systems", 3rd Edition, McGraw-Hill
19. C. J. Date, “Introduction to Database Management Systems”, 8th Edition, Pearson
20. Atul Kahate, “Introduction to Database Management System”, 3rd Edition, Pearson.
21. Bharat Lohiya, “Database Systems”, Tenth Edition, Aditya Publication, Amravati.
22. Vijay Krishna Pallaw, “Database Management System”, 2nd, Asian Books Pvt. Ltd.
23. Database Management Systems, Database Management Systems.
24. Mrs. Jyoti G. Mante (Khurpade), Mrs. Smita M. Dandge, “Database Mangement System”, Nirali Prakashan.
25. Step by Step Database Systems (DBMS), Shiv Krupa Publications, Akola

Prof. Dr. K.P. Adhiya

26. Mrs. Sheetal Gujar –Takale, Mr. Sahil K. Shah, "Database Management System", Nirali Prakashan.
27. Mrs. Jyoti G. Mante (Khurpade), U.S. Shirshetti, M.V. Salvi, K.S. Sakure, "Relational Database Management System", Nirali Prakashan.
28. Seema Kedar, Rakesh Shirsath, "Database Management Systems", Technical Publications.
29. Pankaj B. Brahmankar, "Database Management Systems", Tech-Max Publications, Pune.
30. Imran Saeed, Tasleem Mustafa, Tariq Mahmood, Ahsan Raza Sattar, "A Fundamental Study of Database Management Systems", 3rd Edition, IT Series Publication.
31. Database Management Systems Lecture Notes, Malla Reddy College of Engineering and Technology, Secunderabad.
32. Dr. Satinder Bal Gupta, Aditya Mittal, "Introduction to Database Management System, University Science Press.
33. E-Notes BCS 41/ BCA 41 on "Database Management System", Thiruvalluvar University.
34. Bighnaraj Naik, Digital Notes on "Relational Database Management System", VSSUT, Burla.
35. Viren Sir, Relational database Management System", Adarsh Institute of Technolgoyt (Poly), VITA.
36. Sitansu S. Mitra, "Principles of Relational Database Systems", Prentice Hall.
37. Neeraj Sharma, Liviu Perniu, Raul F. Chong, Abhishek Iyer, Chaitali Nandan, Adi-Cristina Mitea, Mallarswami Nonvinkere, Mirela Danubianu, "Database Fundamentals", First Edition, DB2 On Campus Book Series.
38. Database Management System, Vidyavahini First Grade College, Tumkur.
39. Bhavna Sangamnerkar, Revised by: Shiv Kishor Sharma, "Database Management System", Think Tanks Biyani Group of Colleges.
40. Tibor Radvanyi, "Database Management Systems".
41. Ramon A. Mata-Toledo, Pauline K. Cushman, "Fundamentals of Relational Databases", Schaum's Outlies.
42. P.S. Gill, "Database Management Systems", 2nd Edition, Dreamtech Press, WILEY

Acknowledgment

Web Resources:-

www.tutorailspoint.com
https://www.geeksforgeeks.org/second-normal-form-2nf/
https://beginnersbook.com/2015/05/normalization-in-dbms/
https://www.javatpoint.com/
https://www.studytonight.com/dbms/fourth-normal-form.php
https://sis.binus.ac.id/2018/01/17/drawbacks-of-normalization/
https://whatisdbms.com/normalization-in-dbms-anomalies-advantages-disadvantages/

Unit-IV

(Storage, File Structure, Indexing)

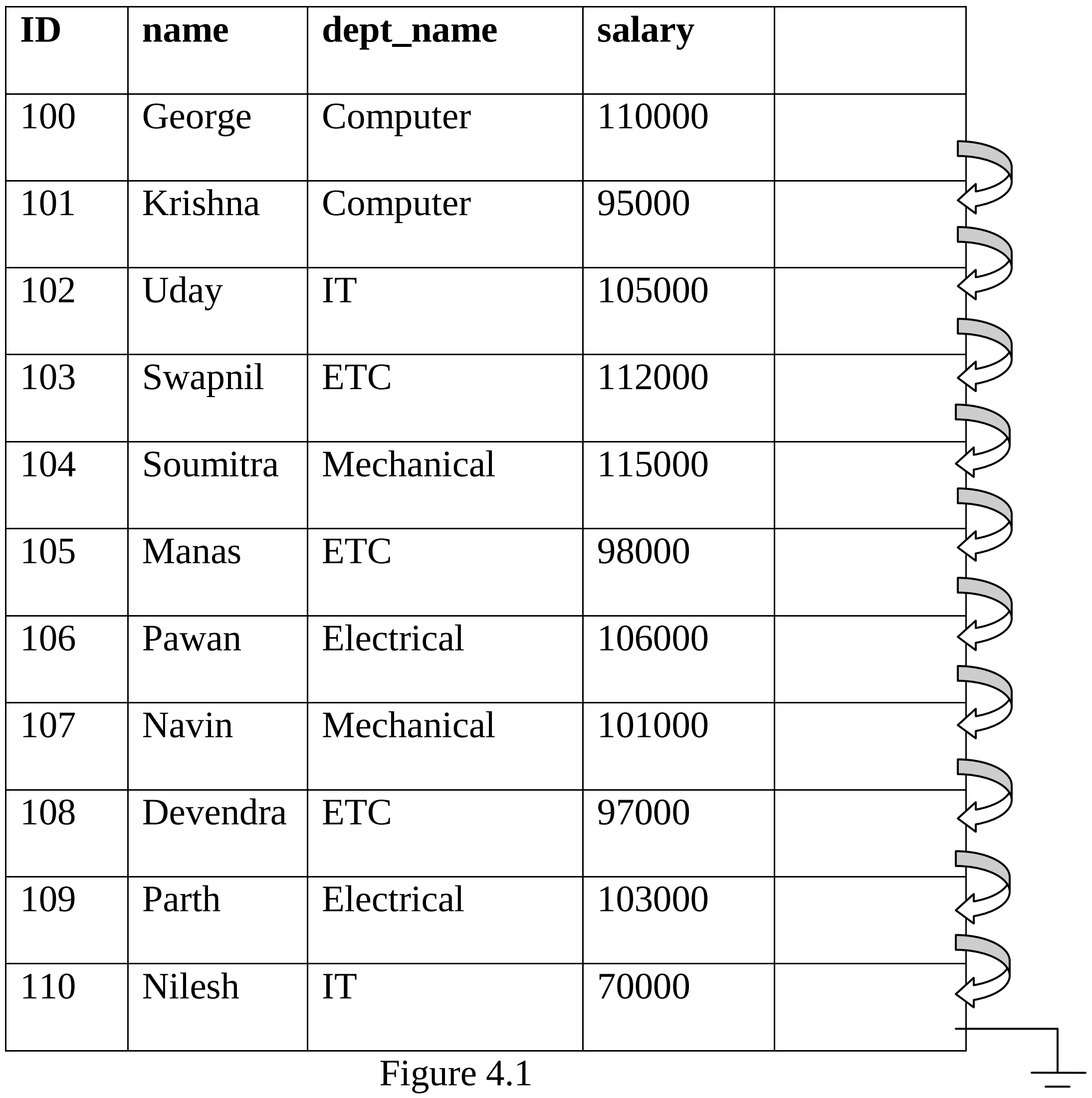
➢ File Organization

- A database is mapped into a number of different files that are maintained by the underlying operating system. These files reside permanently on disks.
- A **file** is organized logically as a sequence of records.
- The arrangement of the records in a file plays a significant role in accessing them. Moreover, proper organization of files on disk helps in accessing the file records efficiently.
- There are various methods (known as **File Organization**) of organizing the records (a relation is a set of records) in a file while storing a file on disk.
- Some of the popular methods of organizing records in files are:-
  - **Heap file organization**: - Any record can be placed anywhere in the file where there is space for the record. There is no ordering of records. Typically, there is a single file for each relation.
  - **Sequential file organization**: - A file organization in which records are organized in sorted order based on some search key (ordering field).
  - **Hash file organization**: - A hash function is computed on some attribute of each record. The result of the hash function specifies in which block of the file the record should be placed.
- Generally, a separate file is used to store the records of each relation. However, in a multitable clustering file organization, records of several different relations are stored in the same file.

✓ <u>**Sequential file organization:**</u> -

  - A **sequential file** is designed for efficient processing of records in sorted order based on some search key. A **search key** is any attribute or set of attributes; it need not be the primary key, or even a superkey. To permit fast retrieval of records in search-key order, we chain together records by pointers.

- The records in the file are ordered by a search-key.
- The following figure shows a sequential file of instructor records. Here, the records are stored in search-key order, using ID as search key.



ID	**name**	**dept_name**	**salary**	
100	George	Computer	110000	
101	Krishna	Computer	95000	
102	Uday	IT	105000	
103	Swapnil	ETC	112000	
104	Soumitra	Mechanical	115000	
105	Manas	ETC	98000	
106	Pawan	Electrical	106000	
107	Navin	Mechanical	101000	
108	Devendra	ETC	97000	
109	Parth	Electrical	103000	
110	Nilesh	IT	70000	

Figure 4.1

- Suitable for applications that require sequential processing of the entire file.

- Sequential method is based on tape model. Devices who support sequential access are magnetic tapes, cassettes, card readers etc.
- Insertion –locate the position where the record is to be inserted
    - if there is free space insert there
    - if no free space, insert the record in an overflow block
    - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order.

➢ Data-Dictionary Storage

- A relational database system needs to maintain data about the relations, such as the schema of the relations. In general, such "data about data" is referred to as **metadata**.
- Relational schemas and other metadata about relations are stored in a structure called the **data dictionary** or **system catalog**
- Generally, Data Dictionary consists of the following information:-
    - Names of the relations (tables).
    - Names of the attributes of each relation
    - Domains and length of attributes.
    - Integrity constraints. E.g. key constraints
      Example: - consider instructor relation-

instructor (ID, name, dept_name, salary)

Field Name	**Data Type**	**Field Length**	**Constraint**	**Description**
ID	Int	--	Primary key	instructor id
name	varchar	25	Not null	Name of the instructor
dept_name	varchar	15	Not null	Name of the department
salary	numeric	(10,3)	No constraint	Salary of the instructor

- Many systems keep the following data also:-
  - Names of the authorized users
  - Authorization and accounting information about users
  - Passwords or other information used to authenticate users
- Number of tuples(records) in each relation
- Methods of storage for each relation (e.g. clustering)
- It may also store the file organization information (e.g. sequential, hash or heap)
  - The data dictionary would store the names of the files containing each relation.

- All this metadata information constitutes, in effect, miniature database.
- It is generally preferable to store the metadata as relations in the database itself.
- The exact choice of how to represent system metadata by relations must be made by the system designers.
- The data dictionary is often stored in a non-normalized form to achieve fast access.
- Whenever the database system needs to retrieve records from a relation, it must first consult the Relation_metadata relation to find the location and storage organization of the relation, and then fetch records using this information.

➢ Basic Concepts of Indexing

- Once the records of a file are placed on the disk using some file organization method, the main issue is to provide a quick response to different queries. For this, an additional structure called **index** on the file can be created, which provides efficient access to the file records. Index can be created on any field of the file, and that field is called **indexing field** or **indexing attribute.**
- An index for a file in a database system works in much the same way as the index in the textbook.
- Database-system indices play the same role as book indices in libraries. For example, to retrieve a instructor record given an *ID*, the database system would look up an index to find on which disk block

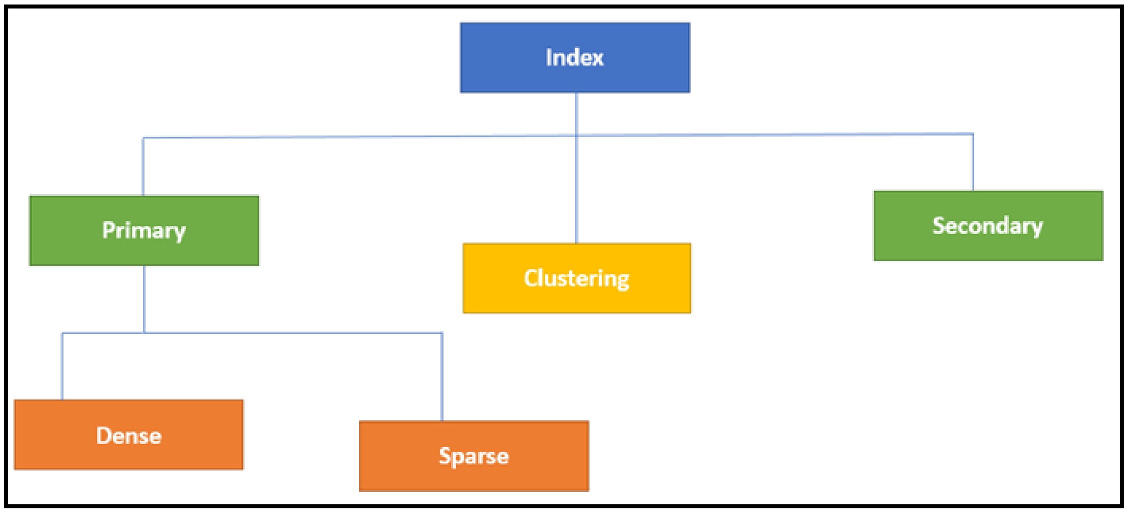
the corresponding record resides, and then fetch the disk block, to get the appropriate *instructor* record.

- There are two basic kinds of indices:
    - **Ordered indices**. Based on a sorted ordering of the values.
    - **Hash indices**. Based on a uniform distribution of values across a range of buckets. The bucket to which a value is assigned is determined by a function, called a *hash function*.
- No one technique is the best. Rather, each technique is best suited to particular database applications. Each technique must be evaluated on the basis of these factors:-
    - **Access types**: The types of access that are supported efficiently. E.g. access types can include finding records with a specified attribute value.
    - **Access time**: The time it takes to find a particular data item, or set of items.
    - **Insertion time**: The time it takes to insert a new data item. This value includes the time it takes to find the correct place to insert the new data item, as well as the time it takes to update the index structure.
    - **Deletion time**: The time it takes to delete a data item. This value includes the time it takes to find the item to be deleted, as well as the time it takes to update the index structure.
    - **Space overhead**: The additional space occupied by an index structure. It is usually meaningful to sacrifice the space to achieve improved performance.
- We often want to have more than one index for a file. For example, we may wish to search for a book by author, by subject, or by title.

➢ Ordered Indices

Different types of single-level indexes are as follows:-

- Primary index
- Clustering index
- Secondary index



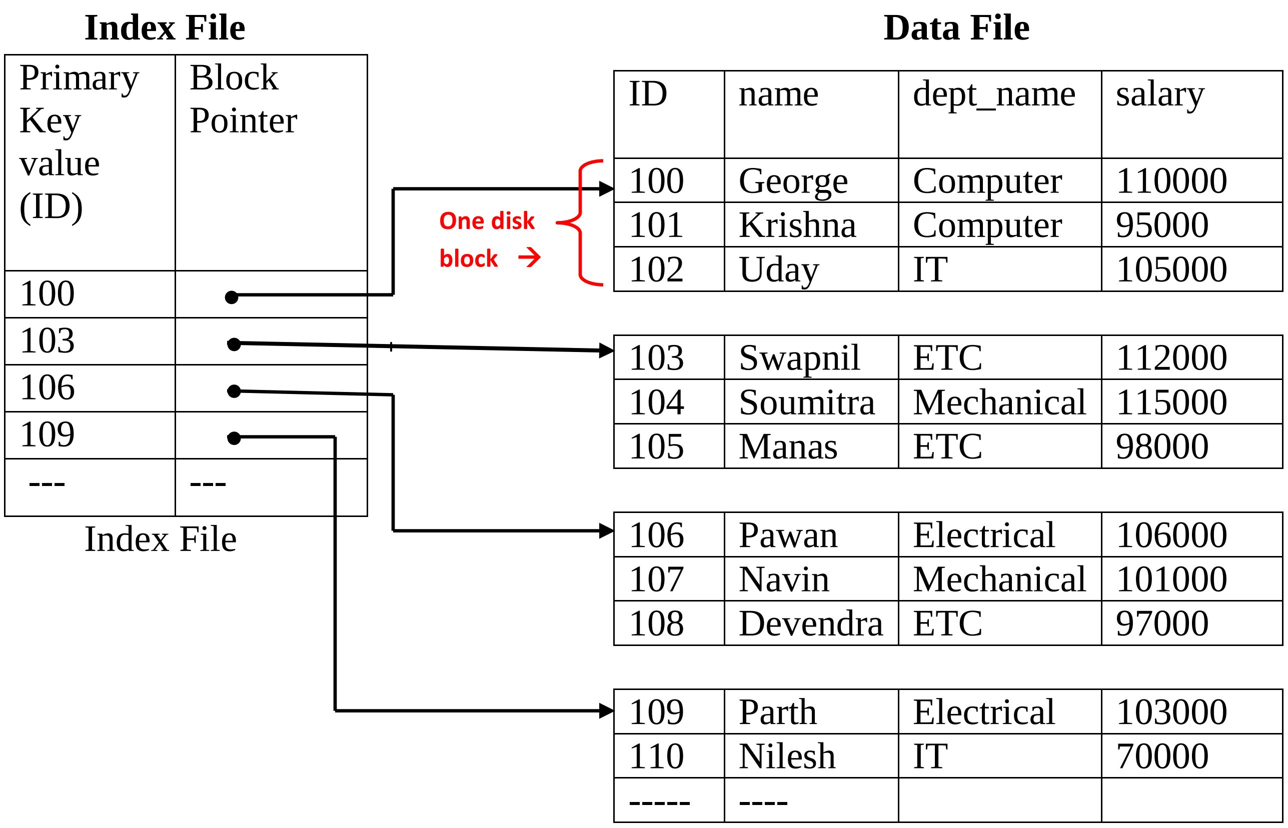
✓ <u>Primary Index</u>

- It refers to the index created on the field on which the data file is ordered (e.g. if data file is ordered on ID field, then ID will be used as index in the index file), that is primary key of relation (often primary key of relation, although that is not necessary).
- The index file contains two attributes-

Value of indexing attribute (e.g. ID)	**Pointer to the record in data file**

- The first record in each block of the data file is called the **anchor record** of the block, or simply the **block anchor.**
- The index file for a primary index needs fewer blocks than does the data file, for two reasons-
    - First, there are fewer index entries than there are records in the data file.
    - Second, each index entry is typically smaller in size than a data record, because it has only two fields.
- So, a binary search on the index file requires fewer block accesses than a binary search on the data file.

- The following figure 4.3, illustrates the primary index:-



Primary Key value (ID)	Block Pointer
100	●
103	●
106	●
109	●
---	---

ID	name	dept_name	salary
100	George	Computer	110000
101	Krishna	Computer	95000
102	Uday	IT	105000
103	Swapnil	ETC	112000
104	Soumitra	Mechanical	115000
105	Manas	ETC	98000
106	Pawan	Electrical	106000
107	Navin	Mechanical	101000
108	Devendra	ETC	97000
109	Parth	Electrical	103000
110	Nilesh	IT	70000
-----	----		

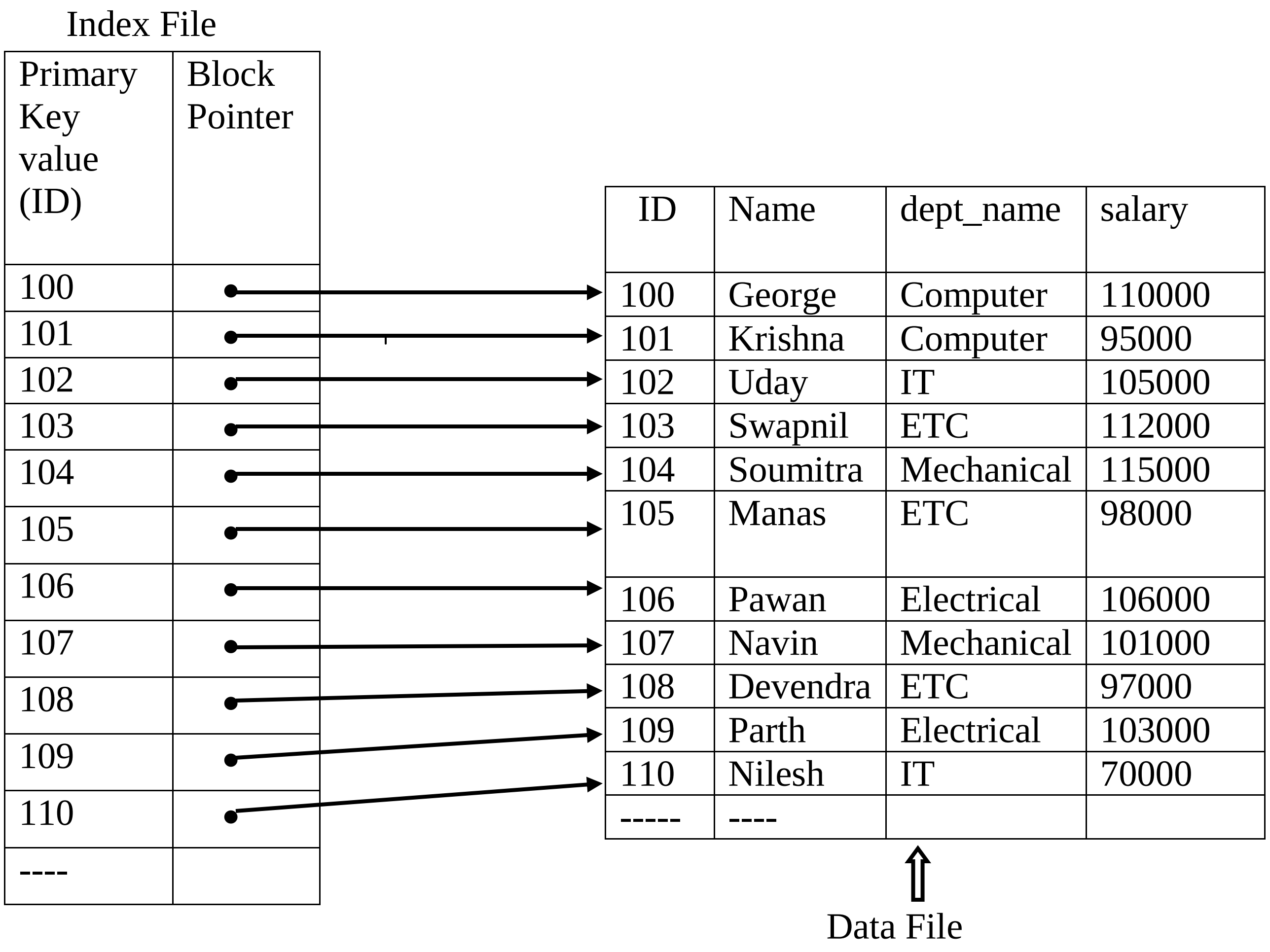
Figure 4.3:- Primary Index

- There is one **index entry** (or **index record**) in the index file for each block in the data file.
- The indexes can also be characterized as Dense or Sparse.
- <u>**Dense and Sparse Indices:-**</u>

  There are two types of ordered indices:-

  - **Dense Index: -** A dense index has an index entry for every search key value (and hence every record) in the data file. In a dense index, the index file contains two entries-
    - Value of indexing attribute (e.g. ID) for each record in the data file.
    - Pointer to each record in the data file.

  The following figure 4.4, illustrates the dense index:-



Primary Key value (ID)	Block Pointer
100	●
101	●
102	●
103	●
104	●
105	●
106	●
107	●
108	●
109	●
110	●

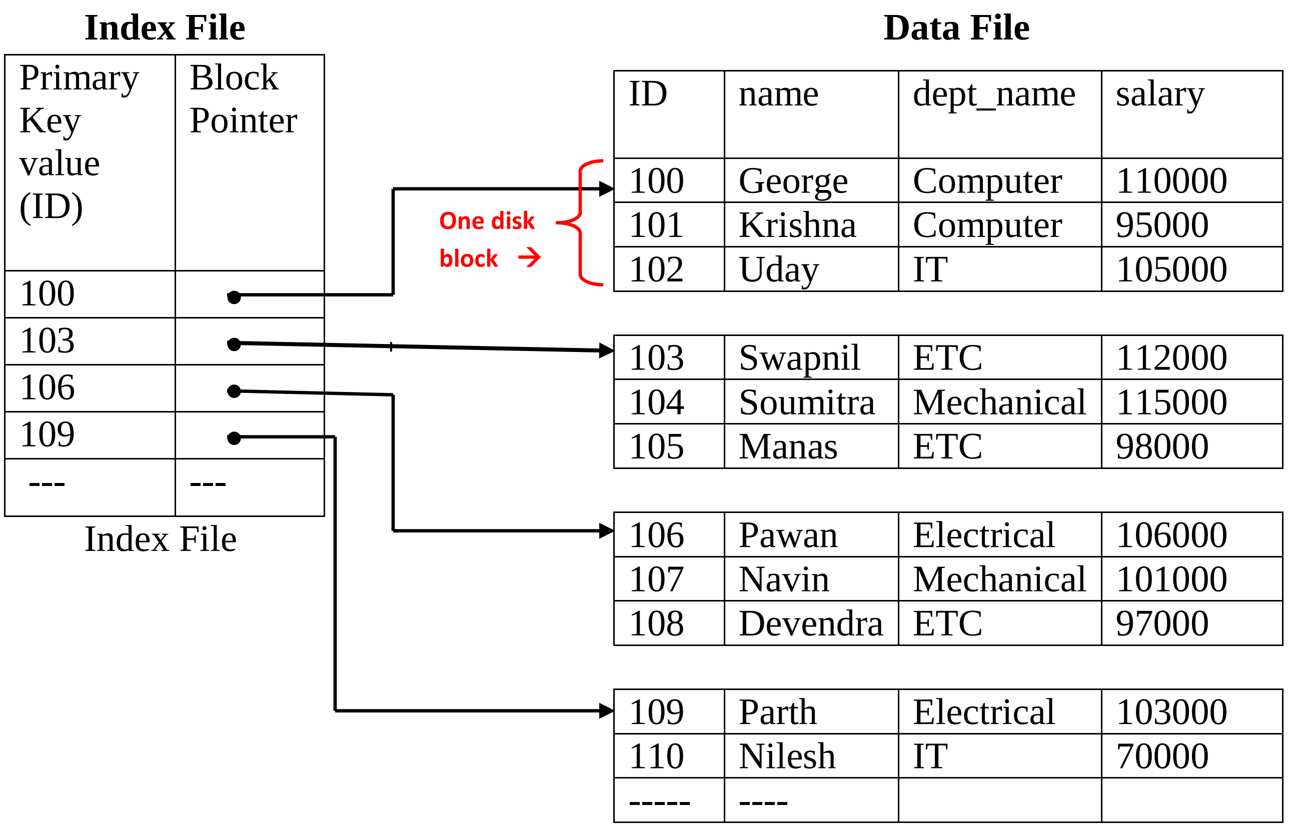
ID	Name	dept_name	salary
100	George	Computer	110000
101	Krishna	Computer	95000
102	Uday	IT	105000
103	Swapnil	ETC	112000
104	Soumitra	Mechanical	115000
105	Manas	ETC	98000
106	Pawan	Electrical	106000
107	Navin	Mechanical	101000
108	Devendra	ETC	97000
109	Parth	Electrical	103000
110	Nilesh	IT	70000
-----	----		

Figure 4.4:- Dense Index

- **Sparse Index: -** A sparse (or nondense) index, has index entries for only some of the search key values. Therefore, 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 (or every record).

  Compared to dense indices:
  - It has less space and less maintenance overhead for insertions and deletions.
  - It is generally slower than dense index for locating records.

The figure 4.5 illustrates the Sparse index.



Index File

Primary Key value (ID)	Block Pointer
100	●
103	●
106	●
109	●
---	---

Data File

ID	name	dept_name	salary
100	George	Computer	110000
101	Krishna	Computer	95000
102	Uday	IT	105000
103	Swapnil	ETC	112000
104	Soumitra	Mechanical	115000
105	Manas	ETC	98000
106	Pawan	Electrical	106000
107	Navin	Mechanical	101000
108	Devendra	ETC	97000
109	Parth	Electrical	103000
110	Nilesh	IT	70000
-----	----		

Figure 4.5:- Sparse Index

Figures 4.4 and 4.5 show dense and sparse indices, respectively for the instructor file. Suppose that we are looking up the record of instructor with ID "102". Now, using the dense index of figure 4.4, we follow the pointer directly to the desired record and the search is complete. If we are using the sparse index of figure 4.5, then we do not find an index entry for "102". Since the last entry before "102" is "100", we follow that pointer. We then read the instructor file in sequential order until we find the desired record.

✓ <u>Clustering Index</u>

- It refers to the index created on the ordered non-key attribute – which does not have a distinct value for each record – that attribute (field) is called **clustering attribute (**or **clustering field)**.
- This differs from a primary index, which requires that the ordering field of the data file have a distinct value for each record.
- A clustering index is also an ordered file with two fields:
  - The first field is of the same type as the clustering field of the data file
  - Second field is a block pointer.
- Figure 4.6 illustrates the clustering index.
- There is one entry in the clustering index for each distinct value of the clustering field, containing the value and a pointer to the first block in the data file that has a record with that value for its clustering field.
- To alleviate the problem if insertion, it is common to reserve whole block (or a cluster of continuous blocks) for each value of the clustering field; all records are with that value are placed in the block (or block cluster). This makes insertion and deletion relatively straightforward. The figure 4.6 shows this scheme.

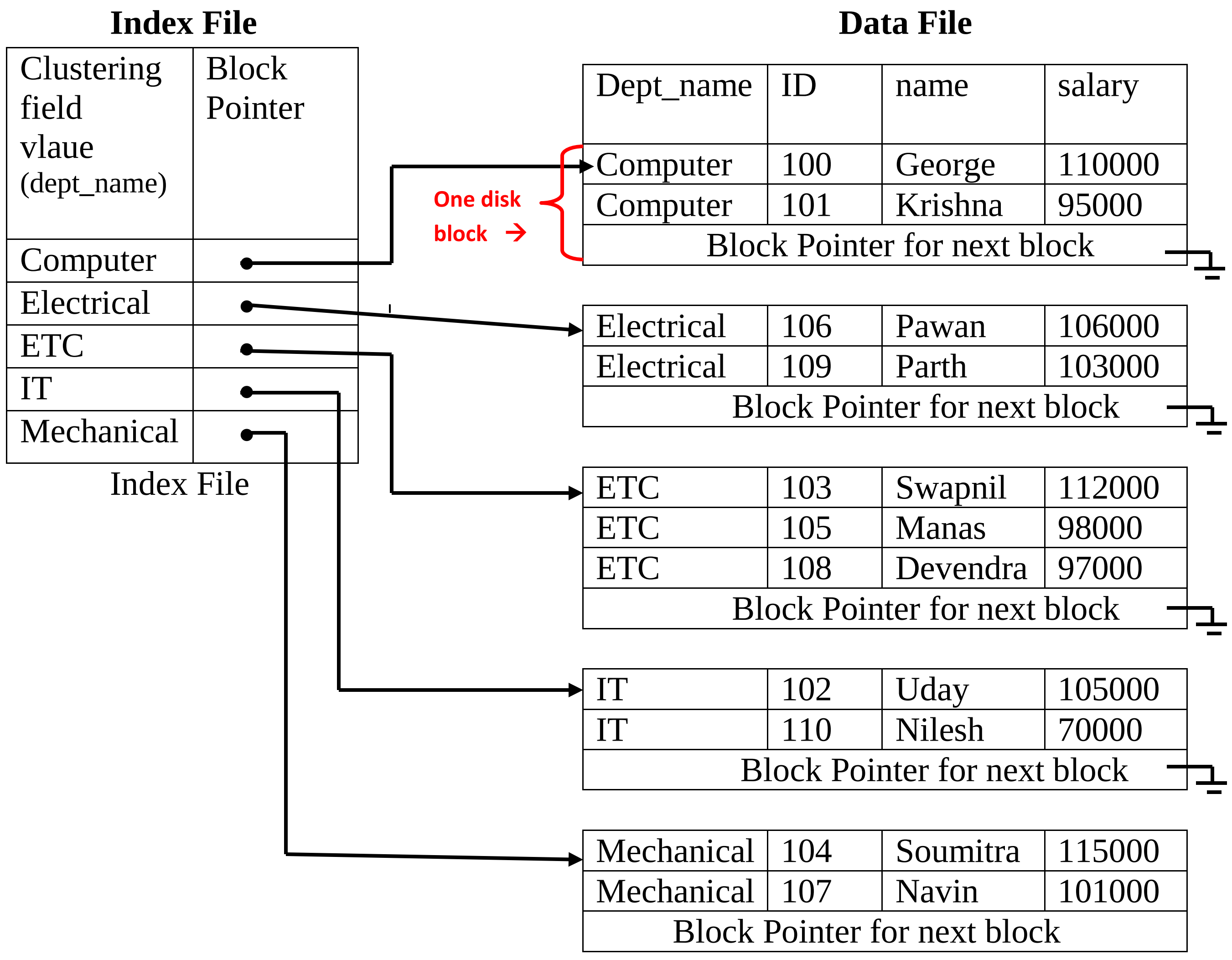

Index File
Data File
Clustering field vlaue (dept_name)
Block Pointer
Computer
Electrical
ETC
IT
Mechanical
Index File
One disk block
Dept_name
ID
name
salary
Computer 100 George 110000
Computer 101 Krishna 95000
Block Pointer for next block
Electrical 106 Pawan 106000
Electrical 109 Parth 103000
Block Pointer for next block
ETC 103 Swapnil 112000
ETC 105 Manas 98000
ETC 108 Devendra 97000
Block Pointer for next block
IT 102 Uday 105000
IT 110 Nilesh 70000
Block Pointer for next block
Mechanical 104 Soumitra 115000
Mechanical 107 Navin 101000
Block Pointer for next block

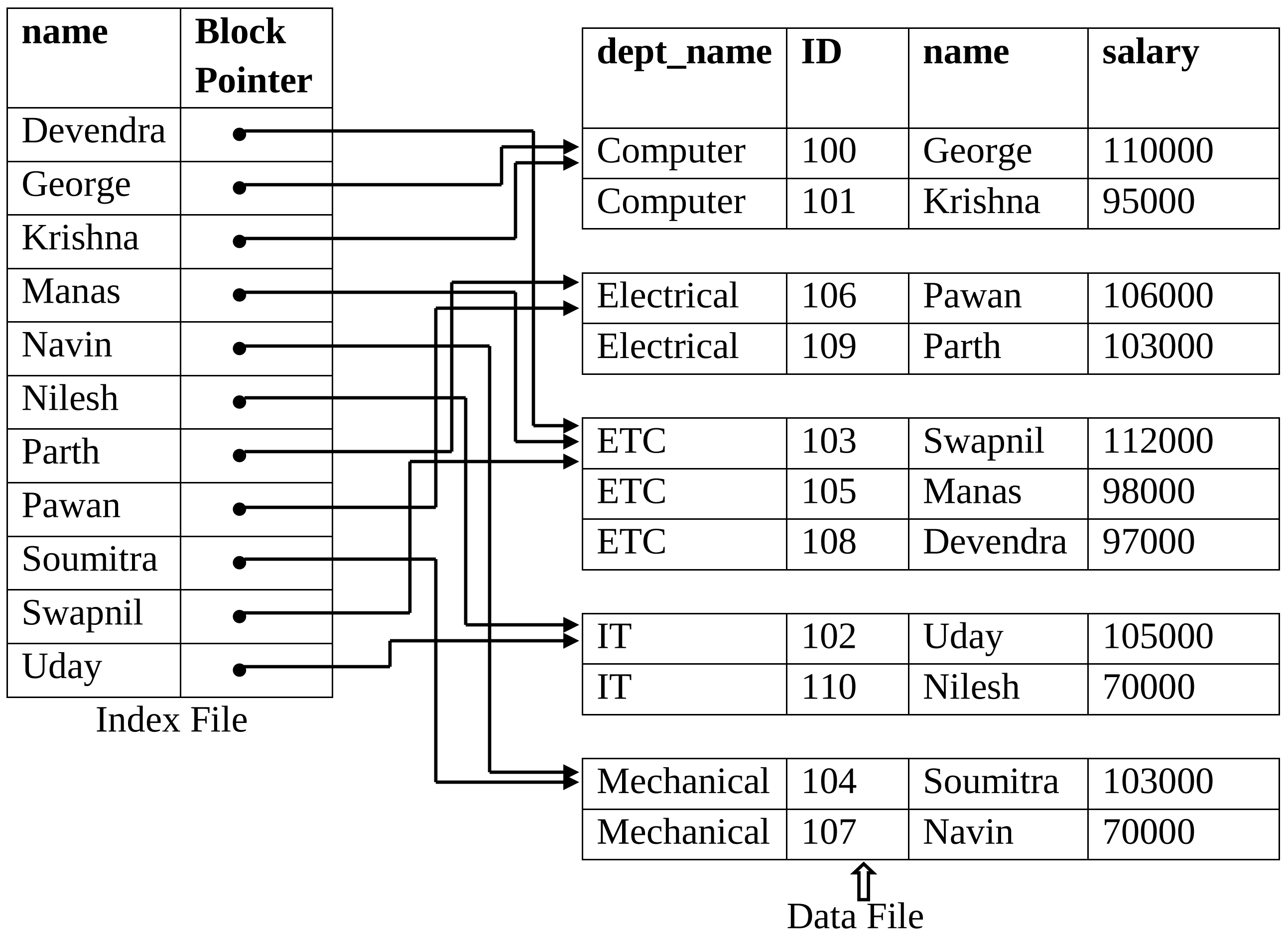
Figure 4.6:- Clustering Index

✓ <u>Secondary Index</u>

- Here only one of the various possible situations is considered. There may be other situations also.
- Consider the instructor relation and its primary and clustering index (only one will be applicable at a time). Now consider the situation when the database is to be searched or accessed in the alphabetical order of names. Any search on a instructor name would require sequential data file search, thus, is going to be very time consuming. Such a search on an average would require reading of half of the total number

of blocks. Thus, we need **secondary indices** in database systems. A **secondary index** is a file that contains records containing a secondary index field value which is not the ordering field of the data file, and a pointer to the block that contains the data record. Although a data file can have only one primary index (as there can be only one ordering of a database file), it can have many secondary indices.

- Secondary index can be defined on an alternate key or non-key attributes. A secondary index that is defined on the alternate key will be dense while secondary index on non-key attributes would require a bucket of pointers for one index entry.
- Following figure 4.7, illustrates the concept.



name	Block Pointer
Devendra	•
George	•
Krishna	•
Manas	•
Navin	•
Nilesh	•
Parth	•
Pawan	•
Soumitra	•
Swapnil	•
Uday	•

Index File

dept_name	ID	name	salary
Computer	100	George	110000
Computer	101	Krishna	95000
Electrical	106	Pawan	106000
Electrical	109	Parth	103000
ETC	103	Swapnil	112000
ETC	105	Manas	98000
ETC	108	Devendra	97000
IT	102	Uday	105000
IT	110	Nilesh	70000
Mechanical	104	Soumitra	103000
Mechanical	107	Navin	70000

Data File

Figure 4.7:- A dense secondary index on a non-ordering key field of a file

✓ Multilevel Indices

- Suppose we build a dense index on a relation with 1,000,000 tuples. Index entries are smaller than data records, so let us assume that 100 index entries fit on a 4 kilobyte block. Thus, our index occupies 10,000 blocks. If the relation instead had 100,000,000 tuples, the index would instead occupy 1,000,000 blocks, or 4 gigabytes of space. Such large indices are stored as sequential files on disk.
- If an index is small enough to be kept entirely in main memory, the search time to find an entry is low. However, if the index is so large that not all of it can be kept in memory, index blocks must be fetched from disk when required. (Even if an index is smaller than the main memory of a

computer, main memory is also required for a number of other tasks, so it may not be possible to keep the entire index in memory.) The search for an entry in the index then requires several disk-block reads.

- Solution: treat primary index kept on disk as a sequential file and construct a sparse index on it.
  - outer index :– a sparse index of primary index
  - inner index :– the primary index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.
- Indices at all levels must be updated on insertion or deletion from the file.
- Figure 4.7 illustrates the concept of two-level sparse index.

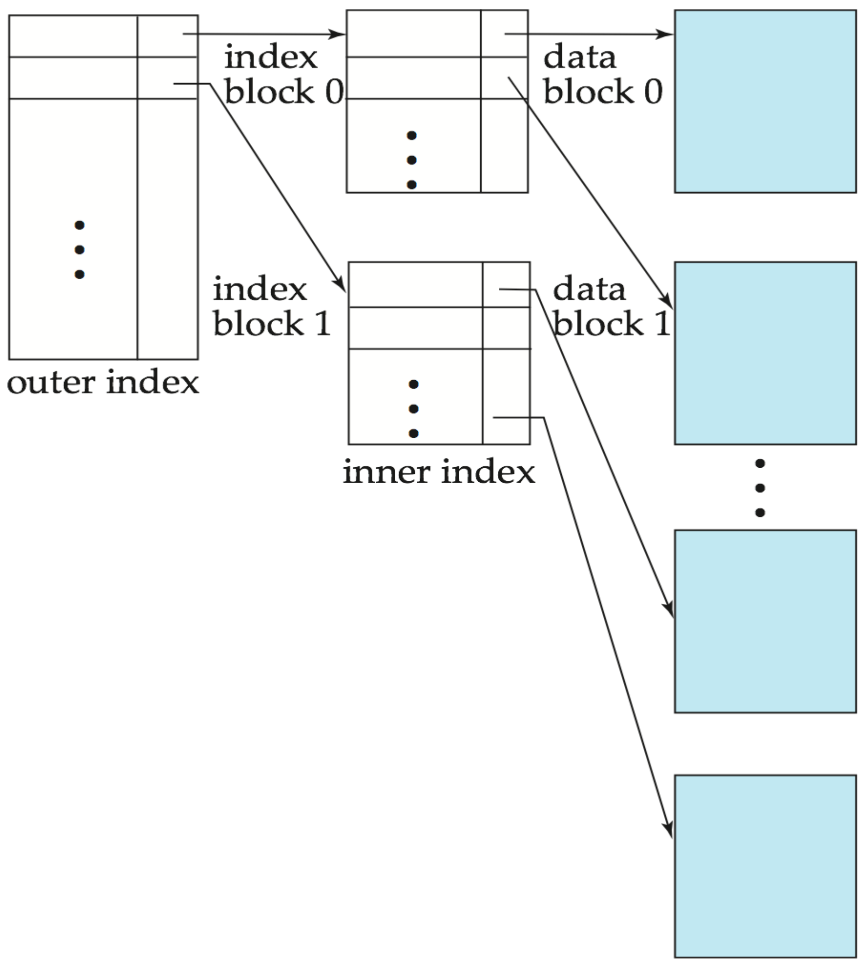
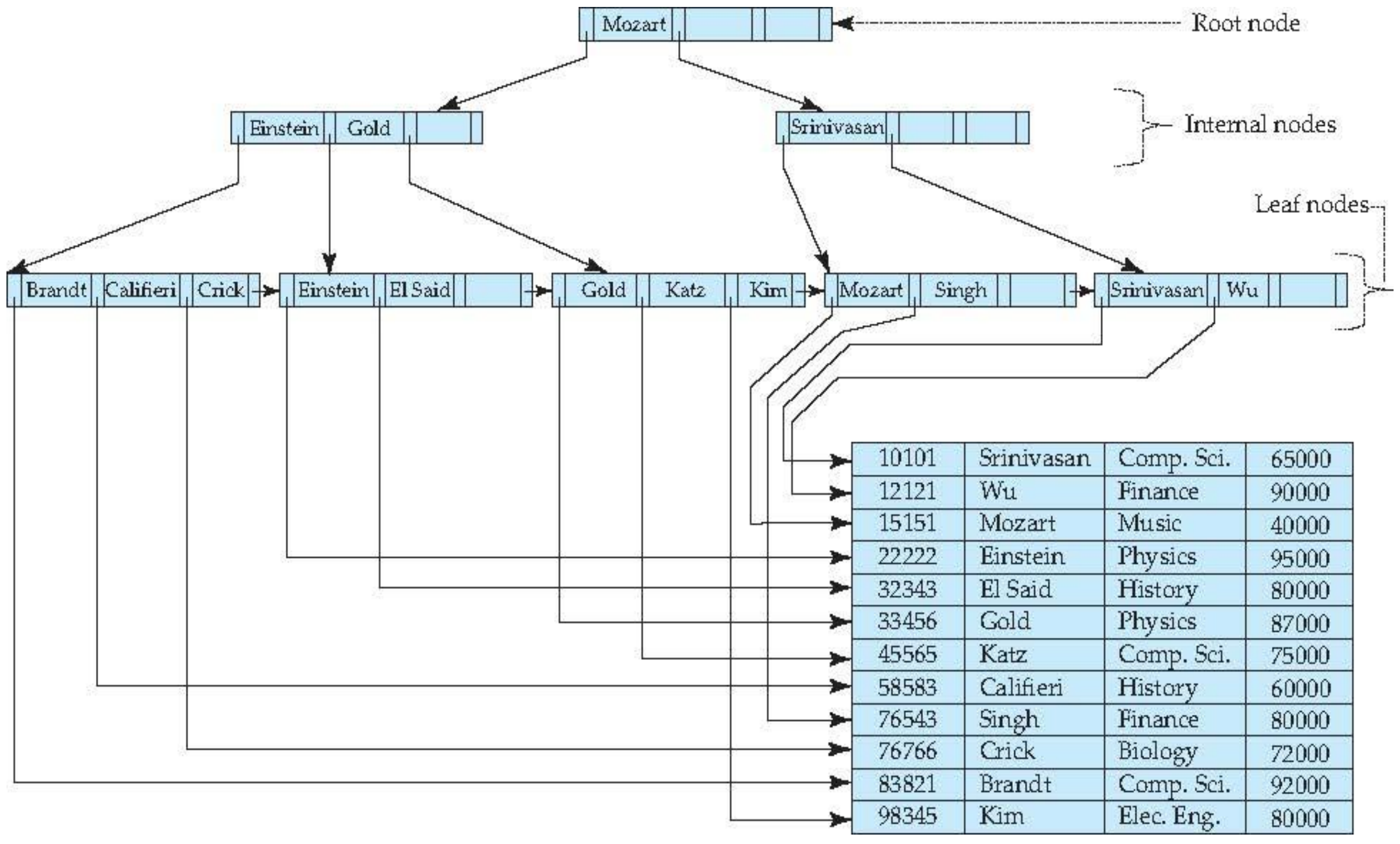


Figure 4.7:- Two-level sparse index.

➢ B^+ Tree Index File

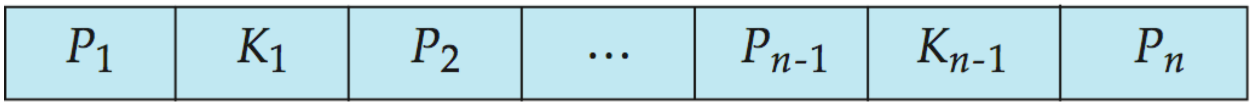
- B^+-tree indices are an alternative to indexed-sequential files.
- Disadvantage of indexed-sequential files:
  - Performance degrades as file grows, since many overflow blocks get created.
  - Periodic reorganization of entire file is required.
- Advantage of B^+-tree index files:
  - Automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
  - Reorganization of entire file is not required to maintain performance.
- (Minor) disadvantage of B^+-trees:
  - extra insertion and deletion overhead, space overhead.
- Advantages of B^+-trees outweigh disadvantages
  - B^+-trees are used extensively
- Example of B^+ Tree for instructor file (n=4)



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

- The B$^+$-tree is a rooted tree satisfying the following properties:
  - All paths from root to leaf are of the same length
  - Each node that is not a root or a leaf has between $\lceil n/2 \rceil$ and n children.
  - A leaf node has between $\lceil (n-1)/2 \rceil$ and $n-1$ values
  - Special cases:
    - If the root is not a leaf, it has at least 2 children.
    - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and $(n-1)$ values.

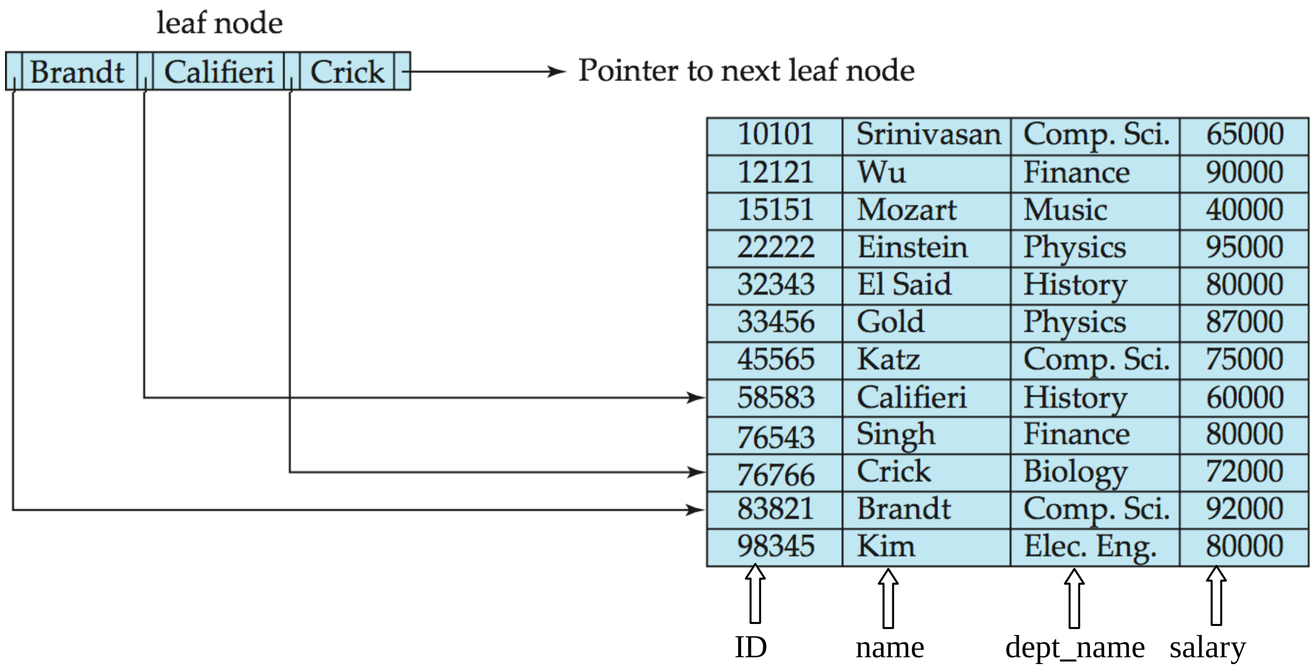
- Typical node of a B$^+$ tree



  - K_i are the search-key values
  - P_i are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).
- The search-keys in a node are ordered

  $K_1 < K_2 < K_3 < \ldots < K_{n-1}$

  (Initially assume no duplicate keys, address duplicates later)
- Properties of a leaf node:-
  - For $i = 1, 2, \ldots, n-1$, pointer P_i points to a file record with search-key value K_i,
  - If L_i, L_j are leaf nodes and $i < j$, L_i's search-key values are less than or equal to L_j's search-key values
  - P_n points to next leaf node in search-key order
- A leaf node for instructor B$^+$ tree index (n=4) is shown in following figure:-



ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor file

Unit-IV

(Relational Database Design)

➢ Features of Good Relational Designs

- In general, the goal of **relational database design** is to generate a set of relation schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily. This is accomplished by designing schemas that are in an appropriate ***normal form*.**
- It is possible to generate a set of relation schemas directly from the E-R design.
- The goodness (or badness) of the resulting set of schemas depend on how good the E-R design was in the first place.
- Consider our College example:- there are four relation schemas to form the College database schema –

instructor (<u>ID</u>, name, dept_name, salary)

department (<u>dept_name</u>, building, budget)

course (<u>course_id</u>, title, dept_name, credits)

prereq (<u>course_id</u>, <u>prereq_id</u>)

✓ <u>Design Alternatives: Larger Schemas</u>

- Suppose that instead of having the schemas *instructor* and *department* separate, we have the schema:

  ***inst_dept* (*ID, name, salary, dept name, building, budget*)**
- This represents the result of a natural join on the relations corresponding to *instructor* and *department*.
- To understand the concept again consider the instance of *instructor* and *department* relations as:

ID	**name**	**dept_name**	**Salary**
100	George	Computer	110000
101	Krishna	Computer	95000
102	Uday	IT	105000
103	Swapnil	ETC	112000
104	Soumitra	Mechanical	115000
105	Manas	ETC	98000
106	Pawan	Electrical	106000
107	Navin	Mechanical	101000
108	Devendra	ETC	97000
109	Parth	Electrical	103000
110	Nilesh	IT	70000

Figure 4.8: - The instructor relation

dept_name	**building**	**budget**
Computer	Building No. 1	1000000
IT	Building No. 2	500000
ETC	Building No. 1	800000
Mechanical	Building No. 3	900000
Electrical	Building No. 2	600000

Figure 4.9: - The department relation

- Let us consider the instance of the *inst_dept* relation shown in the figure 4.10. Notice that we have to repeat the department information ("building" and "budget") once for each instructor in the department. For example, the information about the ETC department (Building

No. 1, 800000) is included in the tuples of Swapnil, Manas and Devendra.

ID	**name**	**Salary**	**dept_name**	**building**	**budget**
100	George	110000	Computer	Building No. 1	1000000
101	Krishna	95000	Computer	Building No. 1	1000000
102	Uday	105000	IT	Building No. 2	500000
103	Swapnil	112000	ETC	Building No. 1	800000
104	Soumitra	115000	Mechanical	Building No. 3	900000
105	Manas	98000	ETC	Building No. 1	800000
106	Pawan	106000	Electrical	Building No. 2	600000
107	Navin	101000	Mechanical	Building No. 3	900000
108	Devendra	97000	ETC	Building No. 1	800000
109	Parth	103000	Electrical	Building No. 2	600000
110	Nilesh	70000	IT	Building No. 2	500000

Figure 4.10:- *inst_dept* relation

- This suggests that using *inst_dept* is a bad idea since it stores the budget amounts **redundantly** and runs the risk that some user might update the budget amount in one tuple but not all, and thus create **inconsistency.**
- **Another problem** with this design (inst_dept schema):-
  Suppose we are creating a new department in the college. In this design, we cannot represent directly the information concerning a department (*dept_ name*, *building*, *budget*) unless that department has at least one instructor in the college. This is because tuples in the *ins_dept* table require values for *ID*, *name*, and *salary*. So we have to create a tuple with a null value for ID, name and salary. **In some cases null values are troublesome**, as proved in the study of SQL.

✓ **Design Alternatives: Smaller Schemas**

- Suppose that, we had started out with the schema *inst_dept*. How would we recognize that it requires repetition of information and should be split into the two schemas *instructor* and *department*?
- By observing the contents of actual relations on schema *inst_ dept*, we could note the repetition of information resulting from having to list the building and budget once for each instructor associated with a department. However, this is an **unreliable process**. A real-world database has a large number of schemas and an even larger number of attributes. The number of tuples can be in the millions or higher. Discovering repletion would be costly.
- Therefore the database designer needs to specify some rules e.g. say by using **functional dependency**.
- Above observations and the rules (functional dependencies in particular) allow the database designer to recognize situations where a schema can be split (or *decomposed)*, into two or more schemas. The right way to decompose *inst_dept* is into schemas ***instructor* and *department* as in the original design**. Finding the right decomposition is much harder for schemas with a large number of attributes and several functional dependencies.
- Some decompositions are **lossy decompositions** and some **are lossless decompositions.**
- Consider the example, say Design1:-

  inst (ID, name, street, City, salary);

  E.g. this schema is decomposed into two schemas, say Design2:

  inst1 (ID, name);

  inst2 (name, street, city, salary);
- Now let the instance of inst is as follows:-

ID	**name**	**street**	**city**	**Salary**
100	George	Shiv Colony	Jalgaon	75000
101	Krishna	Pimprala	Jalgaon	85000
102	Uday	Petrol Pump	Dhule	87000
103	Krishna	Bazar Road	Daryapur	90000

- Consider Design2:- so the instance of inst1 will be as follows:

inst1

ID	**name**
100	George
101	Krishna
102	Uday
103	Krishna

And the instance of inst2 will be as follows:

inst2

name	**street**	**city**	**salary**
George	Shiv Colony	Jalgaon	75000
Krishna	Pimprala	Jalgaon	85000
Uday	Petrol Pump	Dhule	87000
Krishna	Bazar Road	Daryapur	90000

- Now when we perform the **Natural join** between inst1 and inst2 to regenerate original inst, then **we will not get** the exact original inst relation.

Natural join between inst1 and inst2 **(inst1 ⋈ inst2)**

name	**ID**	**street**	**city**	**salary**
George	100	Shiv Colony	Jalgaon	75000
Krishna	**101**	**Pimprala**	**Jalgaon**	**85000**
Krishna	**103**	**Pimprala**	**Jalgaon**	**85000**
Uday	102	Petrol Pump	Dhule	87000
Krishna	**101**	**Bazar Road**	**Daryapur**	**90000**
Krishna	**103**	**Bazar Road**	**Daryapur**	**90000**

Here, we have more tuples, but actually less information in the following sense-

For Krishna with the id= "101" and
for Krishna with the id= "103" , there are two values for street, city and salary.

- Such type of decompositions will be refereed as **Lossy Decompositions** (sometimes called as lossy-join decomposition) and we would like to avoid such decompositions.
- In design3:-
  E.g. this schema is decomposed into two schemas:
  inst1 (ID, name); --- same from Design2
  inst3 (ID, street, city, salary);

inst3

ID	**street**	**City**	**salary**
100	Shiv Colony	Jalgaon	75000
101	Pimprala	Jalgaon	85000
102	Petrol Pump	Dhule	87000
103	Bazar Road	Daryapur	90000

- Now when we perform the Natural join between inst1 and inst3, then **we will get the exact original** inst relation as follows:

  Natural join between inst1 and inst3 **(inst1 ⋈ inst3)**

ID	**name**	**Street**	**city**	**salary**
100	George	Shiv Colony	Jalgaon	75000
101	Krishna	Pimprala	Jalgaon	85000
102	Uday	Petrol Pump	Dhule	87000
103	Krishna	Bazar Road	Daryapur	90000

  This is the example of **Lossless Decomposition**.
- The property of lossless decomposition ensures that no **spurious tuples** are generated when relations are re-united through a natural join operation. In lossless decomposition, a relation that is decomposed during normalization can be rejoined using joins (e.g. natural join) to produce original data. Sometimes, a relation is decomposed into more than two relations.

➢ Normalization

- The E-R model based database designing may have the problems of redundancy and inconsistency. So to solve those problems, some solution is required. This solution can be the **Normalization**.
- Normalization is the process of producing a simpler and more reliable database structure.
- It is the process of modifying a relation schema based on its FDs (functional dependency) and primary keys so that it confirms to certain rules called **normal forms.** A relation is said to be in a particular normal form if it satisfies certain specified constraints. Each of the normal forms is stricter than its predecessors. The normal forms are used to ensure that various types of anomalies and inconsistencies are removed from the database.
- In short - **Normalization** is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.
- The opposite of normalization is denormalization, where we want to combine multiple tables together into a larger table. Denormalization is most frequently associated with designing the fact table in a data warehouse.

✓ **The Need for Normalization (i.e. Purpose of Normalization)**

- It makes the database design efficient in performance.
- It makes the database design free of update, insertion and deletion anomalies.
- It makes the design according to the rules of relational database.
- It makes a design that allows simple retrieval of data.
- It simplifies data maintenance and reduces the need to restructure data.
- It minimizes data redundancy (data duplication), which in turn ensures data consistency-

- Problems of data redundancy:
  a) Disk space wastage
  b) Data inconsistency
  c) DML queries can become slow.
- In general, there are six types of Normal Forms: - 1NF, 2NF, 3NF, BCNF, 4NF and 5NF.

➢ Process of Normalization

- Some of the basic terms/concepts/building-blocks used in defining various normal forms are:

✓ Functional Dependency

- Functional dependency is a relationship between attributes. It means that if the value of one attribute is known, it is possible to obtain value of another attribute. Assume STUDENT relation as follows:
  STUDENT (S_RNO, S_NAME, S_EMAIL)
  Here S_RNO means Student's Registration Number.
- If the value of S_RNO is known then it is possible to obtain the value of S_NAME. It means S_NAME is functionally dependent on S_RNO. Similarly S_EMAIL is also functionally dependent on S_RNO.
- Functional dependency is written as:
  A→B
  which means A decides B.
  Means B is determined by A.
  The value on left side (i.e A) is called as **determinant.**
- In our example Functional dependency is written as:
  1) S_RNO →S_NAME
  2) S_RNO →S_EMAIL

  First case: It is read as "S_RNO determines S_NAME" or "S_NAME is functionally dependent on S_RNO". The value on left side i.e. S_RNO is called as **determinant.**

✓ Full Functional Dependency

- In a relation R, attribute B of R is **fully functional dependent** on an attribute or set of attributes A, if B is functionally dependent on A but not functionally dependent on any other proper subset of A.
- E.g. consider
  MARKS (S_RNO, SUBJECT, MARKS)
  One student is studying many subjects. So both S_RNO and SUBJECT are required to determine particular MARKS.
  S_RNO, SUBJECT → MARKS
- So here, MARKS is **fully functionally dependent** on both fields S_RNO and SUBJECT. It is not functionally dependent on either S_RNO or SUBJECT alone.

✓ Partial Dependency

- Consider the following table – here
  C-NO: - Course Number
  C-FEE: - Course Fee

S-RNO	**C-NO**	C-FEE
1	Comp1	2000
2	Comp2	3000
1	Comp2	3000
3	Comp1	2000
2	Comp1	2000

- Let, (S-RNO, C-NO) is a candidate key. Here,
  C-NO→C-FEE
  It means C-FEE is dependent on C-NO alone and C-NO is a proper subset of the candidate key. This is called as **partial dependency.**
- Definition of partial dependency :- A and B are the attributes in a relation R. Attribute B is partially dependent on attribute A only if it is dependent on subset of attribute A.

✓ Transitive Dependency

- Suppose A, B, C are attributes of a relation R.

  If A→B and B→C

  then C is transitively dependent on A via B, provided that A is not functionally dependent on B or C.
- Consider the following table –

S-RNO	C-NO	Marks	Grade
1	Comp1	91	A^+
2	Comp2	84	A
1	Comp2	88	A
3	Comp1	78	B^+
2	Comp1	68	B

Here Grade depends on Marks, and Marks depends on (S-RNO, C-NO).

(S-RNO, C-NO)→Marks

Marks→Grade

Therefore Grade is **fully transitively dependent** on (S-RNO, C-NO)

➢ Types of Normal Forms

- In general, there are six types of Normal Forms: - 1NF, 2NF, 3NF, BCNF, 4NF and 5NF.

  1NF – First Normal Form

  2NF – Second Normal Form

  3NF – Third Normal Form

  BCNF – Boyce Codd Normal Form

  4NF – Fourth Normal Form

  5NF – Fifth Normal Form

- The figure 4.11 illustrates six normal forms along with short explanation - how any normal form is converted to next higher normal form.
- The figure 4.12 is a simple figure to represent various normal forms.

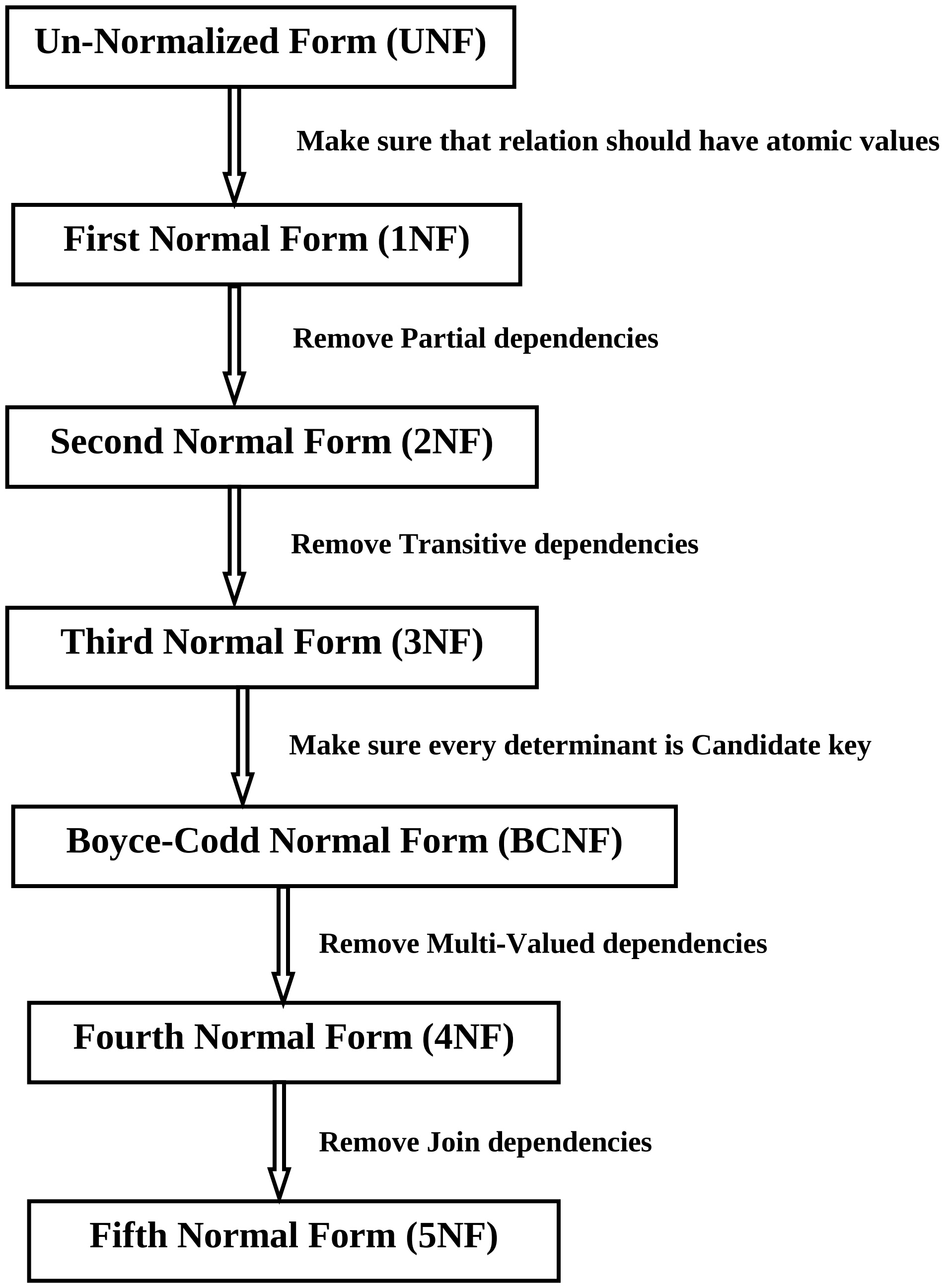

Un-Normalized Form (UNF)
Make sure that relation should have atomic values
First Normal Form (1NF)
Remove Partial dependencies
Second Normal Form (2NF)
Remove Transitive dependencies
Third Normal Form (3NF)
Make sure every determinant is Candidate key
Boyce-Codd Normal Form (BCNF)
Remove Multi-Valued dependencies
Fourth Normal Form (4NF)
Remove Join dependencies
Fifth Normal Form (5NF)

Figure 4.11

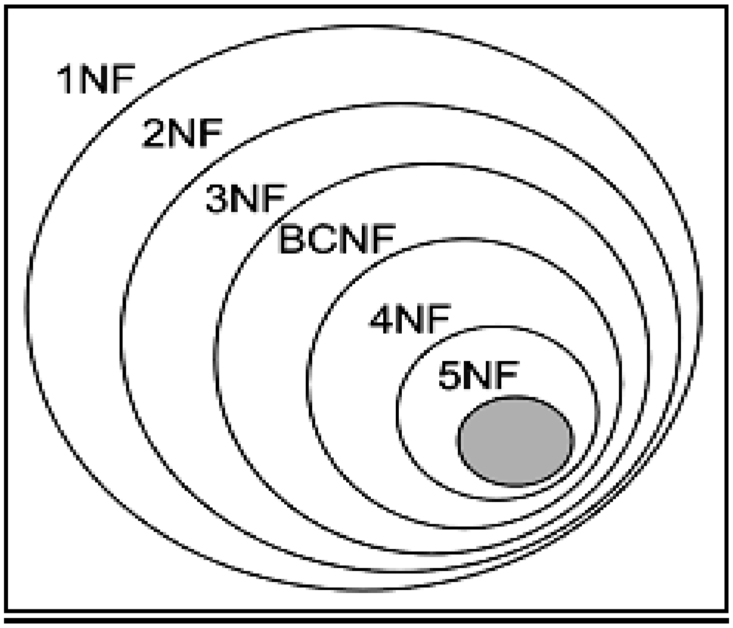


Figure 4.12

✓First Normal Form (1NF):-

- The relation R is in 1NF, if and only if all the attributes of relation R are **atomic** (i.e. indivisible). [i.e. Domain must be atomic]
- It means multi-valued attributes, composite attributes are not allowed in 1NF.
- The example of un-normalized relation is as shown in **fig 4.13:**

A_No	Skill No	Skill Category	P_No	P_Grade	A_Name	A_Email	A_Age	G_No	G_City	G_Guide
210	1130	System	3	B	Ajay	A1@gmail.com	52	520	Pune	Bipin
350	1130 1790 2040	System Tax Audit	5 1 6	A B A	Devang	D1@gmail.com	31	440	Mumbai	Ganesh
500	1790	Tax	2	B	Chirag	C1@gmail.com	48	440	Mumbai	Ganesh
770	1480 1790	Consulting Tax	6 6	A A	Zak	Z1@gmail.com	51	520	Pune	Bipin

Figure 4.13:- Un-normalized Relation

In the figure 4.13:-
A_No:- Accountant Number
A_Name:- Accountant Name
A_Email:- Accountant Email
A_Age:- Accountant Age
P_No:- Proficiency Number
P_Grade:- Proficiency Grade
G_No:- Group Number
G_City:- Group City
G_Guide:- Group Guide (Supervisor)

- The above relation is un-normalized, because it contains repeating groups of three attributes. The three attributes Skill No, Skill Category and P_No contain more than one value (i.e. **not atomic**).
- The above relation can be converted to 1NF by making each attribute atomic, as shown in following figure 4.14:

Primary Key

A_No	Skill No	Skill Category	P_No	P_Grade	A_Name	A_Email	A_Age	G_No	G_City	G_Guide
210	1130	System	3	B	Ajay	A1@gmail.com	52	520	Pune	Bipin
350	1130	System	5	A	Devang	D1@gmail.com	31	440	Mumbai	Ganesh
350	1790	Tax	1	B	Devang	D1@gmail.com	31	440	Mumbai	Ganesh
350	2040	Audit	6	A	Devang	D1@gmail.com	31	440	Mumbai	Ganesh
500	1790	Tax	2	B	Chirag	C1@gmail.com	48	440	Mumbai	Ganesh
770	1480	Consulting	6	A	Zak	Z1@gmail.com	51	520	Pune	Bipin
770	1790	Tax	6	A	Zak	Z1@gmail.com	51	520	Pune	Bipin

Figure 4.14: - First Normal Form (1NF)

- The anomalies in 1NF:
  - **Updation Anomaly**- sometimes Updation is difficult. Suppose the user wants to change the name of A_No 350 to the full name say "Devang Rathi", then he has to change in all records. If all records are not changed then the table will contain inconsistent data.
  - **Insertion Anomaly** (i.e. Addition problem) – sometimes insertion of new data is difficult.
    Suppose the user wants to insert another Skill No say 880. This will not be possible until any Accountant with that skill exists.
  - **Deletion Anomaly** – sometimes deletion of some data is not possible.
    Suppose the user wants to delete Guide Ganesh's record. If his record is deleted then other data (Accountant's data) will also be lost.

✓Second Normal Form (2NF):-

- The relation is in 2NF if:
  - It is in 1NF and
  - All of its non-key attributes are fully functionally dependent on the whole key (i.e. there is no partial dependency between non-key attributes and key attributes).
- In the above 1NF relation there are some attributes which are not depending on the whole primary key. Accountant name, Accountant age is determined by Accountant number only.
- So following changes can be done for 2NF:

Primary Kay

A_No	**A_Name**	**A_Email**	**A_Age**	**G_No**	**G_City**	**G_Guide**
210	Ajay	A1@gmail.com	52	520	Pune	Bipin
350	Devang	D1@gmail.com	31	440	Mumbai	Ganesh
500	Chirag	C1@gmail.com	48	440	Mumbai	Ganesh
770	Zak	Z1@gmail.com	51	520	Pune	Bipin

Figure 4.15:- Accountant Table in 2NF

Primary Kay

Skill No	Skill Category
1130	System
1790	Tax
2040	Audit
1480	Consulting

Figure 4.16: - Skill Table in 2NF

Primary Kay

A_No	**Skill No**	**P_No**	**P_Grade**
210	1130	3	B
350	1130	5	A
350	1790	1	B
350	2040	6	A
500	1790	2	B
770	1480	6	A
770	1790	6	A

Figure 4.17:- Proficiency Table in 2NF

- Analysis of 2NF:-
  - **Update Anomaly**- The updating problem existing in 1NF is eliminated/reduced in 2NF.
  - **Inconsistent data** - The possibility of inconsistent data is eliminated/reduced.
  - **Insertion Anomaly** (i.e. Addition problem) – in 2NF any number of skills can be added in skill relation without an accountant with that skill. So addition problem is eliminated/reduced.
  - **Deletion Anomaly** – it is also eliminated/reduced.

✓ **Third Normal Form (3NF):-**

- A Relation in 2NF may have functional dependency between some non-key attributes. This needs further normalization as the non-keys being dependent leads to unnecessary duplication of data. Normalization to 3NF ensures that there is no functional dependency between non-key attributes.
- The relation is in 3NF if:
  - It is in 2NF and
  - There is no transitive dependency between non-key attributes and key attributes.

- The Accountant table in 2NF contains some attributes which are depending on non-key attributes.
- So following changes can be done for 3NF-

Primary Kay

A_No	**A_Name**	**A_Email**	**A_Age**	**G_No**
210	Ajay	A1@gmail.com	52	520
350	Devang	D1@gmail.com	31	440
500	Chirag	C1@gmail.com	48	440
770	Zak	Z1@gmail.com	51	520

Figure 41.8:- Accountant table in 3NF

Primary Kay

G_No	**G_City**	**G_Guide**
520	Pune	Bipin
440	Mumbai	Ganesh

Figure 4.19:- Group table in 3NF

Primary Kay

Skill No	**Skill Category**
1130	System
1790	Tax
2040	Audit
1480	Consulting

Figure 4.20: - Skill Table in 3NF

Primary Kay

A_No	Skill No	P_No
210	1130	3
350	1130	5
350	1790	1
350	2040	6
500	1790	2
770	1480	6
770	1790	6

Figure 4.21:- Proficiency Table in 3NF

Primary Kay

Lower Limit	Upper Limit	P_Grade
7	9	A^+
4	6	A
1	3	B

Figure 4.22: - Grade

✓Boyce Codd Normal Form (BCNF):-

- BCNF was developed by Raymond F. **Boyce** and Edgar F. **Codd** to address certain types of anomalies not dealt with by 3NF as originally defined.
- BCNF is the advance version of 3NF. It is stricter than 3NF.
- It is an advance version of 3NF that's why it is also referred as 3.5NF.
- A relation is in BCNF, if and only if all the determinants are candidate keys.
- It can be checked by identifying all determinants and then making sure that all these determinants are candidate keys.
- What is determinant?

  E.g. if -

  $A \rightarrow B$

  which means A decides B.

  Means B is determined by A.

  The value on left side (i.e A) is called as **determinant.**
- A relation in BCNF is also in 3NF. But a relation in 3NF may not be in BCNF.

- E.g. assume a company, where an employee can work in more than one department. The relation schema Company is as:

  Company(E-id, E-Country, E-dept, Dept-category, Dept-no-of-emp)

  Assume the relational instance as follows:

E-id	**E-Country**	**E-dept**	**Dept-category**	**Dept-no-of-emp**
101	India	Production	D001	100
101	India	Stores	D001	125
102	America	Design	D002	50
102	America	Purchasing	D002	300

- The following FDs (functional dependencies) are identified:

  E-id→E-Country

  E-dept →{Dept-category, Dept-no-of-emp}

  The candidate key is: {E-id, E-dept}
- The table is not in BCNF, because neither E-id nor E-dept alone are candidate keys. It means all determinants are not candidate keys.
- To convert the table into BCNF, we have to break the table into three tables as follows:

Table1

E-id	**E-Country**
101	India
102	America

Table2

E-dept	**Dept-category**	**Dept-no-of-emp**
Production	D001	100
Stores	D001	125
Design	D002	50
Purchasing	D002	300

Table3

E-id	**E-dept**
101	Production
101	Stores
102	Design
102	Purchasing

- The FDs (functional dependencies) are as follows:

  E-id→E-Country

  E-dept →{Dept-category, Dept-no-of-emp}
- The candidate keys are as follows:

  For Table1:- E-id

  For Table2:- E-dept

  For Table3:- {E-id, E-dept}
- This is now in BCNF, as the left side part of both FDs (i.e. E-id and E-dept) are candidate keys.
- Other Example:-
  - The following relational schema
    inst_dept (ID, name, salary, dept name, building, budget)
    is **not in BCNF**
  - The decomposition of inst_dept into two relations is instructor and department-

    instructor (ID, name, dept_name, salary)

    department (dept_name, building, budget)

    Here both the relations instructor and department **are in BCNF.**
- Generally, any schema with only two attributes is in BCNF by definition.
- When we decompose a schema that is not in BCNF, it may be that one or more of the resulting schemas are not in BCNF. In such cases, further decomposition is required.

✓ Fourth Normal Form (4NF) and Multi-valued Dependency:-

- The relation is in 4NF if it is in BCNF and has no multi-valued dependencies.

❖ Multi-valued Dependency:-

- A multi-valued dependency exists when a relation has at least three attributes, two of them are multi-valued and their values depend only on the third attribute.

- Suppose there is a relation R(A, B, C). The multi-valued dependency exists if:
    - A determines multiple values of B
    - A determines multiple values of C
    - B and C are independent of each other
- Multi-valued dependency is represented by double arrow ->>
- In our example:

  A->>B

  A->>C

  A->>B defines a relationship in which a set of values of attribute B is determined by a single value of A.

  A->>C defines a relationship in which a set of values of attribute C is determined by a single value of A.
- Consider the following table:

Name	**Skill**	**Language**
Amit	C, Java	Marathi, Hindi, English

$\Downarrow$ **1NF**

Name	**Skill**	**Language**
Amit	C	Marathi
Amit	C	Hindi
Amit	C	English
Amit	Java	Marathi
Amit	Java	Hindi
Amit	Java	English

- The multi-valued dependencies are as follows:

  Name ->>Skill

  Name ->>Language
- Insertion and Deletion anomalies can be observed here.

- In order to eliminate those anomalies, multi-valued dependencies should be eliminated. It can be done by creating two relations.
- So following design is in **4NF**:

Table1

Name	**Skill**
Amit	C
Amit	Java

Table2

Name	**Language**
Amit	Marathi
Amit	Hindi
Amit	English

- The third normal form (3NF) is the most optimal normal form and most of the databases are designed in 3NF.

➢ Merits and Demerits of Normalization

- Followings are some of the merits (advantages) of normalization:-
  - Makes the database smaller by eliminating redundant data. By doing this the data will be easier to manage and saves us more space of storage.
  - Better Performance - because smaller the data, faster will be the processing.
  - Removes the various types of anomalies such as – Updation, insertion and deletion.
  - Narrower tables are possible as normalized tables will be fine-tuned and will have lesser columns.

- Followings are some of the demerits (disadvantages) of normalization:-
    - The data is spread into more tables, so there are more tables to join. So the task becomes more tedious. The database becomes harder to realize as well.
    - Making the query becomes more difficult.
    - Requires detailed analysis and design. Normalizing a database is a complex and difficult task.
    - A poorly normalized database may perform badly and store data inefficiently.
    - Sometimes data retrieval performance is severely affected.

Bibliography

1. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, “Database System Concepts”, 6th Edition, McGraw-Hill Hill Education.
2. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, “Database System Concepts”, 4th Edition, McGraw-Hill Hill Education.
3. Ramez Elmasri and Shamkant B. Navathe “Fundamentals of Database Systems”, 5th Edition, Pearson.
4. Express Learning, “Database Management Systems”, ITL Education Solutions Limited.
5. Archana Verma, “Database Management Systems”, GenNext Publication.
6. Dr. Rajiv Chopra, “Database Management Systems (DBMS) – A Practical Approach”, 5th Edition, S. Chand Technical
7. Tanmay Kasbe, “Database Management System Concepts – A Practical Approach”, First Edition, Educreation Publishing.
8. Mahesh Mali, “Database Management Systems”, Edition 2019, TechKnowledge Publications.
9. Rajendra Prasad Mahapatra, Govind Verma, “Database Management System”, Khanna Publishing.
10. Malay K. Pakhira, “Database Management System”, Eastern Economy Edition, PHI.
11. Sarika Gupta, Gaurav Gupta, “Database Management System”, Khanna Book Publishing Edition.
12. Riktesh Srivastava, Rajita Srivastava, “Relational Database Management System”, New Age International Publishers.
13. Peter Rob, Carlos Coronel, “Database System Concepts’, Cenage Learning, India Edition
14. Bipin C. Desai, “An Introduction to Database Systems”, Galgotia Publications.
15. G.K. Gupta, “Database Management Systems”, McGraw Hill Education.
16. Shio Kumar Singh, “Database Systems – Concepts, Design and Applications”, 2nd Edition, PEARSON.
17. S.D.Joshi, “Database Management System”, Tech-Max Publication.
18. R. Ramkrishnan , J. Gehrke, "Database Management Systems", 3rd Edition, McGraw-Hill
19. C. J. Date, “Introduction to Database Management Systems”, 8th Edition, Pearson
20. Atul Kahate, “Introduction to Database Management System”, 3rd Edition, Pearson.
21. Bharat Lohiya, “Database Systems”, Tenth Edition, Aditya Publication, Amravati.
22. Vijay Krishna Pallaw, “Database Management System”, 2nd, Asian Books Pvt. Ltd.
23. Database Management Systems, Database Management Systems.
24. Mrs. Jyoti G. Mante (Khurpade), Mrs. Smita M. Dandge, “Database Mangement System”, Nirali Prakashan.
25. Step by Step Database Systems (DBMS), Shiv Krupa Publications, Akola

26. Mrs. Sheetal Gujar –Takale, Mr. Sahil K. Shah, “Database Management System”, Nirali Prakashan.
27. Mrs. Jyoti G. Mante (Khurpade), U.S. Shirshetti, M.V. Salvi, K.S. Sakure, “Relational Database Management System”, Nirali Prakashan.
28. Seema Kedar, Rakesh Shirsath, “Database Management Systems”, Technical Publications.
29. Pankaj B. Brahmankar, “Database Management Systems”, Tech-Max Publications, Pune.
30. Imran Saeed, Tasleem Mustafa, Tariq Mahmood, Ahsan Raza Sattar, “A Fundamental Study of Database Management Systems”, 3rd Edition, IT Series Publication.
31. Database Management Systems Lecture Notes, Malla Reddy College of Engineering and Technology, Secunderabad.
32. Dr. Satinder Bal Gupta, Aditya Mittal, “Introduction to Database Management System, University Science Press.
33. E-Notes BCS 41/ BCA 41 on “Database Management System”, Thiruvalluvar University.
34. Bighnaraj Naik, Digital Notes on “Relational Database Management System”, VSSUT, Burla.
35. Viren Sir, Relational database Management System”, Adarsh Institute of Technolgoyt (Poly), VITA.
36. Sitansu S. Mitra, “Principles of Relational Database Systems”, Prentice Hall.
37. Neeraj Sharma, Liviu Perniu, Raul F. Chong, Abhishek Iyer, Chaitali Nandan, Adi-Cristina Mitea, Mallarswami Nonvinkere, Mirela Danubianu, “Database Fundamentals”, First Edition, DB2 On Campus Book Series.
38. Database Management System, Vidyavahini First Grade College, Tumkur.
39. Bhavna Sangamnerkar, Revised by: Shiv Kishor Sharma, “Database Management System”, Think Tanks Biyani Group of Colleges.
40. Tibor Radvanyi, “Database Management Systems”.
41. Ramon A. Mata-Toledo, Pauline K. Cushman, “Fundamentals of Relational Databases”, Schaum’s Outlies.
42. P.S. Gill, “Database Management Systems”, 2nd Edition, Dreamtech Press, WILEY

Bibliography

Web Resources:-

www.tutorailspoint.com
https://www.geeksforgeeks.org/second-normal-form-2nf/
https://beginnersbook.com/2015/05/normalization-in-dbms/
https://www.javatpoint.com/
https://www.studytonight.com/dbms/fourth-normal-form.php
https://sis.binus.ac.id/2018/01/17/drawbacks-of-normalization/
https://whatisdbms.com/normalization-in-dbms-anomalies-advantages-disadvantages/