Chapter **3**

FILE ORGANIZATION

3.1 INTRODUCTION

A file organization is a way of arranging the records in a file when the file is stored on secondary storage (disk, tape etc.). The different ways of arranging the records enable different operations to be carried out efficiently over the file. A database management system supports several file organization techniques. The most important task of DBA is to choose a best organization for each file, based on its use. The organization of records in a file is influenced by number of factors that must be taken into consideration while choosing a particular technique. These factors are (*a*) fast retrival, updation and transfer of records, (*b*) efficient use of disk space, (*c*) high throughput, (*d*) type of use, (*e*) efficient manipulation, (*f*) security from unauthorized access, (*g*) scalability, (*h*) reduction in cost, (*i*) protection from failure.

This chapter discusses various file organizations in detail.

3.2 BASIC CONCEPTS OF FILES

A File is a collection of related sequence of records. A collection of field names and their corresponding data types constitutes a *record*. A *data type*, associated with each field, specifies the types of values a field can take. All records in a file are of the same record type.

3.2.1 Records and Record Types

Data is generally stored in the form of records. A record is a collection of fields or data items and data items is formed of one or more bytes. Each record has a unique identifier called record-id. The records in a file are one of the following two types :

(*i*) Fixed Length records.

(*ii*) Variable Length records.

3.2.1.1 Fixed Length Records

Every record in the file has exactly the same size (in bytes). The record slots are uniform and are arranged in a continuous manner in the file. A record is identified using both record-id and slot number of the record. The Figure 3.1 shows a structure of fixed length STUDENT record and the Figure 3.2 shows a portion of a file of fixed length records.

```
type  STUDENT = record
          NAME = char(20);
          Roll No = char(5);
      DOB = char(8);
      end
```

FIGURE 3.1. *Structure of fixed length record STUDENT.*

Name	Roll No.	DOB
Naresh	3234	28-02-75
Suresh	5132	20-05-80
Ramesh	3535	24-10-77
Ashish	3987	15-09-72
Manish	4321	18-11-70
Harish	4983	09-06-73
Manoj	3590	05-01-81

FIGURE 3.2. *Portion of a file of fixed length records.*

Advantage of Fixed Length Records

1. Insertion and deletion of records in the file are simple to implement since the space made available by a deleted record is same as needed to insert a new record.

Disadvantage of Fixed Length Records

1. In fixed length records, since the length of record is fixed, it causes wastage of memory space. For example, if the length is set up to 50 characters and most of the records are less than 25 characters, it causes wastage of precious memory space.
2. It is an inflexible approach. For example, if it is required to increase the length of a record, then major changes in program and database are needed.

3.2.1.2 Variable Length Records

Every record in the file need not be of the same size (in bytes). Therefore, the records in the file have different sizes. The major problem with variable length record is that when a new record is to be inserted, an empty slot of the exact length is required. If the slot is smaller, it cannot be used and if it is too big, the extra space is just wasted. A file may have variable length records due to following reasons :

1. One or more than one fields of a record are of varying size but the records in the file are of same record type.

2. One or more than one fields may have multiple values for individual records and are called repeating fields but the records in the file are of same record type.
3. One or more than one fields are optional *i.e.,* they may have values for some but not for all records. The file records in this case are also of the same record type.
4. The file contains records of different record types and different sizes.

Advantage of Variable Length Records

1. It reduces manual mistakes as database automatically adjust the size of record.
2. It saves lot of memory space in case of records of variable lengths.
3. It is a flexible approach since future enhancements are very easy to implement.

Disadvantage of Variable Length Records

1. It increases the overhead of DBMS because database have to keep record of the sizes of all records.

3.2.2 Types of Files

The following three types of files are used in database systems :

1. Master file
2. Transaction file
3. Report file.

1. **Master file :** This file contains information of permanent nature about the entities. The master file act as a source of reference data for processing transactions. They accumulate the information based on the transaction data.
2. **Transaction file :** This file contains records that describe the activities carried out by the organization. This file is created as a result of processing transactions and preparing transaction documents. These are also used to update the master file permanently.
3. **Report file :** This file is created by extracting data from the different records to prepare a report *e.g.* A report file about the weekly sales of a particular item.

3.3 FILE ORGANIZATION TECHNIQUES

A file organization is a way of arranging the records in a file when the file is stored on secondary storage (disk, tape etc). There are different types of file organizations that are used by applications. The operations to be performed and the selection of storage device are the major factors that influence the choice of a particular file organization. The different types of file organizations are as follows :

1. Heap file organization
2. Sequential file organization
3. Indexed—Sequential file organization
4. Hashing or Direct file organization.

3.3.1 Heap File Organization

In this file organization, the records are stored in the file, in the order in which they are inserted. All the new records are stored at the end of the file. This file organization is also called PILE FILE. This organization is generally used with additional access paths, like secondary indexes. Inserting a new record is very fast and efficient. But searching a record using any search condition involves a linear search through the file, which is comparatively more time consuming. Deleting a record from a heap file is not efficient as deletion of records resulted in wastage of storage space. It is generally used to store small files or in cases where data is difficult to organize. It is also used when data is collected at one place prior to processing.

Advantages of Heap File Organization

1. Insertion of new record is fast and efficient.
2. The filling factor of this file organization is 100%.
3. Space is fully utilized and conserved.

Disadvantage of Heap File Organization

1. Searching and accessing of records is very slow.
2. Deletion of many records result in wastage of space.
3. Updation cost of data is comparatively high.
4. It has limited applications.

3.3.2 Sequential File Organization

In sequential file organization, records are stored in a sequential order according to the "search key". A Search key is an attribute or a set of attributes which are used to serialize the records. It is not necessary that search key must be primary key.

It is the simplest method of file organization. Sequential method is based on tape model. Devices who support sequential access are magnetic tapes, cassettes, card readers etc. Editors and compilers also use this approach to access files.

Structure of a sequential file is shown in Figure 3.3. The records are stored in sequential order one after another. To reach at the consequitive record from any record pointers are used. The Pointers are used for fast retrieval of records.

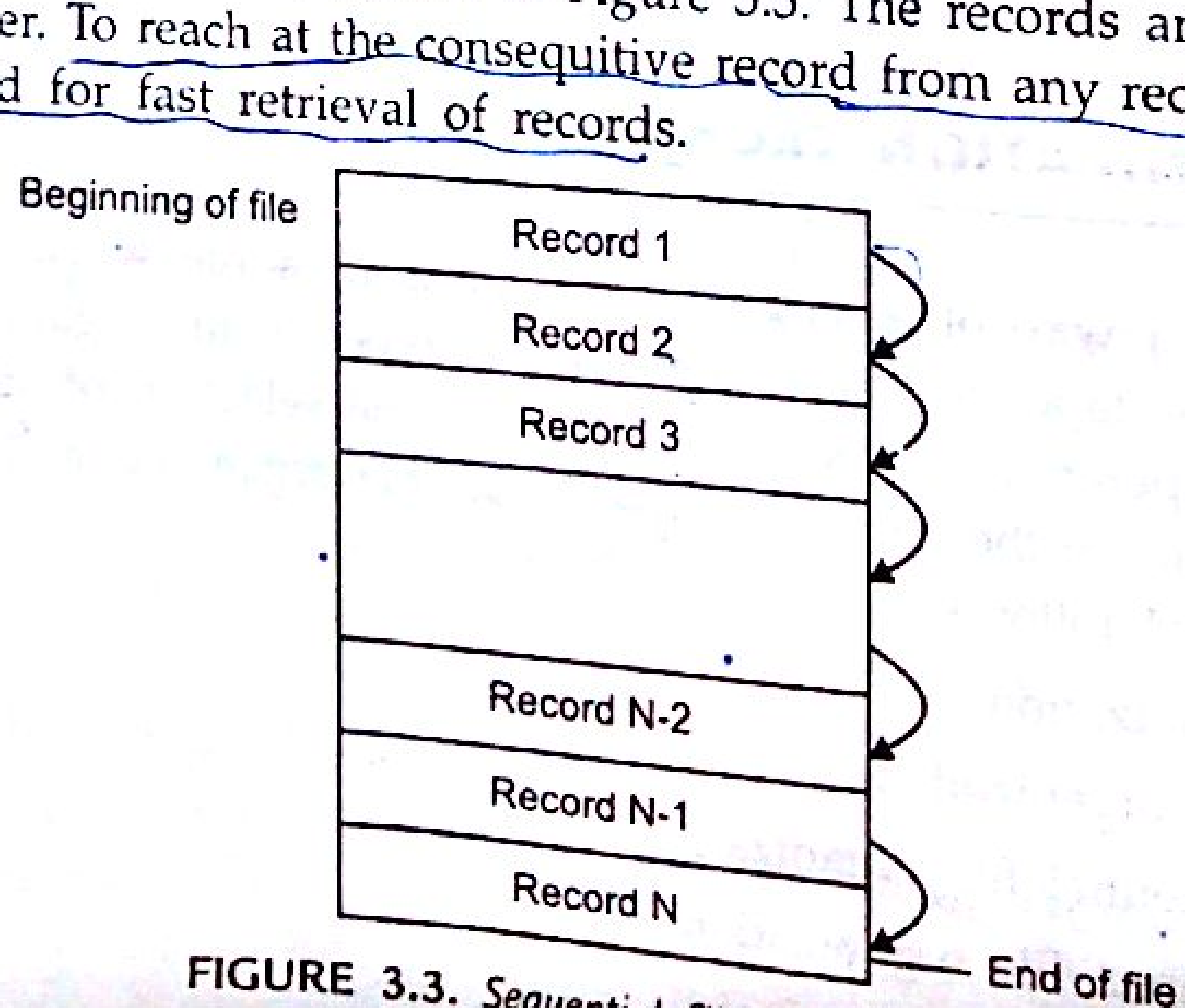


FIGURE 3.3. *Sequential File organization.*

To read any record from the file start searching from the very first record with the help of search key. Sequential file organization gives records in sorted form. This organization is used in small size files.

3.3.2.1 File Operations on Sequentially Organized Files

Various operations that can be performed on sequential files are as follows :

(a) *Creating a sequential file* : To create a new file, first free space is searched in memory. After searching, enough space for file, allocate that space to the new file. Name and physical location of new file is entered in file system directory.

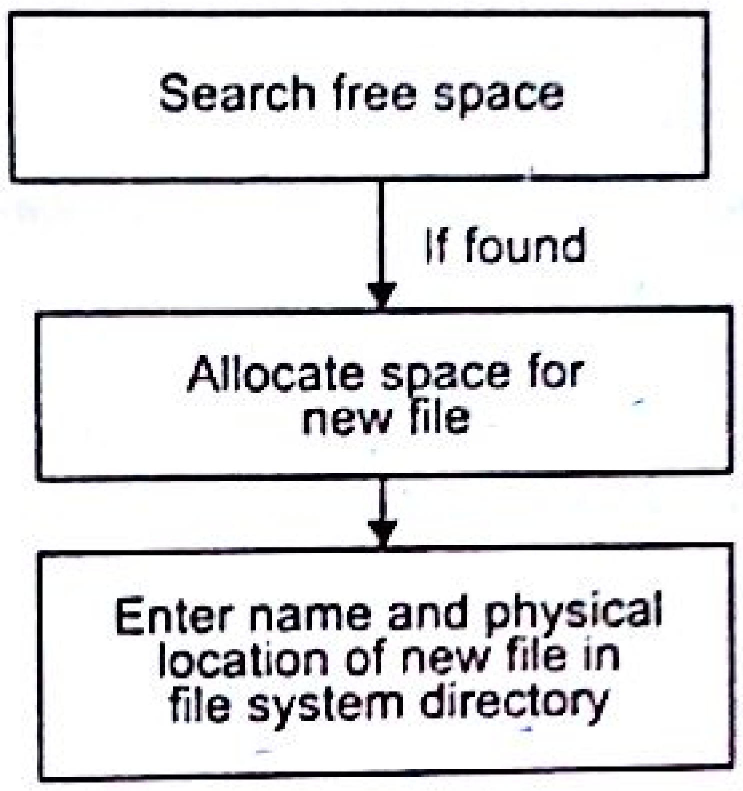

Search free space
If found
Allocate space for new file
Enter name and physical location of new file in file system directory

FIGURE 3.4. *Steps of creating a sequential file.*

(b) *Open an existing file* : To open any file, enter the name of file. This name is searched in file system directory. After matching the name, records in the file are transferred from secondary memory to primary memory. Now, the file is ready to read/write.

(c) *Closing a file* : When task over file is completed then close that file. The memory space allocated for that file in primary memory is deallocated. If file was opened automatically with any programme then it will be closed automatically after ending that programme.

(d) *Reading a sequential file* : To read any record from a sequential file it is necessary to start from beginning of file until record is find or EOF is occured. You cannot go directly to any particular record. Only linear search can be used to search any record.

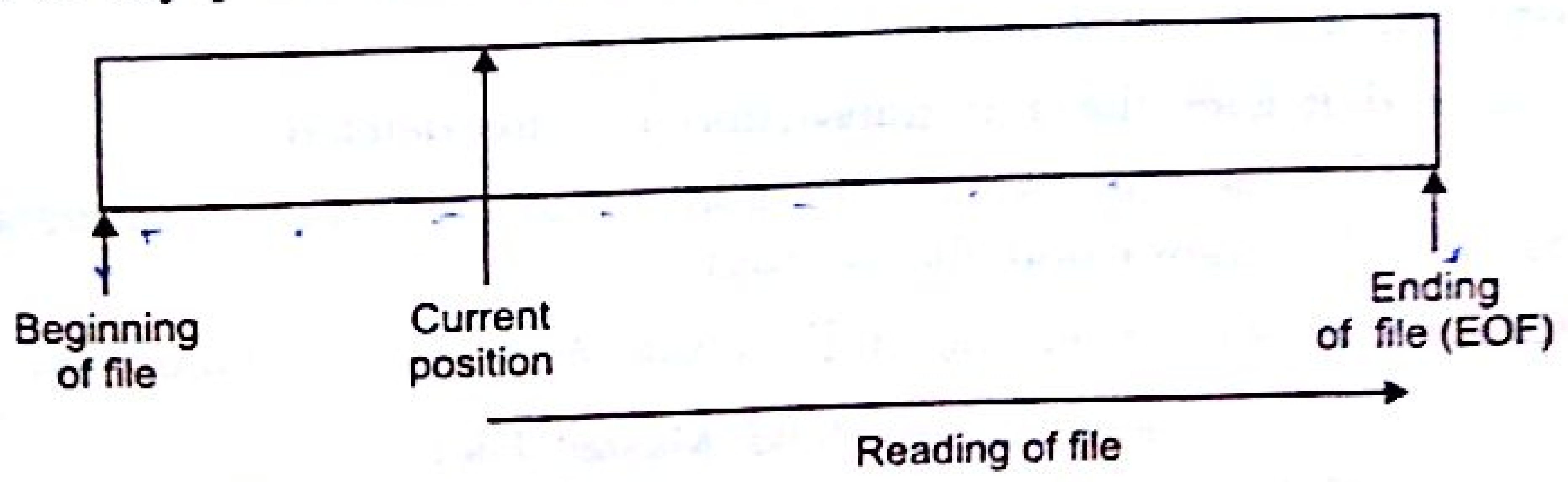


FIGURE 3.5. *Reading of sequential file.*

NOTE *In sequential file system, update, delete, insert and append cannot be performed on any record directly. There is a procedure to perform all these operations.*

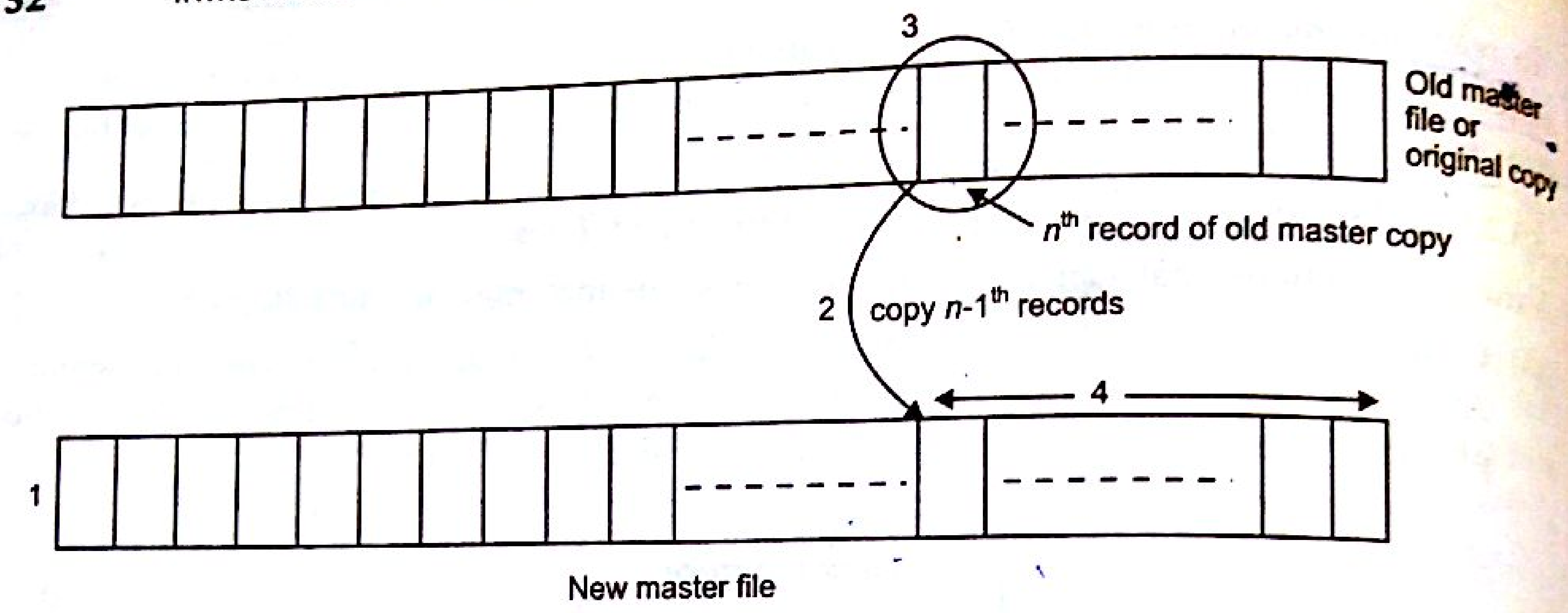

3
Old master file or original copy
n^{th} record of old master copy
2 copy n-1th records
4
1
New master file

FIGURE 3.6. *Updation of sequential file.*

Procedure to Update File :

Step 1. Original file is known as Old Master file. Make a new file which is known as New Master file as shown in Figure 3.6.

Step 2. To update n^{th} record, First copy all the previous n–1 records to new master file

Step 3. Make necessary updations to n^{th} record.

Step 4. Now copy all remaining records including updated n^{th} record to New master file and old master file is deleted.

The major problem is that it is very costly to make New Master file for every single updation. So the modified updation procedure is as follows :

Modification in Procedure : Use a temporary file known as transaction file. The following procedure is followed to make a transaction file and then updating the records.

(*i*) Collect all the requests for updation.

(*ii*) Copy only those records in Transaction file from old Master file on which updation is to be performed.

(*iii*) Update the records.

(*iv*) Now start copying records from Old Master file and Transaction file to New Master file accordingly to the primary key.

(*v*) At last old master file and transaction file are deleted.

(*e*) *Adding or Insertion a new record :* To add or insert a new records in an existing sequential file, the transaction file is used.

Ex. Consider, the old master file emp-records as shown in Figure 3.7.

Emp-records (Old Master File)

Emp-ID	Name	Salary
1	Anil	15,000
7	Sunil	8,000
10	Sahil	10,000

FIGURE 3.7. *Old master file of employee record.*

The records that need to be added are collected in Transaction file as shown in Figure 3.8.

Transaction File

Emp-ID	Name	Salary
5	Sanjay	7,000
12	Amit	9,000

FIGURE 3.8. *Transaction file to add records.*

The EMP-ID is primary key. Now, add records in New Master file as shown in Figure 3.9. If primary key of record in Old Master file is less than record in Transaction file then first copy record of Old Master File into New Master file and vice versa. This process is repeated until EOF is reached in both files.

Emp-records (New Master File)

Emp-ID	Name	Salary
1	Anil	15,000
5	Sanjay	7,000
7	Sunil	8,000
10	Sahil	10,000
12	Amit	9,000

FIGURE 3.9. *New master file for employee records after addition.*

The Old Master file and transaction file are deleted after adding the records.

(*f*) *Deletion of records* : Suppose you want to delete some records from Emp-records (as shown in Figure 3.9) which are no more required. Then emp-records shown in Figure 3.9 is taken as old master file. Records that need to be deleted are stored in Transaction file.

Transaction File

Emp-ID	Name	Salary
1	Anil	15,000
12	Amit	9,000

FIGURE 3.10. *Transaction file to delete records.*

Primary Key of Old Master file is matched with primary key of each record in Transaction file. If primary key is not matched then record is copied from Old Master file to New Master file otherwise discard that record. The process is repeated until EOF of the Old Master file is reached. After this process, Old Master file and transaction file are deleted.

Emp-records (New Master File)

Emp-ID	Name	Salary
5	Sanjay	7,000
7	Sunil	8,000
10	Sahil	10,000

FIGURE 3.11. *New master file for employee records after deletion.*

(g) *Modification of records* : Suppose you want to modify any record of employee. Consider Emp-records in Figure 3.11 as Old Master file. Records that need to be modified are stored in Transaction file with changed values.

Transaction File

Emp-ID	Name	Salary
7	Sunil	12,000

FIGURE 3.12. *Transaction file to modify records.*

Primary key of Old Master file is matched with primary key of each record in Transaction file. If primary key is matched then modified record from Transaction file is copied into New Master file otherwise record from Old Master file is copied into New Master file. The process is repeated until EOF in Old Master file is reached. After this process, Old Master file and transaction file are deleted.

Advantages

1. It is easy to understand.
2. Efficient file system for small size files.
3. Construction and reconstruction of files are much easier in comparison to other file systems.
4. Supports tape media, editors and compilers.
5. It contains sorted records.

Disadvantages

1. Inefficient file system for medium and large size files.
2. Updations and maintenance are not easy.
3. Inefficient use of storage space because of fixed size blocks.
4. Linear search takes more time.
5. Before updations all transactions are stored sequentially.

3.3.3 Index Sequential File Organization

Index sequential file organization is used to overcome the disadvantages of sequential file organization. It also preserves the advantages of sequential access. This organization enables fast searching of records with the use of index. Some basic terms associated with index sequential file organization are as follows :

Block : Block is a unit of storage in which records are saved.

Index : Index is a table with a search key by which block of a record can be find.

Pointer : Pointer is a variable which points from index entry to starting address of block.

To manipulate any record, search key of index is entered to find the starting address of block and then required record is searched sequentially within the block.

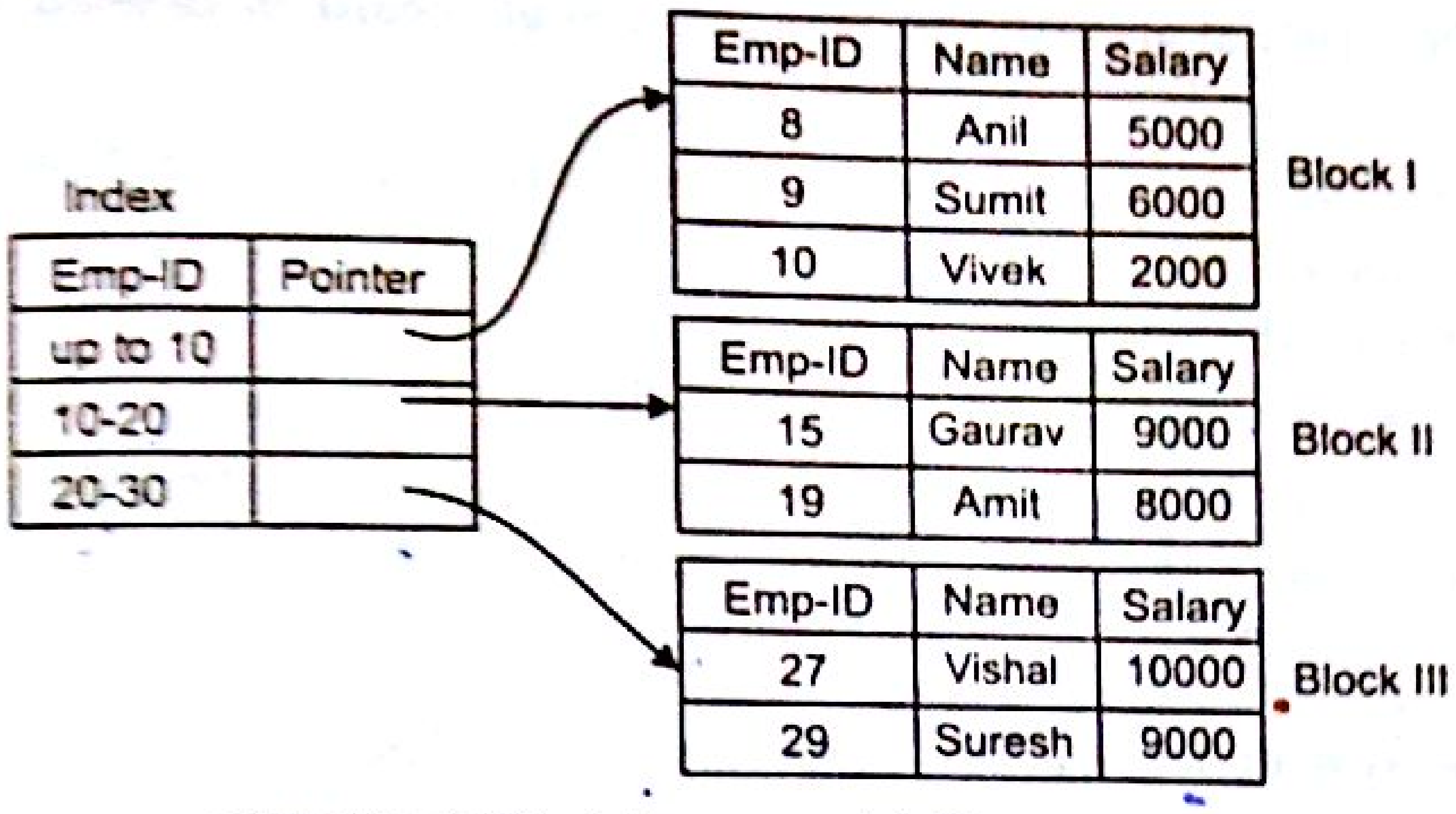


FIGURE 3.13. *Index sequential file organization.*

3.3.3.1 Components of an Indexed Sequential File

(a) *Prime area* : During the creation of index sequential file, records are written in prime area. Records are maintained according to any key. Hence, prime area must be a sequential file.

(b) *Overflow area* : Overflow area is used during addition of new records. There are two types of overflow area :

(i) *Cylinder overflow area* : This area consists of free area on each cylinder which is reserved for overflow records for that particular cylinder.

(ii) *Independent overflow area* : This area is used to store overflow records from anywhere.

(c) *Record characteristics* : Usually fixed length records are used.

(d) *Indexes* : Index is a collection of entries and each entry corresponds to block of storage.

(i) *First level index* : The lowest level index is known as first level index.

(ii) *Higher level index* : Higher level indexing is used when first level index becomes too large.

(iii) *Cylinder index* : The indexes can be made according to the hardware boundaries. Cylinder index entries consists one entry for each cylinder.

(iv) *Master index* : The highest level of index is known as master index.

3.3.3.2 Operations on Index Sequential Files

(a) *Creating a index sequential file* : After allocating free space and make necessary entries in file system directory all records are written into prime area in sequential manner according to key value.

(b) *Opening and closing an existing file* : It is same as sequential file operations.

(c) *Reading records from a index sequential file (or searching any record)* : To search any record enter the key value of record then first search the block of record in index then search record sequentially within the block.

(d) *Modification of records* : To modify a record, first search that record and then modify it. Updated record is stored at same position if it has same key value and size equal to the original record. Otherwise original record is deleted and new record is inserted.

(e) *Deletion of records* : To delete a record, first search that record. Specific codes are used to indicate deleted records. Records consists of flags. Specific code is inserted to the flag of that record that needs to be deleted.

(f) *Insertion of records* : To insert a record, first find the desired block according to key value. Then confirm that record does not exist already. The record is first placed in overflow area and then copied to the desired block.

Advantages

(i) Efficient file system for medium and large size files.

(ii) Easy to update.

(iii) Easy to maintain than direct files.

(iv) Efficient use of storage space.

(v) Searching of records are fast.

(vi) Maintain advantages of sequential file system.

Disadvantages

(i) Inefficient file system for small size files.

(ii) It is expensive method.

(iii) Typical structure than sequential files.

(iv) Indexes need additional storage space.

(v) Performance degradation w.r.t. growth of files.

3.3.4 Hashing

Hashing is a technique by which key field is converted into address of physical location or record by using any function, known as hash function.

$$\begin{tikzcd}\text{Key field} \arrow[r, "\text{Hash function}"] & \text{Address of record}\end{tikzcd}$$

FIGURE 3.14. *Hashing.*

There are various hashing techniques. These are as follows :

(i) **Mid square method** : In mid square hash method, first compute the square of key value and then take the central digits, which is the address of that record. Suppose you have 100 records in one file and the key value is 81. First compute square of 81 which is $(81)^2$ = 6561. After that take central digits of 6561. So, address of record having key value 81 is 56.

(ii) **Folding method** : In this method of hashing, first make partitions of key value and then fold them.

(a) *Boundary folding* : In this technique various parts of key value are aligned by folding them as folding a paper. At last digits are added to get address as shown in Figure 3.15.

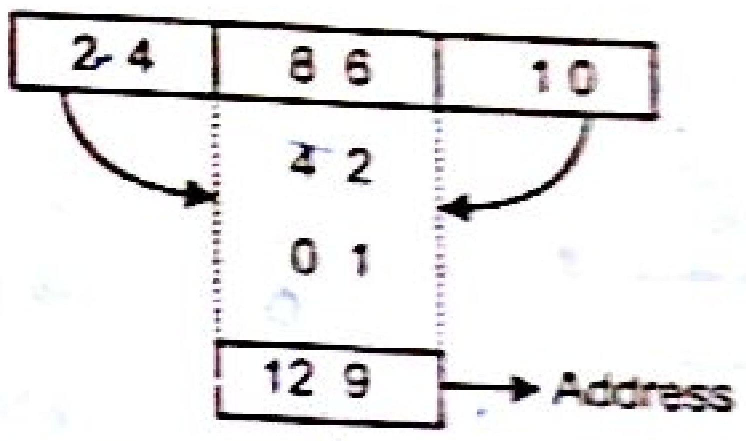


FIGURE 3.15. *Boundary folding*

(*b*) *Shift folding* : In this technique various parts of key value are shifted laterally or you can say that they are simply added as shown in Figure 3.16.

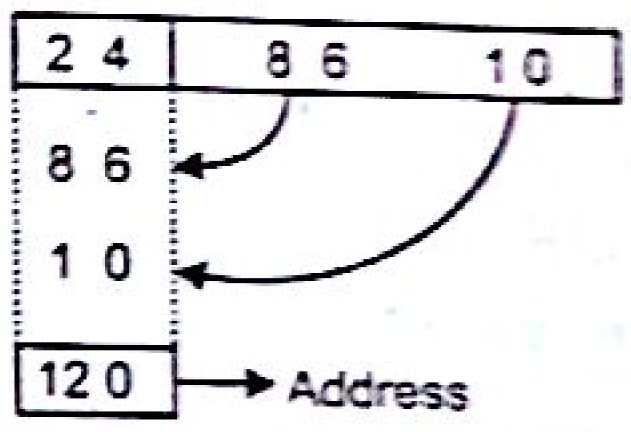


FIGURE 3.16. *Shift folding*

(*iii*) **Division method :** In division method, divide the key value with any number and take quotient as address of that record. Mostly prime number is taken as divisor. Consider a record having key value 550 which is divided by 5 gives 110 as quotient which is taken as address of that record.

(*iv*) **Division-remainder method :** In division-remainder method, divide the key value with any number (Prime number) and take remainder as address of that record. The divisor must be greater than total number of records and small then all key values. Suppose there are 100 records in any file. A record has key value 660 which is divided by 109 gives 6 as remainder which is taken as address of that record.

(*v*) **Radix transformation method (Radix conversion method) :** In radix conversion method, first key is converted into binary string. Then this string is partitioned into small strings. Then these strings are converted into decimal format. Suppose, a record has key value 269 whose binary equivalent is 100001101. Take small strings of 3 bits each. These are 100, 001, 101. Now convert them into decimal format which gives 4, 1, 5. These digits are treated as numbers with radix *r*. Suppose *r* is 5. Then address is

$$4 \times 5^2 + 1 \times 5^1 + 5 \times 5^0 = 110$$

(*vi*) **Polynomial conversion method :** In polynomial conversion method every digit of key value is taken as coefficient of polynomial. After obtaining that polynomial, divide it by another polynomial. The remainder polynomial gives the basis of address of that record.

(*vii*) **Truncation method :** In truncation method, some digits of key value are truncated. They may be left most digits, right most digits or central digits. The remaining digits are taken as address of record. Suppose a record is having key value 543189. Then truncate first two left most digits, which gives 3189 as address for that record.

(*viii*) **Conversion using digital gates :** In this method, convert key value into binary format then apply different operations on these digits and at last convert the result into decimal format as shown in Figure 3.17.

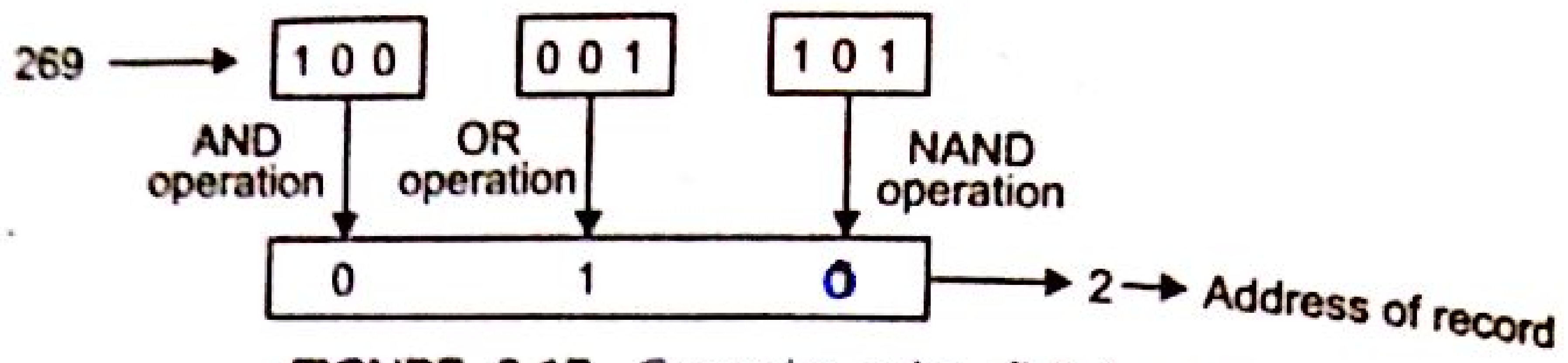


FIGURE 3.17. *Conversion using digital gates.*

3.3.4.1 Collision and Synonyms

The main disadvantage of hashing is collision.

Collision : A collision occurs when a hash function f mapped more than one key value into same physical address.

Synonym : The keys which are mapped into same location are known as Synonyms. Consider the folding method in which key 248610 gives address 129 and key 925005 also gives address 129. This is a collision and key 248610 and 925005 are synonyms.

3.3.4.2 Techniques to Avoid Collisions

There are two main collision resolution technique :

(*i*) Open Addressing

(*ii*) Overflow Chaining.

Both techniques are based on the fact that in case of collision, the synonym key record is write on another location.

(*i*) ***Open Addressing*** : In open addressing, the next free or empty position is searched to store synonym key record. If there is no empty position within a block then empty position in next block is searched. There are various probing methods in open addressing but the simplest one is linear probing.

(*a*) *Linear probing* : Linear probing is the simplest technique which is based on cyclic proble sequence. Suppose, there are total n locations in a block. Suppose a key k is mapped into location d which is not empty then the searching sequence becomes

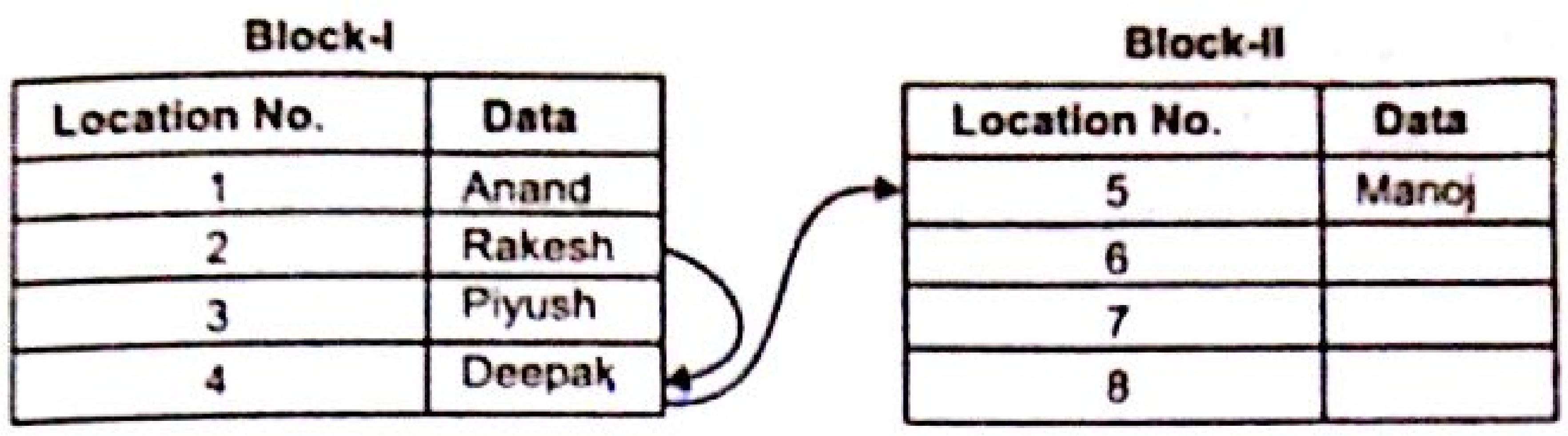
$$d,\ d + 1,\ d + 2,\ d + 3,\ \ldots,\ n,\ 1,\ 2,\ 3,\ \ldots,\ d - 1$$

If there is no free location in that block then, start searching in next consecutive block. It is also known as Consecutive Spill Method.

Consider the example of relation Employee (Dept-ID, Name) where Dept-ID is key field for mapping.

Employee.

Dept-ID	Name
1	Anand
2	Rakesh
3	Piyush
2	Deepak
2	Manoj



Location No.	Data
1	Anand
2	Rakesh
3	Piyush
4	Deepak

Location No.	Data
5	Manoj
6	
7	
8	

FIGURE 3.18. *Linear probing.*

- For employee Anand, Dept-ID is mapped into 1.
- For employee Rakesh, Dept-ID is mapped into 2.
- For employee Piyush, Dept-ID is mapped into 3.
- For employee Deepak, Dept-ID is mapped into 2. A collision occurs because location 2 is not empty. So, searching is started from location 3. Location 4 is empty which is given to Deepak.
- For employee Manoj, again Dept-ID is mapped into 2. A collision occurs because location 2 is not empty. So, searching is started for an empty location. Here Block-I is full so searching is started in consecutive block, Block-II and location 5 is given to Manoj.

Disadvantages

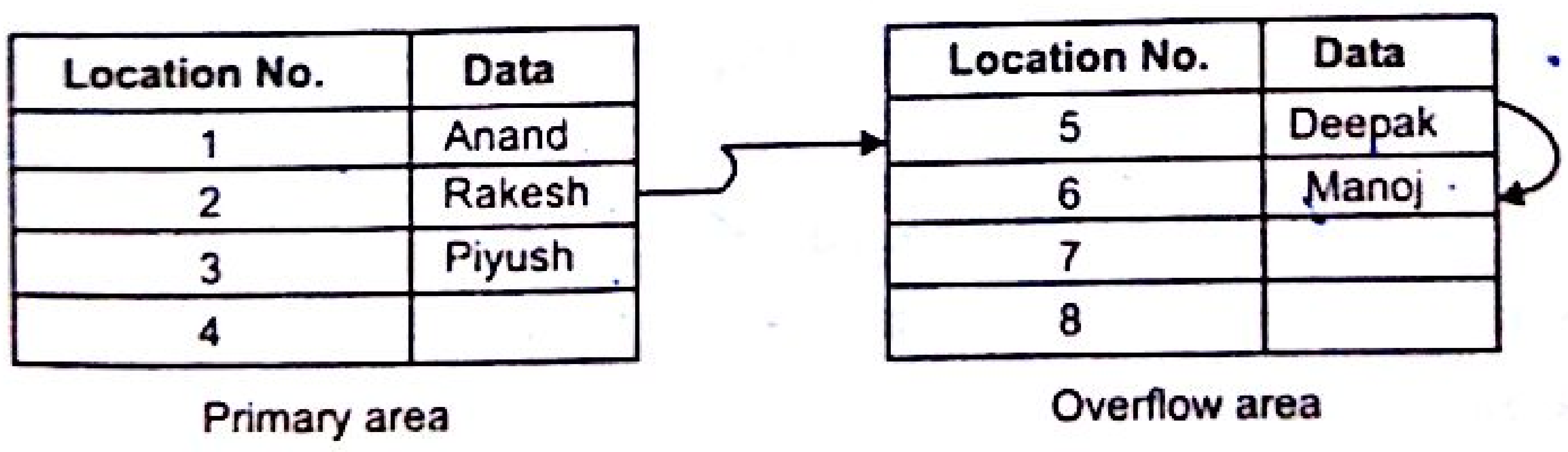
(*i*) Creation of cluster of records (clustering effect).

(*ii*) Searching time for free location is more.

(*b*) *Rehashing* : To avoid clustering effect in linear probing more than one hashing functions can be used. If first hash function results collision then another hash function is used to resolve collision. But this method results extra overhead.

(*ii*) ***Overflow Chaining*** : In this method, there are two areas for storing records, **primary** area in which record is stored in normal conditions, if any collision occurs then synonym key record is stored in **overflow area**.

Consider the example of relation employee shown in Figure 3.18. Now the employee Deepak and Manoj are stored as shown in Figure 3.19.



Location No.	Data
1	Anand
2	Rakesh
3	Piyush
4	

Location No.	Data
5	Deepak
6	Manoj
7	
8	

FIGURE 3.19. *Overflow chaining.*

Advantages

1. Less searching time required.
2. No clustering effect.
3. More efficient than open addressing method.

3.3.5 Direct File Organization

To meet the requirement to access records randomly direct file organization is used. In direct file organization records can be stored anywhere in storage area but can be accessed directly, without any sequential searching. It overcomes the drawbacks of sequential, index sequential and B-trees file organization.

For an efficient organization and direct access of individual record, some mapping or transformation procedure is needed that converts key field of a record into its physical storage location.

Actually, direct file organization depends upon hashing that provides the base of mapping procedure. To overcome the drawbacks of hashing algorithm, collision resolution technique is needed. Devices that support direct access are CD's, Floppy etc.

Direct file organization is also known as Random File Organization. Choice of hashing algorithm and collision resolution technique is crucial point in direct file organization.

3.3.5.1 File Operations on Direct Files

(*a*) *Creating a direct file* : After searching and allocating free space for file, necessary entries in system directory are made. In direct file organization, a hashing algorithm for mapping procedure and any collision resolution technique to avoid collisions during mapping are specified. The key field is also specified (It may not be primary key).

(*b*) Open an existing file and closing a file are same as in other file organizations.

(*c*) *Searching (Reading or retrieving) from direct file* : To read any record from direct file, just enter the key field of that record. With the help of hashing algorithm that key field is mapped into physical location of that record. In case of any collision, collision resolution technique is used.

(*d*) *Updation of records in direct file* :

(*i*) *Adding a new record* : To add a new record in direct file, specify its key field. With the help of mapping procedure and collision resolution technique, get the free address location for that record.

(*ii*) *Deleting record from direct file* : To delete a record, first search that record and after searching, change its status code to deleted or vacant.

(*iii*) *Modify any record* : To modify any record, first search that record, then make the necessary modifications. Then re-write the modified record to the same location.

Advantages

1. Records are not needed to be sorted in order during addition.
2. It gives fastest retrieval of records.
3. It gives efficient use of memory.
4. Operations on direct file are fast so there is no need to collect same type of operations in a file, as in sequential file system.
5. Searching time depends upon mapping procedure not logarithm of the number of search keys as in B-trees.
6. Supports fast storage devices.

Disadvantages

1. Wastage of storage space (Clustering) if hashing algorithm is not chosen properly.
2. It does not support sequential storage devices.
3. Direct file system is complex and hence expensive.
4. Extra overhead due to collision resolution techniques.

3.4 INDEXING

An index is a collection of data entries which is used to locate a record in a file. Index table records consist of two parts, the first part consists of value of prime or non-prime attributes of file record known as **indexing field** and, the **second part** consists of a pointer to the location where the record is physically stored in memory. In general, index table is like the index of a book, that consists of the name of topic and the page number. During searching of a file record, index is searched to locate the record memory address instead of searching a record in secondary memory. On the basis of properties that affect the efficiency of searching, the indexes can be classified into two categories.

1. Ordered indexing
2. Hashed indexing.

3.4.1 Ordered Indexing

In ordered indexing, records of file are stored in some sorted order in physical memory. The values in the index are ordered (sorted) so that binary search can be performed on the index. Ordered indexes can be divided into two categories.

1. Dense indexing
2. Sparse indexing.

3.4.1.1 Dense and Sparse Indexing

- *Dense index* : In dense indexing there is a record in index table for each unique value of the search-key attribute of file and a pointer to the first data record with that value. The other records with the same value of search-key attribute are stored sequentially after the first record. The order of data entries in the index differs from the order of data records as shown in Figure 3.20.

  Advantages of Dense index

  (*i*) It is efficient technique for small and medium sized data files.

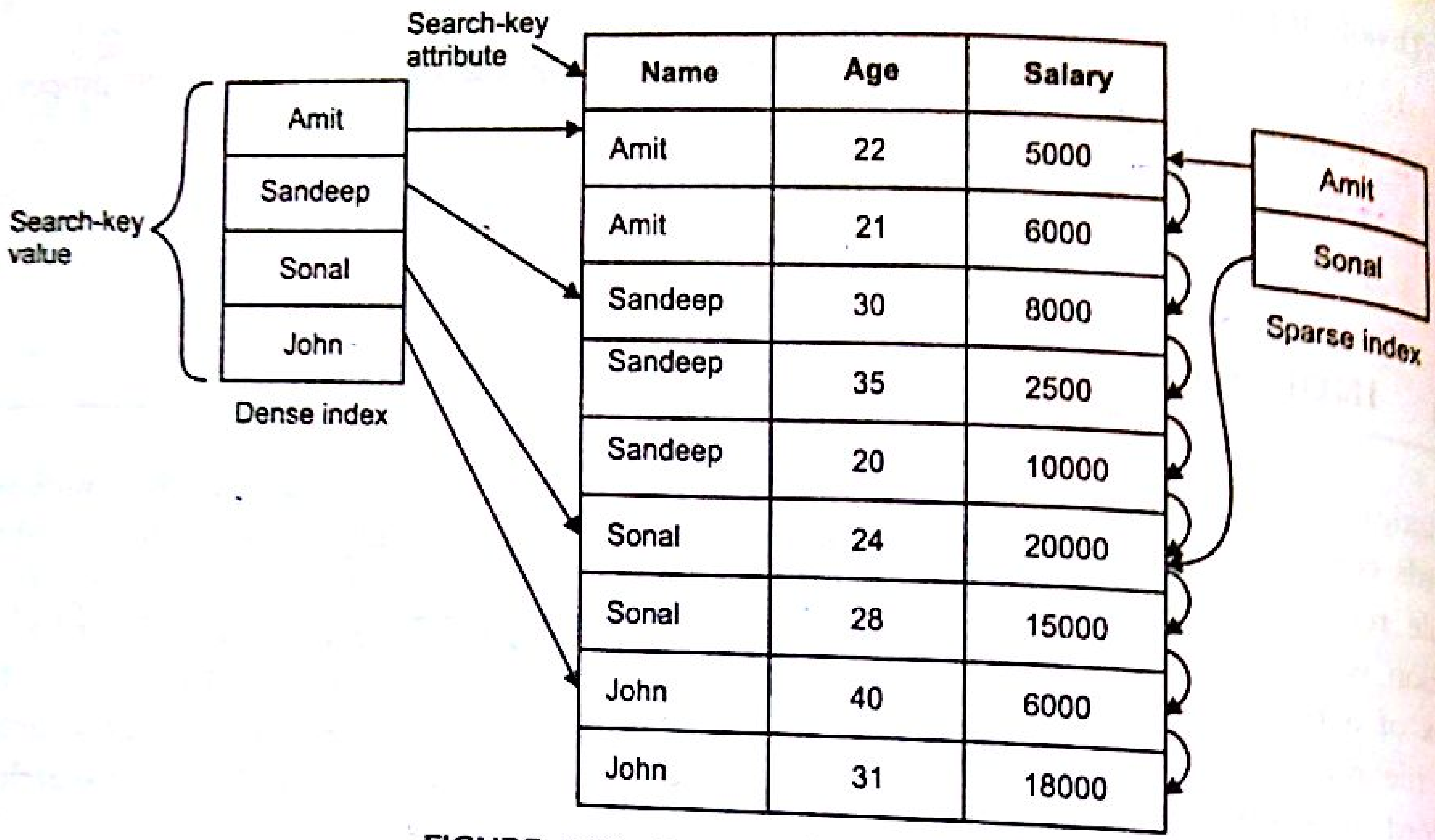
  (*ii*) Searching is comparatively fast and efficient.

  Disadvantages of Dense index

  (*i*) Index table is large and require more memory space.

  (*ii*) Insertion and deletion is comparatively complex.

  (*iii*) In-efficient for large data files.



Name	Age	Salary
Amit	22	5000
Amit	21	6000
Sandeep	30	8000
Sandeep	35	2500
Sandeep	20	10000
Sonal	24	20000
Sonal	28	15000
John	40	6000
John	31	18000

FIGURE 3.20. *Dense and sparse index.*

- *Sparse index* : On contrary, in sparse indexing there are only some records in index table for unique values of the search-key attribute of file and a pointer to the first data record with that value. To search a record in sparse index we search for a value that is less than or equal to value in index for which we are looking. After getting the first record, linear search is performed to retrieve the desired record. There is at most one sparse index since it is not possible to build a sparse index that is not clustered.

Advantages of Sparse index

(*i*) Index table is small and hence save memory space (specially in large files).

(*ii*) Insertion and deletion is comparatively easy.

Disadvantages of Sparse index

(*i*) Searching is comparatively slower, since index table is searched and then linear search is performed inside secondary memory.

3.4.1.2 Clustered and Non-Clustered Indexes

- *Clustered index* : In clustering, index file records are stored physically in order on a non-prime key attribute that does not have a unique value for each record. The non-prime key field is known as clustering field and index in known as clustering index. It is same as dense index. A file can have at most one clustered index as it can be clustered on at most one search key attribute. It may be sparse.
- *Non-Clustered index* : An index that is not clustered is known as non-clustered index. A data file can have more than one non-clustered index.

3.4.1.3 Primary and Secondary Index

- *Primary index* : A primary index consists of all prime-key attributes of a table and a pointer to physical memory address of the record of data file. To retrieve a record on the basis of all primary key attributes, primary index is used for fast searching. A binary search is done on index table and then directly retrieve that record from physical memory. It may be sparse.

Advantages of Primary index

(*i*) Search operation is very fast.

(*ii*) Index table record is usually smaller.

(*iii*) A primary index is guaranteed not to duplicate.

Disadvantages of Primary index

(*i*) There is only one primary index of a table. To search a record on less than all prime-key attributes, linear search is performed on index table.

(*ii*) To create a primary index of an existing table, records should be in some sequential order otherwise database is required to be adjusted.

- *Secondary index* : A secondary index provides a secondary means of accessing a data file. A secondary index may be on a candidate key field or on non-prime key attributes of a table. To retrieve a record on the basis of non-prime key attributes, secondary index can be used for fast searching. Secondary index must be dense with a index entry for every search key value and a pointer to every record in a file.

Advantages of Secondary index

(*i*) Improve search time if search on non-prime key attributes.

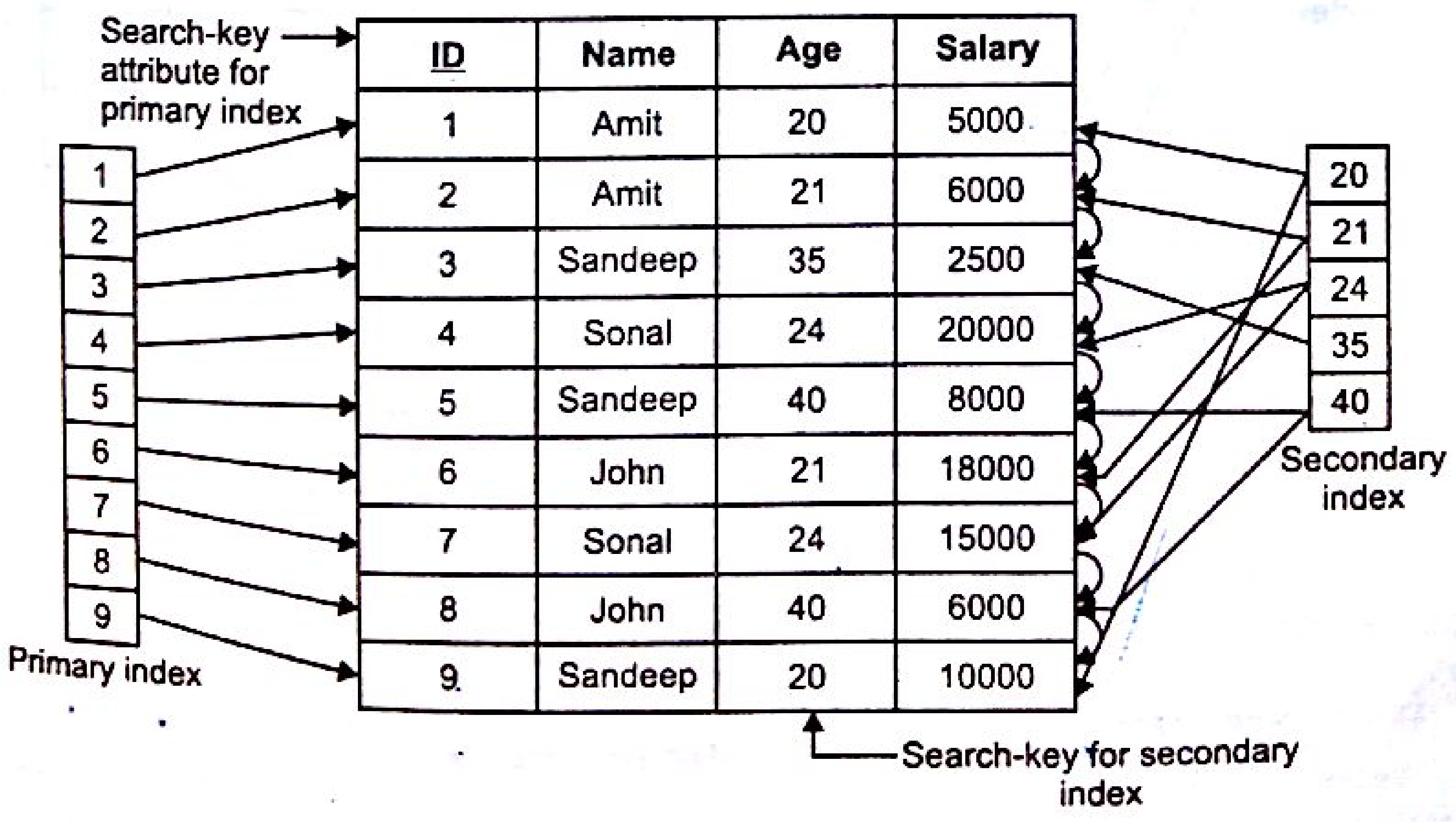
(*ii*) A data file can have more than one secondary index.

Disadvantages of Secondary index

(*i*) A secondary index usually needs more storage space.

(*ii*) Search time is more than primary index.

(*iii*) They impose a significant overhead on the modification of database.


Search-key attribute for primary index
1
2
3
4
5
6
7
8
9
Primary index
20
21
24
35
40
Secondary index
Search-key for secondary index

ID	Name	Age	Salary
1	Amit	20	5000
2	Amit	21	6000
3	Sandeep	35	2500
4	Sonal	24	20000
5	Sandeep	40	8000
6	John	21	18000
7	Sonal	24	15000
8	John	40	6000
9	Sandeep	20	10000

FIGURE 3.21. *Primary and secondary index.*

3.4.1.4 Single and Multilevel Indexes

- *Single level indexes* : A single stage index for a data file is known as single level index. A single level index cannot be divided. It is useful in small and medium size data files. If the file size is bigger, then single level, indexing is not a efficient method. Searching is faster than other indexes for small size data files.
- *Multilevel indexes* : A single index for a large size data file increases the size of index table and increases the search time that results in slower searches. The idea behind multilevel indexes is that, a single level index is divided into multiple levels, which reduces search time.

  In multilevel indexes, the first level index consists of two fields, the first field consists of a value of search key attributes and a second field consists of a pointer to the block (or second level index) which consists that values. and so on.

  To search a record in multilevel index, binary search is used to find the largest of all the small values or equal to the one that needs to be searched. The pointer points to a block of the inner index. After reaching to the desired block, the desired record is searched (in case of two-level indexing) otherwise again the largest of the small values or equal to the one that needs to be searched and so on.

  Benefits of multilevel indexes are they reduce search time significantly for large size data files.

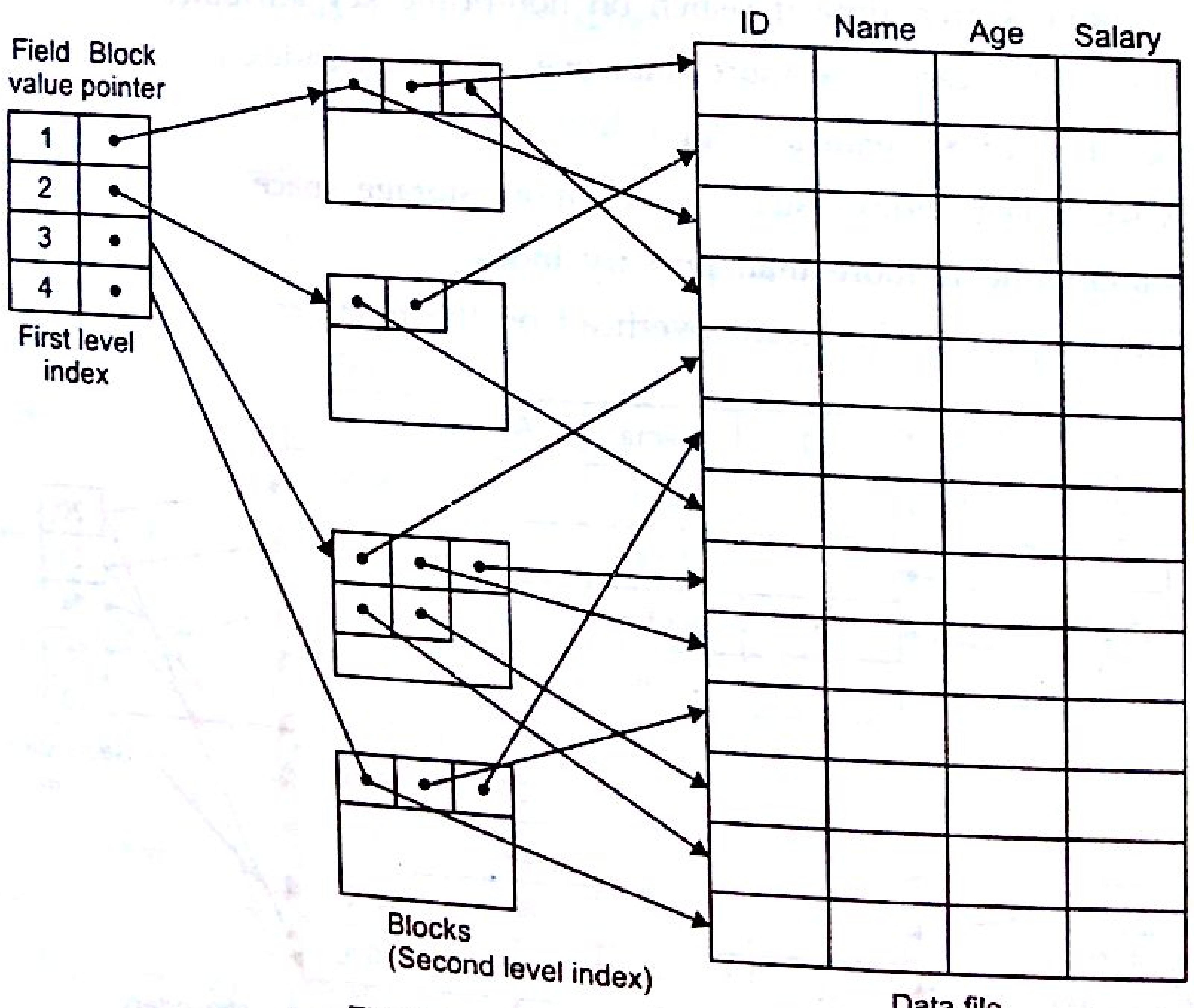


FIGURE 3.22. *Multilevel indexing.*

3.4.2 Hashed Indexing

To overcome the disadvantages of ordered indexing, a hash index can be created for a data file. Hashing allow us to avoid accessing an index structure. A hashed index consists of two fields, the first field consists of search key attribute values and second field consists of pointer to the hash file structure. Hashed indexing is based on values of records being uniformly distributed using a hashed function.

3.5 B-TREE INDEX FILES

In B-Tree index files, tree structure is used. A B-tree of order m is an m-way search tree with the following properties.

1. Each node of the tree, except the root and leaves, has at least $\left\lceil \frac{1}{2}n \right\rceil$ subtrees and no more than n subtrees. It ensures that each node of tree is at least half full.
2. The root of the tree has at least two subtrees, unless it is itself a leaf. It forces the tree to branch early.
3. All leaves of the tree are on the same level. It keeps the tree nearly balanced.

To overcome the performance degradation w.r.t. growth of files in index sequential files B-Tree index files are used. It is a kind of multilevel index file organisation. In tree structure, search starts from the root node and stops at leaf node.

(*i*) **Non-Leaf Node :**

P_1	R_1	K_1	P_2	R_2	K_2	....	P_{m-1}	R_{m-1}	K_{m-1}	P_m

In non-leaf node, there are two pointers. Pointer P_i points to any other node. Pointer R_i points to the block of actual records or storage area of records. K_i represents the key value.

(*ii*) **Leaf Node :**

P_1	K_1	P_2	K_2	....	P_{n-1}	K_{n-1}	P_n

In leaf node, there is only one pointer P_i which points to block of actual records or storage area of records. K_i represents the key value. A B-tree for file employee is shown in Figure 3.23.

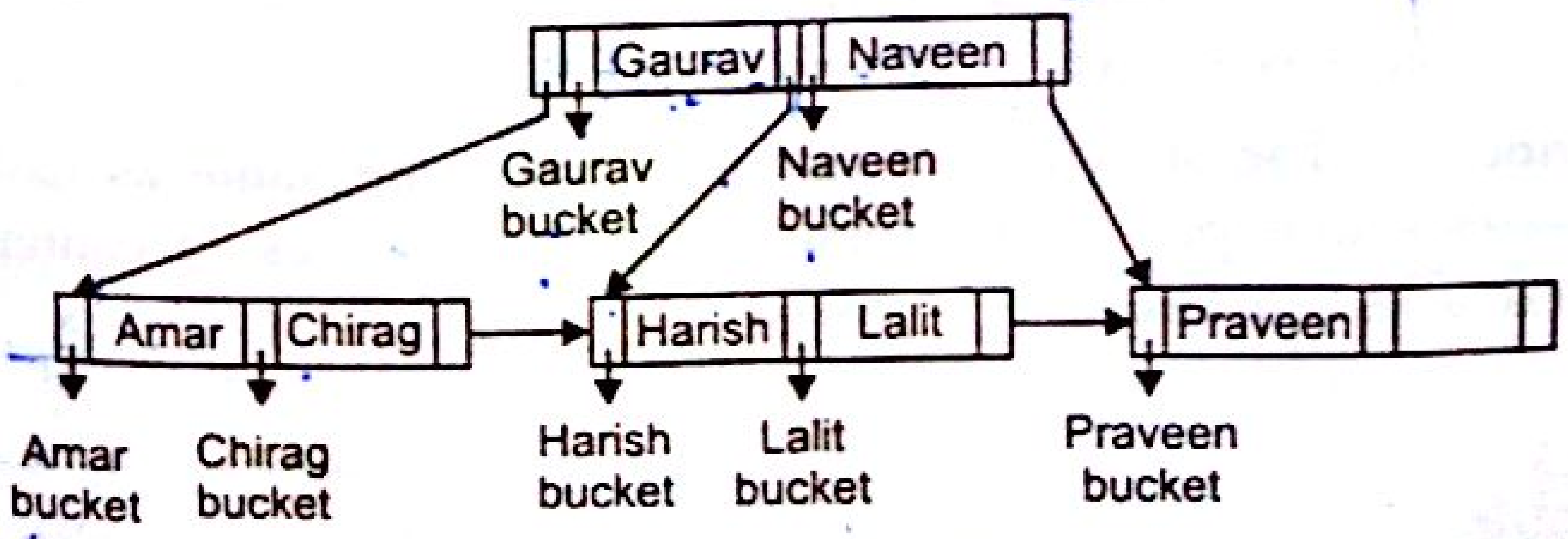


FIGURE 3.23. *B-tree for file Employee.*

(*ii*) Deletion is more complex than B-trees.

(*iii*) Search key values are duplicated which results in wastage of memory space.

3.7 COMPARISON OF DIFFERENT FILE ORGANIZATIONS

S. No.	Sequential	Indexed	Hashed/Direct
1.	Random retrieval on primary key is impractical.	Random retrieval of primary key is moderately fast.	Random retrieval of primary key is very fast.
2.	There is no wasted space for data.	No wasted space for data but there is extra space for index.	Extra space for addition and deletion of records.
3.	Sequential retrieval on primary key is very fast.	Sequential retrieval on primary key is moderately fast.	Sequential retrieval of primary key is impractical.
4.	Multiple key retrieval in sequential file organization is possible.	Multiple key retrieval is very fast with multiple indexes.	Multiple key retrieval is not possible.
5.	Updating of records generally requires rewriting the file.	Updating of records requires maintenance of indexes.	Updating of records is the easiest one.
6.	Addition of new records requires rewriting the file.	Addition of new records is easy and requires maintenance of indexes.	Addition of new records is very easy.
7.	Deletion of records can create wasted space.	Deletion of records is easy if space can be allocated dynamically.	Deletion of records is very easy.

3.8 FACTORS AFFECTING CHOICE OF FILE ORGANIZATION

(*i*) *Access type* : In order to search a record, whether random access or sequential access is required.

(*ii*) *Access time* : The total time taken by file organization to find a particular record.

(*iii*) *File size* : Choice is also dependent upon file size. If file size is large then choose direct access otherwise choose sequential access.

(*iv*) *Overhead* : Each technique has some overhead (it may be space overhead, time overhead etc.). Any technique giving fast access may waste more space then slower techniques.

(*v*) *Updation time* : Time required to add, delete and modify any record also plays important role in efficiency.

(*vi*) *Complexity* : If technique is fast then it is complex and expensive. Whether funds are available to adopt new techniques.

(*vii*) *Availability of hardware* : The hardware that supports file organization. Example tape reader supports only sequential file organization.

EXERCISES

1. What is an index on a file of records ? What is a search key of an index ? Why do we need indexes ?
2. Explain index sequential file organization with example.
3. What do you mean by file organization ? Describe various ways to organize a file.
4. List two advantages and two disadvantages of indexed-sequential file organization.
5. List out advantages and disadvantages of variable-length records and fixed-length records.
6. Discuss disadvantages of using sequential file organization. How do you overcome those problems with direct-indexed file organization.
7. List advantages and disadvantages of direct file access.
8. What do you mean by collision ? Discuss techniques to avoid collision.
9. Define hashing. Discuss various hashing techniques.
10. Define B-trees. How insertion and deletion is made in B-trees ? Explain.
11. What is hashing and index-sequential file organization ? Give an example of an application of each.
12. Compare and contrast sequential and index sequential file organizations.
13. Why is a B^+-tree a better structure than a B-tree for implementation of an indexed sequential file ? Discuss.
14. Explain the structure of an index sequential file organization, with a suitable diagram. Write three differences between index sequential and B-tree file organizations.
15. Why can we have a maximum of one primary or clustering index on a file but several secondary indices ? Your explanation should include an example.
16. What is the difference between primary index and secondary index? What is non-dense indexing ?
17. Construct a B^+-tree for the following set of key values.

    (2, 3, 5, 7, 11, 17, 19, 23, 29, 31)

    Assume that the tree is initially empty and the number of pointer that will fit in one node is four or six.

(iii) Reducing the Null Values in tuples.

(iv) Not allowing the possibility of generating spurious tuples.

(i) **Meaning of the relation attributes :** When the attributes are grouped to form a relation schema, it is assumed that attributes belonging to one relation have certain real-word meaning and a proper interpretation associated with them. This meaning (Semantics) specifies how the attribute values in a tuple relate to one another. The conceptual design should have a clear meaning, if it is done carefully, followed by a systematic mapping into relations and most of the semantics will have been accounted for. Thus the easier it is to explain the meaning of the relation, the better the relation schema design will be.

GUIDELINE 1 : Design a relation schema so that it is easy to explain its meaning. The attributes from multiple entity types and relationship types should not be combined into a single relation. Thus a relation schema that corresponds to one entity type or one relationship type has a straight forward meaning.

(ii) **Redundant information in tuples and update anomalies :** One major goal of schema design is to minimize the storage space needed by the base relations. A significant effect on storage space occurred, when we group attributes into relation schemas. The second major problem, when we use relations as base relations is the problem of update anomalies. There are mainly three types of update anomalies in a relation *i.e.*, Insertion Anomalies, deletion anomalies and modification Anomalies. These anomalies are discussed in the section 6.3 in detail.

GUIDELINE 2 : Design the base relation schema in such a way that no updation anomalies (insertion, deletion and modification) are present in the relations. If present, note them and make sure that the programs that update the database will operate correctly.

(iii) **Null values in tuples :** When many attributes are grouped together into a "fat" relation and many of the attributes do not apply to all tuples in the relation, then there exists many NULL'S in those tuples. This wastes a lot of space. It is also not possible to understand the meaning of the attributes having NULL Values. Another problem occur when specifying the join operation. One major and most important problem with Null's is how to account for them when aggregate operation (*i.e.*, COUNT or SUM) are applied. The Null's can have multiple interpretations, like:·

(a) The attribute does not apply to this rule.

(b) The attribute is unknown for this tuple.

(c) The value is known but not present *i.e.*, cannot be recorded.

GUIDELINE 3 : Try to avoid, placing the attributes in a base relation whose value may usually be NULL. If Null's are unavoidable, make sure that apply in exceptional cases only and majority of the tuples must have some not NULL Value.

(iv) **Generation of spurious tuples :** The Decomposition of a relation schema R into two relations R1 and R2 is undesirable, because if we join them back using NATURAL Join, we do not get the correct original information. The join operation generates spurious tuples that represent the invalid information. More about join in section (6.4.2.6).

GUIDELINE 4 : The relation Schemas are designed in such a way that they can be joined with equality conditions on attributes that are either **primary key** or **foreign key**. This guarantees that no spurious tuples will be generated. Matching attributes in relations that are not (foreign key, primary key) combinations must be avoided, because joining on such attributes may produce spurious tuples.

6.3 FUNCTIONAL DEPENDENCIES

Functional dependencies are the result of interrelationship between attributes or in between tuples in any relation.

Definition : In relation R, X and Y are the two subsets of the set of attributes, Y is said to be functionally dependent on X if a given value of X (all attributes in X) uniquely determines the value of Y (all attributes in Y).

It is denoted by X → Y(Y depends upon X).

Determinant : Here X is known as determinant of functional dependency.

Consider the example of Employee relation :

Employee

EID	Name	Salary
1	Aditya	15,000
2	Manoj	16,000
3	Sandeep	9,000
4	Vikas	10,000
5	Manoj	9,000

FIGURE 6.1. *Employee relation.*

In Employee relation, EID is primary key. Suppose you want to know the name and salary of any employee. If you have EID of that employee, then you can easily find information of that employee. So, Name and Salary attributes depend upon EID attribute.

Here, X is (EID) and Y is (Name, Salary)

X (EID) : Y (Name, Salary)

The determinant is EID

Suppose X has value 5 then Y has value (Manoj, 9,000)

6.3.1 Functional Dependency Chart

It is the graphical representation of function dependencies among attributes in any relation. The following four steps are followed to draw FD chart.

1. Find out the primary key attributes.
2. Make a rectangle and write all primary key attributes inside it.
3. Write all non-prime key attributes outside the rectangle.
4. Use arrows to show functional dependency among attributes.

Consider the example of Figure 6.1. Its functional dependency chart is shown in Figure 6.2.

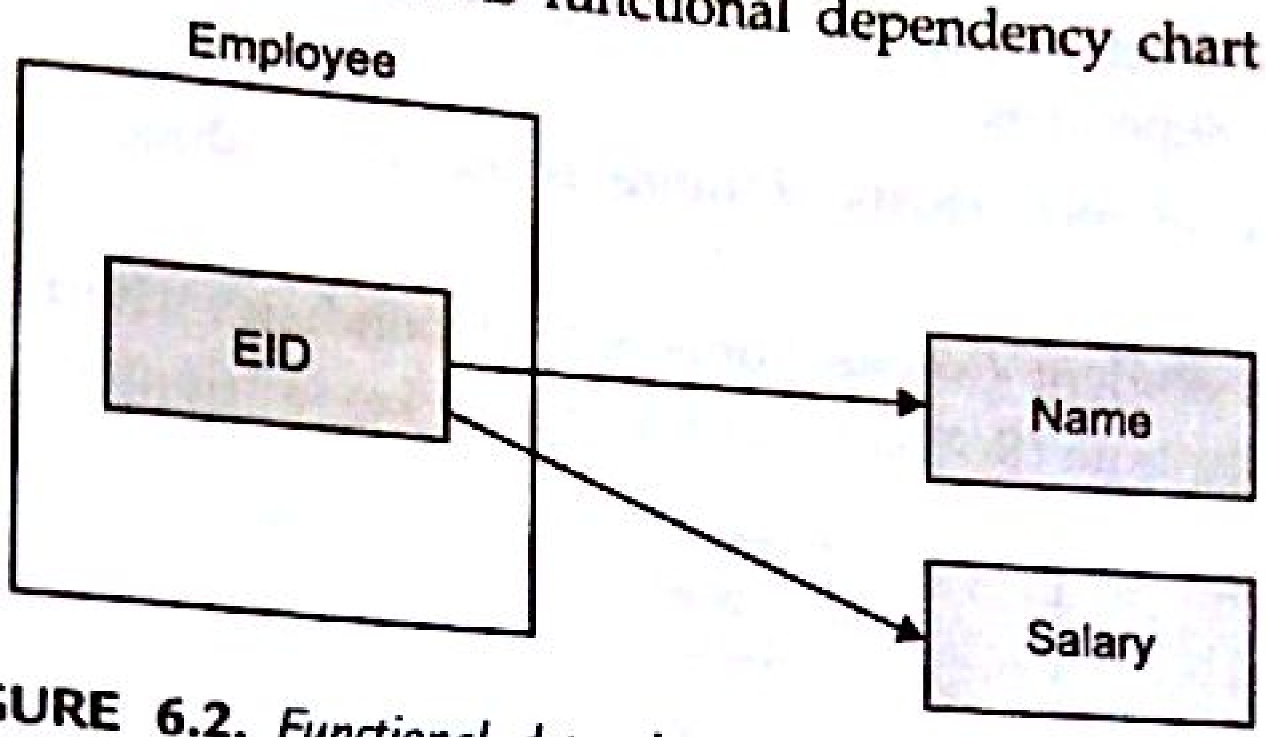


FIGURE 6.2. *Functional dependency chart of Employee relation.*

It is easier to remember all dependencies by making FD charts.

6.3.2 Types of Functional Dependencies

There are four major types of FD's.

1. **Partial Dependency and Fully Functional Dependency**

- *Partial dependency* : Suppose you have more than one attributes in primary key. Let A be the non-prime key attribute. If A is not dependent upon all prime key attributes then partial dependency exists.
- *Fully functional dependency* : Let A be the non-prime key attribute and value of A is dependent upon all prime key attributes. Then A is said to be fully functional dependent. Consider a relation student having prime key attributes (RollNo and Game) and non-prime key attributes (Grade, Name and Fee).

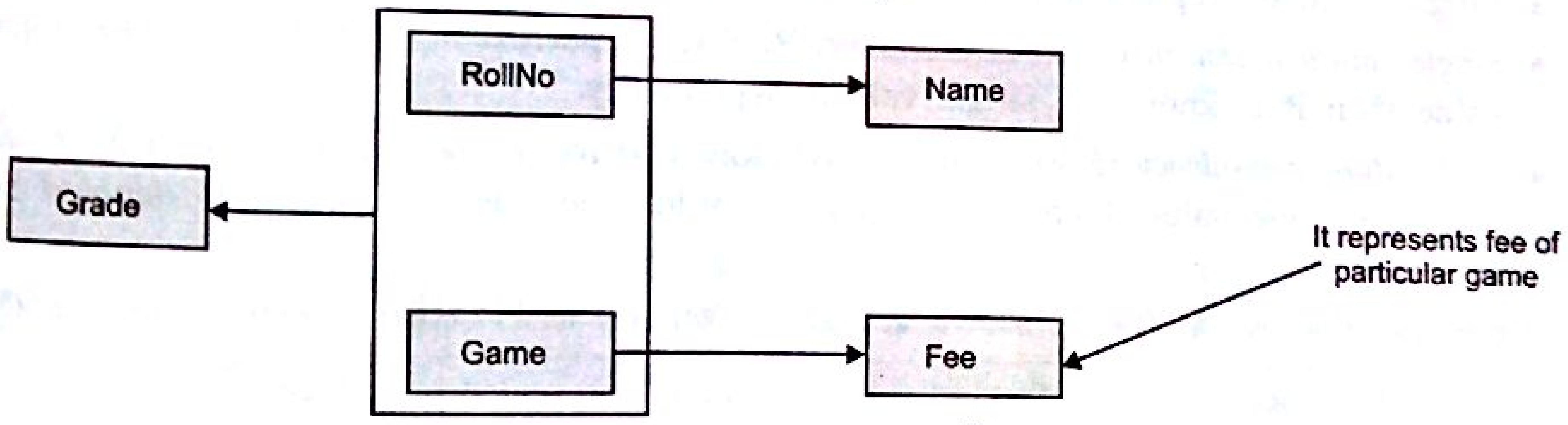


FIGURE 6.3. *Functional dependency chart showing Partial and Fully functional dependency of student relation.*

As shown in Figure 6.3, Name and Fee are **partially dependent** because you can find the name of student by his RollNo. and fee of any game by name of the game.

Grade is **fully functionally dependent** because you can find the grade of any student in a particular game if you know RollNo. and Game of that student. Partial dependency is due to more than one prime key attribute.

2. **Transitive Dependency and Non Transitive Dependency**

- *Transitive dependency* : Transitive dependency is due to dependency between non-prime key attributes. Suppose in a relation R, $X \rightarrow Y$ (Y depends upon X), $Y \rightarrow Z$ (Z depends upon Y), then $X \rightarrow Z$ (Z depends upon X). Therefore, Z is said to be transitively dependent upon X.

- *Non-Transitive dependency* : Any functional dependency which is not transitive is known as Non-Transitive dependency.

  Non-Transitive dependency exists if there is no dependency between non-prime key attributes.

  Consider a relation student (whose Functional dependency chart is shown in Figure 6.4) having prime key attribute (RollNo) and non-prime key attributes (Name, Semester, Hostel).

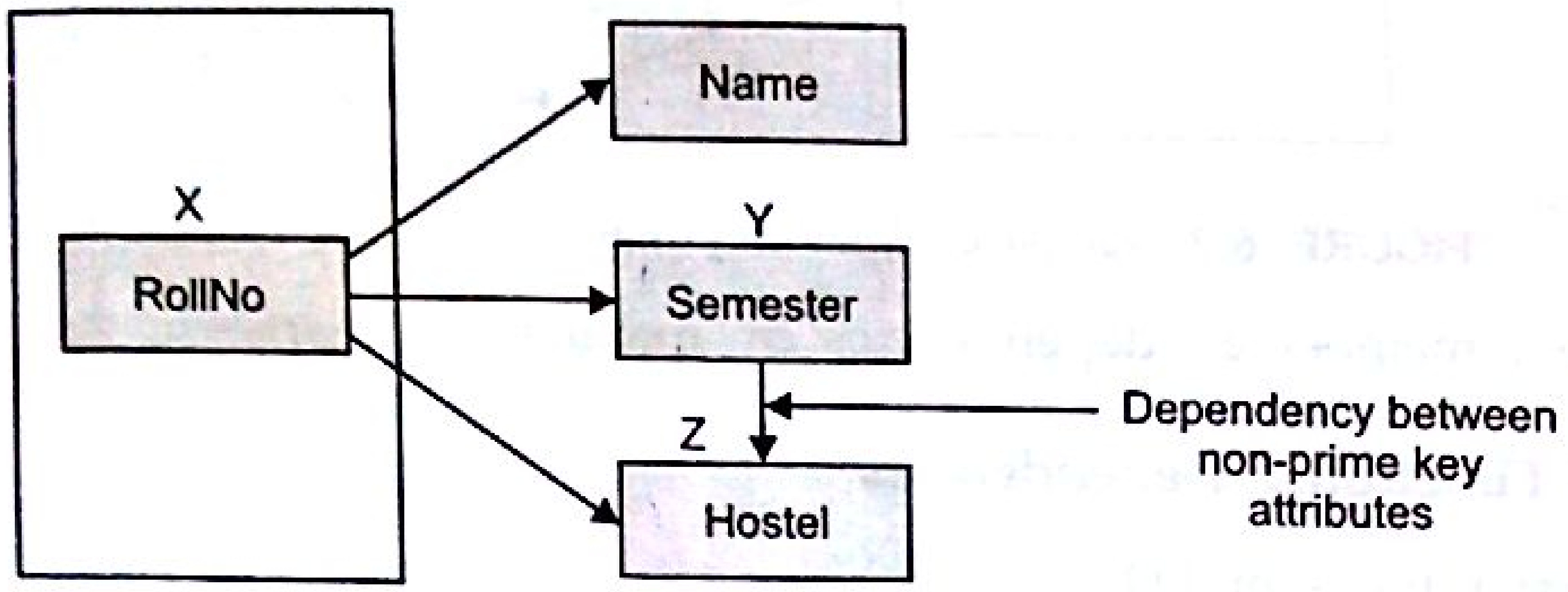


FIGURE 6.4. *Functional dependency chart showing transitive and non-transitive dependency on relation student.*

For each semester there is different hostel

Here Hostel is transitively dependent upon RollNo. Semester of any student can be find by his RollNo. Hostel can be find out by semester of student.

Here, Name is non-transitively dependent upon RollNo.

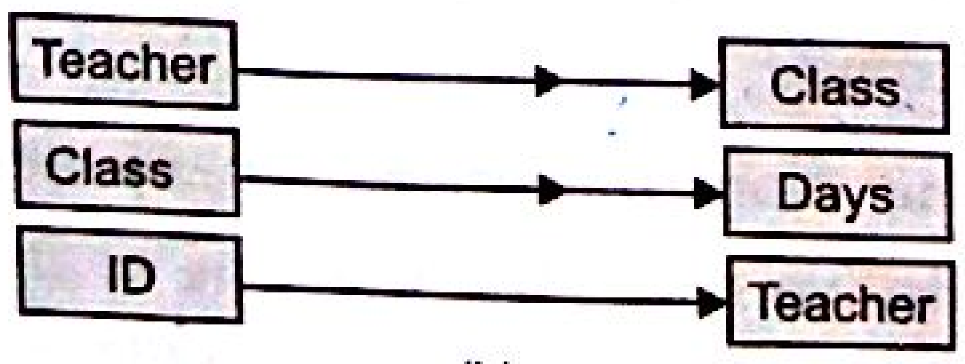
3. Single Valued Dependency and Multivalued Dependency

- *Single valued dependency* : In any relation R, if for a particular value of X, Y has single value then it is known as single valued dependency.
- *Multivalued dependency (MVD)* : In any relation R, if for a particular value of X, Y has more then one value, then it is known as multivalued dependency. It is denoted by $X \twoheadrightarrow Y$.

Consider the relation Teacher shown in Figure 6.5(*a*) and its FD chart shown in Figure 6.5(*b*).

ID	Teacher	Class	Days
1Z	Sam	Computer	1
2Z	John	Computer	6
1Z	Sam	Electronics	3
2Z	John	Mechanical	5
3Z	Nick	Mechanical	2

(a)



(b)

FIGURE 6.5. *Functional dependency chart showing single valued and multivalued dependency on relation teacher.*

There is MVD between **Teacher** and **Class** because a teacher can take more than one class. There is another MVD between Class and Days because a class can be on more than one day.

There is **single valued dependency** between ID and Teacher because each teacher has a unique ID.

4. Trival Dependency and Non-Trival Dependency

- *Trival FD* In any relation R, $X \rightarrow Y$ is trival if $Y \subset X$ (Y is the subset of X).
- *Non-Trival FD* In any relation R, $X \rightarrow Y$ is non trival if $Y \not\subset X$ (Y is not the subset of X)

Consider the Supplier-Product relation shown in Figure 6.6.

S#	City	P#	Quantity
1	Delhi	1P	100
2	Rohtak	8P	200
3	Pune	3P	50
4	Pune	5P	75
5	Rohtak	1P	99
6	Mau	5P	105

FIGURE 6.6. *Supplier-Product relation.*

Here, (S#, P#) → S# is trival FD

S# : Supplier ID

P# : Product ID

6.4 ANOMALIES IN RELATIONAL DATABASE

There are various anomalies or pitfalls in relational database. Various dependencies in relational database cause these anomalies.

Anomalies Anomalies refer to the undesirable results because of modification of data. Consider the relation Employee with attributes EID, Name, Salary, Dept.No, Dept.Name, as shown in Figure 6.7. The various anomalies are as follows :

Employee

EID	Name	Salary	Dept.No	Dept.Name
1	Shiv Goyal	10,000	2	Accounts
2	Amit Chopra	9,000	2	Accounts
3	Deepak Gupta	11,000	1	Sales
4	Sandeep Sharma	8,500	5	Marketing
5	Vikas Malik	7,000	5	Marketing
6	Gaurav Jain	15,000	2	Accounts
7	Lalit Parmar	14,000	5	Marketing
			10	Finance
8	Vishal Bamel	10,500	2	Accounts

Cannot be inserted

FIGURE 6.7. *Employee relation with anomalous data.*

6.4.1 Insertion Anomaly

Suppose you want to add new information in any relation but cannot enter that data because of some constraints. This is known as Insertion anomaly. In relation Employee, you cannot add new department Finance unless there is an employee in Finance department. Addition of this information violates Entity Integrity Rule 1. (Primary Key cannot be NULL). In other words, when you depend on any other information to add new information then it leads to insertion anomaly.

6.4.2 Deletion Anomaly

The deletion anomaly occurs when you try to delete any existing information from any relation and this causes deletion of any other undesirable information.

In relation Employee, if you try to delete tuple containg Deepak this leads to the deletion of department "Sales" completely (there is only one employee in sales department).

6.4.3 Updation Anomaly

The updation anomaly occurs when you try to update any existing information in any relation and this causes inconsistency of data.

In relation Employee, if you change the Dept.No. of department Accounts.

NOTE *We can update only one tuple at a time.*

This will cause inconsistency if you update Dept.No. of single employee only otherwise you have to search all employees working in Accounts department and update them individually.

6.5 NORMALIZATION

Normalization is a process by which we can decompose or divide any relation into more than one relation to remove anomalies in relational database.

It is a step by step process and each step is known as **Normal Form.**

Normalization is a reversible process.

6.5.1 Properties of Normalization

1. Remove different anomalies.
2. Decomposition must be lossless.
3. Preserve necessary dependency.
4. Reduce redundancy.

6.5.2 Various Normal Forms

The different normal forms are as follows. Each of which has its importance and are more desirable than the previous one.

6.5.2.1 First Normal Form (1NF)

A relation is in first normal form if domain of each attribute contains only atomic values. It means atomicity must be present in relation.

Consider the relation Employee as shown in Figure 6.7. It is not in first normal form because attribute Name is not atomic. So, divide it into two attributes First Name and Last Name as shown in Figure 6.8.

Employee

EID	First Name	Second Name	Salary	Dept. No.	Dept. Name
1	Shivi	Goyal	10,000	2	Accounts
2	Amit	Chopra	9,000	2	Accounts
3	Deepak	Gupta	11,000	1	Sales
4	Sandeep	Sharma	8,500	5	Marketing
5	Vikas	Malik	7,000	5	Marketing
6	Gaurav	Jain	15,000	2	Accounts
7	Lalit	Parmar	14,000	5	Marketing
8	Vishal	Bamel	10,500	2	Accounts

FIGURE 6.8. *Employee relation in 1NF.*

Now, relation Employee is in 1NF.

Anomalies in First Normal Form : First Normal form deals only with atomicity. Anomalies described earlier are also applicable here.

6.5.2.2 Second Normal Form (2NF)

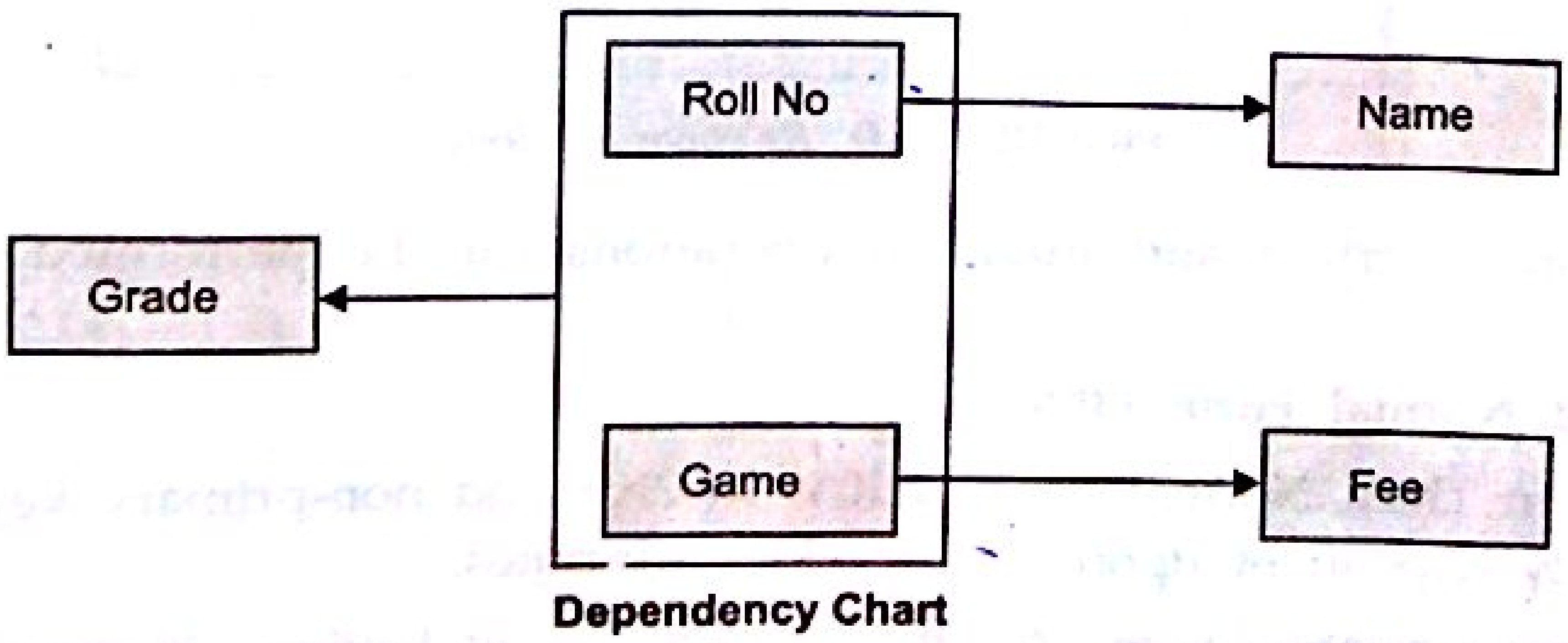
A relation is in second normal form if it is in 1NF and all non-primary key attributes must be fully functionally dependent upon primary key attributes.

Consider the relation Student as shown in Figure 6.9(*a*) :

Student

RollNo.	Game	Name	Fee	Grade
1	Cricket	Amit	200	A
2	Badminton	Dheeraj	150	B
3	Cricket	Lalit	200	A
4	Badminton	Parul	150	C
5	Hockey	Jack	100	A
6	Cricket	John	200	C

(a)



(b)

FIGURE 6.9. *Student relation with anomalies.*

The Primary Key is (RollNo., Game). Each Student can participate in more than one game. Relation Student is in 1NF but still contains anomalies.

1. *Deletion anomaly* : Suppose you want to delete student Jack. Here you loose information about game Hockey because he is the only player participated in hockey.
2. *Insertion anomaly* : Suppose you want to add a new game Basket Ball having no student participated in it. You cannot add this information unless there is a player for it.
3. *Updation anomaly* : Suppose you want to change Fee of Cricket. Here, you have to search all the students participated in cricket and update fee individually otherwise it produces inconsistency.

The solution of this problem is to separate Partial dependencies and Fully functional dependencies. So, divide Student relation into three relations Student(RollNo., Name), Games (Game, Fee) and Performance(RollNo., Game, Grade) as shown in Figure 6.10.

Student

RollNo.	Name
1	Amit
2	Dheeraj
3	Lalit
4	Parul
5	Jack
6	John

Games

Game	Fee
Cricket	200
Badminton	150
Hockey	100

Performance

RollNo.	Game	Grade
1	Cricket	A
2	Badminton	B
3	Cricket	A
4	Badminton	C
5	Hockey	A
6	Cricket	C

FIGURE 6.10. *Relations in 2NF.*

Now, Deletion, Insertion and updation operations can be performed without causing inconsistency.

6.5.2.3 Third Normal Form (3NF)

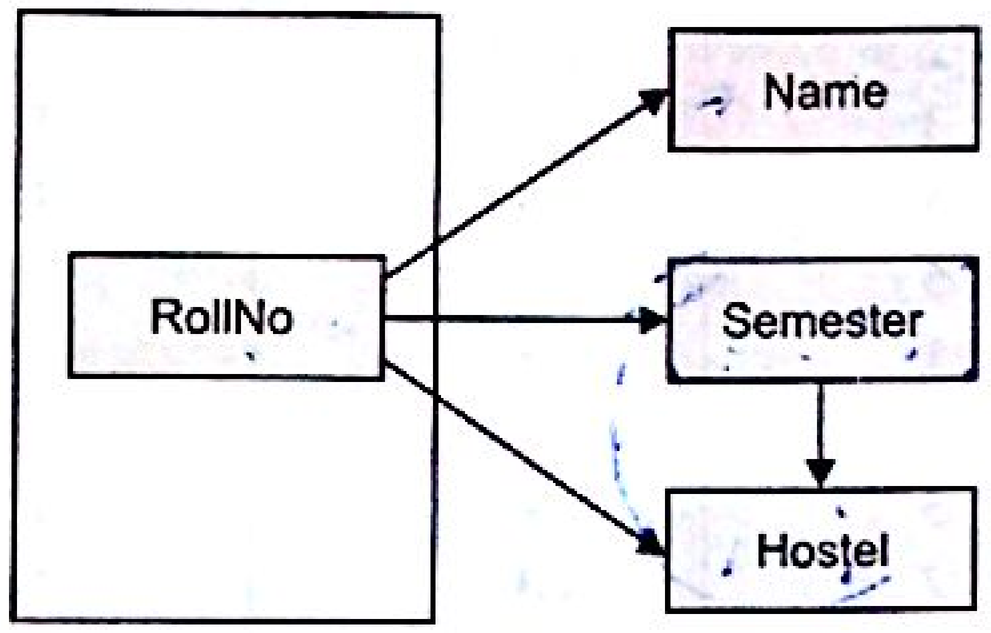
A relation is in Third Normal Form if it is in 2NF and non-primary key attributes must be non-transitively dependent upon primary key attributes.

In other words a relation is in 3NF if it is in 2NF and having no transitive dependency. Consider the relation Student as shown in Figure 6.11(*a*).

Student

RollNo.	Name	Semester	Hostel
1	Lalit	1	H1
2	Gaurav	2	H2
3	Vishal	1	H1
4	Neha	4	H4
5	John	3	H3

(a)



(b)

FIGURE 6.11. *Student relation.*

The Primary Key is (RollNo.). The condition is different Hostel is allotted for different semester. Student relation is in 2NF but still contains anomalies.

1. *Deletion anomaly :* If you want to delete student Gaurav. You loose information about Hostel H2 because he is the only student staying in hostel H2.
2. *Insertion anomaly :* If you want to add a new Hostel H8 and this is not allotted to any student. You cannot add this information.
3. *Updation anomaly :* If you want to change hostel of all students of first semester. You have to search all the students of first semester and update them individually otherwise it causes inconsistency.

The solution of this problem is to divide relation Student into two relations Student(RollNo. Name, Semester) and Hostels(Semester, Hostel) as shown in Figure 6.12.

Student

RollNo.	Name	Semester
1	Lalit	1
2	Gaurav	2
3	Vishal	1
4	Neha	4
5	John	3

Hostels

Semester	Hostel
1	H1
2	H2
3	H3
4	H4

FIGURE 6.12. *Relations in 3NF.*

Now, deletion, insertion and updation operations can be performed without causing inconsistency.

6.5.2.4 Boyce Codd Normal Form (BCNF)

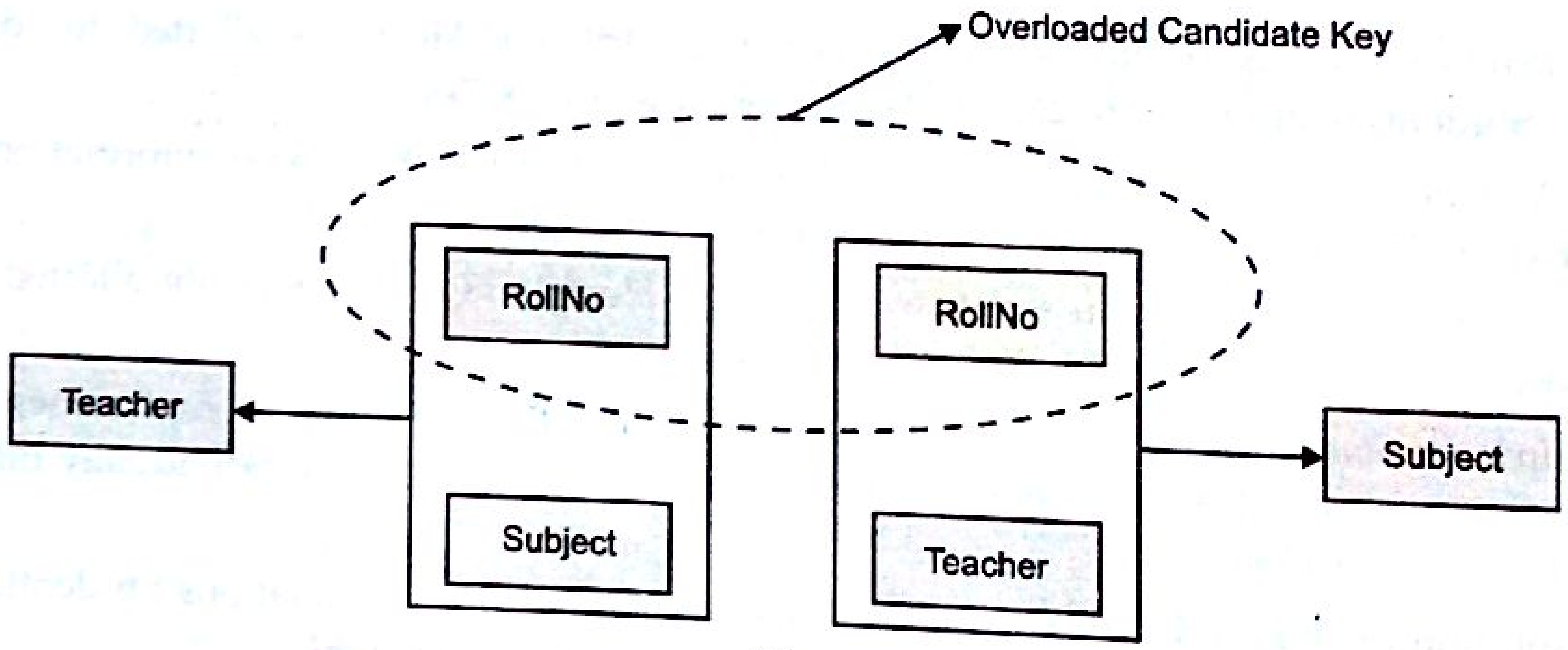
BCNF is a strict format of 3NF. A relation is in BCNF if and only if all determinants are candidate keys. BCNF deals with multiple candidate keys.

Relations in 3NF also contains anomalies. Consider the relation Student as shown in Figure 6.13(*a*).

Student

RollNo.	Subject	Teacher
1	C	T1
2	C++	T2
3	C	T1
4	Java	T3
5	Java	T3
1	Oracle	T5
6	Oracle	T5
3	C++	T2
7	VB	T4
8	Oracle	T6

(a)



(b)

FIGURE 6.13. *Student relation.*

Assumptions :

— Student can have more than 1 subject.

— A Teacher can teach only 1 subject.

— A subject can be taught by more than 1 teacher

There are two candidate keys (RollNo., Subject) and (RollNo., Teacher)

Relation Student is in 3NF but still contain anomalies.

1. *Deletion anomaly* : If you delete student whose RollNo. is 7. You will also loose information that Teacher T4 is teaching the subject VB.
2. *Insertion anomaly* : If you want to add a new Subject VC++, you cannot do that until a student chooses subject VC++ and a teacher teaches subject VC++.

3. *Updation anomaly* : Suppose you want to change Teacher for Subject C. You have to search all the students having subject C and update each record individually otherwise it causes inconsistency.

In relation, Student, candidate key is overloaded. You can find teacher by RollNo. and Subject. You can also find subject by RollNo. and Teacher. Here RollNo. is overloaded. You can also find subject by teacher.

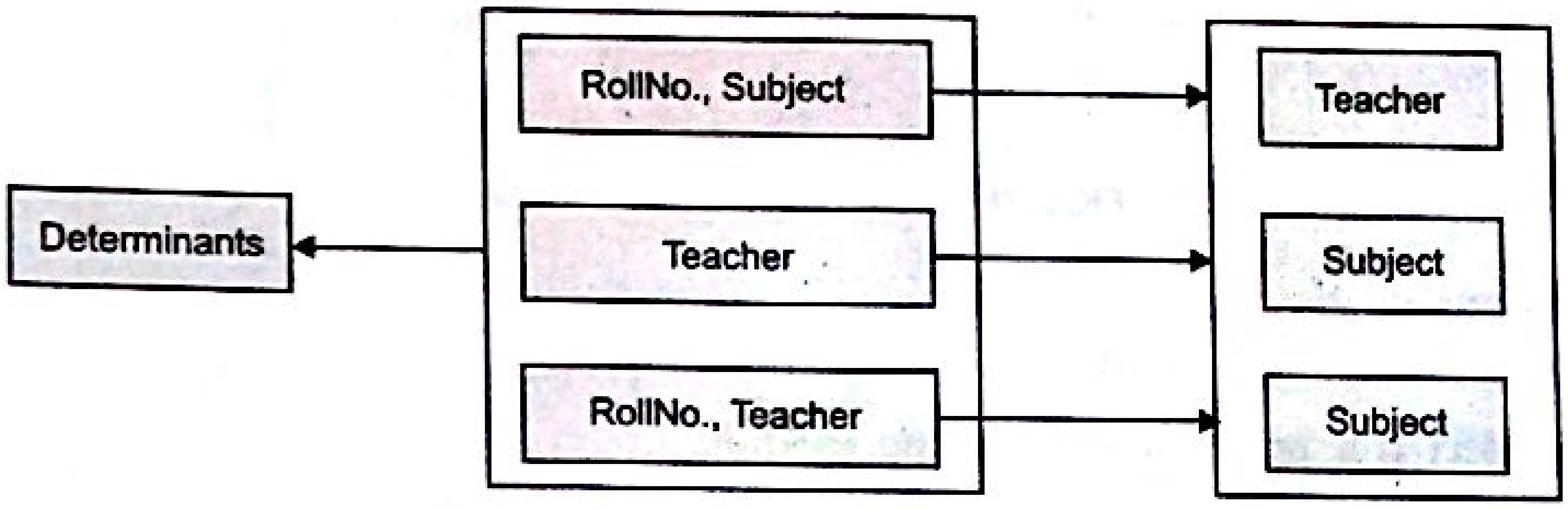


FIGURE 6.14. *Determinants in relation student.*

The solution of this problem is to divide relation Student in two relations Stu-Teac and Teac-Sub as shown in Figure 6.15.

Stu-Teac

RollNo.	Teacher
1	T1
2	T2
3	T1
4	T3
5	T3
1	T5
6	T5
3	T2
7	T4
8	T6

Stu - Teac (RollNo., Teacher)
Candidate Key (RollNo., Teacher)

Teac-Sub

Teacheer	Subject
T1	C
T2	C++
T3	Java
T4	VB
T5	Oracle
T6	Oracle

Teac-Sub
Candidate Key (Teacher)

FIGURE 6.15. *Relations in BCNF.*

In this solution all determinants are candidate keys.

6.5.2.5 Fourth Normal Form (4NF)

A relation is in 4NF if it is in BCNF and for all Multi Valued Functional Dependencies (MVD) of the form X →→ Y either X → Y is a trival MVD or X is a super key of relation.

Relations in BCNF also contains anomalies. Consider the relation project-Work as shown in Figure 6.16.

Project-Work

Programmer	Project	Module
P1	1	M1
P2	1	M2
P3	2	M1
P1	3	M1
P4	3	M2

FIGURE 6.16. *Project-Work relation.*

Assumptions :

— A Programmer can work on any number of projects.

— A project can have more then one module.

Relation Project-work is in BCNF but still contains anomalies.

1. *Deletion anomaly* : If you delete project 2. You will loose information about Programmer P3.
2. *Insertion anomaly* : If you want to add a new project 4. You cannot add this project until it is assigned to any programmer.
3. *Updation anomaly* : If you want to change name of project 1. Then you have to search all the programmers having project 1 and update them individually otherwise it causes inconsistency.

Dependencies in Relation Project-work are

$$\text{Programmer} \rightarrow\!\!\rightarrow \text{Project}$$
$$\text{Project} \rightarrow\!\!\rightarrow \text{Module}$$

The solution of this problem is to divide relation Project-Work into two relations Prog-Prj (Programmer, Project) and Prj-Module (Project, Module) as shown in Figure 6.17.

Proj-Prj

Programmer	Project
P1	1
P2	1
P3	2
P1	3
P4	3

Here Programmer is the super key

Prj-Module

Project	Module
1	M1
1	M2
2	M1
3	M1
3	M2

Here Project is the super key

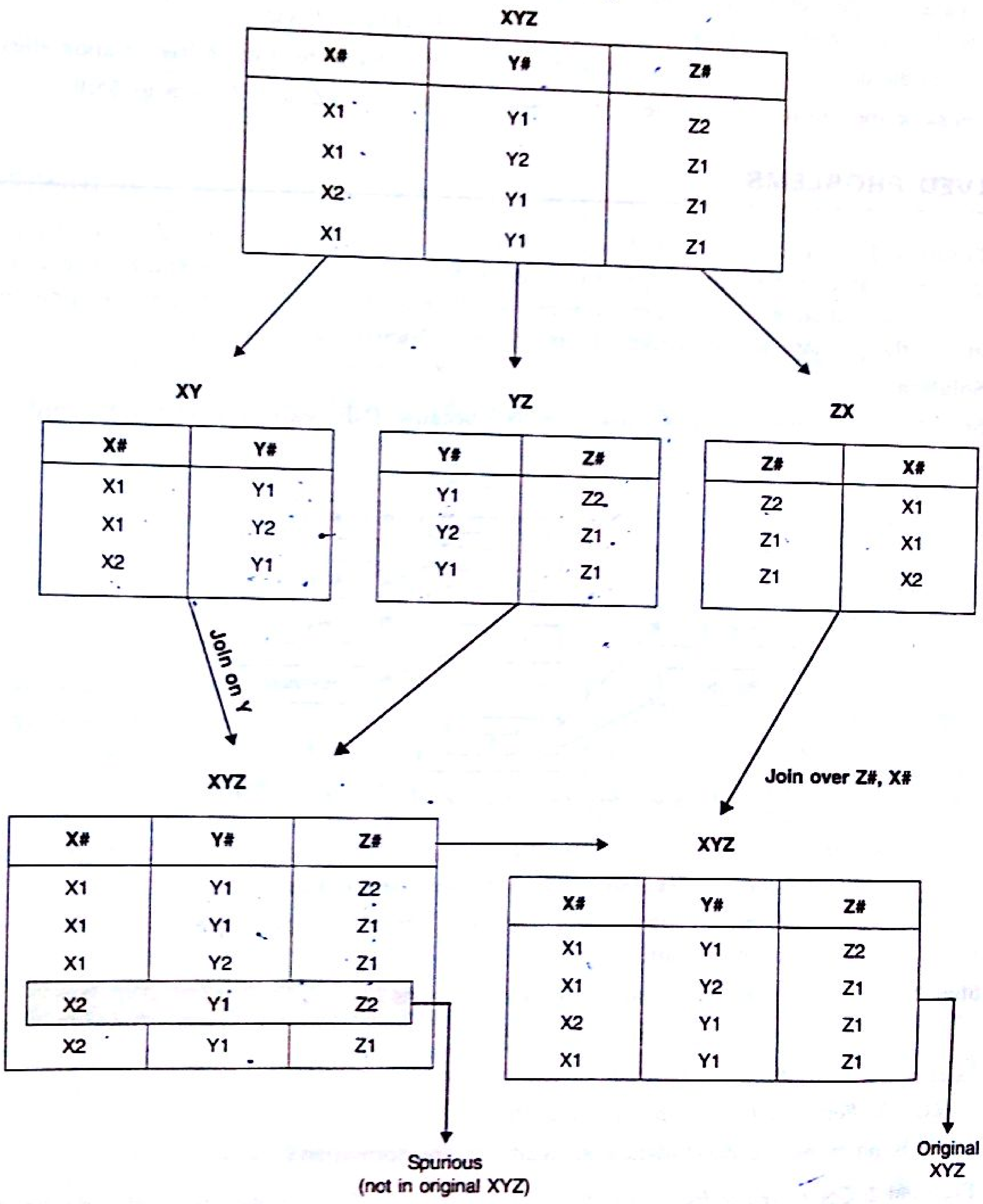
FIGURE 6.17. *Relations are in 4NF.*

6.5.2.6 Project Join Normal Form (5NF) and Join Dependency

Join Dependency : Let R be a given relation upto 4NF and it decompose (Projected or divided) into $\{R_1, R_2, R_3..., R_n\}$. The relation R satisfy the join dependency * $\{R_1, R_2, R_3, ... R_n\}$ if and only if joining of R_1 to R_n = R.

Consider the Relation XYZ with attributes X# (Customer_ID), Y# (Account_ID) and Z# (Branch_ID) as shown in Figure 6.18.

First, decompose XYZ into three relations XY, YZ and ZX.



XYZ

X#	Y#	Z#
X1	Y1	Z2
X1	Y2	Z1
X2	Y1	Z1
X1	Y1	Z1

XY

X#	Y#
X1	Y1
X1	Y2
X2	Y1

YZ

Y#	Z#
Y1	Z2
Y2	Z1
Y1	Z1

ZX

Z#	X#
Z2	X1
Z1	X1
Z1	X2

XYZ

X#	Y#	Z#
X1	Y1	Z2
X1	Y1	Z1
X1	Y2	Z1
X2	Y1	Z2
X2	Y1	Z1

XYZ

X#	Y#	Z#
X1	Y1	Z2
X1	Y2	Z1
X2	Y1	Z1
X1	Y1	Z1

FIGURE 6.18. *Relations XY, YZ, ZX are in 5NF and relations XYZ satisfy the Join dependency.*

When joined, these three relations give original form of XYZ. So relation XYZ satisfy the join dependency.

Project Join Normal Form (5NF) : Let R is a relation and D is a set of all dependencies (Multivalued dependency, Functional dependency, Join dependency etc.) The relation R is in 5NF w.r.t. D if for every Join Dependency, Join Dependency is trivial.

5NF is the ultimate Normal form. A relation in 5 NF is guaranteed to be free of anomalies.

Consider the example in Figure 6.18, where relations XY, YZ and ZX are in 5NF.

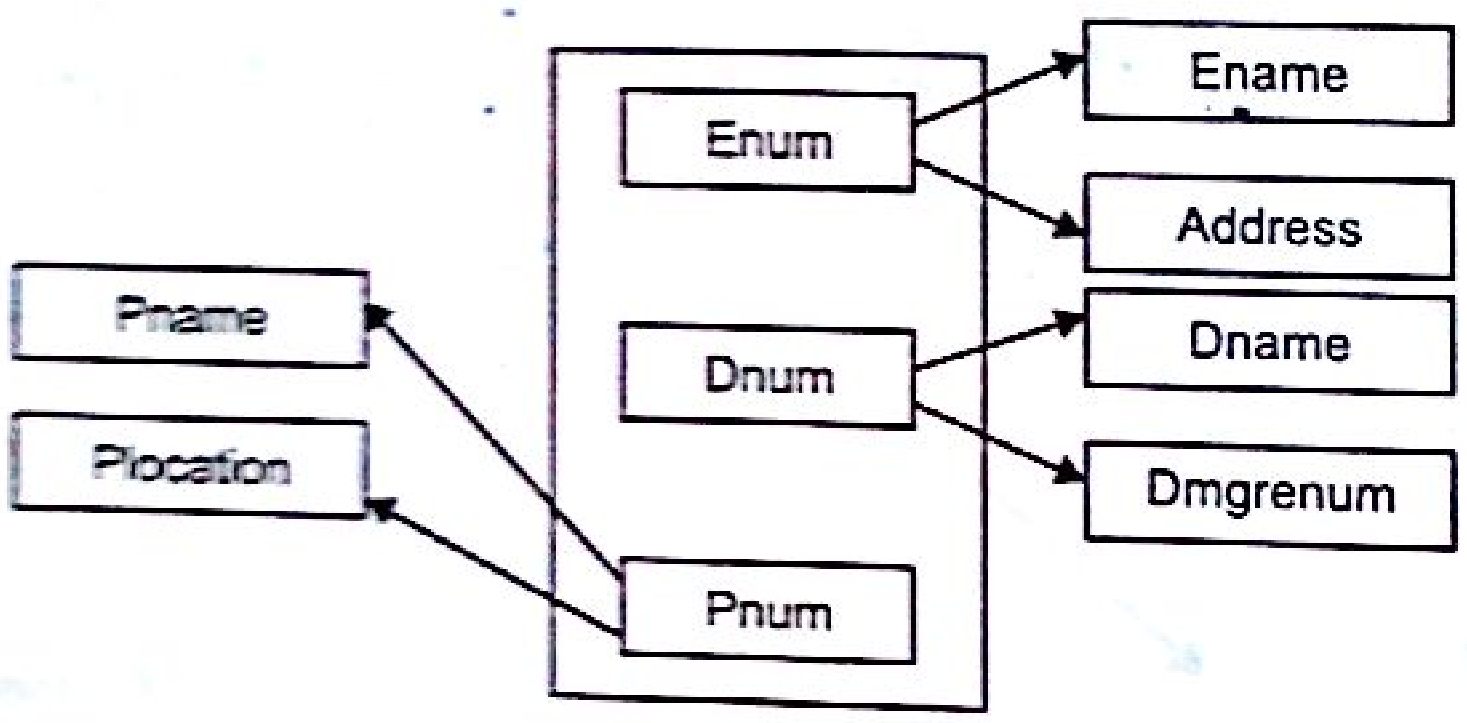
SOLVED PROBLEMS

Problem 1. Normalize the following relation EDP (ename, enum, address, dnum, dmagenum, dname, pname, pnum, plocation), given functional dependencies enum → {ename, address, dnum, dname, dmgrenum}, dnum → {dname, dmagrenum}, pnum → {pname, plocation}, the primary key is {enum, dnum, pnum}. Discuss each step.

Solution.

Step 1. We assume that it is already in INF because INF deals with atomicity only.

Now dependency chart of EDP is



Functional dependency chart for relation EDP.

A relation is in 2NF if it is in 1NF and all non-primary key attributes must be fully functionally dependent upon primary key attributes.

So, In EDP none of the non-primary key attributes is fully functional dependent upon primary key attributes.

Step 2. So, decompose EDP into three new relations

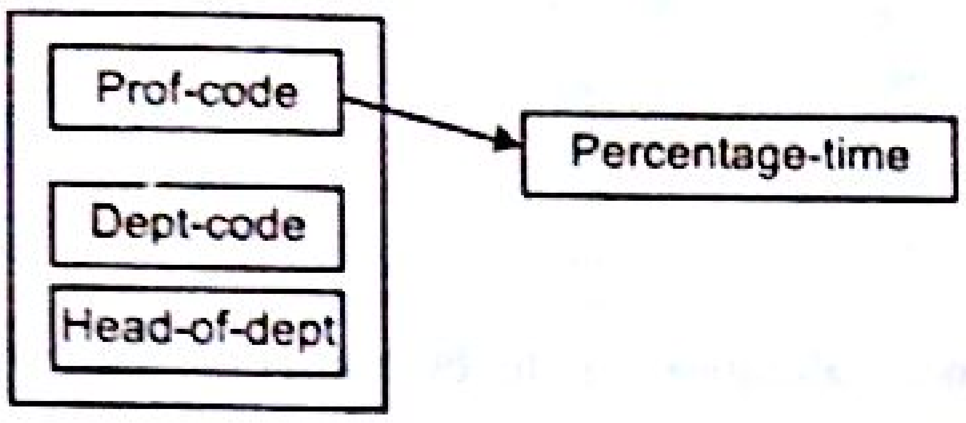
1. Employee (<u>enum</u>, ename, address)
2. Department (<u>dnum</u>, dname, dngrenum)
3. Personal (<u>pnum</u>, pname, plocation)

There is no transitive dependency, so relations are normalized

Problem 2. Consider the following database schema: College (prof-code, dept-code, head-of-dept, percentage-time).

You can find dept-code by knowing head-of-dept and vice-versa and percentage-time is the time ratio, which a prof-code spend for a particular department. Draw the functional dependency diagram for above schema.

Solution.



EXERCISES

1. Explain what is meant by normalization and its advantages. Explain and define 1NF, 2NF, 3NF, BCNF and 4NF by giving suitable examples for each.
2. Compare and contrast Full/Partial/Transitive dependencies.
3. What do you understand by functional dependency? Explain with examples.
4. Explain 1NF, 2NF, 3NF, BCNF and 4NF with help of suitable examples.
5. Explain multi-valued dependencies and Fourth Normal Form.
6. Explain Functional and transitive dependencies.
7. "Non-loss decomposition is an aid to relational databases". Is it true ? If yes, then justify it through an example.
8. Explain join dependency and 5th Normal Form.
9. What is normalization ? Explain successive normalization in designing a relational database by taking a suitable example.
10. Explain join dependency and multivalued dependency.
11. What is normalization ? What is the need of normalization? What do you understand by loss less decomposition ? Define and discuss 3NF and BCNF using suitable examples.
12. Define normalization. Take a relation and normalize it upto 3NF, explaining each step.
13. Define BCNF. Using suitable example distinguish between 3NF and BCNF, which is better and why ?
14. What are the problems of bad database design ? Write the procedure of normalizing database while discussing the various normal forms.
15. Define functional dependence and full functional dependence and then explain the concept of 2NF also with example.
16. Define Functional Dependencies. Write Armstrong rule and show that other rules are derived from Armstrong rules. Also distinguish between full FD and partial FD.
17. State, by giving examples, the conditions that are necessary for a relation to be in INF, 2NF, 3NF, and BCNF.
18. What is functional Dependency ? How does, it relate with multivalued dependency ? Explain.
19. Why are certain functional dependencies called "trivial functional dependencies" ? Explain.

20. Consider the following table.

Insurance No.	Contract No.	HRS.	Ename	Hotel No.	Location
1135	C1024	16	Sharma	H25	Delhi
1057	C1024	20	Sahni	H25	Delhi
1068	C1025	24	Gupta	H04	Mumbai
1140	C1025	16	Jindal	H04	Mumbai

Describe, and illustrate, the process of normalization up to BCNF. State and make assumptions, if any.

21. Give major differences between 4NF and 5NF.

22. How functional dependency is different from multi-valued and join dependencies ? Give an example each of multi-valued and join dependency ?

Chapter **8**

TRANSACTIONS AND CONCURRENCY CONTROL

8.1 INTRODUCTION

Transaction is a logical unit of work that represents the real-world events. A transaction is also defined as any one execution of a user program in a Database Management System (DBMS). The transaction management is the ability of a database management system to manage the different transactions that occur within it. A DBMS has to interleave the actions of many transactions due to performance reasons. The interleaving is done in such a way that the result of the concurrent execution is equivalent to some serial execution of the same set of transactions.

Concurrency control is the activity of coordinating the actions of transactions that operate, simultaneously or in parallel to access the shared data. How the DBMS handles concurrent executions is an important aspect of transaction management. Other related issues are how the DBMS handles partial transactions, or transactions that are interrupted before successful completion. The DBMS makes it sure that the modifications done by such partial transactions are not seen by other transactions. Thus, transaction processing and concurrency control form important activities of any Database Management System (DBMS) and is the subject-matter of this chapter.

8.2 TRANSACTION

A transaction can be defined as a unit or part of any program at the time of its execution. During transactions data items can be read or. updated or both.

8.2.1 ACID Properties of Transaction

The database system is required to maintain the following properties of transactions to ensure the integrity of the data.

1. *Atomicity* : A transaction must be atomic. Atomic transaction means either all operations within a transaction are completed or none.
2. *Consistency* : A transaction must be executed in isolation. It means variables used by a transaction cannot be changed by any other transaction concurrently.
3. *Isolation* : During concurrent transaction each transaction must be unaware of other transactions. For any transaction T_i, it appears to T_i that any other transaction T_k is either finished before starting of T_i or started after T_i finished.
4. *Durability* : Changes are made permanent to database after successful completion of transaction even in the case of system failure or crash.

8.2.2 Transaction States

A transaction must be in one of the following states as shown in Figure 8.1.

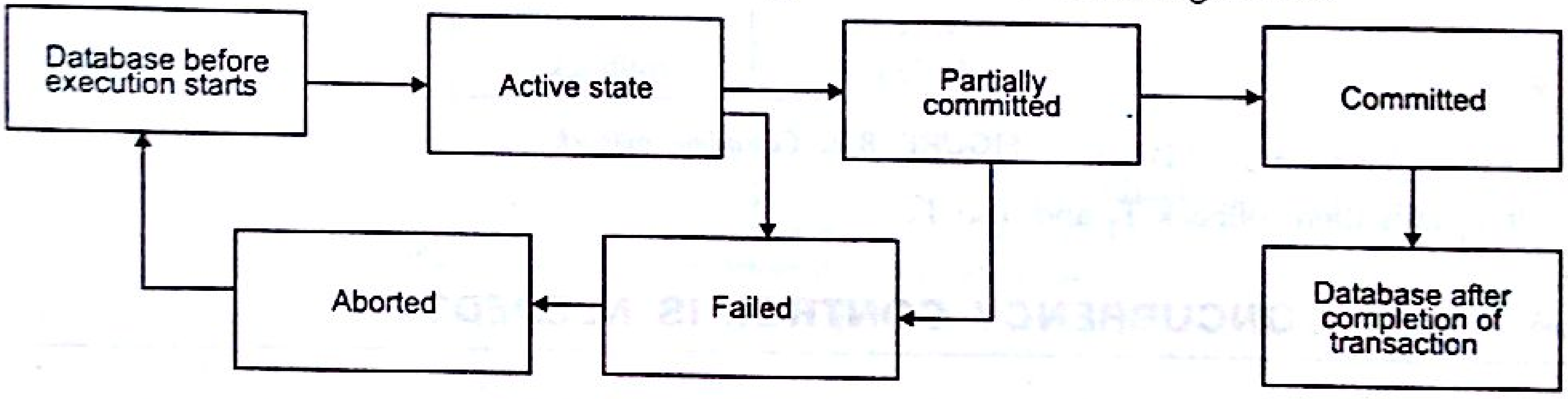

Database before execution starts
Active state
Partially committed
Committed
Aborted
Failed
Database after completion of transaction

FIGURE 8.1. *States of the transaction.*

1. *Active state* : It is the initial state of transaction. During execution of statements, a transaction is in active state.
2. *Partially committed* : A transaction is in partially committed state, when all the statements within transaction are executed but transaction is not committed.
3. *Failed* : In any case, if transaction cannot be proceeded further then transaction is in failed state.
4. *Committed* : After successful completion of transaction, it is in committed state.

8.3 SOME DEFINITIONS

8.3.1 Serializability

Consider a set of transactions (T_1, T_2 ..., T_i). S_1 is the state of database after they are concurrently executed and successfully completed and S_2 is the state of database after they are executed in any serial manner (one-by-one) and successfully completed. If S_1 and S_2 are same then the database maintains serializability.

8.3.2 Concurrent Execution

If more than one transactions are executed at the same time then they are said to be executed concurrently.

8.3.3 Recoverability

To maintain atomicity of database, undo effects of any transaction has to be performed in case of failure of that transaction. If undo effects successfully then that database maintains recoverability. This process is known as **Rollback.**

8.3.4 Cascading Rollback

If any transaction T_i is dependent upon T_j and T_i is failed due to any reason then rollback T_j. This is known as cascading rollback. Consider Figure 8.2. Here T_2 is dependent upon T_1 because T_2 reads value of C which is updated by T_1.

T_1	T_2
read A	
read B	
read C	read A
write C	read B
read A	read C
write A	
Failed	rollback

FIGURE 8.2. *Cascading rollback.*

If T_1 fails then rollback T_1 and also T_2.

8.4 WHY CONCURRENCY CONTROL IS NEEDED?

To make system efficient and save time, it is required to execute more than one transaction at the same time. But concurrency leads several problems. The three problems associated with concurrency are as follows :

8.4.1 The Lost Update Problem

If any transaction T_j updates any variable v at time t without knowing the value of v at time t then this may leads to lost update problem. Consider the transactions shown in Figure 8.3.

Transaction T_i	Time	Transaction T_j
–	↓	–
read(v)	t_1	–
–	↓	–
–	t_2	read(v)
–	↓	–
update(v)	t_3	–
–	↓	–
–	t_4	update(v)
–	↓	

FIGURE 8.3. *The lost update problem.*

At time t_1 and t_2, transactions t_i and t_j reads variable v respectively and get some value of v. At time t_3, t_i updates v but, at t_4, t_j also updates v (old value of v) without looking the new value of v (updated by t_i). So, updation made by t_i at t_3 is lost at t_4 because T_j overwrites it.

8.4.2 The Uncommitted Dependency Problem

This problem arises if any transaction T_j updates any variable v and allows retrieval or updation of v by any other transaction but T_j rolled back due to failure. Consider the transactions shown in Figure 8.4.

Transaction T_i	Time	Transaction T_j
—	│	—
—	t_1	write(v)
—	│	—
read(v) or write(v)	t_2	—
—	│	—
—	t_3	—
—	↓	Rollback

FIGURE 8.4. *The uncommited dependency problem.*

At time t_1, transaction T_j, updates variable v which is read or updated by T_i at time t_2. Suppose at time t_3, T_j is rollbacked due to any reason then result produced by T_i is wrong because it is based on false assumption.

8.4.3 The Inconsistent Analysis Problem

Consider the transactions shown in Figure 8.5.

Transaction T_i	Time	Transaction T_j
—	│	—
read(a) a = 10	t_1	—
read(b) b = 20	t_2	—
—	│	—
—	t_3	write(a) a = 50
—	│	—
—	t_4	Commit
—	│	—
Add a, b $a + b$ = 30, not 60	t_5	
—	↓	

FIGURE 8.5. *The inconsistent analysis problem.*

The transaction T_i reads variable a and b at time t_1 and t_2 respectively. But at time t_3, T_j updates value of a from 10 to 50 and commits at t_4 that makes changes permanent. So, addition of a and b at time t_5 gives wrong result. This leads inconsistency in database because of inconsistent analysis by T_i.

8.5 CONCURRENCY CONTROL TECHNIQUES

To avoid concurrency related problems and to maintain consistency, some rules (protocols) need to be made so that system can be protected from such situations and increase the efficiency.

8.5.1 Lock-Based Protocols

To maintain consistency, restrict modification of the data item by any other transaction which is presently accessed by some transaction. These restrictions can be applied by applying locks on data items. To access any data item, transaction have to obtain **lock** on it.

8.5.1.1 Types of Lock

There are two types of locks that can be implemated on a transaction.

(i) *Shared lock* : Shared lock is Read-Only lock. If a transaction T_i has obtained shared lock on data item A then T_i can read A but cannot modify A. It is denoted by S and T_i is said to be in Shared lock mode.

(ii) *Exclusive lock* : Exclusive lock is Read-Write lock. If a transaction T_i has obtained exclusive lock on data item A then T_i can both read and modify (write) A. It is denoted by X and T_i is said to be in Exclusive lock mode.

8.5.1.2 Concurrency Control Manager

The working of Concurrency Control Manager is to grant locks to different transactions. Concurrency Control manager is basically a programme.

8.5.1.3 Compatible Lock Modes

Suppose transaction T_i has obtained a lock on data item V in mode A. If transaction T_j can also obtain lock on V in mode B then, A is compatible with B or these two lock modes are compatible. Matrix of compatibility of shared and Exclusive lock modes are shown in Figure 8.6.

	S	X
S	Possible	Not possible
X	Not possible	Not possible

FIGURE 8.6. *Compatibility of shared and exclusive lock mode.*

The table shows that if a transaction T_i has shared lock on data item V then any other transaction T_j can obtain only shared lock on V at the same time. All other possible combinations are incompatible.

S - lock(A)	—	Shared lock on data item A
X - lock(A)	—	Exclusive lock on data item A
read(A)	—	Read operation on A
write(A)	—	Write operation on A
unlock(A)	—	A is free now.

A transaction T_i can hold only those items which are used in it. If any data item V is not free then T_i is made to wait until V is released. It is not desirable to unlock data item immediately after its final access.

Example : Consider the banking system in which you have to transfer Rs. 500/- from account A to account B. Transaction T_1 in Figure 8.7 shows transfer of amount and T_2 shows sum of accounts A and B. Initially account A has Rs. 1000/- and B has Rs. 300/-.

T_1	T_2
X - lock(A);	S - lock(A);
read(A);	read(A);
A = A - 500;	unlock(A);
Write(A);	S-lock(B);
Unlock(A);	read(B);
X-lock(B);	unlock(B);
read(B);	display(A + B);
B = B + 500;	
Write(B);	
unlock(B);	

FIGURE 8.7. *Transactions T_1 and T_2 for banking example.*

The possible schedule for T_1 and T_2 is shown in Figure 8.8.

T_1		T_2	
X-lock(A)			
read(A)	A = 1000		
A = A - 500	A = 500		
Write(A)			
unlock(A)			
		S-lock(A)	
		read(A)	A = 500
		unlock(A)	
		S-lock(B)	
		read(B)	B = 300
		unlock(B)	
		display(A + B)	A + B = 800
X-lock(B)			
read(B)	B = 300		
B = B + 500	B = 800		
write(B)			
Unlock(B)			

FIGURE 8.8. *Schedule for transaction T_1 and T_2 in case locked data items are unlocked immediately after its final access.*

In the schedule shown in Figure 8.8, T_2 gives wrong result *i.e.*, Rs.800/- instead of Rs.1300/- because unlock A is performed immediately after its final access.

8.5.1.4 Improvement in Basic Structure of Locking

Keep all locks until the end of transaction. The modified transaction T_1 and T_2 are shown in Figure 8.9.

T_1	T_2
X-lock(A);	S-lock(A);
read(A);	read(A);
A = A - 500;	
write(A);	S-lock(B);
X-lock(B);	read(B);
read(B);	display(A + B);
B = B + 500;	unlock(A);
write(B);	unlock(B);
unlock(A);	
unlock(B);	

FIGURE 8.9. *Improved structure of locking on transactions.*

The transaction T_2 cannot obtain lock on A until A is released by T_1 but at that time changes are made in both accounts.

Advantages

1. In maintains serializability.
2. It solves all the concurrency problems.

Disadvantages

1. It leads to a new problem that is Deadlock.

Two possible schedules S_1 and S_2 for transactions T_1 and T_2 of Figure 8.9 are shown in Figure 8.10(*a*, *b*).

In S_1, right result is obtained but S_2 leads to deadlock because both T_1 and T_2 are in waiting state and wait for release of B and A respectively. (Shared mode and exclusive mode are incompatible).

8.5.1.5 Two-phase Locking Protocol

Two-phase locking protocol *guarntees serializability*. In this protocol, every transaction issue lock and unlock requests in two phases.

(*i*) *Growing phase* : In this phase, transaction can obtain new locks but cannot release any lock.

(*ii*) *Shrinking or contracting phase* : In this phase, transaction can release locks, but cannot obtain new locks.

Lock Point : In growing phase, the point at which transaction has obtained its final lock is called Lock Point.

In two phase locking protocol, all transactions can be ordered according to their lock points. Transactions T_1 and T_2 in Figure 8.9 follows two phase locking protocol.

S_1

T_1		T_2	
X-lock(A)			
read(A)	A = 1000		
A = A - 500	A = 500		
Write(A)			
X-lock(B)			
read(B)	B = 300		
B = B + 500	B = 800		
write(B)			
Unlock(A)			
Unlock(B)			
		S-lock(A)	
		read(A)	A = 500
		S-lock(B)	
		read(B)	B = 800
		display(A + B)	A + B = 1300
		unlock(A)	
		unlock(B)	

(a)

S_2

T_1	T_2
X-lock(A)	
read(A)	
A = A - 500	
write(A)	
	S-lock(B)
	read(B)
	$\boxed{\text{S-lock(A)}}$ $\downarrow$ Not-possible because A is locked exclusively by T_1 and T_2 have to wait for A.
$\boxed{\text{X-lock(B)}}$ $\downarrow$ Not-possible because B is locked in shared mode by T_2 and T_1 have to wait for B.	

(b)

FIGURE 8.10. *Schedules S_1 and S_2 for transactions T_1 and T_2.*

1. ***Basic Two phase Locking Protocol :*** The technique described above is known as basic two phase locking Protocol.

Advantages

1. It ensures serializability.

Disadvantages

1. It may cause deadlock.
2. It may cause cascading rollback.

2. ***Strict Two phase Locking Protocol :*** In strict two phase locking protocol, all exclusive locks are kept until the end of transaction or until the transaction commits.

Advantages

1. There is no cascading rollback in strict two phase locking protocol.

Example. Consider partial schedules S_1 (in basic two-phase locking) and S_2 (in Strict two-phase locking) as shown in Figure 8.10.

In S_1 there is cascading rollback of T_4 due to rollback of T_3 but in S_2, all exclusive lock are kept till end and avoid cascading rollback as shown in Figure 8.11(*a, b*).

3. ***Rigorous Two-phase Locking Protocol :*** In rigorous two phase locking protocol, all locks are kept until the transaction commits. It means both shared and exclusive locks cannot be released till the end of transaction.

S_1

T_3	T_4
S-lock(A)	
read(A)	
X-lock(B)	
read(B)	
write(B)	
X-lock(C)	
unlock(B)	
read(C)	X-lock(B)
write(C)	read(B)
—	write(B)
—	—
—	—
--	—
Rollback	Rollback due to roll back of T_3

Basic two-phase locking

(a)

S_2

T_3	T_4
S-lock(A)	
read(A)	
X-lock(B)	
read(B)	
write(B)	
X-lock(C)	
read(C)	
write(C)	
commit	
unlock(B)	
unlock(A)	X-lock(B)
unlock(C)	read(B)
	write(B)
	—
	—

Strict two-phase locking

(b)

FIGURE 8.11. *Partial schedules S_1 and S_2.*

8.5.1.6 Lock Conversion

To improve efficiency of two-phase locking protocol, modes of lock can be converted.

(i) *Upgrade* : Coversion of shared mode to exclusive mode is known as upgrade.

(ii) *Downgrade* : Conversion of exclusive mode to shared mode is known as downgrade.

Consider transaction T_5 as shown in Figure 8.12.

In T_5, an exclusive lock of A is needed after sometime so first obtain shared lock on A and then upgrade it. During this period any other transaction can also obtain shared lock of A and hence increase efficiency.

T_5
–
–
read(A)
read(B)
read(C)
–
–
–
write(A)
–
–

FIGURE 8.12. *Lock conversion (Upgrade).*

8.5.2 Graph Based Protocols

In graph based protocols, an additional information is needed for each transaction to know, how they will access data items of database. These protocols are used where two-phase protocols are not applicable but they required additional information that is not required in two-phase protocols. In graph based protocols partial ordering is used.

Consider, a set of data items D = $\{d_1, d_2, ..., d_i, d_j, ...d_z\}$. If $d_i \rightarrow d_j$ then, if any transaction T want to access both d_i and d_j then T have to access d_i first then d_j.

Here, **tree protocol** is used in which D is viewed as a directed acyclic graph known as *database graph.*

Rules for tree protocol :

1. Any transaction T can lock any data item at first time.
2. After first lock if T wants to lock any other data item *v* then, first, T have to lock its parent.
3. T cannot release all exclusive locks before commit or abort.
4. Suppose T has obtained lock on data item *v* and then release it after sometime. Then T cannot obtain any lock on *v* again.

All the schedules under graph based protocols maintain serializability. Cascadelessness and recoverability are maintained by rule (3) but this reduces concurrency and hence reduce efficiency.

8.5.5.2 Multiversion Two-phase Locking

In multiversion two-phase locking, advantages of both multiversion concurrency control techniques and two-phase locking technique are combined. It is also helpful to overcome the disadvantages of multiversion timestamp ordering.

Single Timestamp is given to every version of each data item. Here timestamp counter (TS-counter) is used instead of system clock and logical counter. Whenever a transaction commits, TS-counter is incremented by 1.

Read only transactions are based upon multiversion timestamp ordering. These transactions are associated with timestamp equal to the value of TS-counter.

Updations are based upon rigrous two-phase locking protocol.

8.6 DEADLOCKS

A system is said to be in deadlock state if there exist a set of transactions $\{T_0, T_1, ..., T_n\}$ such that each transaction in set is waiting for releasing any resource by any other transaction in that set. In this case, all the transactions are in waiting state.

8.6.1 Necessary Conditions for Deadlock

A system is in deadlock state if it satisfies all of the following conditions.

(*i*) *Hold and wait* : Whenever a transaction holding at least one resource and waiting for another resources that are currently being held by other processes.

(*ii*) *Mutual Exclusion* : When any transaction has obtained a nonsharable lock on any data item and any other transaction requests that item.

(*iii*) *No Preemption* : When a transaction holds any resource even after its completion.

(*iv*) *Circular Wait* : Let T be the set of waiting processes $T = \{T_0, T_1, ..., T_i\}$ such that T_0 is waiting for a resource that is held by T_1, T_1 is waiting for a resource that is held by T_2, T_i is waiting for a resource that is held by T_0.

8.6.2 Methods for Handling Deadlocks

There are **Two** methods for handling deadlocks. These are :

8.6.2.1 Deadlock Prevention

It is better to prevent the system from deadlcok condition then to detect it and then recover it. Different approaches for preventing deadlocks are :

1. ***Advance Locking*** : It is the simplest technique in which each transaction locks all the required data items at the beginning of its execution in atomic manner. It means all items are locked in one step or none is locked.

Disadvantages

1. It is very hard to predict all data items required by transaction at starting.
2. Under utilization is high.
3. Reduces concurrency.

2. ***Ordering Data Items*** : In this approach, all the data items are assigned in order say 1,2,3,... etc. Any transaction T_i can acquire locks in a particular order, such as in ascending order or descending order.

Disadvantages : It reduces concurrency but less then advance locking.

ransaction chooses as

can be added in cost factor to reduce

3. *Ordering Data Items with Two-phase Locking* : In this approach, two-phase locking with ordering data items is used to increase concurrency.

4. *Techniques based on Timestamps* : There are Two different techniques to remove deadlock based on time stamps.

(*i*) **Wait-die** : In wait-die technique if any transaction T_i needs any resource that is presently held by T_j then T_i have to wait only if it has timestamp smaller than T_j otherwise T_i is rolled back.

```
If  TS(T_i)<TS(T_j)
        T_i waits;
    else
        T_i is rolled back;
```

It is a nonpreemptive technique.

(*ii*) **Wound-wait** : In wound-wait technique, if any transaction T_i needs any resource that is presently held by T_j then T_i have to wait only if it has timestamp larger then T_j otherwise T_j is rolled back.

```
If  TS(T_i)>TS(T_j)
        T_i waits;
    else
        T_j is rolled back;
```

It is a *preemptive technique.*

Disadvantage : There are unnecessary roll backs in both techniques that leads to starvation.

8.6.2.2 Deadlock Detection and Recovery

In this approach, allow system to enter in deadlock state then detect it and recover it. To employ this approach, system must have :

1. Mechanism to maintain information of current allocation of data items to transactions and the order in which they are allocated.
2. An algorithm which is invoked periodically by system to determine deadlocks in system.
3. An algorithm to recover from deadlock state.

Deadlock Detection : To detect deadlock, maintain **wait-for graph**.

1. *Wait-for Graph* : It consists of a pair G = (V, E), where V is the set of vertices and E is the set of edges. Vertices show transactions and edges show dependency of transactions.

If transaction T_i request a data item which is currently being held by T_j then, an edge $T_i \rightarrow T_j$ is inserted in graph. When T_i has obtained the required item from T_j remove the edge. Deadlock occurs when there is cycle in wait-for graph. Consider wait-for graph as shown in Figure 8.17.

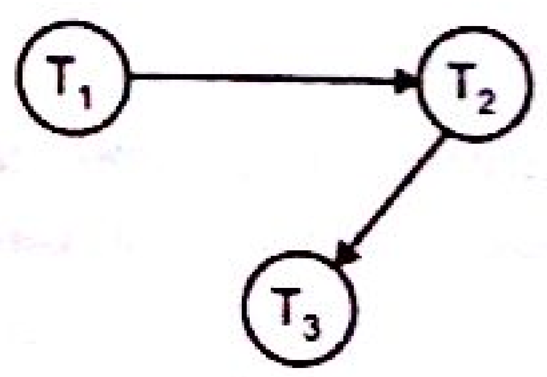


FIGURE 8.17. *Wait-for graph without deadlock situation.*

There are three transactions T_1, T_2 and T_3. The transaction T_1 is waiting for any data item held by T_2 and T_2 is waiting for any data item held by T_3. But there is no deadlock because there is no cycle in wait-for graph.

Consider the graph as shown in Figure 8.18, where T_3 is also waiting for any data item held by T_1.

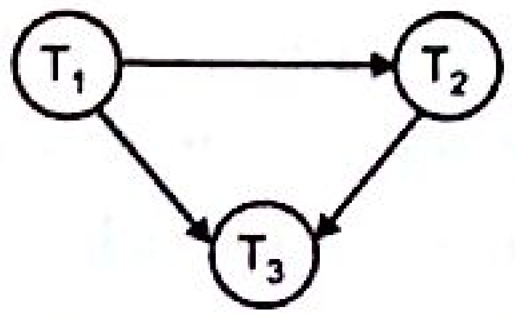


FIGURE 8.18. *Wait-for graph with deadlock.*

Thus, there exists a cycle in wait-for graph.

$$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$$

which results system in deadlock state and T_1, T_2 and T_3 are all blocked.

Now a major question arises : How much will be the time interval after which system invokes deadlock detection algorithm? It depends upon **Two** factors :

1. After how much time a deadlock occurs.
2. How many transactions are deadlocked.

2. *Recovery from Deadlock* : When deadlock detection algorithm detects any deadlock in system, system must take necessary actions to recover from deadlock.

Steps of recovery algorithm are :

1. **Select a victim :** To recover from deadlock, break the cycle in wait-for graph. To break this cycle, rollback any of the deadlocked transaction. The transaction which is rolled back is known as **victim**.

   Various factors to determine victim are :

   (*i*) Cost of transaction.

   (*ii*) How many more data items it required to complete its task.

   (*iii*) How many more transactions affected by its rollback.

   (*iv*) How much the transaction is completed and left.

2. **Rollback** : After selection of victim, it is rollbacked.

   There are **Two** types of rollback to break the cycle :

   (*i*) *Total rollback* : This is simple technique. In total rollback, the victim is rolled back completely.

   (*ii*) *Partial rollback* : In this approach, rollback the transaction upto that point which is necessary to break deadlock. Here transaction is partially rolled back. It is an effective technique but system have to maintain additional information about the state of all running transactions.

3. **Prevent Starvation** : Sometimes, a particular transaction is rolledback several times and never completes which causes starvation of that transaction. To prevent starvation, set the limit of rollbacks or the maximum number, the same transaction chooses as victim. Also, the number of rollback can be added in cost factor to reduce starvation.

5.6.2.3 Timeout-based Schemes

Timeout-based scheme exists in between deadlock prevention and deadlock detection and recovery. In this scheme, the transaction waits for at most a fixed time for any resource. If transaction is not able to get resource in that fixed time then it is said to be timeout and transaction is rollbacked and restarted after sometime.

This approach is preferred where transactions are short and long waits cause deadlocks.

It is typical to choose appropriate time for time-out.

If it is too long then it results unnecessary delays and if it is too short then it cause unnecessary rollbacks.

EXERCISES

1. What is concurrency ? Discuss the various problems associated within DBMS.
2. Discuss the locking techniques for concurrency control with examples.
3. Discuss the following concurrency control techniques with examples:
   (*a*) Locking techniques
   (*b*) Techniques based on time stamp ordering
   (*c*) Multiversian concurrency control techniques.
4. What do you understand by concurrency in databases ? Discuss the various concurrency control methods in databases with examples.
5. Explain the multiversion concurrency control techniques.
6. Discuss the concurrency and the various possible problems associated with it in DBMS. Discuss various concurrency control techniques through suitable examples. Also discuss, what do you understand by serializability ?
7. What are distributed locking methods ? When and how are they used ? Also discuss advantages and disadvantages of distributed locking methods.
8. Explain the concept of concurrency and concurrency control in database.
9. Explain the various concurrency control techniques without locking with examples.
10. During its execution a transaction passes through several states, until it finally commits or aborts. List all possible sequence of states through which a transaction may pass. Explain why each state transition may occur ?
11. What is a transaction ? What are the properties of a transaction ? How is transaction recovered when a system failure occurs?
12. What do you mean by concurrency? Explain how the concurrency problems can be solved with the use of locks.
13. What problems will occur due to concurrent execution of two transactions? What are their solutions.
14. What benefits does strict two phase locking provide ? What disadvantages result?
15. What are ACID properties ? Explain.
16. What do you understand by serializability ? Explain.
17. What do you mean by concurrent-access anomolies in databases ?
18. What are concurrency problems ? How are they solved ?

19. What is time stamping ? How can it be used for concurrency control ? Describe this with the help of an example.

20. An on-line books issues/return system in a library allows various members to reserve books on-line. A member may also like to drop the reservation request, if so desired. This system is also used by the librarian to find how many books are outstanding for more than a month. Write the pseudocodes of the necessary transactions needed for the system above. What are the problems that may occur due to concurrent transactions as above ? How can you solve these problems.

21. Explain the concepts of two-phases locking with an example. How is it related to serializability ? Describe the role of two-phased locking in the process of deadlock detection and avoidance.

22. An airline reservation allows many customers to book tickets simultaneously. What are the basic concurrency related problems that you may encounter, in case no concurrency control mechanism is inplace ? What is the solution proposed by you to overcome the concurrency related problem as above ?

23. Compare and contrast the features of simple locking, intention mode locking and time stamping mechanisms from the viewpoint of transaction control under concurrent transitions.

24. Discuss "wait-die" and "wound-wait" approaches of deadlock avoidance. Compare these approaches of deadlock avoidance with a deadlock aviodance approach in which data items are locked in a particular order (according to their rank.)

25. Suppose you are working in a bank and have been assigned as a transaction programmer, what measures will you take such that your transaction programs do not cause any concurrency related problem ? Explain with help of example.

26. What is deadlock in the contest of concurrent transaction execution ? When does it occur ? How is it detected in centralized database system? How can it be avoided ?

27. A company has increased the basic salary of its employees by Rs. 2000. The salary was further upgraded by giving an additional increment of 10%. The company also has a concurrent transaction, which evaluates average salary of its employees. Write the necessary transactions, which can run concurrently to give appropriate results. Use appropriate schemes to avoid undesirable results.

28. Assume the following concurrent transactions:
    — Company X buys 500 shares from company Y making a financial transaction of Rs. 500.
    — Company Z provides loan of Rs. 6000 to companies X and Y.
    — The monthly average of financial transaction is to be calculated.

    Write the pseudocode programs of the transactions that can run concurrently. Use appropriate schemes to avoid undesirable results.

29. Assume the following concurrent banking transactions.
    — Transfer of Rs. 2000 from Account number 250 to Account number 500
    — Withdrawal by account 500, an amount of Rs. 5000.
    — Finding out the average balance for various accounts of the bank.

    Write the necessary transaction pseudocode program, which can run concurrently. Use appropriate schemes to avoid undesirable results.

Chapter 10

DATABASE RECOVERY SYSTEM

10.1 INTRODUCTION

A Computer system can be failed due to variety of reasons like disk crash, power fluctuation, software error, sabotage and fire or flood at computer site. These failures result in the lost of information. Thus, database must be capable of handling such situations to ensure that the **atomicity** and **durability** properties of transactions are preserved. An integral part of a database system is the *recovery manager* that is responsible for recovering the data. It ensures **atomicity** by undoing the actions of transactions that do not commit and **durability** by making sure that all actions of committed transactions survive system crashes and media failures. The recovery manager deal with a wide variety of database states because it is called during system failures. Furthermore, the database recovery is the process of restoring the database to a consistent state when a failure occurred. In this chapter, we will discuss different failures and database recovery techniques used so that the database system be restored to the most recent consistent state that existed shortly before the time of system failure.

10.2 CLASSIFICATION OF FAILURES

Failure refers to the state when system can no longer continue with its normal execution and that results in loss of information. Different types of failure are as follows :

1. **System crash** : This failure happens due to the bugs in software or by hardware failure etc.
2. **Transaction failure** : This failure happens due to any **logical error** such as overflow of stack, bad input, data not found, less free space available etc., or by **system error** such as deadlocks etc.
3. **Disk failure** : This failure happens due to head crash, tearing of tapes, failure during transfer of data etc.

10.3 RECOVERY CONCEPT

Recovery from failure state refers to the method by which system restore its most recent consistent state just before the time of failure. There are several methods by which you can recover database from failure state. These are defined as follows :

10.3.1 Log Based Recovery

In log based recovery system, a **log** is maintained, in which all the modifications of the database are kept. A log consists of **log records.** For each activity of database, separate log record is made. Log records are maintained in a serial manner in which different activities are happened. There are various log records. A typical update log record must contain following fields:

(i) *Transaction identifier* : A unique number given to each transaction.

(ii) *Data-item identifier* : A unique number given to data item written.

(iii) *Date and time* of updation.

(iv) *Old value* : Value of data item before write.

(v) *New value* : Value of data item after write.

Logs must be written on the non-volatile (stable) storage. In log-based recovery, the following two operations for recovery are required :

(i) *Redo* : It means, the work of the transactions that completed successfully before crash is to be performed again.

(ii) *Undo* : It means, all the work done by the transactions that did not complete due to crash is to be undone.

The redo and undo operations must be idempotent. An idempotent operation is that which gives same result, when executed one or more times.

For any transaction T_i, various log records are :

$[T_i$ start$]$	: It records to log when T_i starts execution.
$[T_i, A_j]$	: It records to log when T_i reads data item A_j.
$[T_i, A_j, V_1, V_2]$	: It records to log when T_i updates data item A_j, where V_1 refers to old value and V_2 refers to new value of A_j.
$[T_i$ Commit$]$	: It records to log when T_i successfully commits.
$[T_i$ aborts$]$	: It records to log if T_i aborts.

There are *two types of Log Based Recovery techniques* and are discussed below :

[illegible] 10.3.1.1 Recovery Based on Deferred Database Modification

In deferred database modification technique, deferred (stops) all the write operations of any Transaction T_i until it partially commits. It means modify real database after T_i partially commits. All the activities are recorded in log. Log records are used to modify actual database. Suppose a transaction T_i wants to write on data item A_j, then a log record $[T_i, A_j, V_1, V_2]$ is saved in log and it is used to modify database. After actual modification T_i enters in committed state. *In this technique, the old value field is not needed.*

Consider the example of Banking system. Suppose you want to transfer Rs.200 from Account A to B in Transaction T_1 and deposit Rs.200 to Account C in T_2. The transactions T_1 and T_2 are shown in Figure 10.1.

T_1	T_2
read(A);	read(C);
A = A - 200;	C = C + 200;
write(A);	write(C);
read(B);	
B = B + 200;	
write(B);	

FIGURE 10.1. *Transactions T_1 and T_2.*

Suppose, the initial values of A, B and C Accounts are Rs. 500, Rs. 1,000 and Rs. 600 respectively, Various log records for T_1 and T_2 are as shown in Figure 10.2.

[T_1 start]
[T_1, A]
[T_1, A, 300]
[T_1 , B]
[T_1, B, 1200]
[T_1 commit]
[T_2 start]
[T_2, C]
[T_2, C, 800]
[T_2 commit]

FIGURE 10.2. *Log records for transactions T_1 and T_2.*

For a redo operation, log must contain [T_i start] and [T_i commit] log records.

(a)	(b)
[T_1 start]	[T_1 start]
[T_1, A]	[T_1, A]
[T_1, A, 300]	[T_1, A, 300]
	[T_1, B]
	[T_1, B, 1200]
	[T_1 commit]
	[T_2 start]
	[T_2, C]
	[T_2, C, 800]

FIGURE 10.3. *Log of transactions T_1 and T_2 in case of crash.*

Crash will happen at any time of execution of transactions. Suppose crash happened

(*i*) **After write (A) of T_1:** At that time log records in log are shown in Figure 10.3(*a*). There is no need to redo operation because no commit record appears in the log. Log records of T_1 can be deleted.

(*ii*) **After write (C) of T_2 :** At that time log records in log are shown in Figure 10.3(*b*). In this situation, you have to redo T_1 because both [T_1 start] and [T_1 commit] appears in log. After redo operation, value of A and B are 300 and 1200 respectively. Values remain same because redo is idempotent.

(*iii*) **During recovery :** If system is crashed at the time of recovery, simply starts the recovery again.

10.3.1.2 Recovery Based on Immediate Database Modification

In immediate database modification technique, database is modified by any transaction T_i during its active state. It means, real database is modified just after the write operation but after log record is written to stable storage. This is because log records are used during recovery. Use both Undo and Redo operations in this method. Old value field is also needed (for undo operation). Consider again the banking transaction of Figure 10.1. Corresponding log records after successful completion of T_1 and T_2 are shown in Figure 10.4.

[T_1 start]
[T_1, A]
[T_1, A, 500, 300]
[T_1, B]
[T_1, B, 1000, 1200]
[T_1, commit]
[T_2, start]
[T_2, C]
[T_2, C, 600, 800]
[T_2 commit]

FIGURE 10.4. *Log records for transactions T_1 and T_2.*

- For a transaction T_i to be redone, log must contain both [T_i start] and [T_i commit] records.
- For a transaction T_i to be undone, log must contain only [T_i start] record. ✓

(a)

[T_1 start]
[T_1, A]
[T_1, A, 500, 300]

(b)

[T_1 start]
[T_1, A]
[T_1, A, 500, 300]
[T_1, B]
[T_1 , B, 1000,1200]
[T_1 commit]
[T_2 start]
[T_2, C]
[T_2, C, 600,800]

FIGURE 10.5. *Log of transactions T_1 and T_2 in case of crash.*

Crash will happen at any time of execution of transaction. Suppose crash happened.

(i) **After write (A) of T_1 :** At that time log records in log are shown in Figure 10.5(*a*). Here only [T_1 start] exists so undo transaction T_1. As a result, Account A restores its old value 500.

(ii) **After write (C) of T_2 :** At that time log records in log are shown in Figure 10.5(*b*). During back-up record [T_2 start] appears but there is no [T_2 commit], so undo transaction T_2. As a result, Account C restores its old value 600. When you found both [T_1 start] and [T_1 commit] records in log, redo transaction T_1 and account A and B both keep their new value.

(iii) **During recovery :** If system is crashed at the time of recovery simply starts recovery again.

10.3.1.3 Checkpoints

Both the techniques discussed earlier ensures recovery from failure state, but they have some disadvantages such as :

(i) They are time consuming because successfully completed transactions have to be redone.

(ii) Searching procedure is also time consuming because the whole log has to be searched.

So, use checkpoints to reduce overhead. Any of previous recovery techniques can be used with checkpoints. All the transactions completed successfully or having [T_i commit] record before [checkpoint] record need not to be redone.

During the time of failure, search the most recent checkpoint. All the transactions completed successfully after checkpoint need to be redone. After searching checkpoint, search the most recent transaction T_i that started execution before that checkpoint but not completed. After searching that transaction, redo/undo transaction as required in applied method.

Advantages

1. No need to redo successfully completed transactions before most recent checkpoints.
2. Less searching required.
3. Old records can be deleted.

10.4 SHADOW PAGING

Shadow Paging is an alternative technique for recovery to overcome the disadvantages of log-based recovery techniques. The main idea behind the shadow paging is to keep two page tables in database, one is used for *current operations* and other is used *in case of recovery*.

Database is partitioned into fixed length blocks called *pages*. For memory management, adopt any paging technique.

Page Table

To keep record of each page in database, maintain a page table. Total number of entries in page table is equal to the number of pages in database. Each entry in page table contains a pointer to the physical location of pages.

[illegible]

The two page tables in Shadow Paging are :

(*i*) *Shadow page table* : This table cannot be changed during any transaction.

(*ii*) *Current page table* : This table may be changed during transaction.

Both page tables are identical at the start of transaction. Suppose a transaction T_i performs a write operation on data item V, that resides in page *j*. The write operation procedure is as follows :

1. If j^{th} page is in main memory then OK otherwise first transfer it from secondary memory to main memory by instruction input (V).
2. Suppose page is used first time then :
   (*a*) System first finds a free page on disk and delete its entry from free page list.
   (*b*) Then modify the current page table so that it points to the page found in step 2(*a*).
3. Modify the contents of page or assign new value to V.

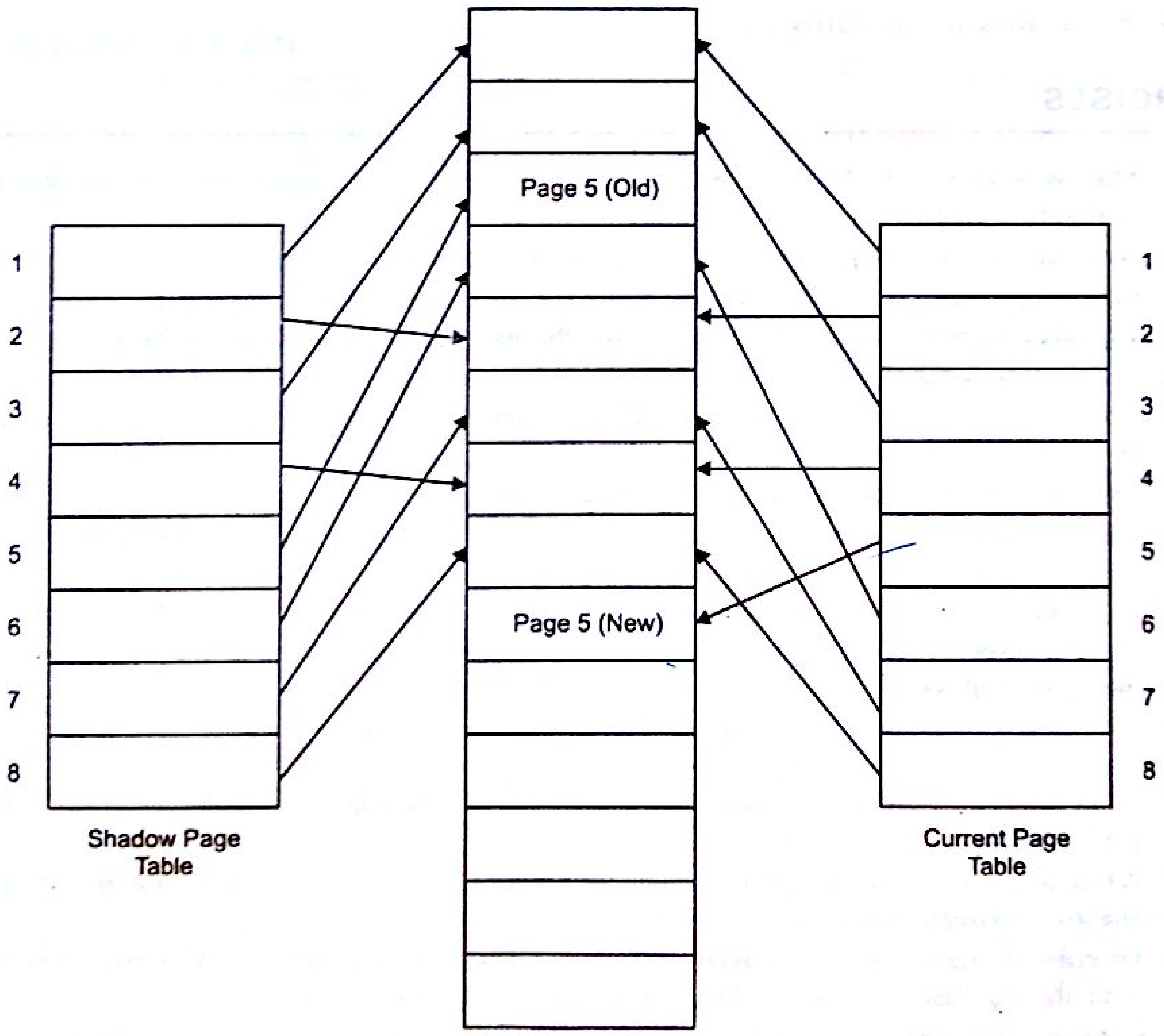


FIGURE 1(.6. *Shadow paging.*

After performing all the above operations, the following steps are needed to commit Ti.

1. Modified pages are transferred from main memory to disk.
2. Save current page table on disk.
3. Change current page table into shadow page table (by changing the physical address).

Shadow page table must be stored in non-volatile or stable storage because it is used at the time of recovery. Suppose the system crashed and you have to recover it. After every successful completion of transaction, current page table is converted into shadow page table. So, shadow page table has to be searched. For making the search simple, store shadow page table on fixed location of stable storage. No redo operations need to be invoked. Shadow page table and Current page table are shown in Figure 10.6 after write operation on page 5.

Disadvantages

1. Data fregmentation due to fixed sized pages.
2. Wastage of memory space.
3. Extra overhead at the time of commit (by transferring of page tables)
4. It provides no concurrency.

EXERCISES

1. Discuss recovery. What is the need of performing recovery ? Discuss the various methods of performing recovery.
2. Describe log-based and checkpoint schemes for data recovery.
3. Explain log-based recovery methods. What are the role of checkpoints in it. Discuss.
4. Explain anyone of the methods to recover the lost data in a database system.
5. Which type of failures can exist in a system ?
6. Compare the deferred and immediate-modification versions of the log-based recovery in terms of overhead and implementation.
7. Explain techniques of recovery of database in detail.
8. Discuss various data recovery issues. What are the techniques available for data recovery?
9. What is a log file ? What does it contain? How can a log file be used for recovery ? Describe this with the help of an example that includes all recovery operations (UNDO, REDO, etc).
10. How can you recover from media failure on which your database was stored ? Describe the mechanism of such recovery.
11. Why is recovery needed in databases ? How can log be used to recover from a failure? How is log-based recovery different from that of checkpoint based recovery ?
12. What do you understand by recovery and reliability in the context of database systems? Explain with the help of an example.
13. When is a system considered to be reliable ? Explain two major types of system failures. List any two recovery schemes.
14. Describe the shadow paging recovery scheme along with a diagram. Compare this scheme with the log-based recovery scheme in terms of implementation.
15. Compare and contrast the features of log-based recovery mechanism with checkpointing based recovery. Suggest applications where you will prefer log-based recovery schemes over checkpointing. Give an example of checkpointing based recovery scheme.
16. What is checkpoint ? Why is it needed ? What are the actions which are taken by DBMS at a checkpoint ? Explain with the help of an example.

Chapter **11**

QUERY PROCESSING AND OPTIMIZATION

11.1 INTRODUCTION

Query processing requires that the DBMS identify and execute a strategy for retrieving the results of the query. The query determines what data is to be found, but does not define the method by which the data manager searches the database. Therefore Query optimization is necessary to determine the optimal alternative to process a query. There are two main techniques for query optimization. The first approach is to use a rule based or heuristic method for ordering the operations in a query execution strategy. The second approach estimates the cost of different execution strategies and chooses the best solution. In general most commercial database systems use a combination of both techniques.

11.2 BASICS OF QUERY PROCESSING

Query Processing : Query Processing is a procedure of converting a query written in high-level language (*Ex.* SQL, QBE) into a correct and efficient execution plan expressed in low-level language, which is used for data manipulation.

Query Processor : Query processor is responsible for generating execution plan.

Execution Plan : Query processing is a stepwise process. Before retrieving or updating data in database, a query goes through a series of query compilation steps. These steps are known as execution plan.

The success of a query language also depends upon its query processor *i.e.,* how much efficient execution plan it can create? The better execution plan leads to low time and cost.

In query processing, the first phase is *transformation* in which *parser* first checks the syntax of query and also checks the relations and attributes used in the query that are defined in the database. After checking the syntax and verifying the relations, query is transformed into equivalent expression that are more efficient to execute. Transformation, depends upon various factors like existence of certain database structures, presence of different indexes, file is sorted or not, cost of transformation, physical characteristics of data etc. After transformation of query, transformed query is evaluated by using number of strategies known as *access plans*. While generating access plans, factors like physical properties of data and storage are taken into account and the optimal access plan is executed. The next step is to validate the user privileges and ensure that the query does not disobey the relevant integrity constraints. Finally, execution plan is executed to generate the result.

11.2.1 General Strategy for Query Processing

The general strategy for query processing is as follows :

(*i*) **Representation of query :** Query written by user cannot be processed directly by system. Query processor first checks the syntax and existence of relations and their attributes in database. After validations, query processor transform it into equivalent and more efficient expression for example query will be converted into a standard internal format that *parser can manipulate. Parser* also adds some additional predicates to the query to enforce security. Internal form may be relational algebra, relational calculus, any low-level language, operator graphs etc.

(*ii*) **Operator graphs :** Operator graphs are used to represent query. It gives the sequence of operations that can be performed. It is easy to understand the query represented by operator graphs. It is useful to determine *redundancy in* query expressions, result of transformation, simplify the view etc.

(*iii*) **Response time and Data characteristics consideration :** Data characteristics like length of records, expected sizes of both intermediate and final results, size of relations etc., are also considered for optimizing the query. In addition to this overall response time is also determined.

11.2.2 Steps in Query Processing

Various steps in query processing are shown in Figure 11.1. Suppose that user inputs a query in general query language say QBE, then it is first converted into high-level query language say SQL etc. Other steps in query processing are discussed below in detail :

(*i*) Syntax Analysis : Query in high-level language is parsed *into* tokens and tokens are analyzed for any syntax error. Order of tokens are also maintained to *make* sure that all the rules of language *grammars* are followed. In case of any error, query is rejected and an error code with explanation for rejection is returned to the user. (Only syntax is checked in this step).

(*ii*) Query Decomposition : In this step, query is decomposed into query blocks which are the low-level operations. It starts with the high-level query that is transformed into low-level operations and checks whether that query is syntactically and semantically correct. For example, a SQL query is decomposed into blocks like Select block, From block, Where block etc. Various stages in query decomposition are shown in Figure 11.2.

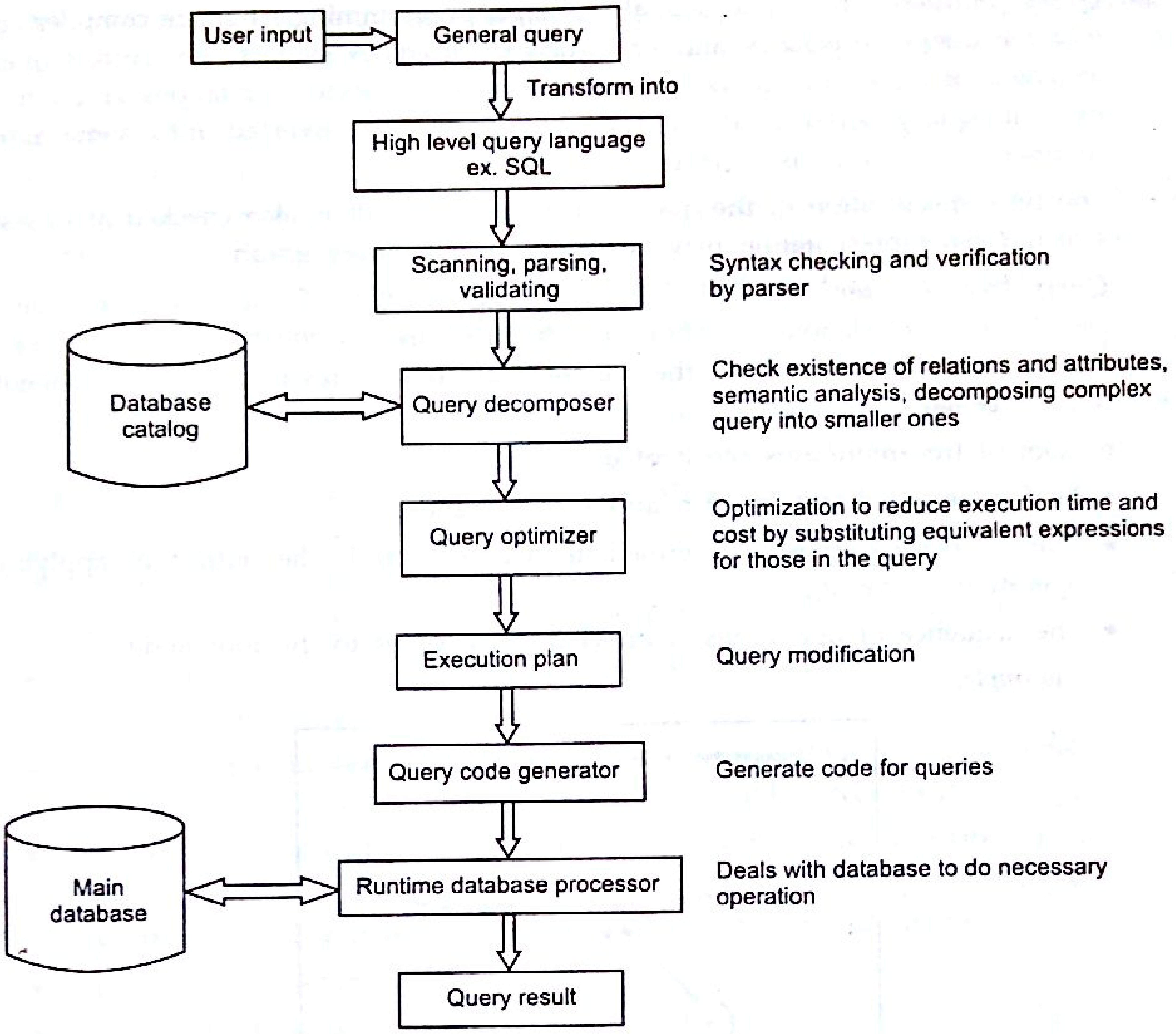

User input
General query
Transform into
High level query language ex. SQL
Scanning, parsing, validating
Syntax checking and verification by parser
Database catalog
Query decomposer
Check existence of relations and attributes, semantic analysis, decomposing complex query into smaller ones
Query optimizer
Optimization to reduce execution time and cost by substituting equivalent expressions for those in the query
Execution plan
Query modification
Query code generator
Generate code for queries
Main database
Runtime database processor
Deals with database to do necessary operation
Query result

FIGURE 11.1. *Steps in query processing.*

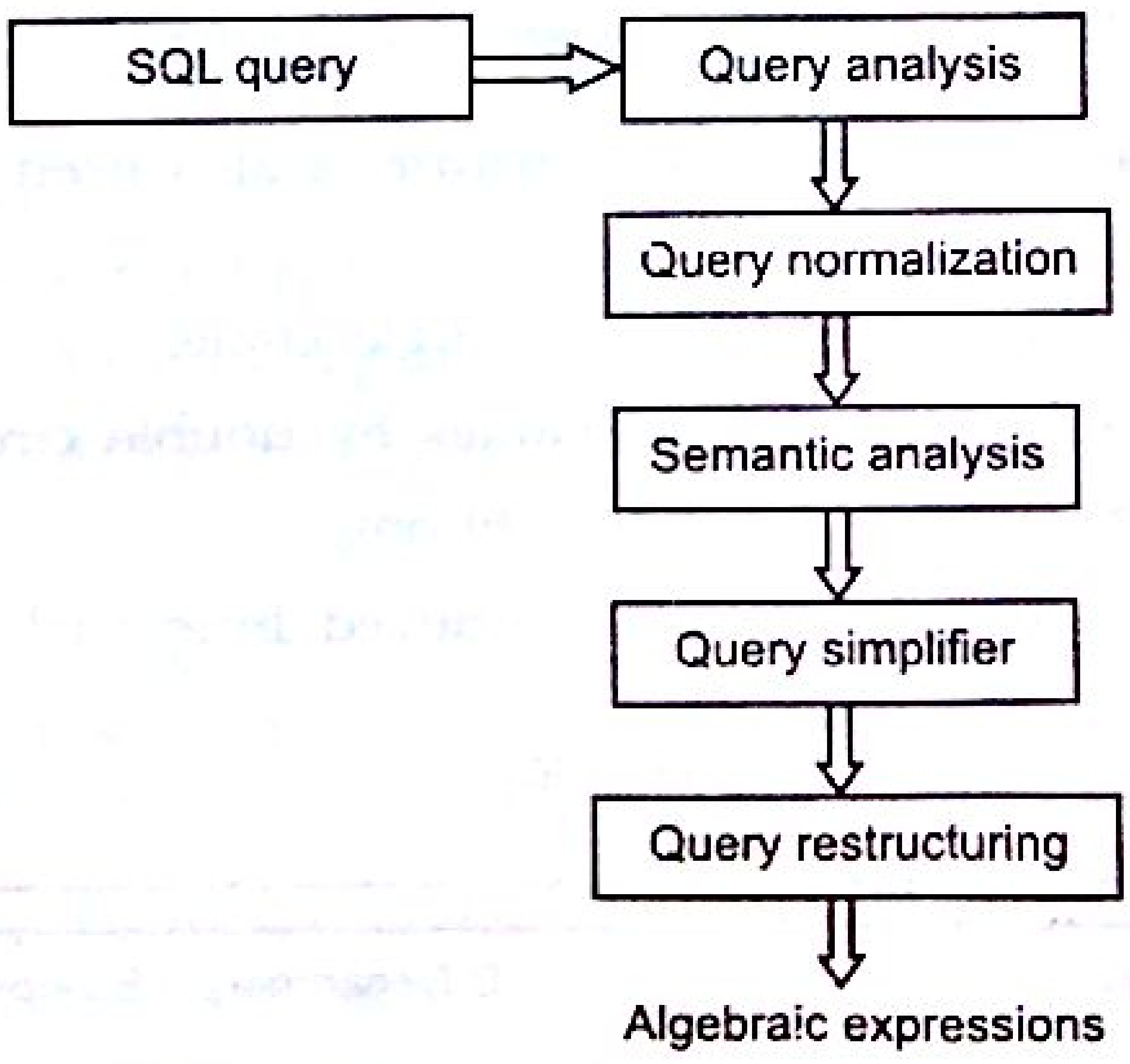


FIGURE 11.2. *Steps in query decomposition.*

(a) *Query Analysis :* In the query analysis stage, programming language compiler checks that the query is lexically and syntactically correct. A syntactically correct query is analyzed using system catalogues to verify the existence of relations and attributes used in query. After analysis a correct query is converted into some internal representation, which is more efficient for processing.

The type specification of the query qualifier and result is also checked at this stage.

The internal representation may be, query tree or query graph.

Query tree notation : A Typical internal representation of query is query tree. It is also known as relational algebra tree. A query tree is constructed using tree data structure that corresponds to the relational algebra expression. Main components of query tree are :

- Root of tree–represents result of query.
- Leaf nodes–represent input relations of the query.
- Internal nodes–represent intermediate relation that is the output of applying an operation in the algebra.
- The sequence of operations is directed from leaves to the root node.

For example,

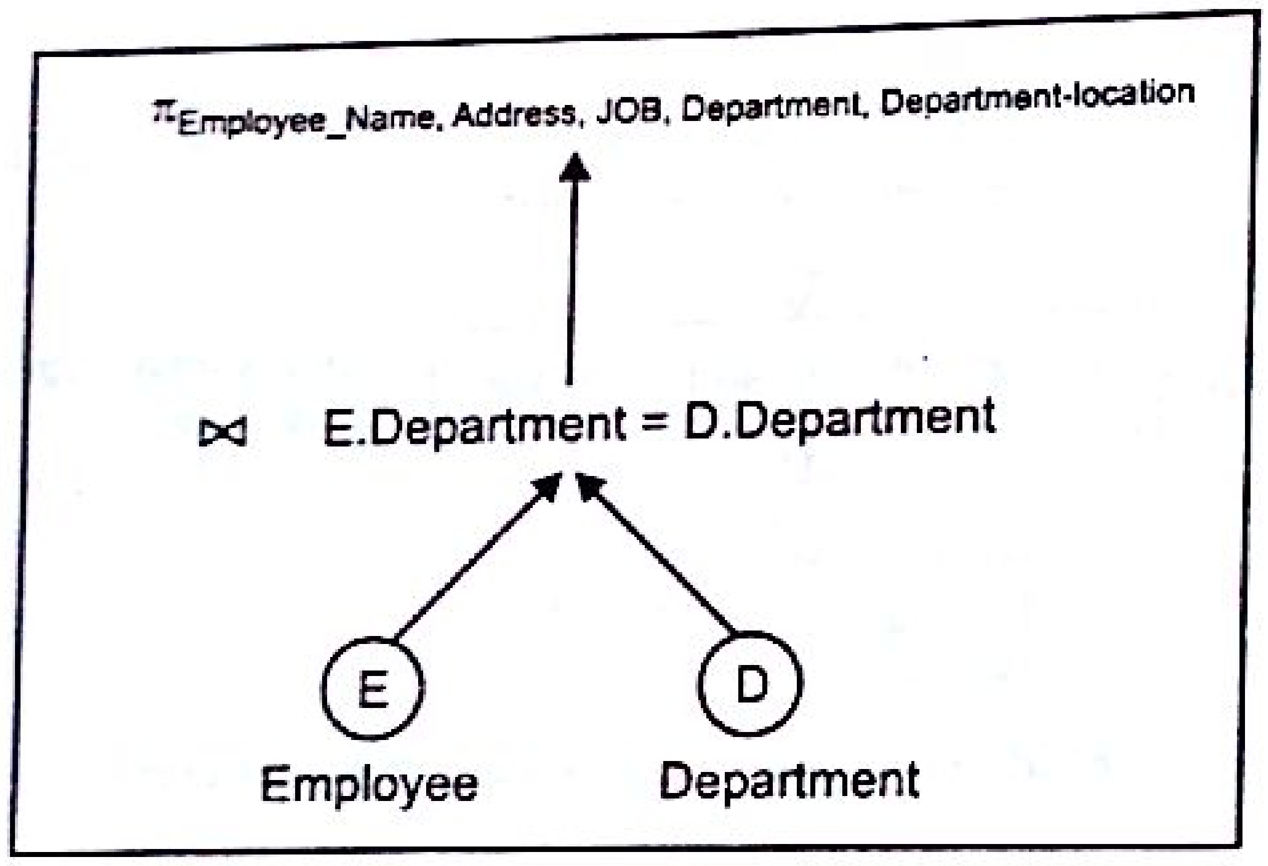

$\pi_{\text{Employee\_Name, Address, JOB, Department, Department-location}}$
$\bowtie$ E.Department = D.Department
E
D
Employee
Department

FIGURE 11.3. *Query treee notation.*

Query graph notation : Graph data structure is also used for internal representation of query. In graphs:

- Relation nodes–represent relations by single circle.
- Constant nodes–represent constant values by double circle.
- Edges–represent relation and join conditions.
- Square brackets–represent attributes retrieved from each relation.

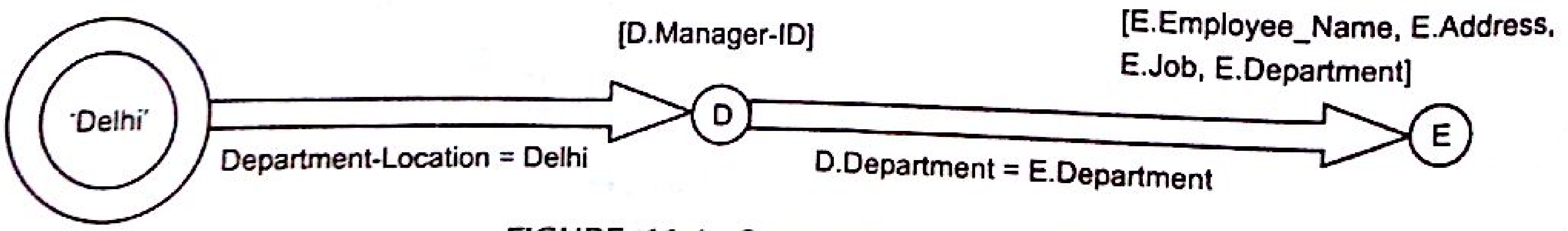

'Delhi'
[D.Manager-ID]
D
Department-Location = Delhi
[E.Employee_Name, E.Address, E.Job, E.Department]
E
D.Department = E.Department

FIGURE 11.4. *Query graph notation.*

(*b*) ***Query normalization :*** After query analysis, it is normalized to remove any redundancy. In this phase query is converted into normalized form that can be easily manipulated. A set of *equivalency rules* are applied to query to simplify the projection and selection operations to avoid redundancy. Query can be converted into one of the following two normal forms :

- *Conjunctive normal form :* It is a sequence of conjuncts that are connected with 'AND' operator. A *conjunct* consists of one or more terms connected with 'OR' operator. A conjuctive selection consists only those tuples that satisfy all *conjuncts.*

  Example. (Emp_Job = 'Analyst' $\vee$ salary < 50000) $\wedge$ (Hire_Date > 1–1–2000)

- *Disjunctive normal forms :* It is a sequence of disjuncts that are connected with 'OR' operator. A disjunct consists of one or more terms connected with 'AND' operator. A disjunctive selection contains those tuples that satisfy anyone of the disjunct.

  Example. (Emp_Job= 'Analyst' $\wedge$ salary < 5000) $\vee$ (Hire_Date > 1–1–2000)

Disjunctive normal form is more useful as it allows the query to break into a series of independent sub-queries linked by union.

(*c*) ***Semantic analyzer :*** The semantic analyzer performs the following tasks :

- It helps in reducing the number of predicates.
- It rejects contradictory normalized forms.
- In case of missing join specification, components of query do not contribute to generation of results. It identifies these queries and rejects them.
- It makes sure that each object in query is referenced correctly according to its data type.

(*d*) ***Query simplifier :*** The major tasks of query simplifier are as follows :

- It eliminates common sub-expressions.
- It eliminates redundant qualification.
- It introduces integrity constraints, view definitions into the query graph representation.
- It eliminates query that voids any integrity constraint without accessing the database.
- It transforms sub-graphs into semantically equivalent and more efficient form.
- It deals with user access rights.

Idempotence rules of Boolean Algebra are applied to get final form of simplification.

(*e*) ***Query Restructuring :*** At the final stage of query decomposition, transformation rules are applied to restructure the query to give a more efficient implementation.

(*iii*) Query Optimization : The aim of the query optimization step is to choose the best possible query execution plan with minimum resources required to execute that plan. Query optimization is discussed in detail in section 11.3.

(*iv*) Execution Plan : Execution plan is the basic algorithm used for each operation in the query. Execution plans are classified into following Four types : (*a*) Left-deep tree query execution plane, (*b*) Right-deep tree query execution plan, (*c*) Linear tree execution plan, (*d*) Bushy execution plan.

(a) ***Left-deep tree query execution plan :*** In left-deep tree query execution plan, development of plan starts with a single relation and successively adding a operation involving a single relation until the query is completed. For example, Only the left hand side of a join is allowed to participate in result from a previous join and hence named left-deep tree. It is shown in Figure 11.5.

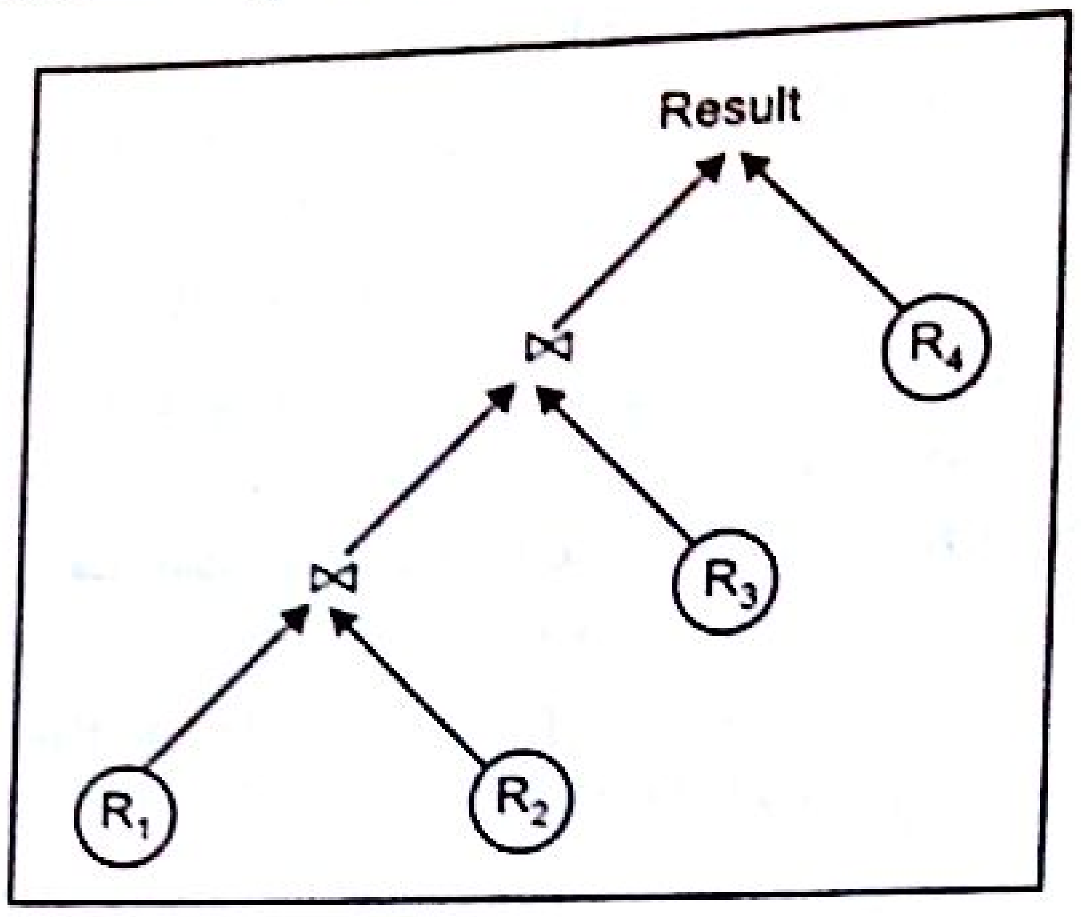


FIGURE 11.5. *Left-deep execution plan.*

Advantages

- It reduces search space.
- Query optimiser is based on dynamic programming techniques.
- It is convenient for pipelined evaluation as only one input to each join is pipelined.

Disadvantage

- Reduction in search space leads to miss some lower cost execution strategies.

(b) ***Right-deep tree query execution plan :*** It is almost same as left-deep query execution plan with the only difference that only the right hand side of a join is allowed to participate in result from a previous join and hence named right-deep tree. It is applicable on applications having a large main memory. It is shown in Figure 11.6.

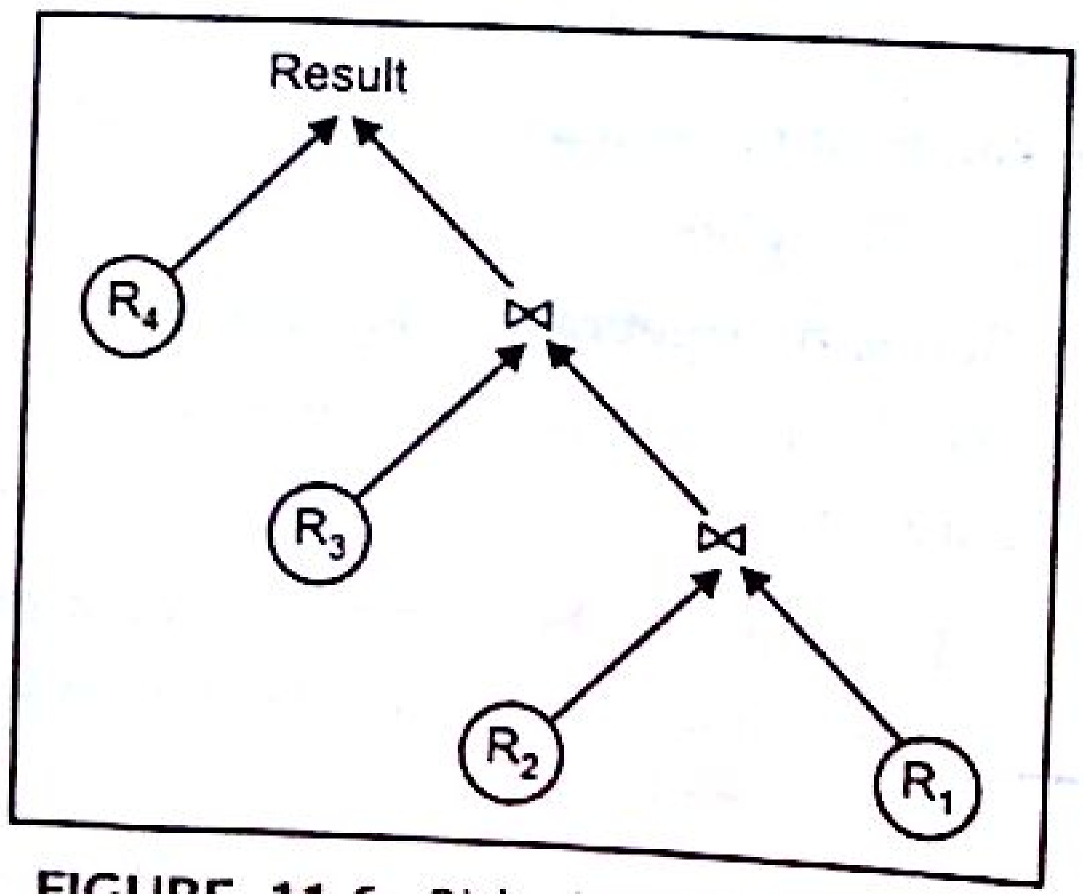


FIGURE 11.6. *Right-deep execution plan.*

(c) ***Linear tree execution plan :*** The combination of left-deep and right-deep execution plans with a restriction that the relation on one side of each operator is always a base relation is known as linear trees. It is shown in Figure 11.7.

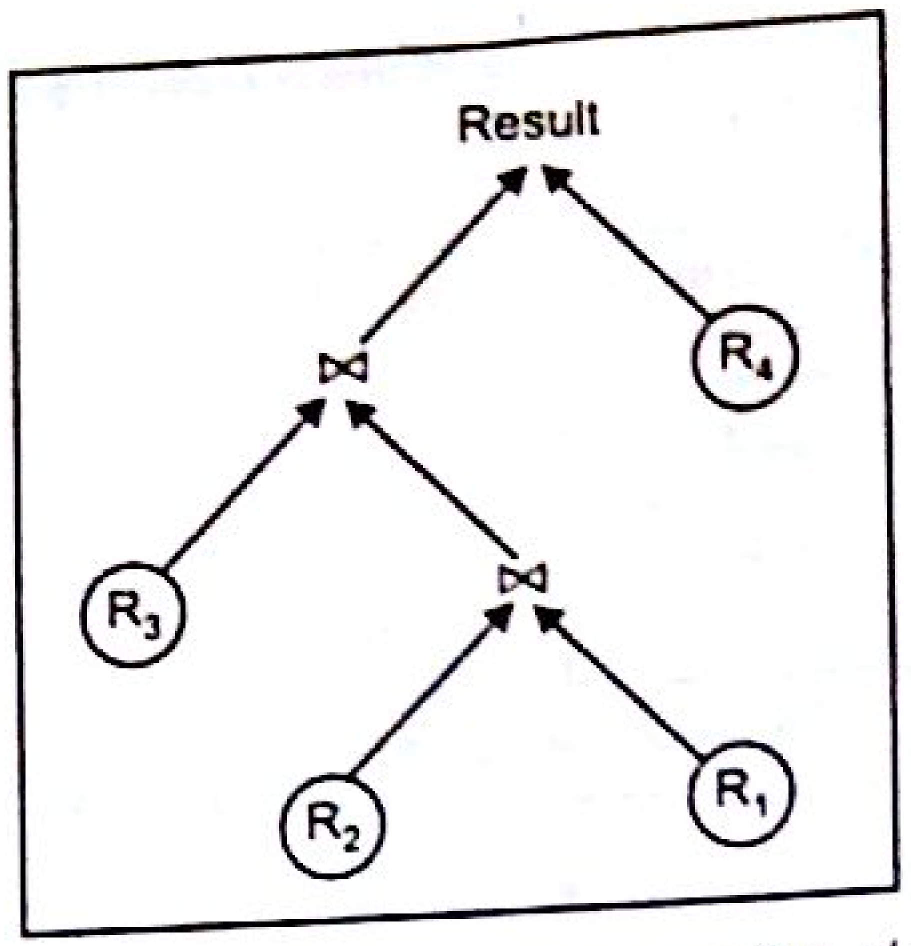


FIGURE 11.7. *Linear tree execution plan.*

(*d*) ***Bushy execution plan*** : Bushy execution plan is the most general type of execution plan. More than one relation can participate in intermediate results. It is shown in Figure 11.8.

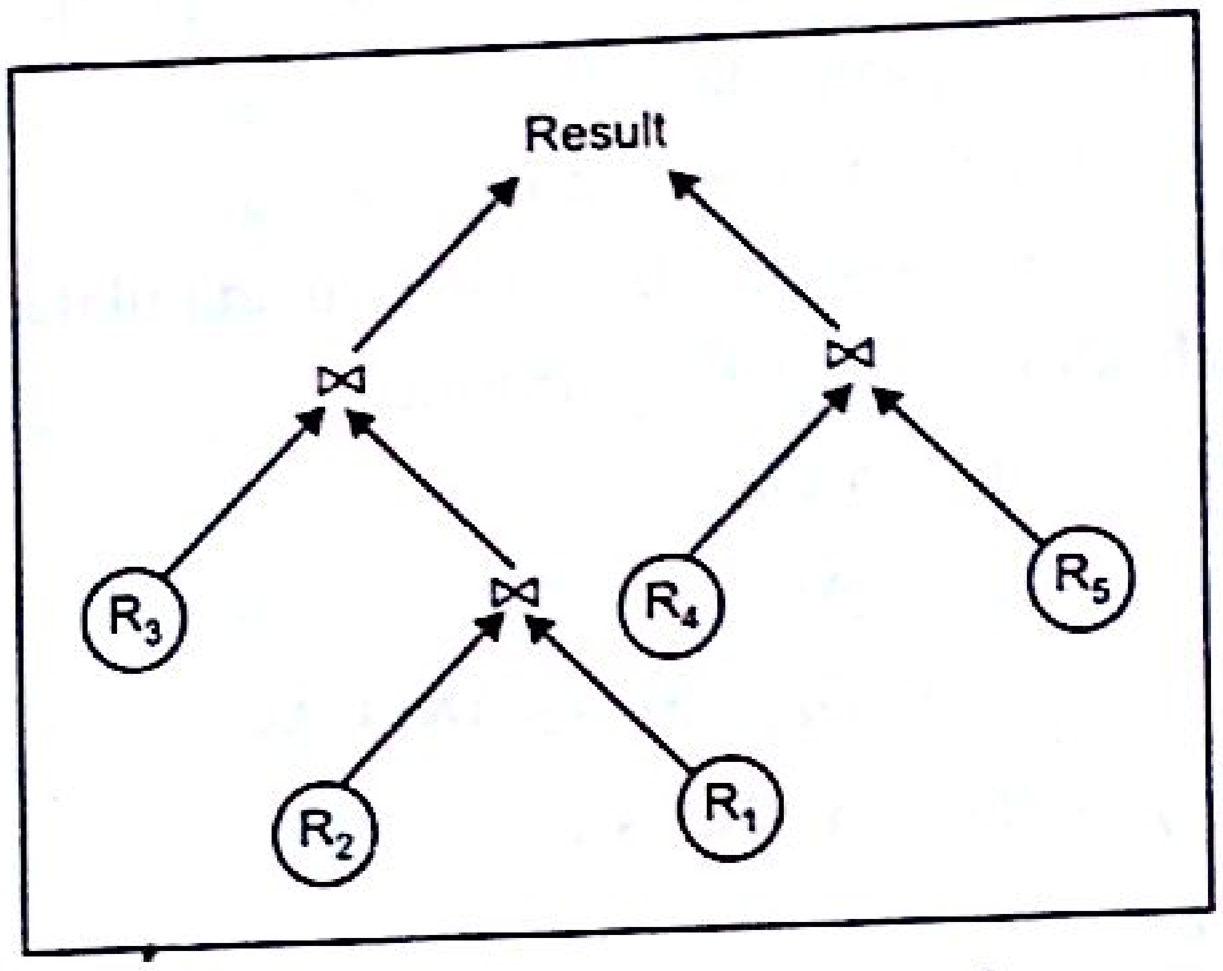


FIGURE 11.8. *Bushy execution plan.*

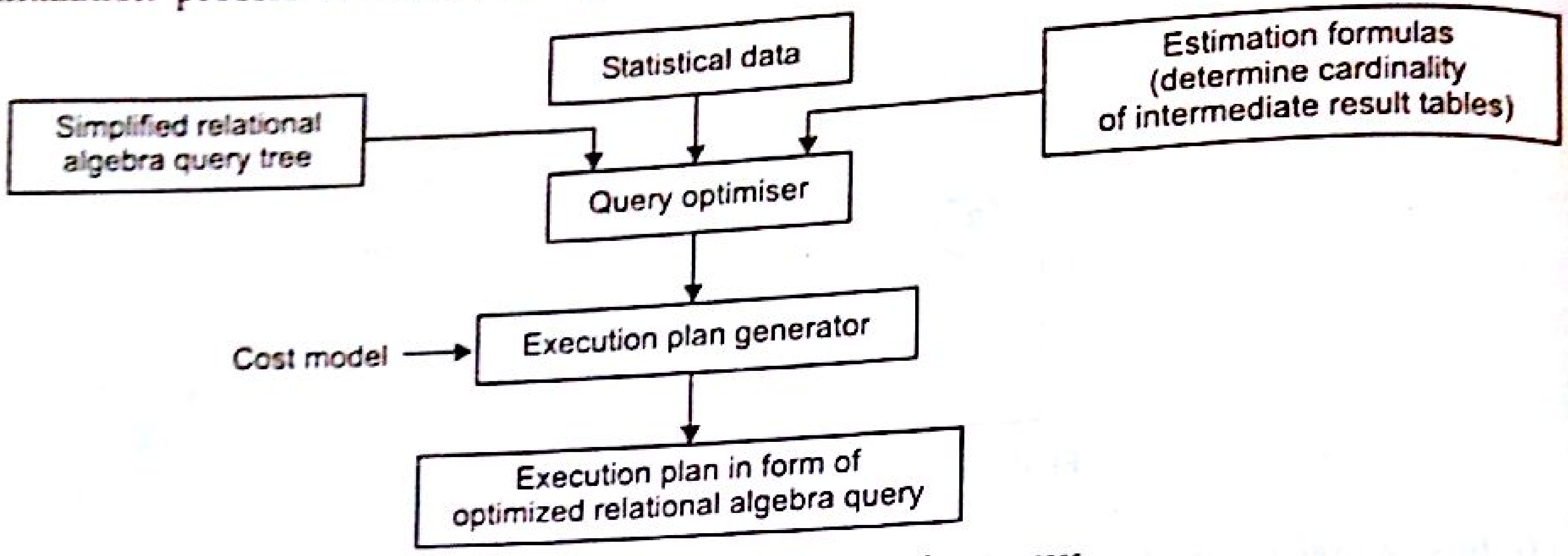
The main advantage of bushy execution plan is the flexibility provided by it in choosing the best execution plan by increasing search space but this flexibility may leads to considerably increase the search space.

(*v*) Query Code Generator: After selecting the best possible execution plan query is converted into low-level language so that it can be taken as input by runtime database process.

(*vi*) Runtime Database Processor : It deals directly with main database and do the necessary operation mentioned in query and returns the result to user.

11.3 QUERY OPTIMIZATION

Query performance of a database systems is dependent not only on the database structure, but also on the way in which the query is optimized. Query optimization means converting a query into an equivalent form which is more efficient to execute. It is necessary for high-level relation queries and it provides an opportunity to DBMS to systematically evaluate alterative query execution strategies and to choose an optimal strategy. A typical query optimization process is shown in Figure 11.9.



Statistical data
Estimation formulas (determine cardinality of intermediate result tables)
Simplified relational algebra query tree
Query optimiser
Cost model
Execution plan generator
Execution plan in form of optimized relational algebra query

FIGURE 11.9. *Query optimization process.*

The main issues that need to be considered in query optimization are:

1. Reduction of data transfer with database.
2. Use of available indexes for fast searching.
3. Reduction of number of times the database is manipulated.
4. The order in which joins should be performed.
5. How to store intermediate results.

Following are the three relations we used in each example:

Employee (Emp-ID, Emp-Name, Age, Salary, Dept-ID)

Department (Dept-ID, Proj-ID, Dept-Name)

Project (Proj-ID, Name, Location, Duration)

There are two main techniques used to implement query optimization. These are *heuristic query optimization* and *cost based query optimization.*

11.3.1 Transformation Rules for Relational Algebra

The transformation rules are used to formulate a relational algebra expression into different ways and query optimizer choose the most efficient equivalent expression to execute. Two expressions are considered to be equivalent if they have same set of attributes in different order but representing the same information.

Let us consider relations R, S, and T with set of attributes

$$X = \{X_1, X_2 \ldots X_n\},\ Y = \{Y_1, Y_2, \ldots Y_n\} \text{ and } Z = \{Z_1, Z_2 \ldots Z_n\}$$

respectively, where X, Y, and Z represent predicates and L, L_1, L_2, M, M_1, M_2, and N denote sets of attributes.

Rule 1. Cascading of selection (σ)

$$\sigma_{X \wedge Y \wedge Z}(R) \equiv \sigma_X(\sigma_Y(\sigma_Z(R))).$$

EXERCISES

1. What are the general strategies for query processing ? Explain. Also discuss query optimization techniques through suitable example.
2. What is meant by query processing ? Discuss the various methods for query processing and query optimization with the help of suitable examples.
3. What do you mean by query optimization ? Explain the various query optimization methods.
4. Explain the basic algorithm for executing query-processing operation with examples.
5. Discuss various Heuristic based query optimization techniques with examples.
6. Discuss the issues that are considered in designing of query optimizer.
7. What is query processing ? Describe the steps involved in query processing.
8. Describe how a query that involves join, selection and projection operations can be optimized. Explain the above with a suitable example.
9. Write the relational algebraic expression and create the query graph for the query. List the names of the employees who draw a salary above. Rs. 10,000 and are working on a project whose title is "Disaster management". Suggest query improvements in the query graph, if any.
10. A query-involving Join on three relations is to be executed. What factors will a DBMS consider to do evaluation of the query in an optimal fashion ? Explain with the help of an example.
11. Does the data dictionary have any role to play in query processing? Describe with the help of an SQL query requiring **Join** operation, SELECTION and PROJECTION.