UNIT 1 DATABASE MANAGEMENT SYSTEM – AN INTRODUCTION

Structure

1.0 INTRODUCTION

In the present time, most of your online activities require interaction with a database system running as the backend of an application, such as purchasing from supermarkets or e-commerce website, depositing and/or withdrawing from a bank, booking hotel, airline or railway reservation, accessing a computerised library, ordering a magazine subscription from a publisher, using your smartphone apps to purchase goods. In all the above cases a database is accessed. Most of these backend database systems may be called Traditional Database Applications. In these types of databases, the information stored and accessed is textual or numeric. However, with advances in technology in the past decades, different newer database models have been developed, which will be discussed in Block 4 of this course. In this unit, we will be introducing the architecture and structure of a traditional Database Management System.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- describe the File Based system and its limitations;
- describe the structure of DBMS;
- define the functions of DBA;
- explain the three-tier architecture of DBMS and its need; and
- appreciate different database models.

1.2 NEED FOR A DATABASE MANAGEMENT SYSTEM

A Database is an organised, persistent collection of data of an organisation. The database management system manages the database of an enterprise. Prior to use of the database systems, file-based systems were popularly used. In order to appreciate the strengths of database management systems, you may first list the problems associated with the file base systems, which are discussed next.

1.2.1 File Based System

Computerised file-based systems were primarily designed to make electronic versions of physical filing systems used by businesses. For example, a physical file can be set up to hold all the correspondence relating to a particular matter of a project, product, task, client or employee. In an organisation there could be many such files, which may be labeled and stored. Similarly, at homes you may have to maintain files relating to bank statements, loans, receipts, tax payments, etc. You can use electronic means to create these files. How do you search specific information from these files? For finding information, the entries could be searched sequentially. Alternatively, an indexing system could be used to locate the information directly, for example, you search for specific content in different files in your computer file search options.

The filing system works well when the number of items to be stored is small. It even works quite well when the number of items stored is quite large and they are only needed to be stored. However, a manual file system crashes when cross-referencing and processing of information in the files is carried out. For example, the University has several students who can register for different programmes of the University. The University has several faculty members who teach different courses to the enrolled students. The university may have to maintain separate files for the personal details of students, fees paid by them, and the details of the courses undertaken by them. In addition, the University also needs files for keeping details of each faculty member in various departments, courses taught by them. How would the following queries be answered?

- Fee paid by the students of the Computer Science Department per annum.
- The students who are willing to avail the pickup bus facility.
- Number of students taught by a faculty in a specific academic period.
- The number of students who have passed the programme this year in comparison to earlier years.
- How many students of a specific department have registered for courses other than their own department?

Please refer to *Figure 1*. The answer to all the questions cannot be computed by just simple statements but will require extensive file processing. Thus, each of these queries will require substantial amount of time in the file-based system.

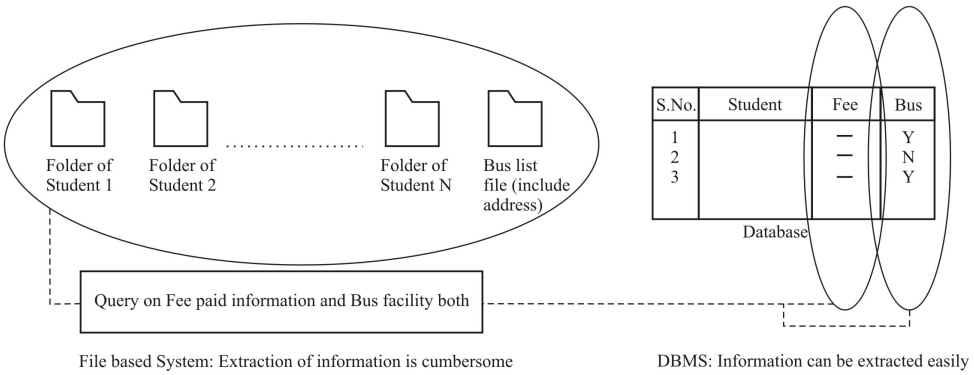


Figure 1: Information extraction using a file based system

1.2.2 Limitations of File Based System

File based systems required that several files should be opened for a particular system application. Several files may consist of duplicate data, which can result in several shortcomings. Some of these shortcomings are listed below:

- **Data isolation**: Since the file system stores data in separate files, which may belong to different applications. These files are not accessible to other applications and are difficult to share, especially when an application needs to use more than one file. Also, as the number of files can be very large for such systems, therefore, it would be difficult to search the relevant data from these files.
- **Data Duplication:** As stated earlier, a file system has different files for different applications, which may have overlapping data requirements. This will result in duplication of data, which can result in inconsistent data when duplicate data is updated. In addition, data duplication can also result in waste of storage. An example of data duplication is shown in Figure 1, where the address of several students may be stored in two different files, viz. student folder and bus list file.
- **Inconsistent Data:** The data in a file system can become inconsistent if more than one person modifies the data concurrently, for example, if any student changes the residence and the change is notified to only his/her file and not to the bus list. Entering wrong data is also another reason for inconsistencies.
- **Data dependence:** In the file systems, you need to clearly define the storage organisation of data files and the structure of the records in the application code. This means that it is extremely difficult to make changes to the existing structure, as any change in structure would require change in all the programs using that structure of the data. The programmer would have to identify all the affected programs, modify them and retest them. This characteristic of the File Based system is called **program data dependence.**
- **Incompatible File Formats:** Since the structure of the files is embedded in application programs, the structure is dependent on application programming languages. Hence the structure of a file generated by COBOL programming language may be quite different from a file generated by 'C' programming language. This incompatibility makes them difficult to process jointly. The application developer may have to develop software to convert the files to some common format for processing. However, this may be time consuming and expensive.
- **Fixed Queries:** File based systems are very much dependent on application programs. Any query or report needed by the organization would require the

application programmer to write a new program. As the type and number of queries or reports are expected to increase with time, producing different types of queries or reports would be difficult to implement in file-based systems. Since applications of file-based systems are designed to answer specific queries, therefore, new queries cannot be answered without generating an application. This entire process is time consuming and complex.

The file-based systems require large number of applications, therefore, are difficult to maintain. Further, each application would require separate provision from security. Besides the above, the maintenance of the File Based System is difficult and there is no provision for security. Further, data recovery from failures is inadequate or non-existent.

1.2.3 The Database Approach

As discussed in the previous section, the file system has many weaknesses. Therefore, a new approach was proposed that eliminates the weaknesses of the file system. This approach, called the database approach, separated data from application programs. The data in this approach is integrated from various applications and securely shared using a management system. *A database stores the integrated data of an organisation in a persistent manner.* The following are some of the characteristics of the database approach:

- The database can store data in a centralised database or a distributed database.
- The data is managed by a database management system (DBMS)
- It contains additional data about data, called metadata, which describes the structure and constraints on the stored data. The metadata is stored in DBMS in a data dictionary or system catalog.
- The database integrates the data of an organisation. This data is shared under the control of DBMS in a secure manner.
- DBMS allows several basic operations related to data, such as creating a database structure, inserting and editing data in a database, enforcing security and constraints on data and allowing access to data to authorised users.

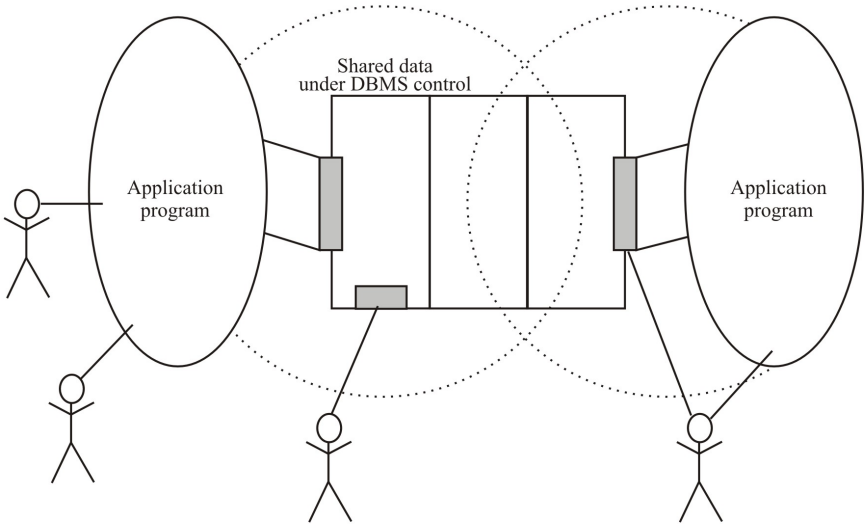
Let us discuss these advantages of database systems in more detail.

Reduction of Redundancies

The data of a database system is integrated, therefore, removes any duplicate data, as was the case of a file-based system, which maintains different files for different applications. The files that contain the copy of the data may become inconsistent as some of the files may be updated, whereas others may not be. The reduction in duplicate data may also help in reducing such data inconsistencies among the duplicated data of files. Thus, in the database approach, data is stored at a single place or with controlled redundancy under the control of DBMS, which helps in reducing data inconsistency.

Sharing of Data

The data of a database is integrated data of the entire organisation, however, the authorised users may be permitted to share partial data (why?). This scheme can be best explained with the help of a logical diagram (*Figure 2*). New applications can be built and added to the current system and data can be shared with these data, as per the need of such applications. Please note that data sharing is controlled by the DBMS, such that only the authorised users or applications can access it.



A user can either access data window through DBMS or use an application program.

Figure 2: User interaction to data through DBMS

Data Independence

In the file-based system, the descriptions of data and logic for accessing the data are built into each application program making the program more dependent on data. A change in the structure of data may require alterations to programs. Database Management systems separate data descriptions from data. Hence it is not affected by changes. This is called Data Independence, where details of data are not exposed. DBMS provides an abstract view and hides details. For example, in Figure 2, you can observe that the interface or window to data provided by DBMS to a user may still be the same although the internal structure of the data is changed.

Improved Integrity

Data Integrity refers to validity and consistency of data. Data Integrity means that the data should be accurate and consistent. This is implemented by enforcing checks or constraints on the data while it is being entered or manipulated in a database. Data of a database is not allowed to violate these constraints or rules. Constraints may apply to data items within a record or relationships between records. For example, a constraint on the age of an employee can be between 18 and 70 years only. While entering or modifying the data of the age of an employee, the DBMS should enforce this constraint. However, please note there is still a possible error if you enter the age of a person as 55 instead of 25. Such errors would require different ways of checking. Database systems support many other types of constraints, which will be discussed in later units.

Efficient Data Access

DBMS utilises techniques to store and retrieve the data efficiently, at least for unforeseen queries. An advanced DBMS allows its users to access data efficiently.

User Interfaces as per Users technical knowledge

A DBMS allows different types of interfaces, based on different levels of their technical knowledge. For example, the following types of interfaces may be provided by a DBMS:

- Menu driven forms and reports-based interfaces
- Application programming interface
- Query language interfaces
- Natural language interfaces

Representing relationship among data

Data of a database system is integrated data of an organization, which includes large number of related data objects. DBMS should maintain and preserve these relationships so that related data can be accessed easily.

Improved Security

The data of an organisation is vital and confidential. DBMS allows users to share only that information that they are authorised to access. For example, the critical information of an employee of an organisation should not be accessible to any other employee of that organisation. Hence, data of the database should be protected from unauthorised users. This is implemented by Database Administrator (DBA), who provides the users with controlled privileges on the data and type of operations. To enforce security, DBMS has a security and authorisation subsystem. Only authorised users may use the specific data of a database. Further, the operations performed by these uses on data can be restricted to data retrieval, insertion, update, deletion etc. or any combination of these operations. For example, the Branch Manager of any company may have access to all data and is allowed to modify the incentive data of employees, whereas the Sales Assistant may not have access to salary details.

Improved Backup and Recovery

A file-based system may fail to provide measures to protect data from system failures, as taking backups periodically is the responsibility of the user. However, most of the DBMSs are designed in such a manner that it can recover from different types of hardware failures. In addition, DBMS also supports data backup. In general, a DBMS has a subsystem to support such provisions.

Support for concurrent Transactions

A transaction, in the context of a database system, is an atomic operation that is either completed fully or not at all. A database should allow multiple transactions to be carried out at the same time. For example, in a bank several money withdrawal and money transfer operations may be carried out at the same time. Most of the commercial DBMSs ensure that all these transactions do not interfere with each other.

Check Your Progress 1

1) What is a DBMS?

……

……

2) What are the advantages of a DBMS?

……

……..

3) Compare and contrast the traditional File based system with Database approach.

……

……..

1.3 THE LOGICAL DBMS ARCHITECTURE

As discussed in the previous section that most of the advantages of database systems are because of the creation of DBMS software. DBMSs must support many services and therefore are complex in nature. In addition, DBMSs are also required to store, manipulate and control a very large amount of data in a reliable manner. In this and subsequent section, we discuss the architecture of DBMS, which will help you with the processes and features of a DBMS.

In this section, two different architectures, which deal with two different aspects of database management, are discussed. The first architecture is the logical architecture,

which defines the data organisation and access at different logical levels of a database. The second architecture defines various components of a DBMS software. This architecture is referred to as physical database architecture.

1.3.1 Three Level Database Architecture

The three-level database architecture of a database defines the three different levels of abstraction of data for different types of users of the database. The proposed architecture was designed and standardised by the American National Standards Institute (ANSI) and is also known as ANSI/SPARC architecture. As per this architecture, a database schema can be visualised at three different levels. Figure 3 shows these three levels of this architecture. These levels are explained next.

The External or View Level

This level of abstraction provides a view of data for the users of a database system. Typically, this abstraction is created based on access rights of the users. Different types of users can be allowed different external views of data, as shown in Figure 3. Users can have different views of data. This level hides the overall structure of a database system from external users. From the database designer's point of view, this level also provides information on the mapping of external records of external views to the conceptual record of conceptual view of data.

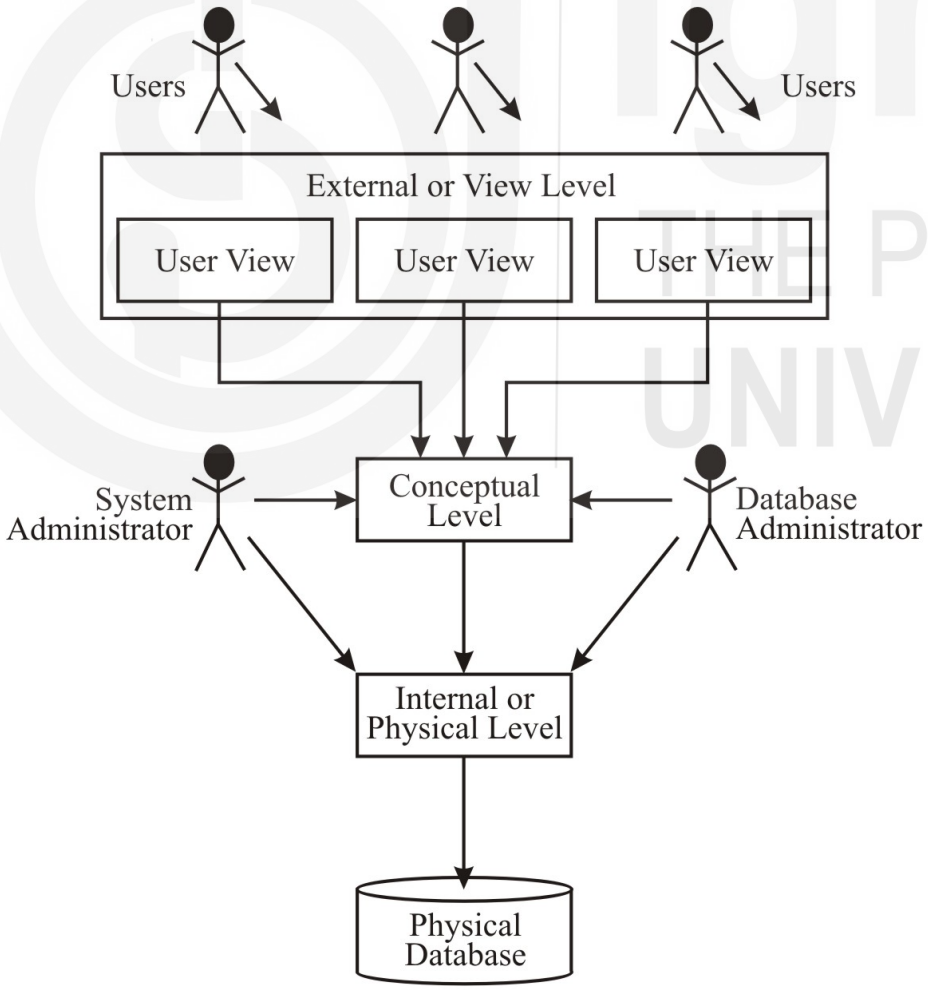


Figure 3: Logical DBMS Architecture

The Conceptual Level

The purpose of the conceptual level is to define the structure, relationships and constraints of a database system. Therefore, the conceptual level includes the structure of the data objects, relationships among those objects and the constraints on the data objects and access control of objects. A standard language called Data Definition Language (DDL) is often used in DBMSs to define the conceptual level or schema of a database system. The conceptual schema can be defined by the database administrator or the system administrator, as shown in Figure 3.

The Internal or Physical Level

A database system consists of many files, which include the data files, the meta data files, the access structure files, etc. The data files contain the actual data. The metadata files contain structure related information of data files, relationship details among data objects, constraints on data items etc. The access structure files define the control related information of the data objects. These files can be controlled by the DBMS software for access and updates. DDL can be used to describe the internal or physical schema too. In addition, this level can store additional files, which are used for enhancing the performance of the database system. These files are stored on an operating system, however, the access to these files is also under the control of the database system administrator.

1.3.2 Mappings between the three Levels and its relationship with Data Independence

The three level architecture defines a single database system across all the three levels. Therefore, different levels must map with each other. This mapping led to the concept of data independence, which was one of the major weaknesses in the file systems. This concept is explained next.

The first mapping is between the conceptual level and the external level. The external level is derived from the conceptual level. It is a part of the conceptual level, however, please note that these parts must be related else the database will lose database integrity. The advantage of this mapping is that an external user only needs to see the external level any change in the conceptual level will be hidden from the user. For example, at the conceptual level, you may keep information about the name of a person using the data items like *title, firstname* and *lastname*. At the external level, you may just map it to a data item *name* field. Thus, there exists a mapping, which will map:

name = title || *firstname* || *lastname* (|| is a concatenation operation)

Suppose, at a later point you decide to add additional data item *middlename* in the conceptual level, then you just need to change your mapping and not the programs which are based on the external level. The mapping would be changed to:

name = title || *firstname* || *middlename* || *lastname*

Thus, the conceptual to external mapping may isolate the external users from the changes in the conceptual level. These changes can be hidden in the conceptual to internal mapping. This is also referred to as logical data independence.

The second mapping is between the physical level and the conceptual level. This mapping insulates the conceptual level from changes in file organisation, indexes etc., without changing the conceptual level. In general, such changes may be performed to enhance the performance of a database management system. For example, instead of organising the student file on enrolment number, which is the primary key, you may organise the student file on Programme + Student name. This may enhance the access time of data for many important queries to the database. The conceptual level, in this case, still remains unchanged. This is also referred to as physical data independence.

1.3.3 Objective of the three-level architecture

The three level architecture separates the data that is presented to the user from the data that is stored in the database physically. The basic objectives of three level architecture are:

- It can support different views for different users. In case of change in any application, the views related to that application are required to change. Thus, three level architecture ensures the independence of data and programs.
- As stated above, due to independence of data and application programs, the applications need not deal with physical file organisation. Therefore, the application programs and users of a database are provided with a higher level of abstraction of data. Thus, a user or application programs are not required to deal with conceptual schema and the physical schema of a database. In general, the Database Administrator (DBA) is responsible for creating and modifying the conceptual schema and physical schema using DDL.

1.4 PHYSICAL DBMS ARCHITECTURE

A Database Management System (DBMS) is a complex software, as it manages many aspects of the database, such as query languages, data integrity, data security, access to files, and performance of data access. Figure 4 shows some of the important software components of a DBMS, which should be present logically in most of the DBMSs of today.

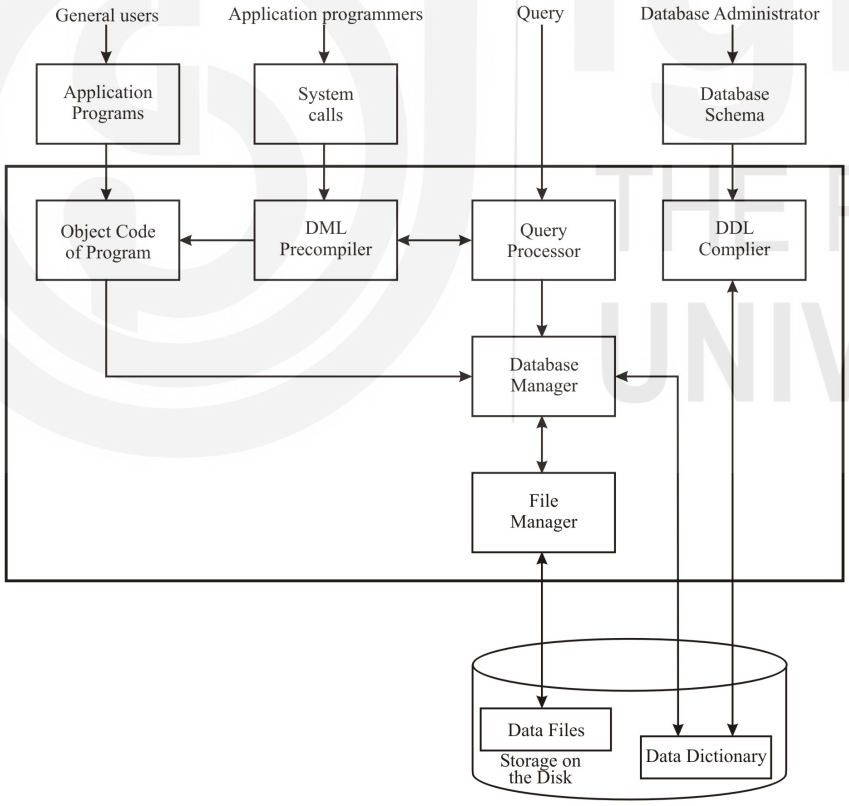


Figure 4: DBMS Structure

The software components of DBMS, as shown in Figure 4 are discussed next.

1.4.1 DML Precompiler

A DBMS consists of several languages, which are used for defining the data and for manipulation of data. For data definition, as stated earlier, a language called Data Definition Language (DDL) is used. Using DDL, you can define the structure of the data objects, data type of different data elements, integrity constraints, storage

constraints and access rights on the data. In addition, every DBMS consists of a Data Manipulation Language (DML), which is used for data insertion, modification, data access etc. The purpose of the DML precompiler is to translate the DML commands, which are written as part of application programs, into related procedure calls so that they can be executed to perform the desired operation. In addition, this component of the DBMS also coordinates the translation of queries, which are input to the query processor component.

1.4.2 DDL Compiler

A DDL compiler is used to compile a DDL statement into the related tables, which include the metadata, constraints, access rights etc. The metadata tables are stored in the data dictionary or system catalog. This metadata is used for providing information about the tables to other components of DBMS. In addition, the meta data is used for protecting data and enforcing data integrity.

1.4.3 File Manager

All the tables and metadata of the database are physically stored as files under the control of the file sub-system of the operating system. The file sub-system of the operating system has generic features, which may not be sufficient for a DBMS file. The File Manager component of the DBMS keeps track of all the data files, their index files and all other related information files of a database system. The final input/output to the database files are performed by the operating system.

1.4.4 Database Manager

Database Manager is one of the most important components of the DBMS. The role of the database manager is to accept the requests of data access or data manipulation by the user queries and application programs and perform the relevant action on the data taking the help of data dictionary and file manager. Database manager protects the data from unauthorised access and manipulation. It also enforces integrity constraints. You may please note that a database contains a large amount of data. The database manager is also responsible for deciding which data should be brought from secondary storage to main memory and back. Database manager allows simultaneous transactions on data and is also responsible for the recovery of the database in case of failures. It is also responsible for enhancing the performance of a database system access. The roles of a database manager are as follows:

- **Collaborating with file manager for file handling:** As explained earlier, the file sub-system of the operating system is responsible for storing or manipulating data and related files of a database system. The file manager component of the DBMS interacts with the operating system for this purpose. The database manager collaborates and controls the file manager for various file operations.
- **Integrity enforcement:** Data integrity relates to correctness of data in a database system. A DBMS forces several constraints on data, which allows only correct data to be stored in a database system. These constraints are stored in the data dictionary and the database manager uses a data dictionary to enforce these integrity constraints. You will learn more about integrity constraints in the later units.
- **Security enforcement:** A DBMS does not allow unauthorised users to access the data of a database. This access control can relate to the entire data or a specific item of the data. For example, as an employee of an organization, you can read only your salary information. You do not have permission to change your salary. In addition, you cannot read or change the salary of any other employee. The security constraints can also be part of the data dictionary and implemented by the database manager.
- **Backup and recovery:** The purpose of the backup and recovery component of data manager is to guard the database against the loss of data at the time of a failure of the computer system. The failure of a computer system can occur due to many different reasons, such as hardware failure communication failure

software failure etc. The database manager ensures that data of various transactions is not corrupted even after a failure.

- **Concurrency control:** One of the major uses of database systems is in transaction management. In such systems, multiple transactions access and modify the database simultaneously. The database manager ensures that consistency of data is ensured even in the presence of concurrent transactions. Transaction management is discussed in block 3 in detail.

To perform the roles or functions as discussed above, the database manager has the following software components. These components are also shown in Figure 5.

- Authorisation control is used to verify the credentials of a user to ascertain if s/he is an authorised person to access or modify the data.
- Command Processor converts the DDL/DML or other programming language commands to an executable sequence of code.
- Integrity checker checks if the data that is being inserted in a database or is being modified follows all the specified constraints on the database.
- Query Optimizer tries to optimise the processing time of different queries.
- Transaction Manager is one of the important components, which checks if the user has the right to perform a transaction.
- Scheduler component manages the consistency of data, while concurrent transactions are being processed by a database. The scheduler ensures an ordering of transaction execution, even though they may be conflicting with each other.
- Recovery Manager is responsible for the recovery of a database system to a consistent state, in case of a failure.
- Buffer Manager optimises the process of transfer of data between the main memory and the disk, where the database is stored. The buffer manager is also termed as the *cache manager*.

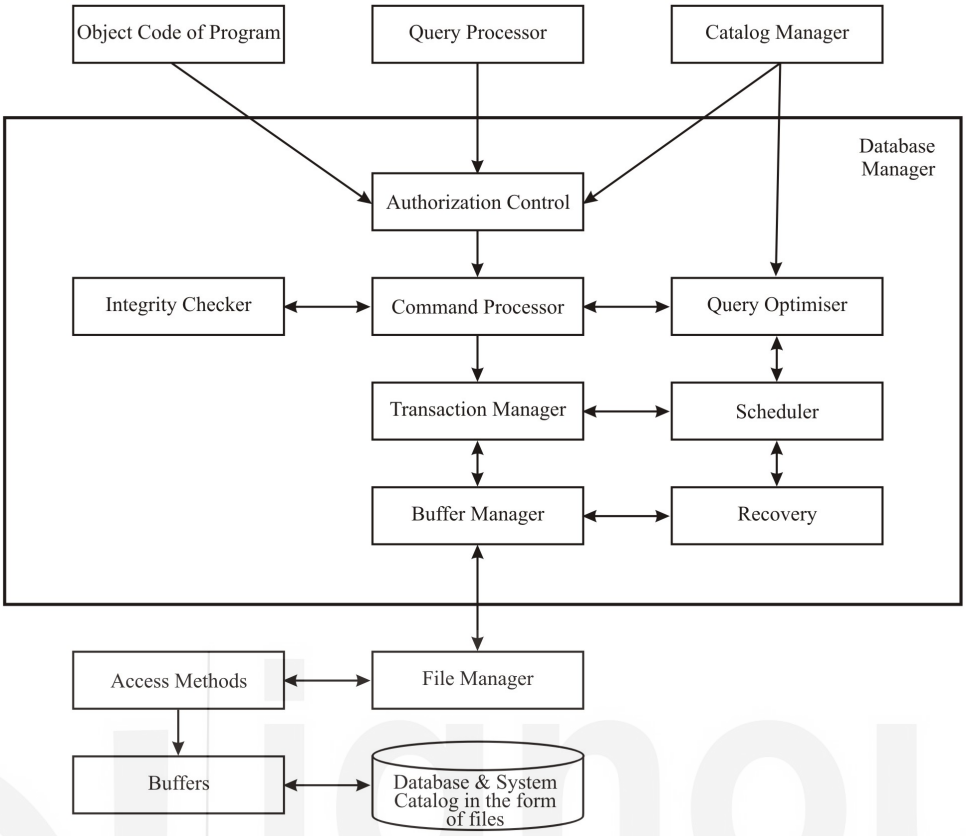


Figure 5: Components of Database Manager

1.4.5 Query Processor

Every DBMS consists of a query language which enables database creation, data manipulation, data retrieval and data control operations. This query language is usually a high level 4th generation language and is translated and optimised so that the required operations can be handled by the DBMS in the most optimal manner. In general, this query language processor consists of two basic components - the first one parses the query language into the native database language and the second optimises the query and prepares the code for query execution. The parser checks the syntax of the query, converts the query to a sequence of tasks and sends these tasks to the query optimiser. The query optimiser is responsible for generating a set of alternative query evaluation plans. It then estimates the expected total query response time, which includes access time of data from the disks, the time to execute a query and the time of communication of data or results over the network, for each of these query evaluation plans. Finally, the best query evaluation plan is selected to execute the query efficiently.

1.4.6 Database Administrator

The role of a database administrator (DBA) is to monitor the database creation (at different levels of 3-level architecture and mapping between the levels), modification, access, query response time, security, recovery in case of failure, etc. The functions of a DBA are given below:

- Defines the database Schema at different levels using DDL
- Specifies the organisation of the files and any other database access methods, including indexes, using DDL
- Performs the changes in the schema definitions, file organisation or indexes, whenever needed.
- Allows authorisation of data access or modification to different users.
- Specifies the integrity constraint on the database.

1.4.7 Data files, Indices and Data Dictionary

The data is stored in the data files. Data files can be standard files or encrypted files. These files, in general, are not accessible directly to any system user. The indices are stored in the index files. Indices provide fast access to data items. For example, a book database may be organised in the order of accession number of books yet may be indexed on Author name and Book titles for allowing faster access for searches on these attributes.

Data Dictionary: The data of an organisation is a valuable corporate resource. Commercial databases have large number of entities, attributes, data interrelationships and other information about the data. A data dictionary/directory is designed to store metadata, which is data about the data in a database system. A good data dictionary allows efficient data access, as it can provide a road map to guide users to access information within a large database. An ideal data dictionary may include the following information about a database:

- Schema definitions of internal, conceptual and external schemas, which includes:
  - the definition of every object or table
    - the attributes and their data types including primary key and attribute constraints.
    - any relationships between different attributes including foreign key constraints etc.
    - authorisation of access or modification at the level of attributes, if any.
  - Authorisations at the level of objects/tables, if any.
- Users and application programs of external schema and their permissions
- Database statistics such as number of records in every object/table, different number of occurrences of a data value in an attribute etc.

Check Your Progress 2

1) What are the major components of Database Manager?

   ……

   ……

   ……

2) Explain the functions of the person who has control of both data and programs accessing that data.

   ……

   ……

1.5 DATABASE SYSTEM ARCHITECTURES

In the present time several database system architectures are popular.

First and earliest architecture of the database system was having a centralised database system. These systems contained a database in a single machine, which was either used by a single user or were shared by several users. A single user application in the present time may be an address book of your mobile device that is being used by a single user. However, today most of the centralised database systems use multiple core processors having large memory and multiple disks, called the database servers. A large number of users are connected to these servers, mostly through remote connections. These systems fall under the category of client-server systems. At its most basic level, this client server database architecture can be broken down into two parts: the *back end* and the *front end*.

The back end are the servers, which are responsible for managing the physical database and providing the necessary support and mappings for the internal, conceptual, and external levels. Other functions of a DBMS, such as security, integrity and access control, are also the responsibility of the back end.

The front end is just any application that runs on the DBMS. These may be applications provided by the DBMS vendor, the user, or a third party. The user interacts with the front end. A front-end may be user friendly interface provided by an application developed for database system access and modifications. These applications may be developed by the vendors themselves or by software developers using an Application Programming Interface (API) or third-party applications.

In this context, many different types of interfaces and utilities are offered in support of commercial database management systems. Some of these are:

- **Interfaces:**
  - **Command line Interface:** This interface existed since the start of DBMS. You can use this interface to write DDL, DML and other permitted commands. These interfaces allow a very rich set of commands and are useful for expert users like DBA.
  - **Graphical User Interface:** Such interfaces are developed to allow users to interact though database applications using menu driven interfaces. Such interfaces have become more useful over the last decade.
- **Utilities for Backup/Restore:** The purpose of these utilities is to ensure that the current state of a database is duly backed up on secondary storage, so that it can be recovered or restored in the case of any failure or even disaster. The backup is done periodically.
- **Utilities for Load/Unload of data:** Life of a database system is quite long. The size of a database system keeps increasing if the old data is not deleted. Also, hardware wears down over a period. Therefore, a database may be required to move from one machine to another machine. Sometimes the change in DBMS software or its versions also necessitates movement of data. The Load and Upload utilities are used for the purposes as above.
- **Utilities for Reporting or Analysis:** Reports form the important component of a database systems, especially management information systems. The reports and analysis of data can be used at various levels of decision making in an organisation. These utilities analyse and report data in various visual forms for decision making.

1.6 DATA MODELS AND TRENDS

A database system requires that the data should be structured in such a manner that it allows easier data access and manipulation operations. These structures are termed as database models. A database model has to address the following issues:

- It should define a logical structure of data, so that different attributes of data are easily identified.
- It should be able to represent relationships between different data elements or objects.
- It should support constraints on data elements and data objects.

The following Table defines some of the Data Models:

Model Type	Examples
Conceptual Model: Uses entities, their attributes and relationships among entities	*Entity-Relationship Model*: This model identifies the physical or logical entities and their attributes. In addition, this model also identifies the relationships among these entities. It is mainly represented in graphical form using E-R diagrams. This is very useful in Database design. The entity relationship diagram is discussed in this Block.
Hierarchical Model: Uses data hierarchy	Hierarchical Model: It defines data as and relationships through hierarchy of data values. *Figure 7* shows an example of hierarchical model. These models are now not used in commercial DBMS products.
Record based Logical Models:	*Network Model*: Network model, as the name suggests, represents data about entities using a set of records. The relationships among these data records are represented using links. A simple network model example is explained in *Figure 6*. It shows a sample diagram for such a system. This model is a very good model as far as conceptual framework is concerned but is nowadays not used in database management systems. *Relational Model*: It models data of both entities and relationships as tables. The relationship tables are related to entity tables using a set of constraints. This model is based on a sound mathematical theory of relations. This is one of the widely used data models and will be discussed in more detail in the subsequent units of this course.
Object-based Models: Use objects as key data representation components.	*Object-Oriented Model*: An object in object-oriented model contains the data of an entity. In addition, the object also allows a set of defined operations on the data stored in an object. Further, the relationships among the objects are implemented by creating links among these objects and by implementing message passing. Object-Models are useful for databases where data interrelationships are complex, for example, Computer Assisted Design based components. These are explained in Block 4.
Semi-structured Model: Uses user defined tags	*Semi-structured data Model*: Relational model is typically called a structured model, whereas, the semi-structured model uses user defined tags, which loosely force integrity contains. However, these models are very flexible in comparison to relational models. These models are popular in web-based data management, as they are very light database systems.
Graph-based Models	*Graph data Model*: Graph is an important data structure consisting of nodes and arcs. The node can represent an entity while the links can be used to represent a relationship.

Figure 6 shows the records of students and the result of the students in various courses taken by them. All the records of the students are shown as a single file. The student records can contain one or many pointers to the course file. Please notice in Figure 6 that the person named Ajay, whose enrolment number is 21026 has obtained 50 marks in course having course code 101. Further, he has obtained 19 marks in a course

having course code 204. Thus, each record on the course is linked to one record of the student.

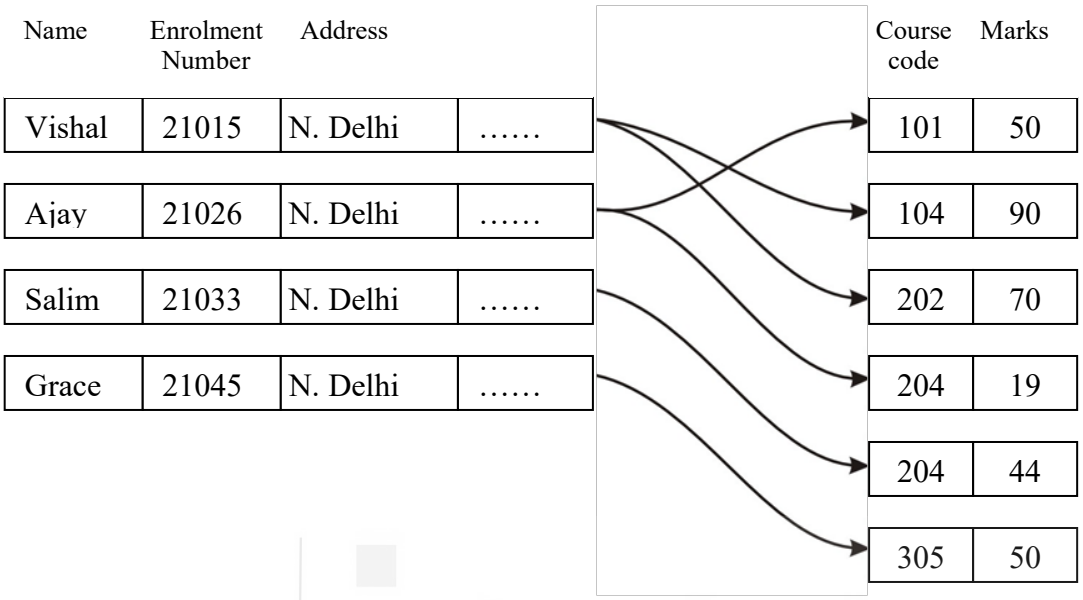


Figure 6: An example of Network Model

Figure 7 shows the data of Figure 6 in a hierarchical data model. Both these models are of historical nature. The relational model, object-oriented model and other model models are explained in the later blocks of this course.

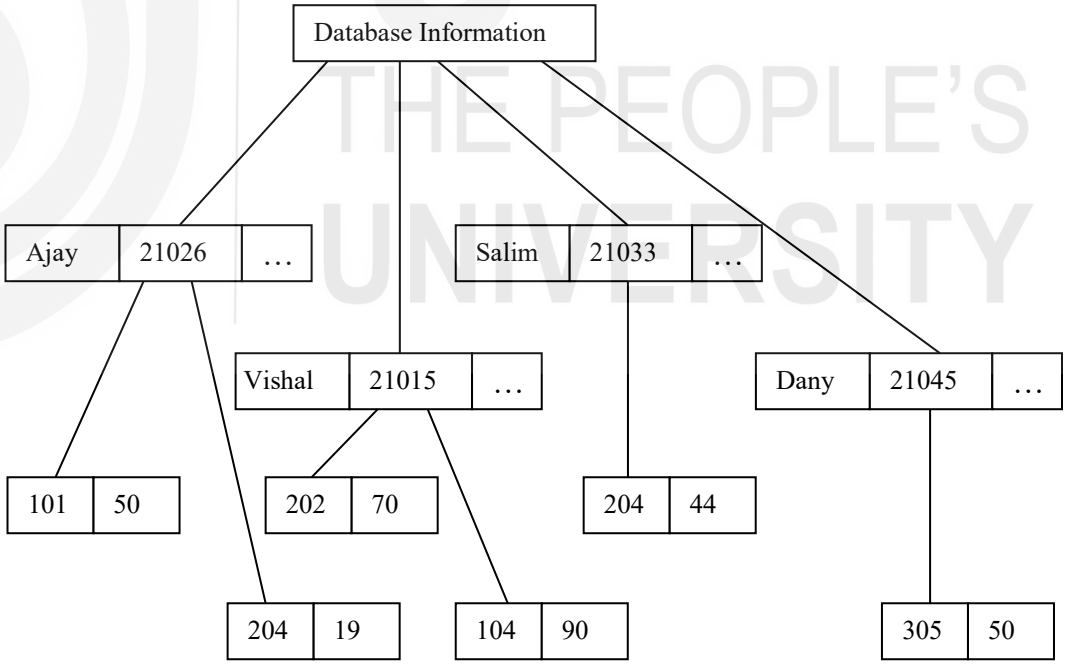


Figure 7: An example of Hierarchical Model

Relational Database management systems, which were supported by structured query language (SQL), became popular in the 1990s and later. These database systems were used to process transactions of a high-end web application. Such systems provided application users a graphical or menu driven interface using devices like computer, mobile devices and had features of user data input through keyboard, mouse, touch screens, forms, speech etc. With the growth of web applications lead to growth of semi-structured data. This led to use of eXtensible Markup Language (XML) and Java Script Object Notation (JSON) as the data exchange formats. Thus, several different database management systems, like geographical database management systems,

emerged that supported such data. With the rapid rate of growth of data newer database management systems like NoSQL databases started to grow. These databases lowered the time of application development but did not have strict control on consistency requirements. In the present time, the features of traditional and NoSQL database management systems are growing. Present day DBMSs are using cloud services for data storage and offer many database services at higher levels of abstraction leading to lesser time for database application deployment.

Check your Progress 3

1) What is a database backend?

   ……

2) What is the need of utilities in a commercial database system? Explain any two such utilities.

   …….

3) What is the difference between network and hierarchical models?

   ……..

1.7 SUMMARY

This unit provides you an introduction about the database system and database management system. It first defines the need of a database management system by comparing the database system approach to file system approach. The file system has several limitations due to data redundancy, however, the database approach integrates and shares the data among several applications. The unit also discusses the basic advantages of the database approach are - sharing of data, data independence, data integrity and security enforcement and transaction management for concurrent users.

The unit also explains the three-level architecture of database systems, which allows a database to be defined in terms of schemas at different levels for different types of users. Such a design allows access of relevant data to relevant users. In addition, the unit explains the physical architecture of DBMS and describes various components of DBMS. Next, the unit discusses the database system architecture for the commercial database and several data models. It also briefly presents the recent trends in DBMSs. You are advised to go through the further readings for more details on these topics.

1.8 SOLUTIONS/ANSWERS

Check Your Progress 1

1) DBMS manages the data of an organisation. It allows facilities for defining, updating and retrieving data of an organisation in a sharable, secure and reliable way.

2) The advantages of DBMS are:
   - Reduces redundancies
   - Provides environment for data independence
   - Enforces data integrity
   - Data Security
   - Answers unforeseen queries
   - Provides support for transactions, recovery etc.

3)

File Based System	Database Approach
Cheaper	Costly
Data dependent	Data independent
Data redundancy	Controlled data redundancy
Inconsistent Data	Consistent Data
Fixed Queries	Unforeseen queries can be answered

Check Your Progress 2

1) Integrity enforcement, control of file manager, security, backup, recovery, concurrency control, etc.

2) A database administrator is normally given such controls. His/her functions are: defining database, defining and optimising storage structures, and control of security, integrity and recovery.

Check Your Progress 3

1. Most of the database systems are deployed in a multi-user environment either as a centralised or distributed system. A database server, which is at the backend, manages and controls the physical data. It also provides integrity, security and access control to multiple client's requests.
2. Utilities are special purpose software, which are provided to support additional features in support of a DBMS. Two such utilities are for backup/restore – to protect data against any failure and load/unload – to load data from one hardware/software combination to a different hardware/software combination.
3. A network model stores basic entities as data records and points are used to represent the relationship between them. A hierarchical model uses almost similar model to a network model to represent data of entities but the relationship between them is implemented as a hierarchy.

UNIT 2 RELATIONAL DATABASE

Structure **Page Nos.**

2.0 INTRODUCTION

In the first unit of this block, you have been provided with the details of the Database Management System, its advantages, structure, etc. This unit is an attempt to provide you information about relational model. The relational model is a widely used model for DBMS implementation. Most of the commercial DBMS products available in the industry are relational at the core.

The relational model is based on the theory of relations and was first proposed by E.F. Codd. In this unit we will discuss the terminology, operators and operations used in relational model.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- describe relational model and its advantages;
- perform basic operations using relational algebra;
- identify the relational constraints in a database system;
- identify the key of a relation.

2.2 THE RELATIONAL MODEL

A model of a database system basically defines the *structure or organisation of data*, the *set of integrity constraints* on the data and a *set of operations* that can be performed on the data. The basic structure of data in a relational model is in the form of two-dimensional tables. A table in this model is called a relation. Such organisation of data simplifies enforcement of integrity constraints and database operations. The following table represents a simple relation:

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	B-4,Modi Nagar, UP
2	Sharada Gupta	30	Pocket 2, Mayur Vihar, Delhi
3	Vibhu Datt	36	C-2, Saket, New Delhi

Figure 1: A Sample Person Relation

Following are some of the advantages of the relational model:

- **Ease of use:** The simple tabular representation of the database helps the user define and query the database conveniently. For example, you can easily find out the age of the person whose first name is "Vibhu".
- **Flexibility:** Since the database is a collection of tables, new data can be added and deleted easily. Also, manipulation of data from various tables can be done easily using various basic operations. For example, you can add a telephone number field in the table in *Figure 1*.
- **Accuracy:** In relational databases the relational algebraic operations are used to manipulate data values in a database. These are mathematical operations and ensure accuracy (and less of ambiguity) as compared to other models. These operations are discussed in more detail in Section 2.4.

2.2.1 Tuple, Attribute, Domain and Relation

Before we discuss the relational model in more detail, let us first define some very basic terms used in this model.

Tuple: Each row in a table represents a record or information of a single object, which is also termed as a tuple. For example, Figure 1 has three records or tuples. A record/tuple consists of a number of attributes, which is defined next.

Attribute: A relation consists of a large number of columns. Each of these columns, which defines a separate data value, is termed as an attribute. The column name in a relation is generally related to the meaning of data items of that column. For example, *Figure 2* represents a relation PERSON. The columns PERSON_ID, NAME, AGE, ADDRESS and TELEPHONE are the attributes of the relation PERSON and each row in the relation represents a separate tuple (record).

Relation Name: PERSON

PERSON_ID	NAME	AGE	ADDRESS	TELEPHONE
1	Sanjay Prasad	35	B-4,Modi Nagar, UP	011-25347527
2	Sharada Gupta	30	Pocket 2, Mayur Vihar, Delhi	023-12245678
3	Vibhu Datt	36	C-2, Saket, New Delhi	033-1601138

Figure 2: An extended PERSON relation

The relation of Figure 2 consists of 5 attributes, therefore, each tuple in this relation is called a 5-tuple. Thus, if a relation has n attributes, then each record in that relation would be termed as *n*-tuple.

Domain: A domain is a set of permissible values that can be accepted by a specific attribute. For example, in Figure 2, if you assume that there may be a maximum of 100 persons in the relation, then you may assign PERSON_ID to a domain of integer values, which should be in the range from 1 to 100 only. Once you have assigned this domain, then you will not be able to assign any value below 1 and above 100 to PERSON_ID. The domain of attribute AGE can be integer values between 0 and 150. The domain can be defined by assigning a type or a format or a range to an attribute. For example, a domain for a number 501 to 999 can be specified by having a 3-digit number format having a range of values between 501 and 999. Domains need not be contiguous numbers. For example, the enrolment number of IGNOU has the last digit as the check digit, thus the enrolment numbers are non-continuous.

Relation: Each table is a relation. A relation is defined using two basic aspects, viz. schema and an instance.

Relational Schema: A relational schema denoted as R, specifies the relation's name, the list of attributes of the relation and the domain of each attribute, which is denoted as R (A_1, A_2, ..., A_n). The relation R has *n* attributes, which is also called the degree of a relation.

For example, the relation schema of the relation PERSON (see Figure 1) can be defined as the follows:

PERSON (PERSON_ID: Integer, NAME: Character, AGE: Integer,
ADDRESS: Character)

Since the PERSON relation contains four attributes, this relation is of degree 4.

Relation Instance or Relation State: A relation instance denoted as **r** is a collection of tuples for a given relational schema at a specific point of time.

A relation state **r** of the relation schema R (A_1, A_2, ..., A_n), also denoted by r(R) is a set of *n*-tuples

r = {t_1,t_2,……t_m} has *m* tuples
Where each *n*-tuple is an ordered list of n values
t = <v_1,v_2,……., v_n>
where each v_i belongs to domain (A_i) or contains null values.

Let us elaborate the definitions above with the help of examples:

Example 1:

RELATION SCHEMA For STUDENT:

STUDENT (RollNo: string, name: string, login: string, age: integer)

RELATION INSTANCE

STUDENT

	ROLLNO	NAME	LOGIN	AGE
t_1	3467	Shikha	xyz@hotmail.com	21
t_2	4677	Kanu	abc@gmail.com	20

Where t_1 = (3467, shikha, xyz@hotmail.com, 20) for this relation instance, the number of tuples *m* = 2 and each tuple contains *n* = 4 values.

Example 2:

RELATIONAL SCHEMA For PERSON:

PERSON (PERSON_ID: integer, NAME: string, AGE: integer, ADDRESS: string)

RELATION INSTANCE

In this instance, the number of tuples m = 3 and each tuple has n = 4 values.

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	B-4,Modi Nagar, UP
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi
3	Vibhu Datt	36	C-2, Saket, New Delhi

If a relation has proper integrity constraints, then each tuple of that relation would be valid tuple. It may be noted that 'Null' value can be assigned for some of the attributes where the values are unknown or missing. However, the Null' value cannot be assigned to certain attributes, this will be explained in the later sections.

Ordering of tuples

In a relation, tuples are not inserted in any specific order. **Ordering of tuples is not defined as a part of a relation definition**. However, records may be organised later according to some attribute value in the storage system. For example, records in the PERSON table may be organised according to PERSON_ID. Such data organisation depends on the requirement of the underlying database application. However, for the purpose of display, you may get these tuples displayed in the sorted order of age. The following table is sorted by age. Please note that this is sorted relation is the same as that or relation displayed in the order of PERSON_ID as each and every tuple is the same. It is also worth mentioning that the relational model **does not allow duplicate tuples**.

PERSON

PERSON_ID	NAME	AGE	ADDRESS
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi
1	Sanjay Prasad	35	B-4,Modi Nagar, UP
3	Vibhu Datt	36	C-2, Saket, New Delhi

2.2.2 Super Keys, Candidate Keys and Primary Keys for the Relations

As discussed in the previous section, ordering of relations does not matter and all tuples in a relation are unique. However, can you uniquely identify a tuple in a relation? To answer this question, let us discuss the concept of keys in relations.

Super Keys

A **super key** is an attribute or set of attributes used to identify the records uniquely in a relation.

For Example, in the Relation PERSON described earlier PERSON_ID is a super key since PERSON_ID is unique for each tuple/record. Similarly (PERSON_ID, AGE) and (PERSON_ID, NAME) are also super keys of the relation PERSON since their combination is also unique for each tuple/record.

Candidate keys

Super keys of a relation may contain extra attributes. A candidate key is a minimal super key, which means that a candidate key does not contain any extraneous attribute. An attribute is called extraneous if even after removing it from the key, the remaining attributes still has the properties of a super key. A relation may have several candidate keys. A candidate key may contain one or many attributes of a relation.

The following properties must be satisfied by a candidate key:

- In any instance of a relation, the value of the candidate key attribute(s) should be unique.
- You cannot put a 'Null' value in any attribute that is a part of the candidate key. This rule is also termed the entity integrity rule. Thus, a candidate key is unique and not null.
- A candidate key should have a minimal set of attributes.
- The value of a candidate key must be **stable,** which means it should not change frequently or its **value** change should not be outside the control of the system.

A relation can have more than one candidate key and one of them can be chosen as a **primary key.**

For example, in the relation PERSON, the two possible candidate keys are PERSON_ID and NAME (assuming unique names exist in the table). PERSON_ID may be chosen as the primary key.

Check Your Progress 1

Answer the following questions for the relational instance s of the following Supplier relation S:

S

SNO	**SNAME**	**CITY**
S1	Smita	Delhi
S2	Jim	Pune

1. What are the different attributes of relation S and how many tuples does S have?
   ……..
2. What are the domains of the attributes of S?
   …….
   ……..…….
3. On sorting this relation on the field CITY, will the relation change? Will the order of tuples in the relation change?
   ……..
4. List the super keys, all the possible candidate keys, and the primary key of the relation.
   ……..

2.3 RELATIONAL CONSTRAINTS

A relational database is a collection of relations. Each relation consists of tuples, which consist of attributes. You can associate constraints, called relational constraints, with the attributes. Such constraints can be of the following types:

- Domain Constraints
- Key Constraint
- Integrity Constraints

2.3.1 Domain Constraint

These constraints bind the possible values of the attribute in a relation to a specific set of values, called the domain of the attribute. For example, an attribute COUNTRY may have a domain of names of all the possible names of the countries. In commercial relational database management systems, the data type associated with an attribute defines the broad domain of an attribute. These domains include:

1) Integral Data type like integer, long etc.
2) Data types related to real numbers like float, double etc.
3) Data types relating to alphabets like characters, strings etc.
4) Boolean

Thus, domain constraint specifies the possible set of values that you want to put in an attribute of a relation. The values that appear in each attribute/column must be drawn from the domain associated with that column.
For example, consider the relation:

STUDENT

ROLLNO	NAME	LOGIN	AGE
3467	Shikha	xyz@hotmail.com	21
4677	Kanu	abc@gmail.com	20

In the relation above, AGE of the relation STUDENT always belongs to the integer domain within a specified range (say 0 representing just born to 150) and not to strings or any other domain. Within a domain, non-atomic values should be avoided. This sometimes cannot be checked by domain constraints. For example, a database which has area code and phone numbers as two different fields will take phone numbers as-

Area_code	Phone
11	29534466

A non-atomic value in this case for a phone can be 1129534466, however, this value can be accepted by the Phone field only.

2.3.2 Key Constraint

This constraint states that the key attribute value in each tuple must be unique, i.e., no two tuples can contain the same value for the key attribute. This is because the value of the primary key is used to identify a unique tuple in a relation.

Example 3: If A is the key attribute in the following relation R, then A1, A2 and A3 must be unique.

R

A	B
A1	B1
A3	B2
A2	B2

Example 4: In relation PERSON, PERSON_ID is the primary key so PERSON_ID cannot be assigned the same for two different persons.

2.3.3 Integrity Constraint

There are two types of integrity constraints:

- Entity Integrity Constraint
- Referential Integrity Constraint

Entity Integrity Constraint:

It states that any attribute of a **primary key cannot take a Null value**. This is because the primary key is used to identify individual tuple in the relation. You will not be able to identify the records uniquely if they contain null values for the primary key attributes. This constraint is specified for each relation of a database.

Example 5: Let R be a relation; an instance *r* of R is given below. Is the instance r valid?

A#	B	C
Null	B1	C1
A1	B2	C2
Null	B2	C3
A2	B3	C3
Null	B1	C5

Note: A '#' in the headings row is indicating that A is the Primary key of R.

In the instance *r* of relation R above, the primary key has Null values in the tuple t_1, tuple t_3 and tuple t_5. As per the entity integrity constraint, Null value in primary key is not permitted. Thus, relation instance *r* is an invalid instance.

Referential integrity constraint

For defining the referential integrity constraint, first we explain the concept of a **foreign key** and **foreign key constraint.**
Consider an attribute set A of a relation R, which references an attribute set B in a relation S, then A will be referred to foreign key in R provided it fulfills the following conditions:

1. B is the Primary key of S.
2. A and B are defined over the same domains. A is called the foreign key in relation R, which references B in relation S.
3. For every value *x* of attribute set A, in any instance *r* of R, there exists a tuple in any instance *s* of S, where the value of attribute set B is *x*. Please note that there will be only one such tuple having the value *x*, as B is the Primary key of S. This is called foreign key constraint.
4. Please note that the relation R is a referencing relation and S is called referenced relation.

A referential integrity constraint is a more generic form of foreign key constraint, which requires that the attribute of the referencing relation must be present in the attribute of referenced relation in some tuple. Thus, foreign key constraint is a specific form of referential integrity constraint. Foreign key constraint is explained with the help of the following example.

Example 6: Consider the two relational instance r of relation R and s of relation S. Now answer the following questions:

a) If C in R is foreign key of C in S, then are the following instances r and s valid?
b) A new tuple having values (A6, B2, C4) is added in r. Does it violate foreign key constraint?

Instance r of R

A#	B	C^
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C3
A5	B1	C5

Instance s of S

E	C#
E1	C1
E2	C3
E3	C5
E2	C2

Notes:

- '#' identifies the Primary key of a relation.
- '^' identifies the foreign key in a relation.

a) In the example above, every value of C in r is matching with the values of C in s in some tuple. The r contains the values C1, C2, C3, C5 and s contains all these values. Thus, the instances r and s does not violate foreign key constraint.
b) If you try to insert the tuple (A6, B2, C4) in r, then it is violating the foreign key constraint for the present instances of relations, as there is no tuple in s, which contains value C4 in s.

2.3.4 Data Updating Operations and Constraint Violations

There are three basic data updating operations to be performed on relations:

- Insertion of tuples
- Deletion of tuples
- Update the value of attribute(s) in a relation

The INSERT Operation:

- To add new records in a database system, you use insert operation. For example, in the instance r of R of example 6, you may like to add a new record <'A5', 25,' C6'>. Addition of a new record may cause constraint violation, which are explained below:

- Violation of Domain constraint: Such a violation occurs if the domain of a value, which you are trying to insert in an attribute, does not match the attribute's domain. e inserting a numeric value 25 into attribute B, whose domain is alphanumeric sting. This will cause a domain constraint violation.
- Violation of Key constraint: This violation occurs when you try to insert a key value, which already exists in some tuple of existing relational instance. For example, when you are trying to insert tuple <'A5', 25, 'C6'> in R, you are inserting a value A5 in the key attribute A. However, the value A5 already exists in attribute A of tuple t_5. Therefore, it is a violation of key constraint.
- Violation of Entity Integrity constraint: This violation will occur; in case you try to insert a Null value into the primary key attribute. For example, if you try to insert a tuple <'Null, 25, 'C6'> in R, you are violating this constraint.
- Violation of Referential Integrity constraint: Consider that while inserting a new tuple, the value that you are trying to insert in a foreign key attribute, which refers to an attribute in another relation where it is the primary key, does not match with any value of referred attribute of the referenced relation. (Please see example 6 par (b)). For example, when you are trying to insert tuple <'A5', 25, 'C6'> in R, you are inserting a value C6 in the foreign key attribute A of R. However, the value C6 is not a value in referred attribute C in referenced relation S. Thus, insertion of this tuple violates the referential integrity constraint.

Dealing with constraints violation during insertion:

If the *insertion* violates one or more constraints, then two options are available:

- Default option: - *Insertion can be rejected and the reason for rejection can also be explained to the user by* DBMS.
- Ask the user to correct the data, resubmit, and give *the reason for rejecting the insertion*.

Example 7:

Consider the Relation PERSON of Example 2:

PERSON

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	B-4,Modi Nagar, UP
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi
3	Vibhu Datt	36	C-2, Saket, New Delhi

(1) Insert <1, 'Vipin', 20, 'Mayur Vihar'> into PERSON
Violated constraint: - Key constraint
Reason: - Primary key 1 already exists in PERSON.
How to resolve: - DBMS could ask the user to provide valid PERSON_ID value and accept the insertion if valid PERSON_ID value is provided.

(2) Insert <'null', 'Anurag', 25, 'Patparganj'> into PERSON
Violated constraint: - Entity Integrity constraint
Reason: - Primary key is 'null'.
How to resolve: - DBMS could ask the user to provide valid PERSON_ID value and accept the insertion if valid PERSON_ID value is provided.

(3) Insert <'abc', 'Suman', 25, 'IP college'> into PERSON
Violated constraint: - Domain constraint
Reason: - value of PERSON_ID is given a string which is not valid.
How to resolve: - DBMS could ask the user to provide valid PERSON_ID value and accept the insertion if valid PERSON_ID value is provided.

(4) Insert <10, 'Anu', 25, 'Patparganj'> into PERSON
Violated constraint: - None

Note: In the first 3 cases of constraint violations above DBMS will reject the insertion.

The Deletion Operation:

The delete operation is used to delete some existing records from a relation. To delete some specific records from the database a condition is specified based on which records are selected for deletion.

Constraints that can be violated during deletion

Only one type of constraints can be violated during deletion, it is referential integrity constraint. When you want to delete a record in a referenced relation, there is a possibility that you may delete a tuple whose attribute is being referenced by the referencing relation, where the attribute is the foreign key. Please go through example 8 very carefully.

Dealing with Constraints Violation

If the *deletion* violates the referential integrity constraint, then three options are available:

- Default option: - *Reject the deletion*. It is the job of the DBMS to explain to the user why the deletion was rejected.
- *Attempt to cascade (or propagate) the deletion* by deleting tuples whose foreign key references the tuple that is being deleted.
- Change the value of the *referencing attribute* that causes the violation.

Example 8:

Let instance r of relation R be:

A#	B	C^
A1	B1	C1
A2	B3	C3
A3	B4	C3
A4	B1	C5

And instance q of relation Q be:

C#	D
C1	D1
C3	D2
C5	D3

Note:

1) '#' identifies the Primary key of a relation.
2) '^' identifies the Foreign key of a relation.

(1) Delete a tuple with C# = 'C1' in Q.
Violated constraint: - Referential Integrity constraint is violated
Reason: - If the stated tuple is deleted from Q, the first tuple of R, which contains the Foreign key (C) value C1, will violate the foreign key constraint.
How to resolve: Options available are

1) Reject the deletion.
2) DBMS may automatically delete all tuples from relation *Q* and *S* with C # = 'C1'. This is called cascade deletion.
3) The third option would result in putting NULL value in R where the value of attribute C is C1, which is the first tuple of R.

The Update Operations:

Update operations are used for modifying the values of attributes in a database. The constraint violations faced by this operation are logically the same as the problem faced by Insertion-Deletion Operation. You may define these actions yourself.

Check Your Progress 2

1 Consider the tables **S**uppliers, **P**arts, pro**J**ect and **SPJ** relation instances in the relations below.

S

SNO	**SNAME**	**CITY**
S1	Smita	Delhi
S2	Jim	Pune
S3	Ballav	Pune

P

PNO	**PNAME**	**COLOUR**	**CITY**
P1	Nut	Red	Delhi
P2	Bolt	Blue	Pune
P3	Pen	White	Mumbai

J

JNO	**JNAME**	**CITY**
J1	Sorter	Pune
J2	Display	Mumbai

SPJ

SNO	**PNO**	**JNO**	**QUANTITY**
S1	P1	J1	200
S2	P2	J2	700

Using the instance of relations as given above, answer the following questions:

1 List a domain constraint for S.
……
……

2 List the Primary keys of all the relations and primary key constraint for the SPJ relation.
……
……

3 List all the Foreign keys and Foreign key constraints.
……
……

4 What would be the constraint violations if the following operations are performed:
(a) Insert <Null, 'Jack', 'Mumbai'> into S
(b) Insert <'S2', Null, 'J3', 200> into SPJ
(c) Insert <'P2', 'Pencil', 'Grey', 'Kolkata'> into P
(d) Insert <'J3', 'Monitor',' Jaipur'> into J
……
……
……

2.4 RELATIONAL ALGEBRA

Relational Algebra is a set of basic operations used to manipulate the data in relational model. These operations enable the user to specify basic retrieval requests. The result of retrieval is a new temporary relation, which is computed after performing these operations on one or two relations. Some of these **operations can be classified in three categories:**

- Relational Operations
  These can be unary operations
  1. SELECTION
  2. PROJECTION

  Or binary operations
  1. CARTESIAN PRODUCT
  2. JOIN
- Basic Set Operations
  1. UNION
  2. INTERSECTION
  3. SET DIFFERENCE
- Assignment Operation, rename and other operations

In this section, we will discuss relational operations and basic set operations. You may refer to the other operations from the further readings.

2.4.1 Relational Operations

Relational algebra forms the basis of the query languages on a database system. Thus, by studying relational algebra, you may be able to write better queries for a database system. Let us discuss these operations in detail.

Selection Operation:

Selection is a unary operator, that is used to choose only those tuples from a relation that fulfill a given criteria. It is represented using the mathematical symbol σ, as given below:

σ <Selection condition> **(Relation)**

You should specify a Boolean expression as a <Selection condition>. A selection condition uses comparison operators like { $\leq, \geq, \neq, =, <, <$} and logical connectors like and ($\wedge$), or ($\vee$) and not ($\neg$).

Example 13:

Consider the relation PERSON. If you want to display details of persons having age less than or equal to 30, then the selection operation will be used as follows:

σ AGE <=30 **(PERSON)**

The resultant relation will be as follows:

PERSON_ID	NAME	AGE	ADDRESS
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi

Note:

1) Selection operation is commutative, i.e.,
   $\sigma_{<condition1>}(\sigma_{<condition2>}(R)) = \sigma_{<condition2>}(\sigma_{<condition1>}(R))$
   Hence, you can apply a sequence of Selection operations in any order

2) You can apply more than one condition by using logical connectors.

Projection operation

The projection operation is used to select specific attributes from amongst all the attributes of records. This is denoted as Π.

Π List of attributes for projection **(Relation)**

Example 14:

Consider the relation PERSON. If you want to display only the names of persons, then the projection operation will be used as follows:

Π_{NAME}**(PERSON)**

The resultant relation will be as follows:

NAME
Sanjay Prasad
Sharad Gupta
Vibhu Datt

Please note that for projection operation:
$\Pi_{<List1>}(\Pi_{<list2>}(R)) = \Pi_{<list1>}(R)$ provided <list2> contains attributes in <list1>.

2.4.2 Cartesian Product

Given two relations R and S, the cartesian product of these two relations can be represented as $T = R \times S$. The cartesian product operation produces all possible combinations of the tuples of tuples of relations R and S. The cartesian product operation is commutative as well as associative. In addition, the degree of the resultant

relation (T) is the sum of the degrees of relation R and relation S. The Cartesian product can be expressed using the following mathematical expression:

$T= \{t_1 \| t_2 \mid t_1 \in R1 \wedge t_2 \in R2\}$. Here, || is a concatenation operator.

Example 12: Consider the following two relational instances of relation R1 and R2. What would be the cartesian product of the relations?

R1

A	B
A1	B1
A1	B2
A2	B2
A2	B3

R2

C
C1
C2

The Cartesian produce of the relations is denoted by R3 = R1 ×R2. The relation R3 will be as shown below:

A	B	C
A1	B1	C1
A1	B1	C2
A1	B2	C1
A1	B2	C2
A2	B2	C1
A2	B2	C2
A2	B3	C1
A2	B3	C2

Please note that in cartesian product every tuple/record of R1 is concatenated with every record of R2. You can observe that the record <A1, B1> is concatenated with both the records of R2, that is why in the output R3 you have tuples < A1, B1, C1> and <A1, B1, C2>, likewise for all the tuples of R1. Please also note that R1 has 4 tuples and R2 has 2 tuples, therefore, R3 has 4×2=8 tuples.

The JOIN operation

JOIN is a binary operator. It combines two relations based on a given join condition. A JOIN operation is represented by mathematical symbol ⋈. But how does JOIN operation combine two relations? It would require that there should be at least one attribute in each of the relations, which have the same domain. Such attributes are called domain compatible attributes.

Syntax:

$R1 \bowtie_{<\text{join condition}>} R2$ is used to combine related tuples from two relations R1 and R2 into a single tuple.

<join condition> is of the form:

<condition>AND<condition>AND……..AND<condition>.

- Degree of Relation:

  Degree ($R1 \bowtie_{<\text{join condition}>} R2$) <= Degree (R1) + Degree (R2).

- Three types of joins are there:

a) **Theta (θ) join**

When each condition is of the form A θ B, where A is an attribute of R1 and B is an attribute of R2; both A and B have the same domain; and θ is one of the comparison operators { $\leq, \geq, \neq, =, <, <$}.

b) **Equijoin**

Equijoin is a restricted form of When each condition appears with equality operator (=) only.

c) **Natural join**

Natural join is defined as a specialised type of join, when the joining attributes in the two joining relations have the identical name, in addition to the identical domain and the condition for the join operation is equality (=) condition. In such cases only one joining attribute is kept in the result.

The following example explains the Natural join operation.

Example 15:

Consider the instance of the following relations. The primary key of STUDENT relation is ROLLNO and foreign key is COURSE_ID, which references the primary key COURSE_ID of COURSE relation.

STUDENT

ROLLNO	NAME	ADDRESS	COURSE_ID
100	Kanupriya	234, Saraswati Vihar.	CS1
101	Rajni Bala	120, Vasant Kunj	CS2
102	Arpita Gupta	75, SRM Apartments.	CS4

COURSE

COURSE_ID	COURSE_NAME	DURATION
CS1	MCA	3yrs
CS2	BCA	3yrs
CS3	M.Sc.	2yrs
CS4	B.Sc.	3yrs
CS5	MBA	2yrs

Display the name and other details of all the students along with their course details.

Solution: You may observe that the course details are existing in the COURSE relation and student names and other details are in the STUDENT relation. Therefore, you need to join these two relations to extract information. The join attributes in this case are COURSE_ID in STUDENT and COURSE_ID in the COURSE relation and equality operator is to be used; therefore, you use natural join operation, which can be represented as:

$$\text{STUDENTCOURSE} = \text{STUDENT} \bowtie \text{COURSE}$$

The output of this natural join will be the following relation. Please note that the output has just one COURSE_ID attribute.

STUDENTCOURSE

ROLLNO	NAME	ADDRESS	COURSE_ID	COURSE_NAME	DURATION
100	Kanupriya	234, Saraswati Vihar.	CS1	MCA	3yrs
101	Rajni Bala	120, Vasant Kunj	CS2	BCA	3yrs
102	Arpita Gupta	75, SRM Apartments.	CS4	B.Sc.	3yrs

There are other types of joins like outer joins. You must refer to further reading for more details on those operations. They are also explained in later blocks of this course.

2.4.3 Basic Set Operation

The set operations are the binary operations, i.e., each is applied to two sets or relations, which should be union compatible. Two relations R (A_1, A_2,…, A_n) and S (B_1, B_2,…, B_n) are said to be union compatible if they have the ***same degree n*** and domains of the corresponding attributes are also the same, i.e., ***Domain (A_i) = Domain (B_i) for 1<=i<=n***.

Union

If R1 and R2 are two union compatible relations, then R3 = R1 ∪ R2 is the relation containing tuples that are either in R1 or in R2 or in both. In other words, R3 will have tuples such that R3 = {t | t ∈ R1 ∨ t ∈ R2}.

Example 9:

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R3 = R1 ∪ R2 is

R3

A	B
A1	B1
A2	B2
A3	B3
A4	B4
A7	B7

Note: 1) Union is a commutative operation, i.e,

R ∪ S = S ∪ R.

2) Union is an associative operation, i.e.

R ∪ (S ∪ T) = (R ∪ S) ∪ T.

Intersection

If R1 and R2 are two union compatible functions or relations, then the result of R3 = R1 ∩ R2 is the relation that includes all tuples that are in both the relations

In other words, R3 will have tuples such that R3 = {t | t ∈ R1 ∧ t ∈ R2}.

Example 10:

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R3 = R1 ∩ R2 is

A	B
A1	B1
A2	B2
A4	B4

Note: 1) Intersection is a commutative operation, i.e.,

$$R1 \cap R2 = R2 \cap R1.$$

2) Intersection is an associative operation, i.e.,

$$R1 \cap (R2 \cap R3) = (R1 \cap R2) \cap R3$$

Set Difference

If R1 and R2 are two union compatible relations, then the result of R3 =R1– R2 is the relation that includes only those tuples that are in R1 but not in R2. In other words, R3 will have tuples such that $R3 = \{t \mid t \in R1 \wedge t \notin R2\}$.

Example 11:

R1

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R2

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R1-R2 =

A	B
A7	B7

R2-R1=

A	B
A3	B3

Note: -1) Difference operation is not commutative, i.e.,

$$\mathbf{R1 - R2 \neq R2 - R1}$$

2) Difference operation is not associative, i. e.,

$$\mathbf{R1 - (R2 - R3) \neq (R1 - R2) - R3}$$

Please note that a relational query language would be relationally complete, if it supports five relational algebraic operators – Selection, Projection, Union, Set Difference and Cartesian Product.

Check Your Progress 1

1) A database system is fully relational if it supports a language as powerful as _____.

2 Primitive operations are union, difference, product, selection and projection. The A ∩ B can be computed using …….

3) Which of the following is not a traditional set operator in relational algebra?

a. Union
b. Intersection
c. Difference
d. Join

4) Consider the relational instances of the relations **S**uppliers, **P**arts, pro**J**ect and **SPJ** relations given below. (Underline represents a key attribute. The SPJ relation has three Foreign keys: SNO, PNO and JNO.)

S

SNO	SNAME	CITY
S1	Smita	Delhi
S2	Jim	Pune
S3	Ballav	Pune
S4	Seema	Delhi
S5	Salim	Agra

P

PNO	PNAME	COLOUR	CITY
P1	Nut	Red	Delhi
P2	Bolt	Blue	Pune
P3	Part1	White	Mumbai
P4	Part2	Blue	Delhi
P5	Camera	Brown	Pune
P6	Part3	Grey	Delhi

J

JNO	JNAME	CITY
J1	Sorter	Pune
J2	Display	Bombay
J3	OCR	Agra
J4	Console	Agra
J5	RAID	Delhi
J6	EDP	Udaipur
J7	Tape	Delhi

SPJ

SNO	PNO	JNO	QUANTITY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J2	400
S2	P2	J7	200
S2	P3	J3	500
S3	P3	J5	400
S3	P4	J3	500
S3	P5	J3	600
S3	P6	J4	800
S4	P6	J2	900
S4	P6	J1	100
S4	P5	J7	200
S5	P5	J5	300
S5	P4	J6	400

Using the in the relations above, which of the following operations and constraints would be valid:

i. UPDATE Project J7 in J setting CITY to Nagpur.
ii. UPDATE part P5 in P, setting PNO to P4.
iii. UPDATE supplier S5 in S, setting SNO to S8, if the relevant update rule is RESTRICT.
iv. DELETE supplier S3 in S, if the relevant rule is CASCADE.
v. DELETE part P2 in P, if the relevant delete rule is RESTRICT.
vi. DELETE project J4 in J, if the relevant delete rule is CASCADE.
vii. UPDATE shipment S1-P1-J1 in SPJ, setting SNO to S2. (shipment S1-P1-J1 means that in SPJ table the value for attributes SNO, PNO and JNO are S1, P1 and J1 respectively)
viii. UPDATE shipment S5-P5-J5 in SPJ, setting JNO to J7.
ix. UPDATE shipment S5-P5-J5 in SPJ, setting JNO to J8
x. INSERT shipment S5-P6-J7 in SPJ.
xi. INSERT shipment S4-P7-J6 in SPJ
xii. INSERT shipment S1-P2-jjj (where jjj stands for a default project number).

……

……

……

5) Find the name of projects in the relations above, to which supplier S1 has supplied using relational algebra.

……

……

……

……

2.5 SUMMARY

This unit is an attempt to provide a detailed viewpoint of database design. The topics covered in this unit include the relational model including the representation of relations, operations such as set type operators and relational operators on relations. The E-R model explained in this unit covers the basic aspects of E-R modeling. E-R modeling is quite common to database design, therefore, you must attempt as many problems as possible from the further reading. The E-R diagram has also been extended. However, that is beyond the scope of this unit. You may refer to further readings for more details on E-R diagrams.

2.6 SOLUTIONS/ ANSWERS

Check Your Progress 1

1. Supplier Number (SNO), Supplier Name (SNAME) and City of the location of the supplier (CITY). The present instance s of the relation S has 2 tuples.
2. Domain of SNO is the codes that are being assigned to suppliers. The present coding uses the first character as S followed by the sequence number of a supplier. The domain of SNAME is strings of characters of some defined length and the domain of CITY is the set of possible cities in India.
3. The relation will not change; the order of tuples will change.
4. The super key of the relation could be (SNO, SNAME, CITY) or (SNO, SNAME) or (SNO, CITY) or assuming every supplier has unique name (SNAME, CITY) or SNO or assuming every supplier has unique name SNAME.
   The possible candidate keys are SNO or assuming every supplier has unique name SNAME
   Primary key may be selected as SNO and alternate key would be SNAME

Check Your Progress 2

1. Domain constraint for S will be for CITY which should be checked to be in a selected list of cities of India. SNO may be checked by a Range constraint and SNAME may be data type with maximum allowable length of name.
2. The Primary keys of relation S, P and J are SNO, PNO and JNO respectively. The primary key of SPJ is a composite key (SNO, PNO, JNO). As per the primary key constraint none of the attributes of primary key in SPJ, viz. SNO, PNO or JNO can be null.
3. There are three foreign keys, all of them are in SPJ relation. These are (i) SNO (ii) PNO (iii) JNO. SNO of SPJ references attribute SNO in S; PNO of SPJ references attribute PNO in P; and JNO of SPJ references attribute JNO in J. As per foreign key constraints the values of SNO, PNO and JNO in SPJ can be only as per the related referenced attributes.
4. The violations are as under:
   a. Primary key of S is SNO, it cannot be Null, you may change it to <S4, Jack, Mumbai>, which will be inserted successfully in S.

b. Primary key violation in SPJ as PNO cannot be Null as it is a part of primary key. There is another violation here, foreign key JNO has a value J3, which does not exist in JNO attribute of relation J for this instance, thus, the allowable values for JNO in SPJ is just J1 or J2. Thus, a correct record insertion in SPJ would be: <'S2', 'P3', 'J2', 200>.
c. The primary key 'P2' already exists and cannot be duplicated, the correct insertion may be to Insert <'P4, 'Pencil', 'Grey', 'Kolkata'> into P.
d. There is no violation.

Check Your Progress 3

1. relational algebra
2. A − (A − B)
3. (d) Join
4.
   i. Accepted
   ii. Rejected (candidate key uniqueness violation)
   iii. Rejected (violates RESTRICTED update rule, as SPJ contains tuples having value S5 in SNO)
   iv. Accepted (supplier S3 and all shipments for supplier S3 in the relation SPJ would be deleted, as the rule is CASCADE)
   v. Rejected (violates RESTRICTED delete rule, as SPJ contains tuples having value P2 in PNO)
   vi. Accepted (project J4 and all shipments for project J4 from the relation SPJ are deleted)
   vii. Accepted
   viii. Rejected (primary/candidate key uniqueness violation as tuple S5-P5-J7 already exists in relation SPJ)
   ix. Rejected (referential integrity violation as there exists no tuple for J8 in relation J)
   x. Accepted
   xi. Rejected (referential integrity violation as there exists no tuple for P7 in relation P)
   xii. Rejected (referential integrity violation – the default project number jjj does not exist in relation J).

5) The answer to this query will require the use of the relational algebraic operations. This can be found by performing selection of supplies made by S1 in SPJ, then taking projection of resultant on JNO and joining the resultant to J relation. Let us show steps:

First find out the supplies made by supplier S1 by selecting those tuples from SPJ where SNO is S1. The relation operator being:

$$SPJT = \sigma_{<SNO = 'S1'>} (SPJ)$$

The resulting temporary relation SPJT will be:

SNO	PNO	JNO	QUANTITY
S1	P1	J1	200
S1	P1	J4	700

Next, you may take projection of SPJT on PNO

$$SPJT2 = \Pi_{JNO} (SPJT)$$

The resulting temporary relation SPJT will be:

JNO
J1
J4

Next, take natural JOIN this table with J:

$$RESULT = SPJT2 \bowtie J$$

The resulting relation RESULT will be:

JNO	JNAME	CITY
J1	Sorter	Pune
J4	Console	Agra

You can write it as a single relational algebraic query as:

$$RESULT = (\Pi_{JNO} (\sigma_{<SNO = 'S1'>} (SPJ))) \bowtie J$$

UNIT 3 ENTITY RELATIONSHIP MODEL

3.0 INTRODUCTION

In the previous unit of this block, you have gone through the concept of relational database management systems and one of the important languages for relational database – relational algebra. This unit explains, you about of an analysis model of the database system, known as Entity-Relationship (E-R) model. The E-R model is a widely used model for analysis of data requirements of an organisation. The E-R model is primarily a semantic model and is very useful in creating raw database design that can be further normalised. With the availability of object-oriented technologies, the E-R model has been extended to include object-oriented features. This unit also discusses these E-R extensions. We will also discuss the conversion of E-R diagrams to tables, in this unit.

3.1 OBJECTIVES

After going through this unit, you should be able to:

- Define and explain various components of E-R model;
- draw an E-R diagram for a given problem;
- convert an E-R diagram to a relational database;
- Explain the role of extended features of E-R model;
- Convert an EER diagram to relations.

3.2 ENTITY RELATIONSHIP (E-R) MODEL

Some of the important characteristics of E-R model are listed below:

- *Entity relationship model is a high-level conceptual data model.*
- *It allows you to describe the data involved in a real-world enterprise in terms of entities and their relationships.*
- *It is widely used to create an initial design of a database.*
- *It provides a set of useful concepts that make it convenient for a developer to move from a basic set of information to a detailed and precise description of information that can be easily implemented in a database system.*
- *It describes data as a collection of entities, relationships and attributes.*

In the following sections, we explain the basic terms used in the E-R model.

3.2.1 Entities

First let us answer the question: **What are entities?**

- An entity is an object of concern, which is used to represent the things in the real world, e.g., car, table, book, etc.
- An entity need not be a physical entity, it can also represent a concept in the real world, e.g., project, loan, etc.
- It represents a class of things, not any one instance, e.g., 'STUDENT' entity has instances of 'Ramesh' and 'Mohan'.

Entity Set or Entity Type: A collection of a similar kind of entity is called an Entity Set or entity type.

For the COLLEGE database, the objects of concern are Student, Faculty, Course and Department. The collections of all the students' entities form an entity set STUDENT. Similarly, collection of all the courses form an entity set COURSE.

You may please note that entity sets need not be disjoint. For example – an entity, say Mohan, may be part of the entity set STUDENT, the entity set FACULTY and the entity set PERSON.

Entity identifier - key attributes: An entity set usually has one or more attributes, which attains unique value for every distinct entity in a given entity set. Such an attribute or set of attributes is/are called key attribute(s) and its values can be used to identify each entity uniquely in the given entity set.

Strong entity set: An entity set which contains at least one key attribute is a Strong entity set. For example, a Student entity set would contain at least one key attribute *Enrolment number*, which is unique for every student, thus, the entity set Student is a strong entity set.

Weak entity set: Entity sets that do not contain any key attribute, and hence cannot be identified independently, are called weak entity sets. A weak entity cannot be identified uniquely by its attributes, therefore, are recognised in conjunction with the

primary key attributes of another strong entity on which its existence is dependent (called owner entity set).

Generally, a primary key of an owner entity set is attached to a weak entity set, which has identifying attributes, called discriminator attributes. These two together form the primary key of the weak entity set. The following restrictions must hold for the above:

- The owner entity set and the weak entity set must participate in one to many relationship set. This relationship set is called the identifying relationship set of the weak entity set.
- The weak entity set must have total participation in the identifying relationship.

One of the most common examples about the weak entity set is an entity set Dependent and the related Strong entity set Employee in an organisation. The Dependent entity set is used to list all the dependents of each employee of an organisation. The attributes of the Dependent entity set are: Dependent name, birth date, gender and relationship with the employee. Each Employee entity is said to own the dependent entities that are related to it. However, please note that the 'Dependent' entity does not exist of its own, it is dependent on the Employee entity. In other words, you can say that in case an employee leaves the organization, all dependents related to him/her also leave along with this employee. Thus, a 'Dependent' entity has no significance without the entity 'Employee'. Thus, it is a weak entity.

3.2.2 Attributes

Let us first define - **What is an attribute?**

An attribute is an element of an entity, which can contain a representative value. In other words, an entity is represented by a set of attributes.

For example, a Student entity set may consist of attributes - Roll no, student's name, age, address, course, etc. An entity will have a value for each of its attributes. For example, for a particular student, the following values can be assigned:

Roll No:	1234
Name:	Mohan
Age:	18
Address:	Z-894, Maidan Garhi, Delhi.
Course:	B.Sc. (H)

Domains: Each simple attribute of an entity type contains a possible set of values that can be attached to it. This is called the domain of an attribute. An attribute cannot contain a value outside this domain.

EXAMPLE- for STUDENT entity Age has a specific domain, integer values say from 15 to 90.

Types of attributes

Attributes attached to an entity can be of various types. They are explained below:

Simple: An attribute that cannot be further divided into smaller parts and represents the basic meaning is called a simple attribute. For example: Each of the attributes - 'FirstName', 'LastName', age of PERSON entity set are simple attributes.

Composite: Attributes that can be further divided into smaller units and each smaller unit contains specific meaning. For example, the attribute NAME of a FACULTY entity can be subdivided into First name, Last name and Middle name.

Single valued: Attributes having a single value for a particular entity. For Example, Age is a single valued attribute of a STUDENT entity.

Multivalued: Attributes that have more than one value for a particular entity is called a multivalued attribute. Different entities may have different numbers of values for these kinds of attributes. For multivalued attributes you must also specify the minimum and maximum number of values that can be attached. For example, the phone number for a PERSON entity is a multivalued attribute.

Stored and derived: Attributes that are directly stored in the database are called stored attributes. For example, 'Birth Date' attribute of a PERSON entity can be a stored attribute. However, there are certain attributes, whose value can be computed from the value of the stored attribute. For example, in the PERSON entity, the attribute 'Birth Date' can be used to compute the attribute Age of a person on a specific day. Thus, 'Birth Date' is a stored attribute, whereas Age may be a derived attribute for this entity.

3.2.3 Relationships

First, let us define the term relationships, i.e. **What Are Relationships?**

A relationship can be defined as:

- a connection or set of associations, or
- a rule for communication among entities:

Example: In a COLLEGE database, the association between student and course entity set, i.e., in the statement "Student **opts** Course" **opts** is an example of a relationship between the two entities Student and Course.

Relationship Sets

A relationship set is a set of relationships of the same type. For example, consider the relationship between two entity sets STUDENT and COURSE. Collection of all the instances of relationship **opts** forms a relationship set.

3.2.4 E-R Diagram Basics

The logical structure of a database is modeled using an E-R model, which is graphically represented with the help of an E-R diagram. The basic symbols using in an E-R diagrams are given in the following table:

Object	Represented by
Entity Set	Rectangle
Attribute	Ellipse
Relationship Set	Diamonds

Figure 3.1 shows different kinds of entities, attributes, relationships and participation constraints, which are explained in this Unit.

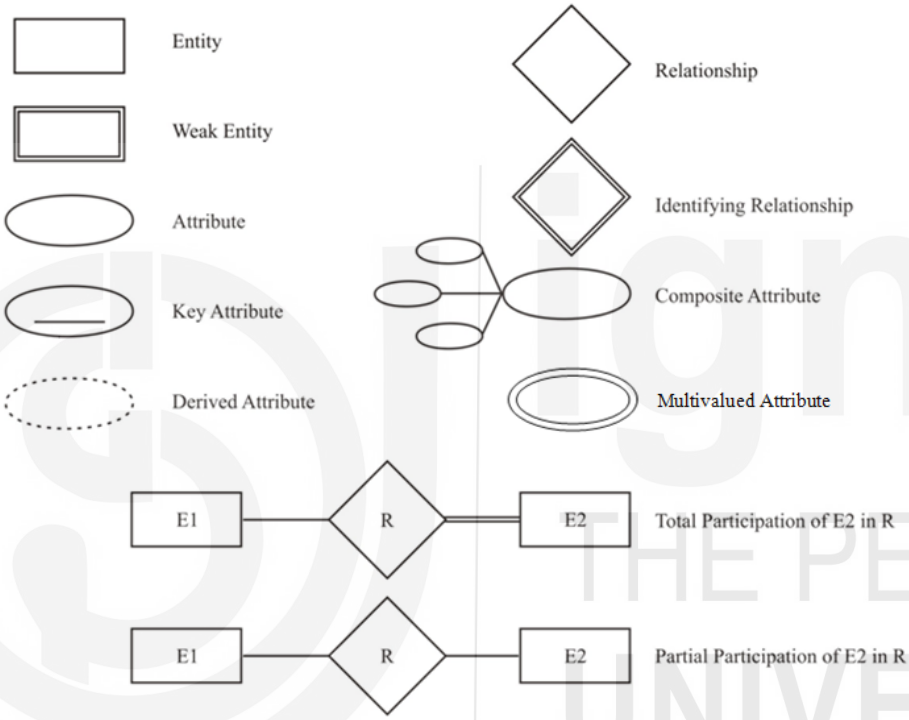


Figure 3.1: Symbols of E-R diagrams

3.2.5 More about Relationships

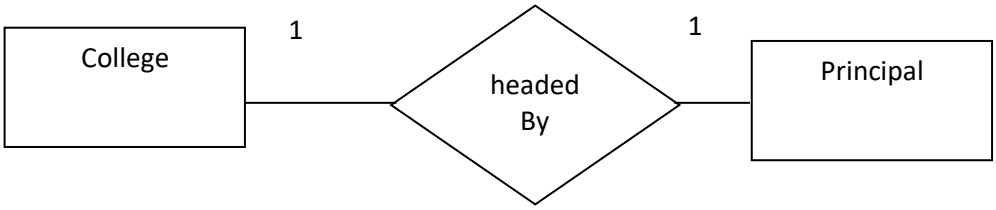
In this section, you will go through some of the important concepts, which are used for making good E-R models.

Degree of a relationship set: The degree of a relationship set is the number of participating Entity sets. The relationship between two entities is called a binary relationship. A relationship among three entities is called a ternary relationship. Similarly, a relationship among n entities is called an n-ary relationship.

Cardinality of a relationship set: Cardinality specifies the number of instances of an entity associated with another entity participating in a relationship. Based on the cardinality, binary relationships can be further classified into the following categories:

- *One-to-one*: An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

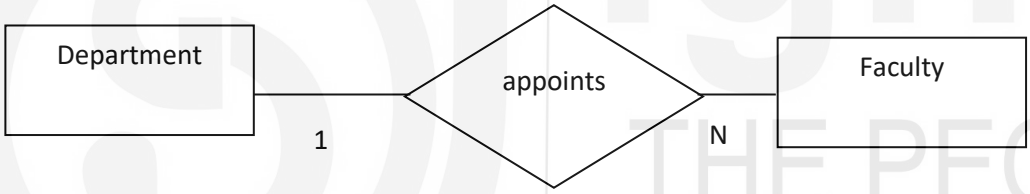
For example, the relationship *headedBy* between college entity set and principal entity set would be one-to-one, as one college can have at most one principal; and one principal can be principal of only one college. (This example assumes that a principal can be head of only one college.)



Similarly, you can define the relationship between University and Vice-Chancellor.

- *One-to-many*: Consider entity sets A and entity set B has a relationship cardinality A : B, as 1:N. This suggests that one specific entity of A may be related with several entities of entity set B.

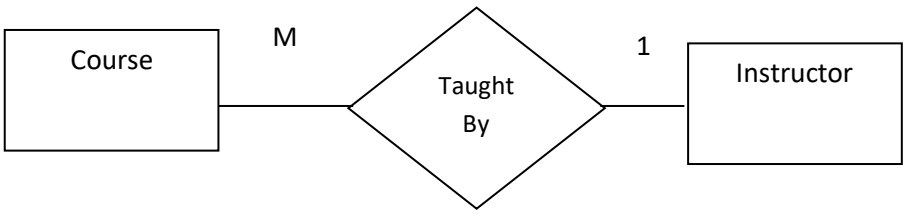
For example, relationship between Department entity set and Faculty entity set (assuming that a faculty member can work in only one department).



For example, in the diagram above, several faculty members may be appointed in one department, however, a specific faculty member will be appointed in precisely one department.

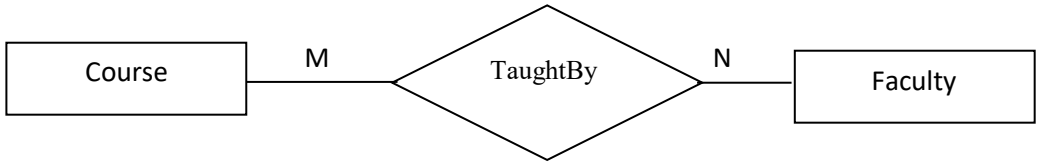
- *Many-to-one*: Consider entity sets A and entity set B has a relationship cardinality A : B, as N : 1. This suggests that several entities of A may be related with one specific entity of entity set B.

For example, the relationship between entity set Course and entity set Instructor. An instructor can teach various courses, but a single course can be taught only by one instructor. Please note this is an assumption.



Many-to-many: Entities in entity set A and entity set B are associated with any number of entities from each other.

For example, consider that a course can be taught jointly by many faculty members and each faculty member can teach several courses, then many-to-many relationship holds, as shown below:



Another example is shown in the diagram given below. The relationship cardinality M : N. This implies that an Author entity can be correlated to many Book entities, which are written by him/her. Further, a Book entity can also be correlated with several Author entities who have written the Book.

Recursive relationships: A recursive relationship is that relationships in which both the participating entity sets are the same entity set, however, the role of the entity set in each participation is different.

Participation constraints: The participation Constraints specify whether the existence of an entity depends on its being related to another entity via the relationship type. There are two types of participation constraints:

Total: When all the entities from an entity set participate in a relationship set, it is termed as the *total* participation. For example, the participation of the entity set Course in the relationship set 'TaughtBy' can be said to be *total* because every Course must be taught by a faculty.

Partial: When it is not necessary for all the entities from an entity set to participate in a relationship set, it is termed as *partial* participation. For example, the participation of the entity set Instructor in the relationship set 'TaughtBy' can be partial, since not every Instructor may be teaching a course.

3.2.6 Extended E-R Features

Although, the basic features of E-R diagrams are sufficient to design many database situations. However, with more complex relations and advanced database applications, it is required to use extended features of E-R models. The three such features are:

- Generalisation
- Specialisation, and
- Aggregation

We have explained them with the help of an example. More details on them are available in the further readings.

Example 1: A bank has an Account entity set. Any accounts of the bank can be one of two types: (1) Savings account and (2) Current account.
The statement above represents a specialisation/ generalisation hierarchy. It can be shown as:

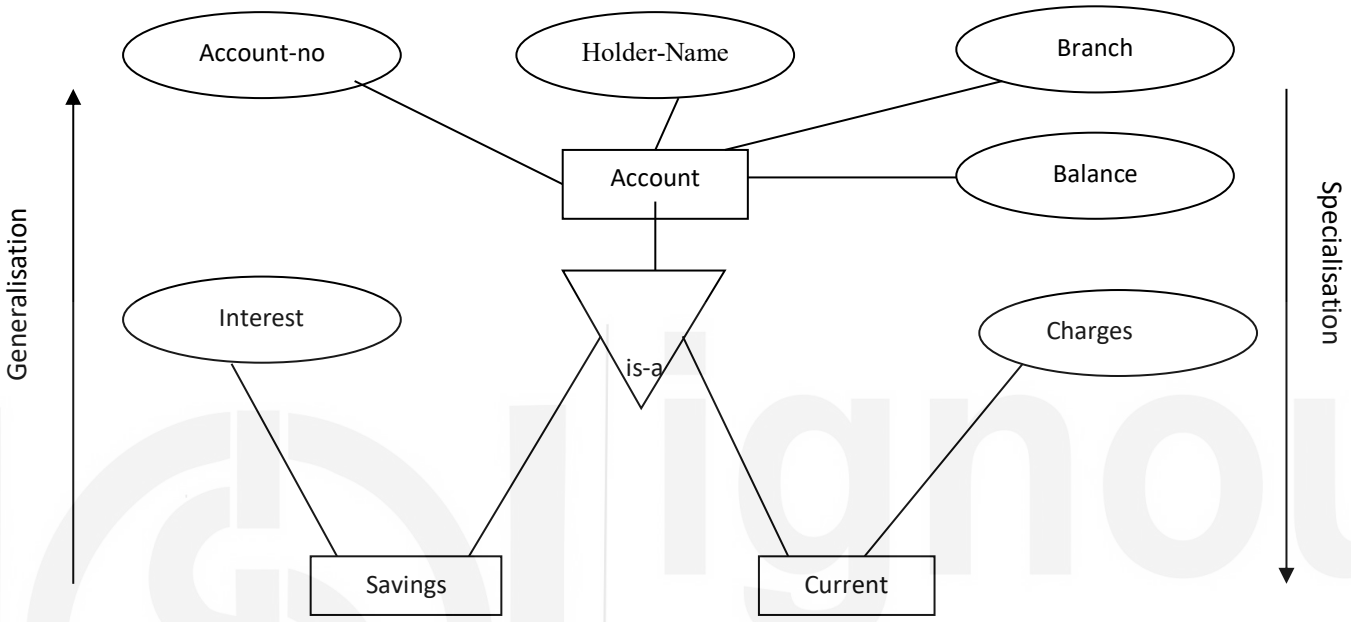


Figure 3.2: Generalisation and Specialisation hierarchy

Aggregation: One limitation of the E-R diagram is that they do not allow representation of relationships among relationships. In such a case the relationship along with its entities are promoted (aggregated to form an aggregate entity which can be used for expressing the required relationships). A detailed discussion on aggregation is beyond the scope of this unit you can refer to the further readings for more detail.

3.3 AN EXAMPLE

Let us explain it with the help of an example application. We will describe here an example database application of a COLLEGE database and use it for illustrating various E-R modeling concepts.

Problem Statement

A college database keeps track of Students, faculty, Departments and Courses. Following paragraphs gives the description of a COLLEGE database system.

A College contains various departments like Department of English, Department of Hindi, Department of Computer Science etc. Each department is assigned a unique

id and name. Some faculty members are appointed to each department and one of them works as head of the department.

There are various courses conducted by each department. Each course is assigned a unique id, name and duration.

The information contained for various objects are stated below:

- Faculty information contains name, address, department, basic salary etc. A faculty member is assigned to only one department but can teach courses of another department.
- Student's information contains Roll no (unique), Name, Address etc. A student can opt only for one course and one department only.
- Student's Parent or Guardian information to be recorded is - name of parent or guardian, age of the parent or guardian, gender and address of parent or guardian.

A student is allowed to take only one course. A student is assisted by his/her guardian. A faculty works in a department, which is headed by a faculty member. A course is taught by a faculty, who is allowed to teach several courses.

Defining Entities and Attributes

One of the simplest ways to identify the entities is to look for the proper nouns. This gives us possible set of entities in the problem statement are:

Student, Faculty, Department, Course, Guardian

The attributes of these entities can be found from the statements. You may also note except of Guardian:

Student (Rollno – Primary Key, Name, Address)
Faculty (Id – Primary Key, Name, Address, Basic_Sal)
Department (D_No, - Primary Key, D_Name)
Course (Course_ID – primary key, Course_Name, Duration)
Guardian (Name, Address, Relationship)

Please note that the Guardian is a weak entity. It is related to the strong entity Student.

Defining Relationship

Using the concepts defined earlier, we have identified that strong entities in COLLEGE database are Student, Faculty, Course and Department. This database also has one weak entity called Guardian. You can specify the following relationships among these entities. Further, these relationships also show the relationship cardinality and participation constraints:

1. **Head_of** is a 1:1 relationship between Faculty and Department. Participation of the entity Faculty is *partial* since not all the faculty members participate in this relationship (as all cannot be the head), while the participation from the Department side is *total* since every department has one head. Please also note that this relationship has an attribute Date_from, which stores the date from which the given appointment was applicable.

2. **Works_in** is a 1:N relationship between Department And Faculty, as one department can have many faculty, whereas a faculty is associated with only one department. Participation from both the sides is total, as every department has at least one faculty and every faculty must belong to a department.

3. **Opts** is a 1:N relationship between Course and Student. Participation from the Student side is *total* because we are assuming that each student opts for one course. But the participation from the course side is *partial*, since there can be courses that have no students.

4. **Taught_by** is a M: N relationship between Faculty and Course, as a Faculty can teach many courses and a Course can be taught by many faculty members.

5. **Enrolled** is a 1:N relationship between Student and Department as a student is allowed to enroll for only one department at a time. A student must enroll in a Department. However, a newly created Department may not have a student.

6. **Assisted_by** is a 1:N relationship between Student and Guardian as a student can have more than one local guardian and one local guardian is assumed to be related to one student only. The weak entity Guardian has *total* participation in the relation "Assisted_by".

Now, you are ready to make an E-R diagram for the college database. The E-R diagram.

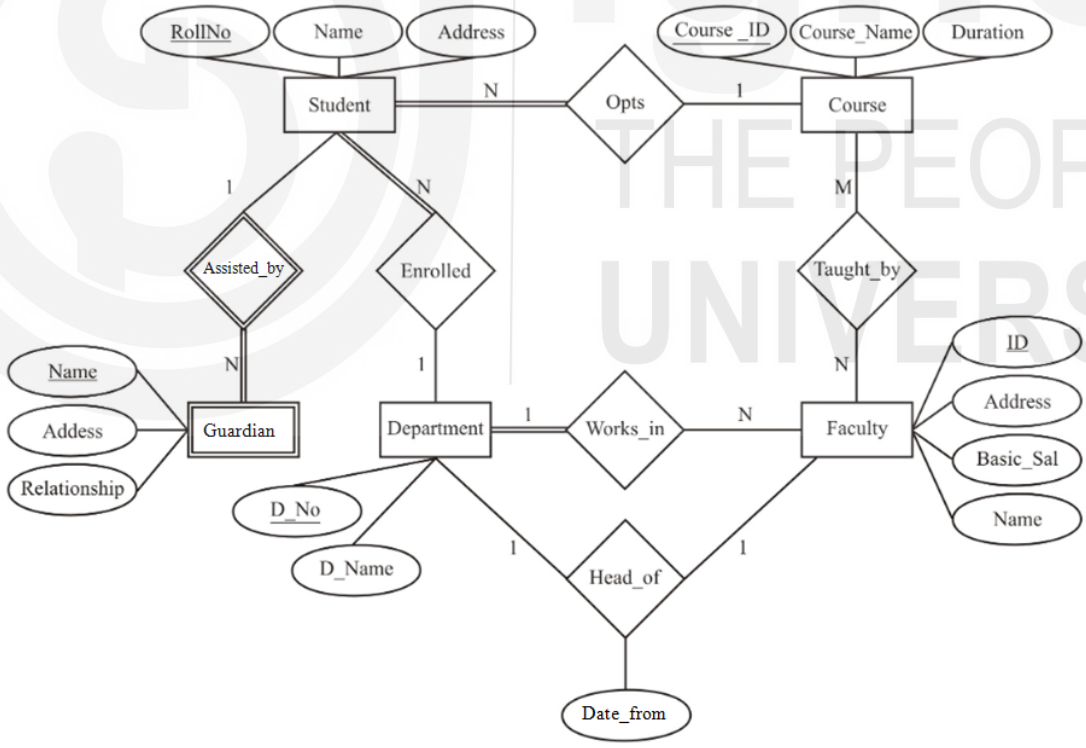


Figure 3.3: E-R diagram of COLLEGE database

3.4 CONVERSION OF E-R DIAGRAM TO RELATIONAL DATABASE

A relational database management system is designed from an E-R diagram, which represents various entities and relationships among entities for an application using the following methods.

Conversion of entity sets:

Strong entity set: For each strong entity set E in the E-R diagram, you create a relation R containing all the simple attributes of E. The primary key of the relation R will be one of the key attributes of R.

For example, the entities Student, Faculty, Course and Department, which are strong entities, relations as shown in Figure 3.4 would be created.

Student (Rollno, Name, Address)
Faculty (Id, Name, Address, Basic_Sal)
Department (D_No, D_Name)
Course (Course_ID, Course_Name, Duration)

Figure 3.4: Conversion of Strong Entities to relations

II) For each weak entity type W in the E-R Diagram, you create another relation R that contains all simple attributes of W. Further, you add the key attribute(s) of the owner entity set (say KP) of W in R. The primary key to this relation R is – <KP + Discriminator attribute of W> and foreign key is KP, which references the owner entity of W.

For example, conversion of weak entity Guardian into relation is shown in Figure 3.5. Please note that the owner entity of the Guardian entity is the strong entity Student, whose key is RollNo. Therefore, the key to Guardian relation is RollNo+Name. The Foreign key in Guardian relation is RollNo, which refers to Student relation.

Guardian (RollNO, Name, Address, Relationship)
Foreign Key: RollNO refers to relation Student

Figure 3.5: Conversion of weak entity Guardian to relation.

Conversion of relationship sets

Binary Relationships

I) One-to-one relationships:

For each 1:1 relationship set in the E-R diagram involving two entities E1 and E2 you choose one of the two entities (say E1), preferably the one with total participation, and add the primary key attribute of the other entity E2, in the relation of entity E1. Make this added attribute in E1 relation as a foreign key attribute to the relation

created from another entity (E2). You should also include all the simple attributes of the relationship type, if any, in the relation E1.

For example, the Head_of relationship in Figure 3.3 is 1:1. The two entities participating in the relationship are - Department and Faculty. The participation of the Department entity is *total* in the Head_of relationship. Therefore, the ID attribute, which is the primary key of the Faculty entity is added to Department relation. Please note, you can rename this attribute in the Department relation, if needed. In addition, this attribute in Department relation will be the foreign key to the Faculty relation. Further, the attribute Date_from of the Head_of Relationship is also added to the Department relation. This is shown in Figure 3.6. Please note that you will keep information in this the relation only stores the ID of the current head and Date from which s/he is the head. Please also note that you will not create a separate table of the relationship Head_of.

Department (D_No, D_Name, Head_ID, Date_from))

The ID attribute of Faculty relations is added to Department relation and has been renamed as Head_ID.

Foreign Key: Head_ID references ID attribute in Faculty relation

Figure 3.6: Converting 1:1 relationship (No new relation is added)

II) One-to-many or many-to-one relationships:

Both relationship sets involve two entity sets, say E1 and E2. Further, assume that E1 is on the many side and E2 is on the one side of the relationship set. You just need to include the primary key of the relation of entity on the one side (E2 in the case as above) to the relation created for the entity set of many side (E1 in the present case). You should include all simple attributes (or simple components of a composite attributes of relationship set, if any) in the relation of E1. Please note that now the relation of E1 has a foreign key reference to the relation of E2.

For example, the Works_in relationship between the entities Department and Faculty. For this relationship, the entity at N side is Faculty, therefore add primary key attribute of entity Department, i.e., D_No as a foreign key attribute in relation created for Faculty entity set. This is shown in Figure 3.7

Faculty (Id, Name, Address, Basic_Sal, D_No))

The Works_in relationship has been included in Faculty relation.

D_No is a foreign key and references the relation Department.

Figure 3.7: Converting 1:N relationship to relation (No new table is added in this case)

III) Many-to-many (M : N) relationship:

Consider that a M : N relationship set is binary with two participating entities, say Entity1 and Entity2. For this type of relationship set a relation is created. This relationship set should contain the Primary key of Entity1 (sat PKE1), as well as the Primary key of Entity2 (say PKE2). In addition, any attribute of the relationship set is added to the relation. The primary key of this newly formed relation of the M : N relationship set is the composite primary key PKE1+PKE2. Please also note that for this new relation of the relationship set, there exists two foreign keys – PKE1, which refers to the relation of Entity1 and PKE2, which refers to the relation of Entity set Entity2.

For example, the m : n relationship Taught_by between entity sets Course and Faculty should be represented as a new table. The structure of the table will include the primary key of Course and primary key of Faculty entities. Please note that both Course and Faculty relations remain unchanged.

Add the following relation into already existing list of relations:

Taught_by (Course_ID, ID)

Primary Key: Course_ID + ID

Foreign Keys:

Course_ID references Course relation

ID references Faculty relation

This relation has no other attribute, as the relationship has no attribute.

Figure 3.8: Converting m : n relationship Taught_by into relations

N-ary Relationships

There are several cases for creating relations for n-ary relationships. A very general case is presented here. For each n-ary relationship set R where n > 2, you create a new table S to represent R. You should include the primary key of all the participating entity sets as the foreign key attributes in S. You should include any simple attributes of the n-ary relationship set (or simple components of complete attributes) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity sets. Figure 3.8 is a special case of n-ary relationship, i.e. a binary relationship.

Multivalued attributes:

For each multivalued attribute 'A' of an entity set E, you can create a new relation R that includes an attribute corresponding to the primary key attribute of the relation entity E that represents the entity set or relationship set that has as an attribute. The primary key of R is then a combination of A and the primary key of relation of E. For example, if a Student entity had RollNo, Name and PhoneNumber attributes, where phone number is a multivalued attribute, then you will create two tables for this entity as given below:

Student (RollNo, Name)

Phone (RollNo, PhoneNumber)

Converting Generalisation / Specialisation hierarchy to tables:

A simple rule for conversation may be to decompose all the specialised entities into relations in case they are disjoint. For example, for the E-R diagram of Figure 3.1, you can create the two tables as:

Saving_account (account-no, holder-name, branch, balance, interest).

Current_account (account-no, holder-name, branch, balance, charges).

The other way might be to create tables that are overlapping (not disjoint) for example, assuming that in the E-R diagram of Figure 3.2 contains overlapping sub-classes, then you would be creating the following three relations:

The first relation would be for the for higher level entity:

account (account-no, holder-name, branch, balance)

The specialisation entities will contain the Primary key of the generalised entity and all the attributes of entity itself, as shown below:

saving (account-no, interest)

current (account-no, charges)

Thus, the information about a single account can be found in all the three relations.

Check Your Progress 1

1) A company wants to develop an application to store information about its clients. Find the possible entities and relationships among the entities. Show the step-by-step procedure to make an E-R diagram for the application.

……..……
……..……
……..……
……

2) An employee works for a department. If the employee is a manager, then s/he manages the department. Every employee works on at least one or more projects, which are controlled by various departments of a company. An employee can have many dependents. Draw an E-R diagram for the above company. Find all possible entities and their relationships.

……..……
……..……
……..……
……

3) A supplier, located in only one-city, supplies various parts for the projects of different companies located in various cities. You can name this database as "supplier-and-parts". Draw the E-R diagram for the supplier-and-parts.

……..……
……..……
……..……..……
……

4) Convert the E-R diagram created for question 2 above into a relational database.
……
……

3.5 ENHANCED E-R MODEL

Enhanced E-R models can help in designing of relational and object-relational database systems. In addition, to E-R modeling concepts, the Enhanced ER model includes:

1) Subclass and Super class
2) Inheritance
3) Specialisation and Generalisation.

As discussed earlier, an entity may be an object with physical existence or it may be an object with a conceptual existence. An entity is depicted by a set of attributes. Sometimes, an entity can consist of explicit sub-groupings. For example, an entity *vehicle* consists of sub-groups – Truck (or Commercial Vehicles), Light Motor Vehicles (or Car), Two-Wheelers (or Scooter), etc. Every sub-grouping must belong to entity set *vehicle*, therefore, these sub-groupings are called a subclass of the *vehicle* entity set and the *vehicle* itself is called the super class for each of these sub-classes.

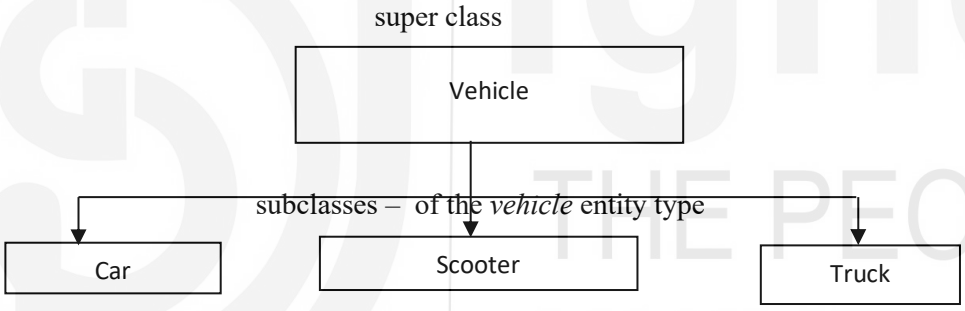


Figure 3.9: A class hierarchy

The relationship between a super class and any of its subclasses is called class/subclass relationship. It is often called an IS-A relationship because of the way we refer to the concept, as you would write, "Car ***is-a*** vehicle". The member entity of the sub-class represents the same real world as the member entity of the super class. If an entity is a member of a sub-class, by default it must also become a member of the super class; whereas it is not necessary that every entity of the super class must be a member of its sub-class. An entity that is a member of a sub-class inherits all the attributes of its super class. An entity set is identified by its attributes and the relationship sets in which it participates; therefore, a sub-class entity also inherits all the relationships in which the super class participates. According to **inheritance** the sub-class inherits attributes and relationships of the super class. In addition, every subclass can contain its own attributes and relationships in which it has participated.

Specialisation is a process in which an entity set (super class) is modeled as a set of sub-entity sets (sub-classes) by using a specific distinguishing characteristic of the

super class. For example, in Figure 3.9, the super class *vehicle* is modeled using sub classes - Truck (or Commercial vehicle), Car (or Light Motor Vehicle), Scooter (or Two-wheelers)) using the *Type* attribute of the *vehicle* class. Please note that several specialisations hierarchies can be modeled using a super class by using different distinguishing characteristics. Figure 3.10 shows how you can represent a specialisation with the help of an EER diagram.

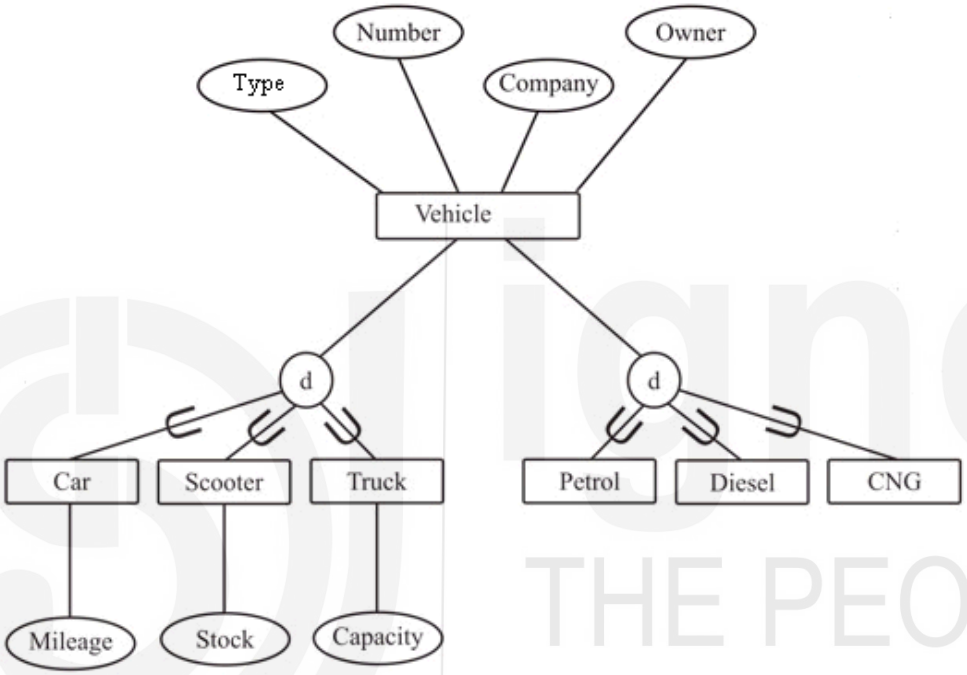


Figure 3.10: EER diagram showing more than one specialization from one super class

In Figure 3.10, letter 'd' in the circle indicates that all these subclasses are **disjoint** in nature, i.e. all the vehicle entities are disjoint, as they can be part of only one of the subclass. Please also notice that in Figure 3.10, common attributes, like vehicle number, owner name etc., are attributes of the super class, whereas attributes like mileage of car, stock of scooter and capacity of truck are the attributes of the sub-classes. Please note that an entity will be appearing twice in the EER diagram – once in the sub-class and the other in the super class (Please refer to Figure 3.11).

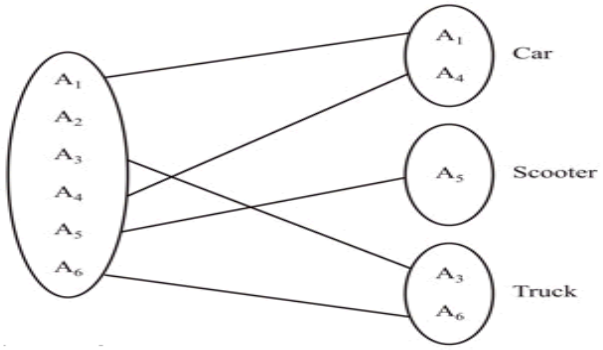


Figure 3.11: Sharing of members of the super class vehicle and its sub-classes

Hence, the specialisation is a set of sub-classes of an entity set, which establishes additional specific attributes with each sub-class and establishes additional specific relationship sets between a sub-class and other entity types or other sub-classes.

Generalisation is the reverse process of specialisation; in other words, it is a process of representing entity sets consisting of entities, which have almost similar attributes excepting few. The similar attributes of these entity set form the super class. For example, the entity set Car and Truck can be generalised into entity set Vehicle. Therefore, Car and Truck can now be sub-classes of the super class generalised class Vehicle (Refer to Figure 3.12).

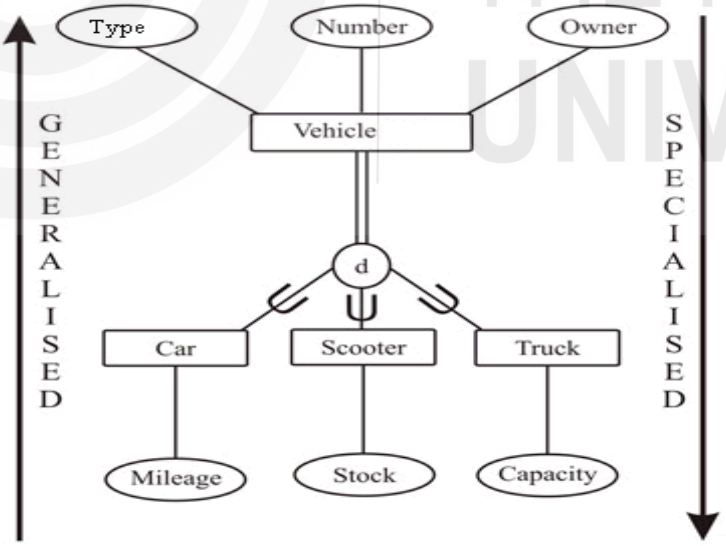


Figure 3.12: Generalisation and Specialisation

Constraints and Characteristics of Specialisation and Generalisation: A super class may either have a single sub-class or many sub-classes in specialisation. In

case of only one sub-class you do not use circle notation to show the relationship of sub-class/super class. Sometimes in specialisation, the subclass becomes the member of the super class after satisfying a condition on the value of some attributes of the super class. Such sub-classes are called *condition-defined* sub-classes or predicate defined subclasses. For example, the Vehicle entity set has an attribute vehicle "Type", as shown in Figure 3.12.

You can specify the condition of membership for a subclass – car, truck, scooter – by the predicate – vehicle 'Type' of the super class vehicle. Therefore, a vehicle object can be a member of the sub-class, if it satisfies the membership condition for that sub-class. For example, to be a member of sub-class Car a vehicle entity must have the condition vehicle "type = car" as true. A specialisation in which an attribute or a set of attributes of the super class (called the *defining attribute* of specialisation) specify membership condition of the sub-classes, is termed as *attribute-defined* specialisation. In case no membership condition is stated for specialisation, a database user determines the membership. Such specialisation is termed as user-defined specialisation.

Disjointness is also the constraints to a specialisation, which specifies that a given entity cannot be a member of more than one sub-classes for a specific specialisation hierarchy. For example, in Figure 3.12, the symbol 'd' in circle stands for disjointness, as an entity can be a member of a single sub-class only. But if the real-world entity is not a disjoint entity sets of the sub-classes may overlap. This is represented by an (o) in the circle. For example, if you classify a class Book into sub-classes Textbooks and Reference Books, then you may like to define a specific book in both the sub-classes. This is a case of Overlapping constraints.

When every entity in the super class must be a member of some subclass in the specialisation it is called total specialisation. But if every entity does not necessarily need to belong to any of the subclasses, it is called partial specialisation. The total is represented by a double line. This is to note that in specialisation and generalisation the deletion of an entity from a super class implies automatic deletion from sub-classes belonging to the same; similarly, insertion of an entity in a super class results in insertion of the entity in all the sub-classes for which the attributes of this entity fulfills the constraints of attribute-defined specialisation. In case of total specialisation, insertion of an entity in a super class implies compulsory insertion in at least one of the sub-classes of the specialisation.

Union: In some cases, a single class has a similar relationship with more than one class. For example, the sub class 'Car' may be owned by two different types of owners: Individual or Organisation. Both these types of owners are different classes; thus, such a situation can be modeled with the help of a Union (Refer to Figure 3.13).

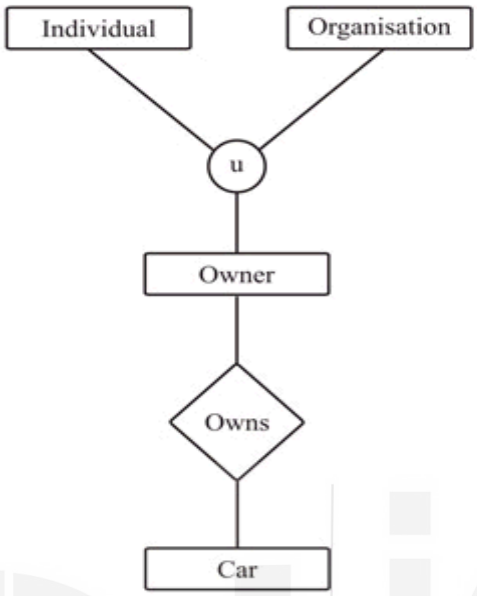


Figure 3.13: Union of classes

In the next section, we discuss how these extended features can be converted to relations.

3.6 CONVERTING EER DIAGRAM TO RELATIONS

The rules for converting the EER diagram, which primarily includes specialisation and generalisation hierarchy are the same as in the E-R diagram. Let us recapitulate these rules:

- Create a relation for each strong entity set.
- Create a relation for a weak entity set. The primary key of this relation would be a composite key involving the attributes of the primary key of the strong entity set on which it depends and discriminator of the weak entity.
- Create a relation for each binary m : n relationship set having the primary keys of both the participating entities, which form the composite primary key to the relation. The individual primary keys are the foreign keys to respective relations.
- For a binary m : 1 or 1 : m relationship, in general, the primary key on the m side is added to 1 side of the entity. This also becomes the foreign key. For abinary 1:1 relationship set the primary key of chosen participating relation is added to the other participating relation.
- Composite attributes may sometimes be converted to a separate relation.

- For generalisation or specialisation hierarchy a relation can be created for higher level and each of the lower-level entities. The higher-level entity would have the common attributes and each lower-level relation would have the primary key of the higher-level entity and the attributes defined at the lower specialised level. However, for a complete disjoint hierarchy no relation may be made at the higher level, but the relations are made at the lower level including the attributes of higher level.
- For an aggregation, all the entities and relationships of the aggregation are transformed into the relation on the basis of rules stated above. A relation is also created for the relationship set that exists between the aggregated entity (say AgE) and another simple entity (SE). The primary key of this new relation is the composite key involving the primary key of the AgE and SE.

So let us now discuss the process of converting the EER diagram into a relation. In case of disjoint constraints with total participation. It is advisable to create separate relations for the sub-classes. But the only problem in such a case will be to implement the referential entity constraints suitably.

For example, assuming that this EER diagram can be converted into a relation as:

Car (Number, owner, mileage)
Scooter (Number, owner, stock)
Truck (Number, owner, capacity)

Please note that referential integrity constraint in this case would require a relationship with three relations and therefore is more complex to implement.

In case, in the EER diagram of Figure 3.12 there is NO total participation of Vehicle super class in the sub-classes, then there will be some vehicles, which are not Car, Scooter and Truck, so how can you represent these? In addition, in case of overlapping constraints, some tuples may get represented in more than one relation. Thus, in such cases, it is ideal to create one relation for the super class and other relations for the sub-classes having the primary key and any other attributes of that sub-class. For example, with NO total participation the following relations would be created for the EER diagram of Figure 3.12:

Vehicle (Number, owner, type)
Car (Number, mileage)
Scooter (Number, stock)
Truck (Number, capacity)

Finally, in the case of union since it represents dissimilar classes, you may represent separate relations. For example, both individual and organisation will be modeled to separate relations.

Check Your Progress 2

1) What is the use of the EER diagram?

……

……

……

2) What are the constraints used in EER diagrams?

……

……

……

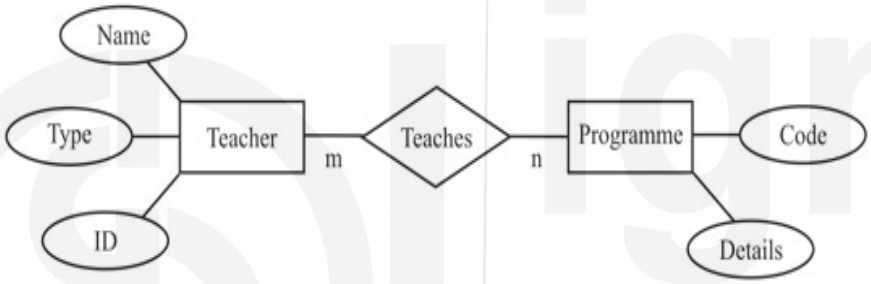
3) How is an EER diagram converted into a relation?

……

……

……

4) Consider the following E-R diagram.



'Type' can be regular or visiting faculty. Visiting faculty members can teach only one programme. Make a suitable EER diagram for this and convert the EER diagram to table.

……

……

……

3.7 SUMMARY

This unit presents the concept of E-R model and EER models. Both these models are represented with the help of E-R diagram and EER diagram. These diagrams are very powerful tools to represent the need of data in a database system and can be used for the design of a good database system. The E-R model explained in this unit covers the basic aspects of E-R modeling. The unit defines the concept of entities, attributes and relationships. Further, it defines different types of entities like strong and weak entities; different types of attributes such as simple, composite, derived etc.; the cardinality and participation constraints in a relationship. These concepts are very useful and you should attempt solving related problems from the further readings. Concepts of EER diagrams including generalisation, specialisation, union

etc. have also been explained in this unit. You may refer to further readings for more details on E-R and EER diagrams.

3.8 SOLUTIONS/ ANSWERS

Check Your Progress 1

1.

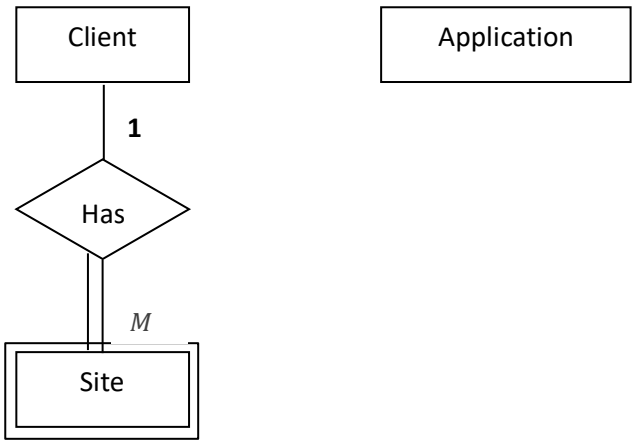
Let us show the step-by-step process of development of Entity-Relationship Diagram for the Client Application system. The first two important entities of the system are **Client** and **Application**. Each of these terms represents a noun, thus, they are eligible to be the entities in the database. But are they the correct entity sets? Client and Application both are **independent** of anything else and the company plans to keep track of its clients and the applications being developed for them. Therefore, each of the entities-Client and Application form an entity set.

But how are these entities related? Are there more entities in the system? Let us first consider the relationship between these two entities, if any. Obviously, the relationship among the entities depends on interpretation of written requirements. Thus, we need to define the terms in more detail.

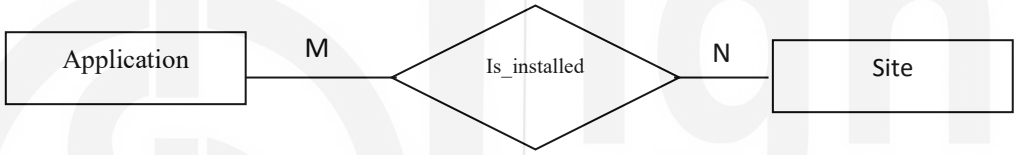
Let us first define the term **Application.** Some of the questions that are to be answered in this regard are: Is keeping track of **Accounts** an application? Is the accounting system installed at each client site regarded as a different application? Can the same application be installed more than once at a particular client site?

Before you answer these questions, do you notice that another entity is in the offering? The client **site** seems to be another candidate entity. This is the kind of thing you need to be sensitive to at this stage of the development of the entity relationship modeling process. So, let us first deal with the relationship between Client and Site before coming back to Application. Just a word of advice: "It is often easier to tackle what seems likely to prove simple before trying to resolve the apparently complex."

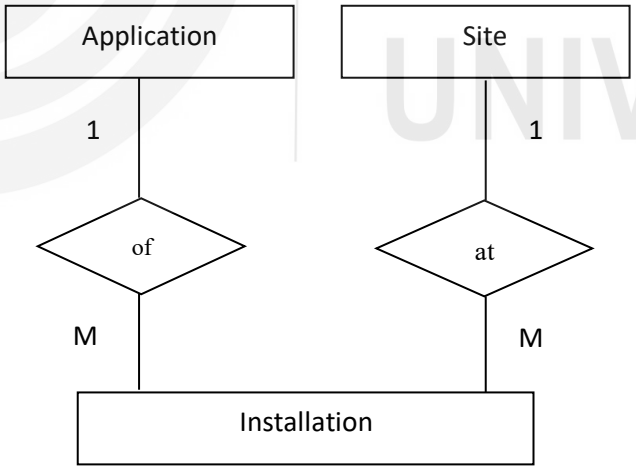
Each Client can have many sites, but each site belongs to one and only one client. Now the question arises what entity type is Site? You cannot have a site without a client. If any site exists without a client, then who would pay the company? This is a good example of an existential dependency and a one-to-many relationship. Thus, Site is a weak entity. This is illustrated in the part E-R diagram given below:



Let us now relate the entity Application to other entities. Please note that several applications developed by the company can be installed at several client sites. Thus, there exists a many-to-many relationship between the entities Site and Application:



However, the M: M relationship "is_installed" has many attributes that need to be stored with it, specifically relating to the authorised persons, setup constraints, dates, etc. Thus, it may be a good idea to promote this relationship as an Entity.



The Figure given above consists of the following relationships:

of - relationship describing the installation of applications and

at - relationship describing the installation at sites.

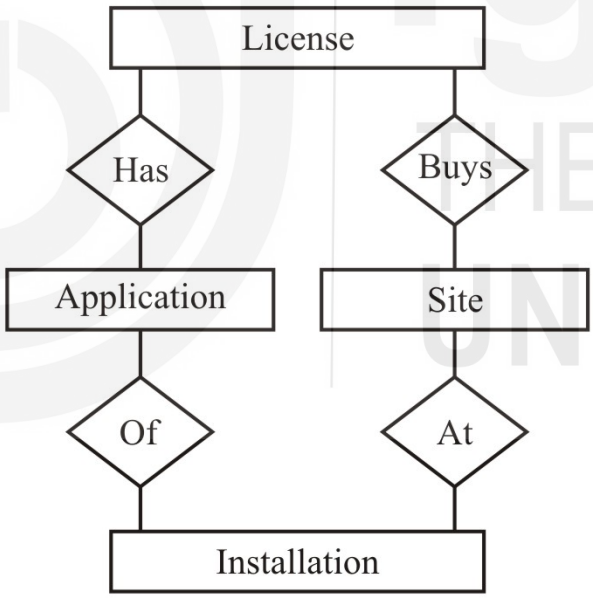
Please note that entities can be recognised in one of two ways – from the nouns of the requirement specification of the system or because of resolving a M:M relationship, as in this case. When you create a new entity in this way, you must find a suitable name for it. This can sometimes be based on the **verbs** used to describe

the M:M. For example, from the statement **you can install the application at many sites,** you can choose the verb install and convert it to related noun **Installation**. But what how will you identify the attributes and relationships of the Installation entity? To find these, you may like to answer the following questions:

- How do you desire to identify each Installation entity?
- What data would be stored in the Installation entity?
- Is an Installation independent of any other entity, that is, can an entity Installation exist without being associated with the entities Client, Site and Application?

In the present design, there cannot be an Installation until you can specify the Client, Site and Application. But since Site is existentially dependent on Client or in other words, Site is subordinate to Client, the Installation can be identified by (Client) Site (it means Client or Site) and Application. You do not want to allow more than one record for the same site and application.

But what if we also sell multiple copies of packages for a site? In such a case, you need to keep track of each individual copy (license number) at each site. In that case, you need another entity named Package (with license number). You may even need separate entities for Application and Package. This will depend on what attributes you want to keep in each of these entities. Thus, with these requirements, the E-R diagram may be extended to as shown below:



Let us define the additional relationships given above:

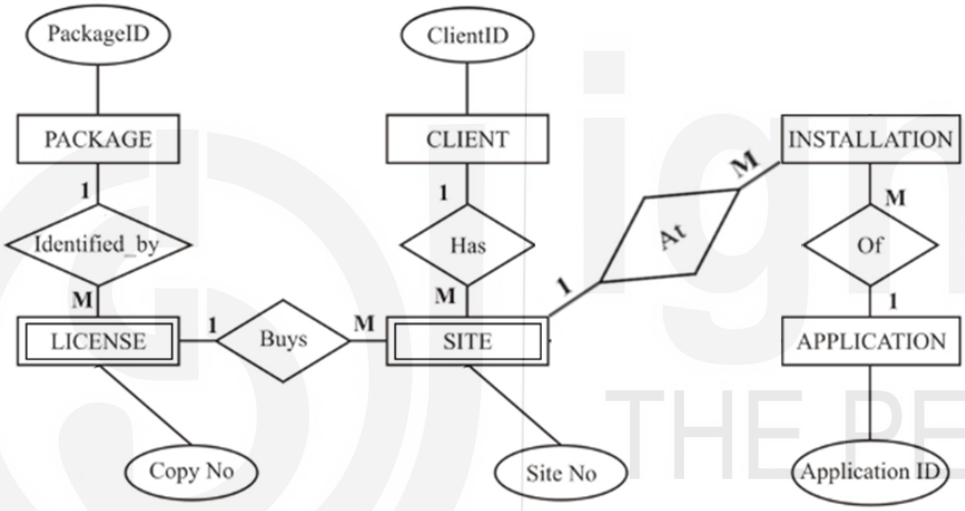
Has: describes that each application has one or more licenses

Buys: describes each site buys the licensed copies of application

You might decide that License should be sub-class to Package entity, therefore, the best unique identifier for the License entity could be Package ID and License serial number. The reason for this relationship is that the license numbers are issued by the companies to whom the Package belongs. The present company does not issue license numbers, and thus have no control over their duration, other service requirements, as

well as the length of data types used for different attributes of the Package and License entity sets. Also, the company does not have control over their uniqueness and changeability of license numbers. Please note that **it is safer to base primary keys on internally assigned values.** What if you make License as a sub-class to Package entity set? It also seems that the client site is not an essential part of the identifier, as the client is free to move the copy of a package to another site. Thus, we should definitely not make client/site part of the primary key of License.

A final proposed E-R diagram for the problem is given below. Please keep thinking and refining your reasoning. Please note that knowing and thinking about a system is essential for making good E-R diagrams. (Please note that in this E-R Diagram, Site and License are modeled as weak entities, though you can decide to change it.)

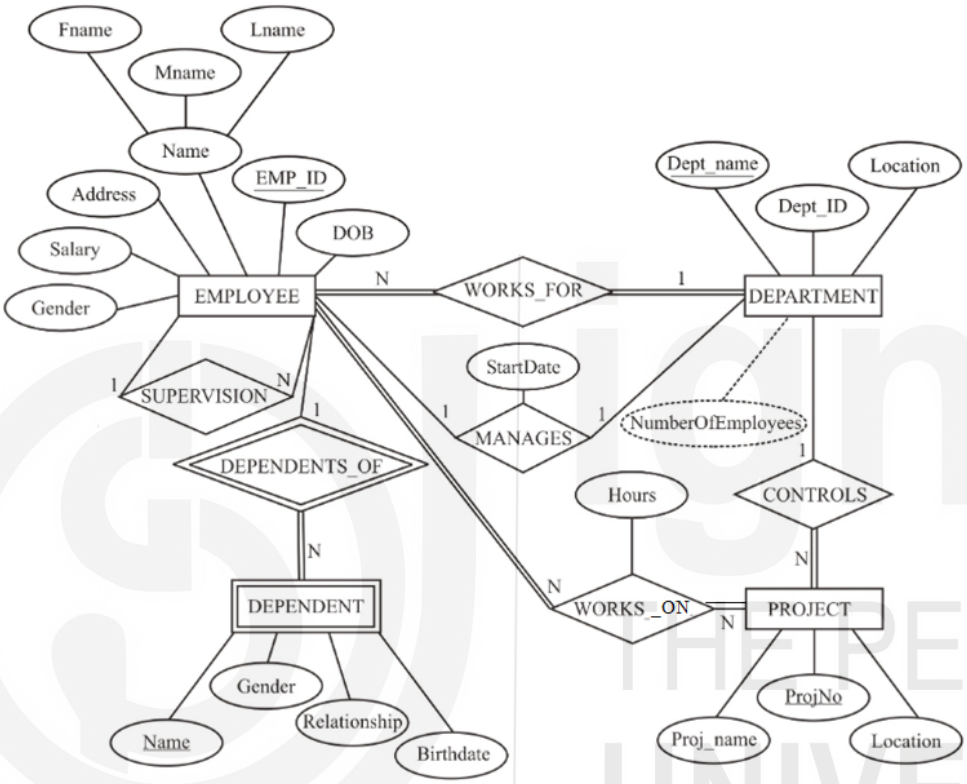


The following table lists probable entities identified so far, together with its superclass, if any, primary keys, and any foreign keys.

Entity	**Super Class (if any)**	**Primary Key**	**Foreign Keys**
Client	-	Client ID	
Site	Client	Client ID, Site No	Client ID
Application	-	Application ID	
Package	-	Package ID	
Installation	Site, Application	Client ID, Site No, Application ID	Client ID, Site No, Application ID

License	Package	Package ID, Copy No	Package ID

2) The E-R diagram is given below:



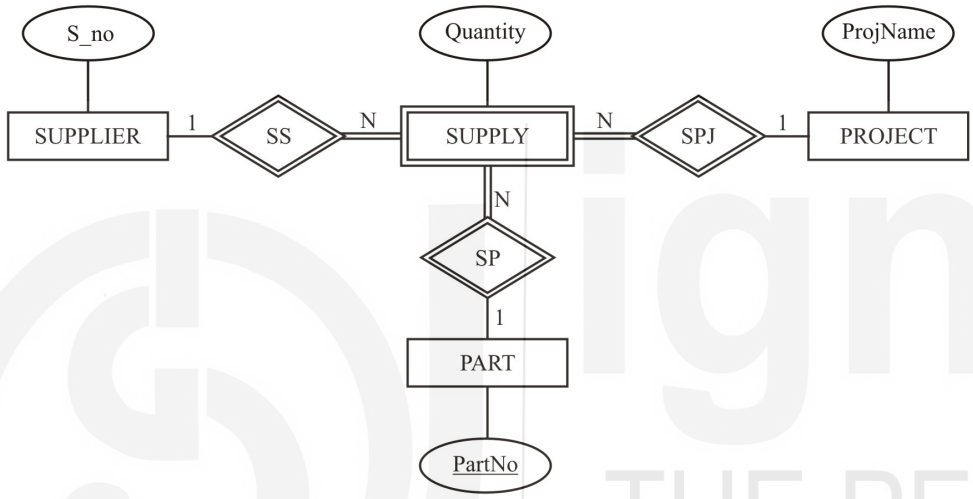
In the E-R diagram given above, EMPLOYEE is an entity, who works for a department, i.e., entity DEPARTMENT, thus, WORKS_FOR is many-to-one relationship, as many employees work for one department. Only one employee (i.e., Manager) manages the department, thus **manages** is the one-to-one relationship. The attribute Emp_Id is the primary key for the entity EMPLOYEE, thus Emp_Id is unique and NOT NULL. The candidate keys for the entity DEPARTMENT are **Dept_name** and **Dept_Id**. Along with other attributes, NumberOfEmployees is the derived attribute on the entity DEPARTMENT, which is the number of employees working for a department. Both the entities EMPLOYEE and DEPARTMENT participate totally in the relationship WORKS_FOR, as at least one employee works for the department, similarly an employee must work in a department.

The entity EMPLOYEES and the entity PROJECTS are related though the many-to many relationship WORKS_ON, as many employees can work for one or more than one projects simultaneously. The entity DEPARTMENT and entity PROJECT has a relationship CONTROLS. Since one department controls many projects, thus,

CONTROLS in a 1:N relationship. The entity EMPLOYEE participates totally in the relationship WORKS_ON, as each employee works on at least one project. A project should also have at least one employee, therefore, its participation is also total in WORKS_ON.

The employees can have many dependents, but the entity DEPENDENTS cannot exist without the existence of the entity EMPLOYEE, thus, DEPENDENT is a weak entity. You can very well see the primary keys for all the entities. The underlined attributes in the eclipses represent the primary key.

3. The E-R diagram for supplier-and-parts database is given as follows:



4. Let us first make a simple relation for the E-R diagram in the answer to question 2:

EMPLOYEE(EMP_ID, Fname, Mname, Lname, DOB, Address, Salary, Gender)
DEPARTMENT(Dept_ID, Dept_name, Location)
DEPENDENT(EMP_ID, Name, Gender, Relationship, Birthdate)
Foreign Key: EMP_ID references EMPLOYEE
PROJECT(ProjNO, Proj_name, Location)
WORKS_FOR: due to this relationship the Primary key of 1 side (Dept_ID) will be added to the EMPLOYEE relation.
SUPERVISION: this relationship is on the same entity, therefore, an attribute Supervisor_ID whose domain is EMP_ID will be added to the EMPLOYEE
MANAGES: It is a 1 : 1 relation, you can choose the Department side as there will be less records. It also has an attribute StartDate. Therefore, MANAGER_ID whose domain is EMP_ID and StartDate attribute would be added to DEPARTMENT.
CONTROLS: It is a 1 : N relationship, so the Primary key of 1 side (Dept_ID) will be added to the PROJECT relation.
DEPENDENT_OF: This relation is already included in DEPENDENT relation.
WORKS_ON: It is a many to many (N : N) relation with total participation on both sides, therefore, a separate table will be created for WORKS_ON with primary key of both the participating entities and attributes of this relation (Hours)

Thus, the final relations would be:

EMPLOYEE (EMP_ID, Fname, Mname, Lname, DOB, Address, Salary,
Gender, Dept_ID, Supervisor_ID)
Foreign Key: Dept_ID references DEPARTMENT.
Domain Constraint: Domain of Supervisor_ID is EMP_ID.
DEPARTMENT (Dept_ID, Dept_name, Location, MANAGER_ID, StartDate)
Foreign Key: MANAGER_ID references EMP_ID of EMPLOYEE.
DEPENDENT (EMP_ID, Name, Gender, Relationship, Birthdate)
Foreign Key: EMP_ID references EMPLOYEE
PROJECT (ProjNO, Proj_name, Location, Dept_ID)
Foreign Key: Dept_ID references DEPARTMENT.
WORKS_ON (EMP_ID, ProjNo, Hours)

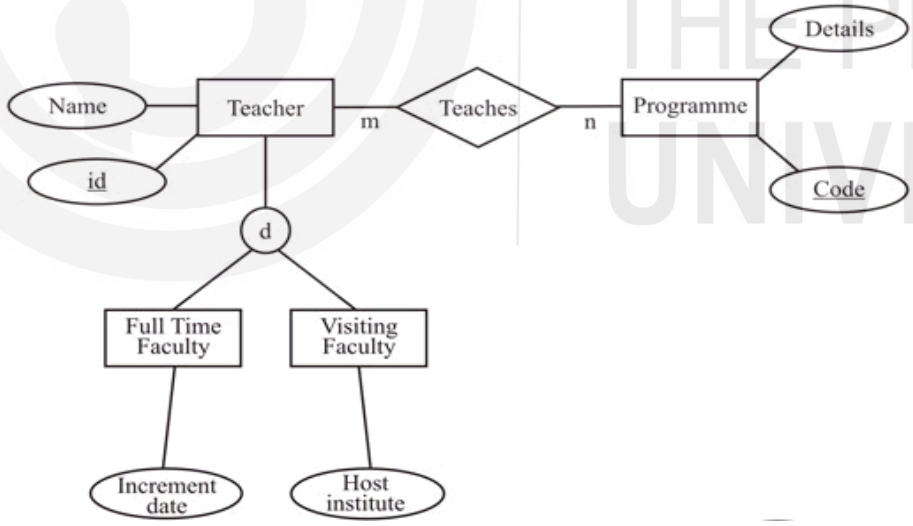
Check Your Progress 2

1) The EER diagrams are used to model advanced data models requiring inheritance, specialisation and generalisation.

2) The basic constraints used in EER diagrams are disjointness, overlapping and unions.

3) For disjointness and union constraints the chances are that you create separate relations for the subclasses and no relation for super class. For overlapping constraints, it is advisable to create a relation of super class and the relations of sub-classes will have only those attributes that are not common to super class except for the primary key.

4) The modified EER diagram is:



FullTimeFaculty (id, Name, Increment-date)
VisitingFaculty (id, Name, Host-institute)
Teachers (id, code)
Programme (code, details)

UNIT 4 FILE ORGANISATION IN DBMS

Structure **Page Nos.**

4.0 INTRODUCTION

In earlier units, you studied the basic concepts of database management systems, entity relationship diagram and relational algebra. Databases are used to store information. Normally, the principal operations you need to perform on database are those relating to:

- Creation of data
- Retrieving the data using conditions
- Modifying
- Deleting some information, which we are sure is no longer useful or valid.

Database structures data as two-dimensional tables, which allows easy processing of these operations. However, as the size of databases are large, the databases are required to be stored on secondary memory of computers (such as hard disk or SSD). Therefore, the secondary storage systems of databases are mainly concerned with the following issues:

- Storing table or tables as files: A single table can be stored as a file or several tables can be put together as a cluster of related records called cluster file.
- Attributes of a table: In general, a table represents the data of one object, therefore, the attributes represent data of a specific object and may be required to be accessed by a specific operation. Thus, it may be a good idea to store all the attributes of an object together. Does the order of attributes in a file matter?
- It seems logical to store all the records of a table contiguously. But, how should such records be ordered? The ordering of records in primary storage does not matter, however, for the secondary storage a specific sequence may be desired. Such decisions may be taken by the database demonstrator and may relate to the performance of the database.
- In the cases of analytical queries, particular attributes are stored together, this approach is called column-oriented approach, this approach is beyond the scope of this Unit. You may refer to further readings for this approach.

This unit focuses on the file Organisation in DBMS, the access methods available and the system parameters associated with them. File Organisation is the way the files are arranged on the disk and access method is how the data can be retrieved based on the file organisation.

4.1 OBJECTIVES

After going through this unit, you should be able to:

- define storage of databases on hard disks and SSD;
- discuss the implementation of various file Organisation techniques;
- discuss the advantages and the limitation of the various file Organisation techniques;
- describe various indexes used in database systems, and
- define the multi-key file organisation.

4.2 PHYSICAL DATABASE DESIGN

The database design involves the process of logical design with the help of E-R diagram, normalisation, etc., followed by the physical design.

The Key issues in the Physical Database Design are:

- The purpose of physical database design is to translate the logical description of data into the technical specifications for storing and retrieving data for the DBMS.
- The goal is to create a design for storing data that will provide adequate performance and ensure database integrity, security and recoverability.

Some of the basic inputs required for Physical Database Design are:

- Normalised relations
- Attribute definitions
- Data usage: entered, retrieved, deleted, updated
- Requirements for security, backup, recovery, retention, integrity
- DBMS characteristics.
- Performance criteria such as response time requirement with respect to volume estimates.

However, for such data some of the Physical Database Design Decisions that are to be taken are:

- Optimising attribute data types.
- Modifying the logical design.
- Specifying the file Organisation.
- Choosing indexes.

Designing the attributes in the database

The following are the considerations one has to keep in mind while designing the attributes in the database.

- Choosing data type
- Coding, compression, encryption
- Controlling data integrity
- Default value
  - Range control
  - Null value control
  - Referential integrity
- Handling missing data

- Substitute an estimate of the missing value
- Trigger a report listing missing values
- In programs, ignore missing data unless the value is significant.

Physical Records

These are the records that are stored in the secondary storage devices. For a database relation, physical records are the group of fields stored in adjacent memory locations and retrieved together as a unit. Considering the page memory system, a data page is the amount of data read or written in one I/O operation to and from a secondary storage device to the memory and vice-versa. In this context we define a term blocking factor that is defined as the number of physical records per page.

The issues relating to the Design of the Physical Database Files

Physical File is a file as stored on the disk or SSD. The main issues relating to physical files are:

- Constructs to link two pieces of data:
  - Sequential storage.
  - Pointers.
- File Organisation: How are the files arranged on the disk?
- Access Method: How can the data be retrieved based on the file Organisation?

Let us see in the next section how the data is stored on the hard disks (HDD) or SSDs

4.3 DATABASE STORAGE ON HDD OR SSD

At this point, it is worthwhile to note the difference between the terms file Organisation and the access method. A file organisation refers to the organisation of the data records, which are part of a file, into a group of records that are placed in a block of secondary storage. The file organisation also entails the access structures and interlinking of records. An access method is the way - how the data can be retrieved based on the file Organisation.

Mostly the databases are stored persistently on HDD or SSD for the reasons given below:

- The databases being very large may not fit completely in the main memory.
- Data of the database is to be stored permanently using non-volatile storage.
- Primary storage is expensive, using secondary storage reduces the cost of the storage per unit.

Each hard drive is usually composed of a set of disk platters. Each disk platter has a layer of magnetic material deposited on its surface. The entire disk can contain a large amount of data, which is organised into smaller packages called BLOCKS (or pages). On most computers, one block is equivalent to 1 KB of data (= 1024 Bytes). A block is the smallest unit of data transfer between the hard disk and the processor of the computer. Each block therefore has a fixed, assigned, address. Typically, the computer processor will submit a read/write request, which includes the address of the block, and the address of RAM in the computer memory area called a buffer (or cache) where the data must be stored/taken from. The processor then reads and modifies the buffer data as required and, if required, writes the block back to the disk. Let us see how the tables of the database are stored on the hard disk.

A Solid-State Disk (SSD) is made up of flash memories. These SSD devices also provide similar block-oriented access to data, however, since they do not have physically moving components, they provide lower latency and lower access time than that of disks. Therefore, in the subsequent sections, we will provide details on the disk oriented approach of data storage.

Fixed and Variable Length Records

There are two basic ways of storing a record on the disks – Fixed Length records and Variable Length Records. In the fixed length records all the attributes are assigned equal space in terms of bytes, just like fixed length structure in C programming. Thus, each record will be of the same size. In such cases, only metadata can be used to identify different records and their attributes.

As far as variable length records are concerned, it may be noted that each record of a table may be of different length. This is because in some records, some attribute values may be 'null'; or some attributes may be of type *varchar*, which allows variable number of characters to be stored in an attribute, therefore each record may have a different length string as the value of this attribute. To store variable length records, it is necessary to use a character that marks the end of an attribute value. In addition, an end of record marker will also be needed, as one disk block may store several records Therefore, each record is separated from the next, again by another special character, the record separator.

The next section discusses different types of file organisation that can be used to store the files on the disks.

4.4 FILE ORGANISATIONS

File organisation is used to organise the content of a file on a secondary storage device. A good file organisation should support efficient data access and update operations on the content of a file, which is stored on the secondary storage.

Data files are organised so as to facilitate access to records and to ensure their efficient storage. A tradeoff between these two requirements generally exists: if rapid access is required, more storage space is required to make it possible. Selection of File Organisations is dependent on two factors, as shown below:

- Typical DBMS applications may need to access a small subset of the data of a database at any given time.
- Whenever a portion of the data is needed by the DBMS; it is located on disk, copied to memory for processing, and rewritten to disk if the data was modified.

A file of record is likely to be accessed and modified in a variety of ways, and different ways of arranging the records enable different operations over the file to be carried out efficiently. A DBMS supports several file Organisation techniques. The important task of the DBA is to choose a good Organisation for each file based on its type of use.

For a database system, you select a file organisation a suitable file organisation based on the parameters like number of keys used to access the files, the ratio of update and access operations, the type of storage technology etc. *Figure 4.1* uses access keys as the basis for classifying different file organisations. This classification will be used in this unit to describe various file organisations.

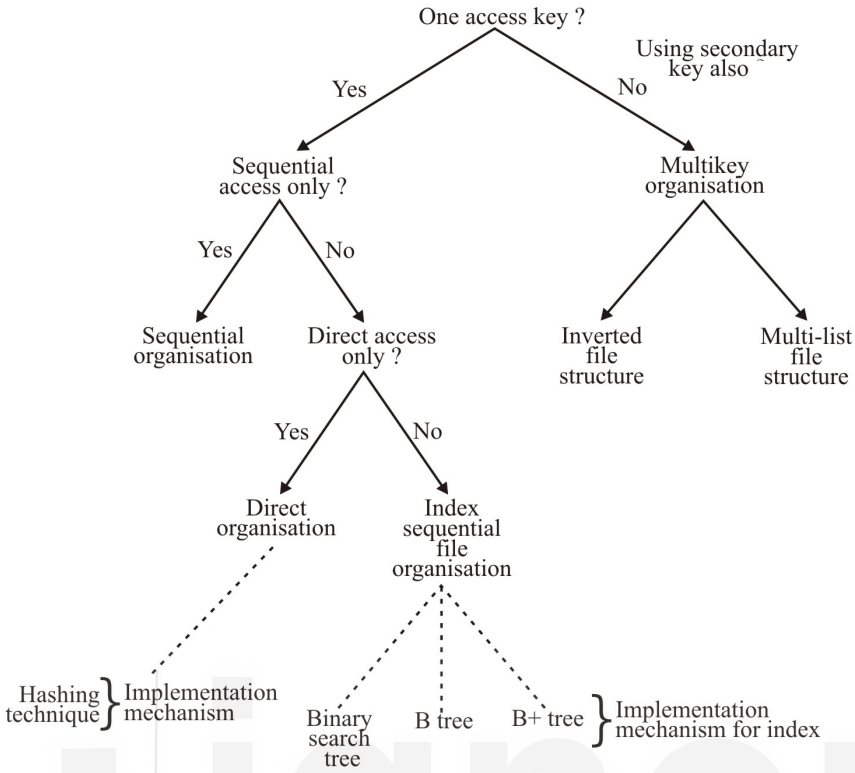


Figure 4.1: Different File Organisation Based on Access Keys

Let us discuss some of these techniques in more detail:

4.4.1 Unordered Heap File Organisation

Basically, these files are unordered files. It is the simplest and most basic type. These files consist of randomly ordered records. The records will have no particular ordering. The operations that you can perform on the records of a heap file are insert, retrieve and delete. The features of the heap file Organisation are:

- New records can be inserted in any empty space that can accommodate them.
- When old records are deleted, the occupied space becomes empty and available for any new insertion.
- If updated records grow, they may need to be relocated (moved) to a new empty space. Thus, this file organisation keeps a list of empty spaces.

Advantages of heap files

1. This is a simple file Organisation.
2. Insertion is somehow efficient.
3. Good for bulk-loading data into a table.
4. Best if file scans are common or insertions are frequent.

Disadvantages of heap files

1. Retrieval requires a linear search and is inefficient.
2. Deletion can result in unused space/need for reorganisation.

4.4.2 Sequential File Organisation

In the sequential Organisation, records of the file are stored in sequence (or consecutively) by the primary key field values. In this organisation, the records can be accessed in the order of its primary key. This kind of file Organisation works well for tasks in which nearly every record of the file is required to be accessed, such as the payroll system. Figure 4.2 shows this file organisation.

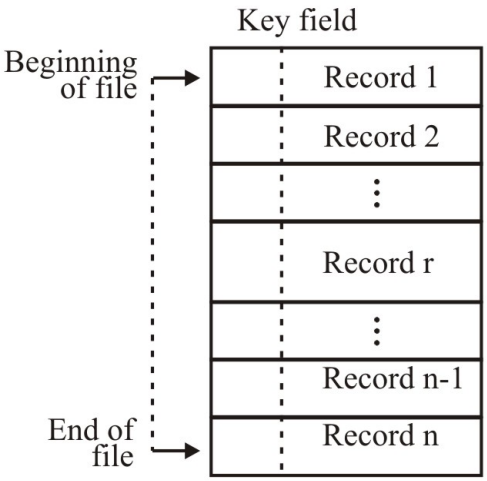


Figure 4.2: Structure of sequential file

A sequential file maintains the records in the logical sequence of its primary key values. Sequential files are inefficient for random access, however, are suitable for sequential access. A sequential file can be stored on devices like magnetic tape that allow sequential access.

On an average, to search a record in a sequential file would require looking into half of the records of the file. However, if a sequential file is stored on a disk (remember disks support direct access of its blocks) with keys stored separately from the rest of the record, then only those disk blocks are needed to be read that contains the desired record or records. This type of storage allows binary search on sequential file blocks, thus, enhancing the speed of access.

Updating a sequential file usually creates a new file so that the record sequence on the primary key is maintained. The update operation first copies the records till the record after which update is required into the new file and then the updated record is put followed by the remainder of records. Thus, the method of updating a sequential file automatically creates a backup copy. However, such update operations are very time consuming.

Addition of records in the sequential files are also handled in a similar manner to update operation. If a record is to be inserted at the last record of the file, it can be performed very easily. However, if a record is required to be inserted in between two records, then such insertion would require shifting down all the subsequent records in the file by one record space. In case of deletion of a record, all the subsequent records need to be shifted up by one record space.

Sequential file organisation is most suitable, if all the records of a file are to be processed in a sequence. For example, processing the monthly payroll of all the employees of an organisation, will require processing of all the employees records sequentially. However, a single update is expensive as a new file must be created, therefore, to reduce the cost per update, all update requests are stored in a single update file, which is sorted in the order of the sequential file ordering key. The file containing the updates is sometimes referred to as a transaction file and is used to update the sequential file in a single processing cycle.

This process is called the batch mode of updating. In this mode, each record of the master sequential file is checked for one or more possible updates by comparing with the update information of the transaction file. The records are written to a new master file in a sequential manner. A record that requires multiple updates is written only when all the updates have been performed on the record. A record that is to be deleted

is not written to a new master file. Thus, a new updated master file will be created from the transaction file and the old master file.

Thus, update, insertion and deletion of records in a sequential file require a new file creation. Can we reduce creation of this new file? Yes, it can be done easily, if the original sequential file is created with holes, which are empty record spaces, as shown in the *Figure 4.3*. Thus, reorganisation of file on addition and update operation can be restricted to only one block, which is read into/ written from the main memory as a single unit. Thus, holes increase the performance of sequential file insertion and deletion. This organisation also supports a concept of overflow area, which can store the spilled over records if a block is full. This technique is also used in index sequential file organisation. A detailed discussion on it can be found in the further readings.

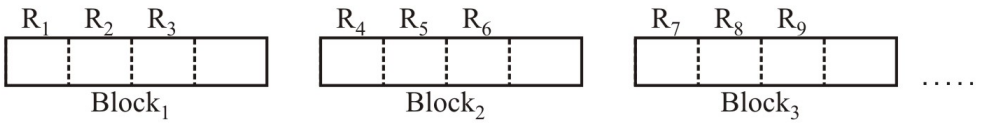


Figure 4.3: A sequential file with empty spaces for record insertions

Advantages of Sequential File Organisation

- It is fast and efficient when dealing with large volumes of data that need to be processed periodically (batch system).

Disadvantages of sequential File Organisation

- Requires all the new transactions to be sorted into a proper sequence for sequential access processing.
- Locating, storing, modifying, deleting, or adding records in the file requires rearranging the file.
- This method is too slow to handle applications requiring immediate updating or responses.

4.4.3 Indexed (Indexed Sequential) File Organisation

It organises the file like a large dictionary, i.e., records are stored in order of the key, but an index is kept which also permits a type of direct access. The records are stored sequentially by primary key values and there is an index built over the primary key field.

To locate a record in a sequential file, which is not indexed sequential, on average, you may read about half the number of records. Therefore, retrieval of a specific record in a sequential file is inefficient, especially if the file has a very large number of disk storage blocks. The use of index in a sequential file improves the query response time by adding an index, which is defined as a set of <*index value, address*> pair. An index is a mechanism for faster search, as the size of the index is much smaller than the original file. Thus, an index may be contained entirely in the main memory of the computer.

An indexed sequential file is a sequential file, which is stored in the order of its primary key, that contains an index on its primary key. Indexed sequential file allows advantages of indexing and sequential organization of records. The sequential organisation facilitates sequential processing of records, whereas the index helps in enhancing the performance of locating specific records based on the index value. In order, to make such files more efficient for insertion and deleting of records, such files are supported with overflow blocks. This reduces the need of moving all the records in a file. Figure 4.6 is an example of an indexed sequential file. Please note that the index is represented in Figure 4.6 as <*Primary Index Value, Block Pointer*>.

4.4.4 Hashed File Organisation

Hashing is the most common form of purely random access to a file or database. It is also used as an optimisation technique to access columns that do not have an index. Hashing involves the use of a hash function. Input to a hash function is the value of the attribute or set of attributes of a record that are to be used for file organisation and the output is the block address or page address, where that record can be found. Figure 4.4 shows a file using hashing file organisation. The hash function used for this file is *key mod 4*. Notice that records with different key values can be placed in a Block based on the hashing function. To search the location of a record, you can apply the hashing function on the key value and then search the hashed block. For example, if you are searching for the location of key 29, then the record can be found in 29 mod 4 = Block 1, read this Block in the main memory and do a linear search on key value to locate the record in the main memory. The most popular form of hashing is division hashing with chained overflow. You can refer to further readings for more details on this file organisation.

Advantages of Hashed File Organisation

1. Insertion or search on hash-key is fast.
2. Best if equality search is needed on hash-key.

Disadvantages of Hashed File Organisation

1. It is a complex file Organisation method.
2. Search is slow.
3. It suffers from disk space overhead.
4. Unbalanced buckets degrade performance.
5. Range search is slow.

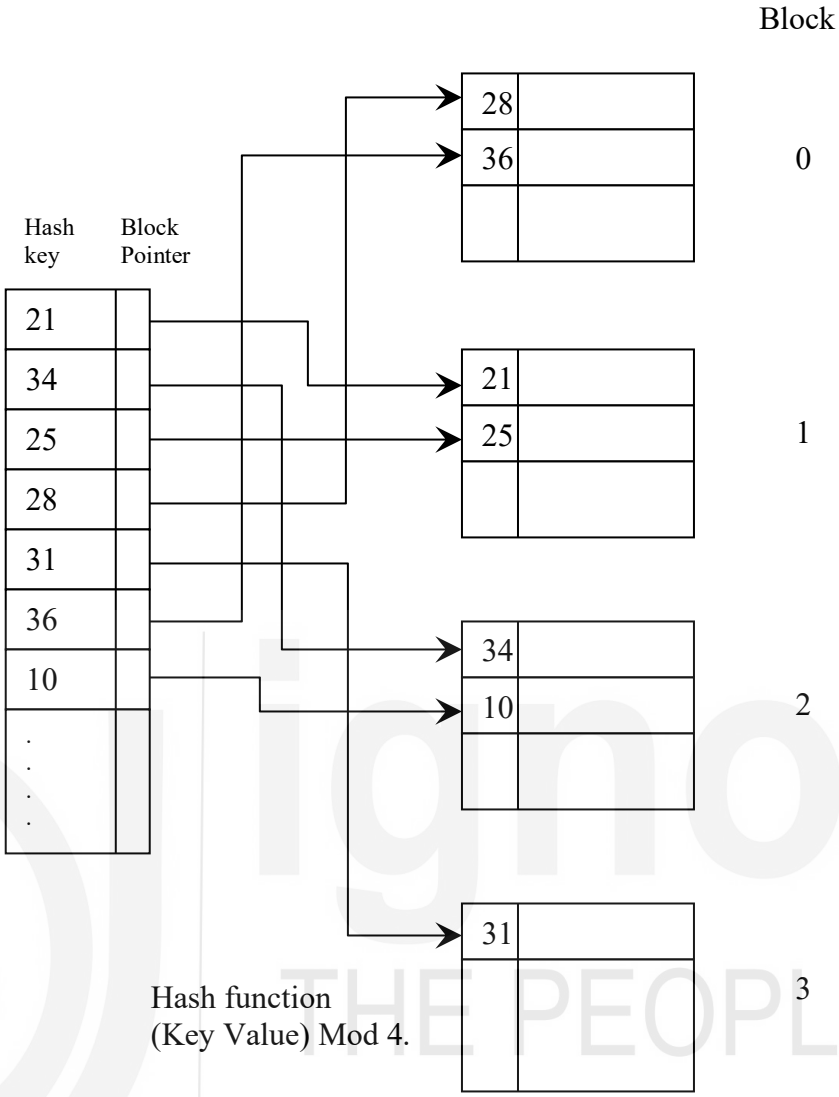


Figure 4.4: A Hashed file

Check Your Progress 1

1) List five operations on sequential file Organisation. Comment on the performance of each of these operations.

……

……

2) What are Direct-Access systems? What can be the various strategies to achieve this?

……

……

……

3) What is file organisation and what are the essential factors that are to be considered for a file organisation?

……

……

……

……

4.5 INDEXES

One of the terms used during the file organisation is the term index. In this section, let us define this term in more detail.

Every printed book that you read, in general, has an index of keywords at the end. Notice that this index is a sorted list of keywords (index values) and page numbers (address) where the keyword can be found. In databases also an index is defined in a similar way, as the <*index value, address*> pair.

The basic advantage of having sorted index pages at the end of the book is that you can locate the description about a desired keyword in the book. You could have used the topic and subtopic listed in the table of contents, but it is not necessary that the given keyword can be found there; also, they are not in any sorted sequence. If a keyword is not listed in index and table of contents, then you need to search each page of the book to find the required keyword, which is very cumbersome. Thus, an index at the back of the book helps in locating the required keyword references very easily in the book.

The same is true for databases that have a very large number of records. A database index allows fast search on the index value in database records. It will be difficult to locate an attribute value in a large database, if the index on that attribute is not provided. In such a case the value is to be searched record-by-record in the entire database, which is cumbersome and time consuming. It is important to note that for a large database all the records cannot be kept in the main memory at a time, thus, data needs to be transferred from the secondary storage device, which is more time consuming.

An index entry consists of a pair consisting of index value and a list of pointers to disk blocks for the records that have that index value. An index contains such information for every stored value of the index attribute. An index file is very small compared to a data file that stores a relation. Also index entries are ordered, so that an index can be searched using an efficient search method like binary search. In case an index file is very large, you can create a multi-level index, that is index on index. Multi-level indexes are defined later in this section.

There are many types of indexes that are categorised as:

Primary index	Single level index	Spare index
Secondary index	Multi-level index	Dense index
Clustering index		

A primary index is defined on the attributes *in the order of which the file is stored*. This field is called the ordering field. A primary index can be on the primary or candidate key of a file. If an index is on the ordering attributes, which are not candidate key attributes, then several records may be related to one ordering field value. This is called clustering index. It is to be noted that there can be only one physical ordering attribute or set of attributes for a file. Thus, a file can have either the primary index or clustering index, not both. Secondary indexes are defined on the non-ordering fields. Thus, there can be several secondary indexes in a file, but only one primary or clustering index.

Primary index

Primary index is a file that contains a sorted sequence of index records having two columns: the ordering key field; and a block address for that key field in the data file. The ordering key field for this index can be the primary key of the data file. Primary index contains one index entry of the ordering key field for each Block of data. An entry in the primary index file contains either the key value of the first record or the key value of the last record, which are stored in that data block; and a pointer to that data block.

Let us discuss the primary index with the help of an example. Let us assume a student database as (Assuming that one block stores only four student records). Figure 4.5 shows a sample of this data file. The sample file is ordered on the attribute - enrolment number.

Block Number	Enrolment Number	Name	City	Programme
BLOCK 1	2109348	…	CHENNAI	CIC
	2109349	…	CALCUTTA	MCA
	2109351	…	KOCHI	BCA
	2109352	…	KOCHI	CIC
BLOCK 2	2109353	…	VARANASI	MBA
	2238389	…	NEW DELHI	MBA
	2238390	…	VARANASI	MCA
	2238411	…	NEW DELHI	BCA
BLOCK 3	2238412	…	AJMER	MCA
	2238414	…	NEW DELHI	MCA
	2238422	…	MUMBAI	BSC
	2258014	…	MUMBAI	BCA
BLOCK 4	2258015	…	MUMBAI	BCA
	2258017	…	NEW DELHI	BSC
	2258018	…	MUMBAI	MCA
	2258019	…	LUCKNOW	MBA
…	…	…	…	…
BLOCK r	2258616	…	AJMER	BCA
	2258617	…	LUCKNOW	MCA
	2258618	…	NEW DELHI	BSC
	2318935	…	FARIDABAD	MBA
…	…	…	…	…
BLOCK N-1	2401407	…	BAREILLY	CIC
	2401408	…	BAREILLY	BSC
	2401409	…	AURANGABAD	BCA
	2401623	…	NEW DELHI	MCA
BLOCK N	2401666	…	MUMBAI	MCA
	2409216	…	LUCKNOW	MBA
	2409217	…	ALMORA	BCA
	2409422	…	MUMBAI	BSC

Figure 4.5: A Student file stored in the order of student enrolment numbers

The primary index on this file would be on the ordering field – enrolment number. The primary index on this file is shown in Figure 4.6. Please note the following points in Figure 4.6.

- An index entry is defined as the attribute value, pointer to the block where that record is stored. The pointer physically is represented as the binary address of the block.
- Since there are four student records, which of the key values should be stored as the index value? We have used the first key value stored in the block as the

index key value. This is also called the anchor value. All the records stored in the given block have ordering attribute value as the same or more than this anchor value.

- The primary index may be smaller in size, as it contains one index entry for each storage data block. Also notice that not all the records need to have an entry in the index file. This type of index is called a non-dense index. Thus, the primary index is non-dense index.
- To locate the record of a student whose enrolment number is 2238422, you need to find two consecutive entries of indexes such that index value 1 < 2238422 < index value 2. In the Figure 4.6, you can find the third and fourth index values as: 2238412 and 2258015 respectively satisfying the properties as above. Thus, the required student record must be found in Block 3.

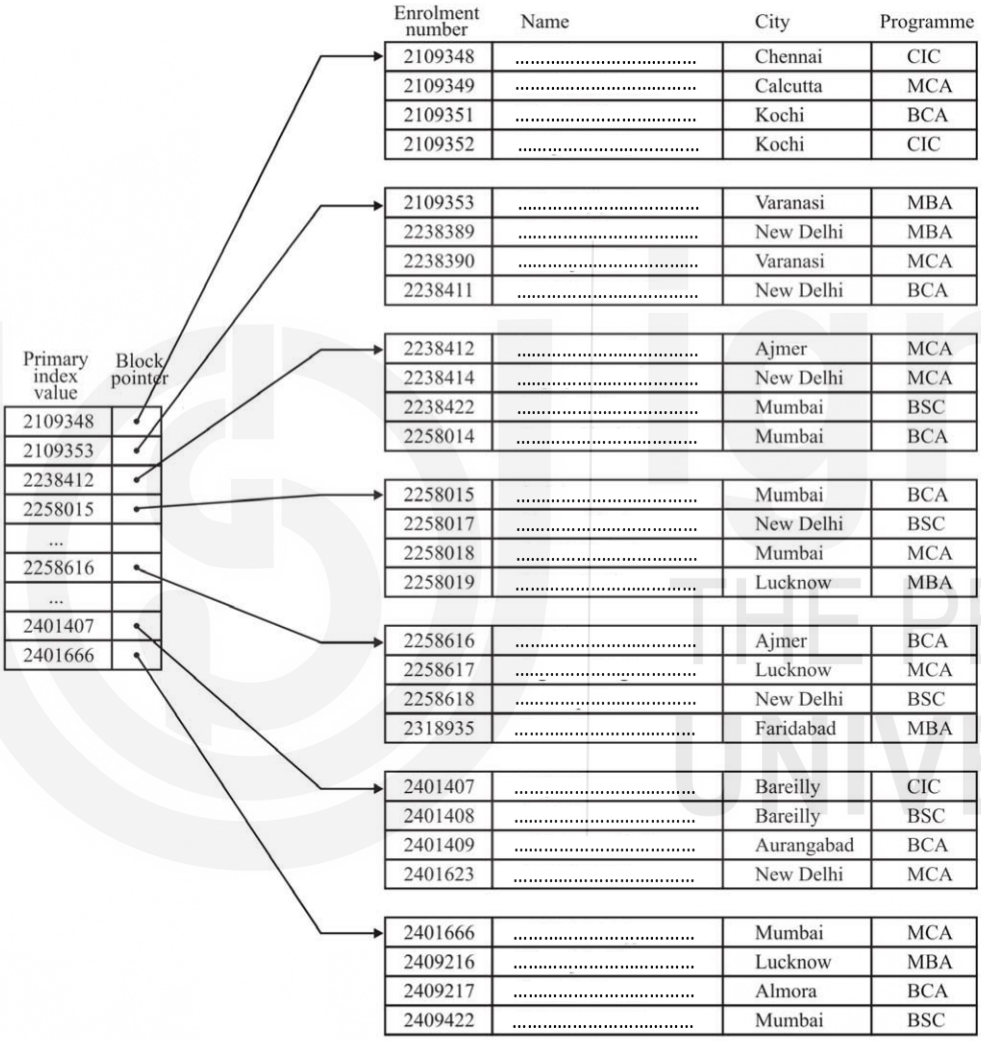


Figure 4.6: A Student file and the Primary Index on Enrolment Number

But does primary index enhance efficiency of searching? Let us explain this with the help of an example (Please note we will define savings in terms of the number of block transfers, as that is the most time-consuming operation during searching).

Example 1: An ordered student file (ordering field is enrolment number) has 20,000 records stored on a disk having the Block size as 1 K. Assume that each student record is of 100 bytes, the ordering field is of 8 bytes, and block pointer is also of 8 bytes, find how many block accesses on average may be saved on using primary index.

Answer:

Number of accesses without using Primary Index:

Number of records in the file = 20000

Block size = 1024 bytes

Record size = 100 bytes

Number of records per block = integer value of [1024 / 100] = 10

Number of disk blocks acquired by the file

= [Number of records / records per block]

= [20000/10] = 2000

Assuming a block level binary search, it would require $\log_2 2000$

= about 11 block accesses.

Number of accesses with Primary Index:

Size of an index entry = 8+8 = 16 bytes

Number of index entries that can be stored per block

= integer value of [1024 / 16] = 64

Number of index entries = number of disk blocks = 2000

Number of index blocks = ceiling of [2000/ 64] = 32

Number of index block transfers to find the value in index blocks = $\log_2 32 = 5$

One block transfer will be required to get the data records using the index pointer after the required index value has been located.

So total number of block transfers with primary index = 5 + 1 = 6.

Thus, the Primary index would save 11 – 6 = 5 block transfers for the given size of data and index.

Is there any disadvantage of using a primary index? Yes, a primary index requires the data file to be ordered, this causes problems during insertion and deletion of records in the file. This problem can be taken care of by selecting a suitable file organisation that allows logical ordering only.

Clustering Indexes.

It may be a good idea to keep records of the students in the order of the programme they have registered, as most of the data file accesses may require programme wise student data. A file can be ordered and physically stored on non-key attributes; an index that is created on such non-key attributes would have multiple records pointed to by a single index entry. Such an index is called a clustering index. *Figure 4.7 and Figure 4.8* show the clustering indexes in the same file organised in different ways.

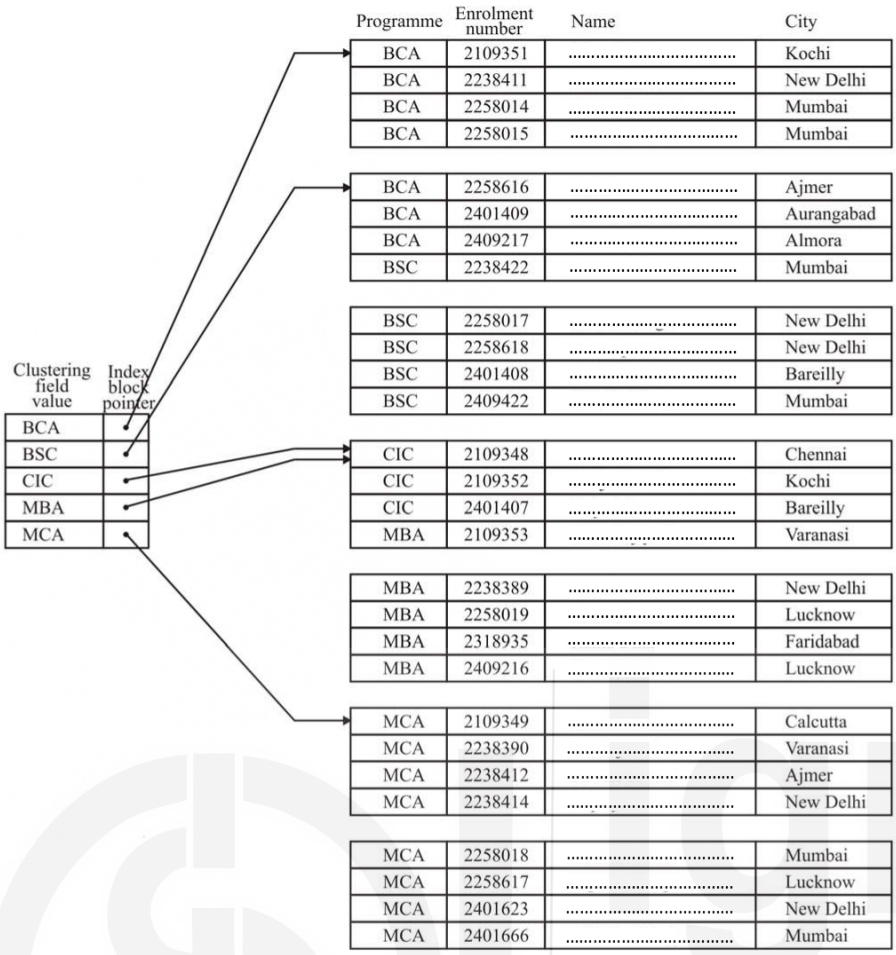


Figure 4.7: A clustering Index on Programme in the Student file

Please note the following points about the clustering index as shown in the *Figure 4.7*

- The clustering index is an ordered file having the clustering index value and a block pointer to the first block where that clustering field value first appears.
- Clustering index is also a sparse index. The size of the clustering index is smaller than the primary index as far as the number of entries is concerned.

Please note that in Figure 4.7, the data file can have a single block in which data of students of multiple programmes are stored. You can improve upon this organisation by allowing only one Programme data in one block. Such an organisation and its clustering index is shown in the *Figure 4.8*:

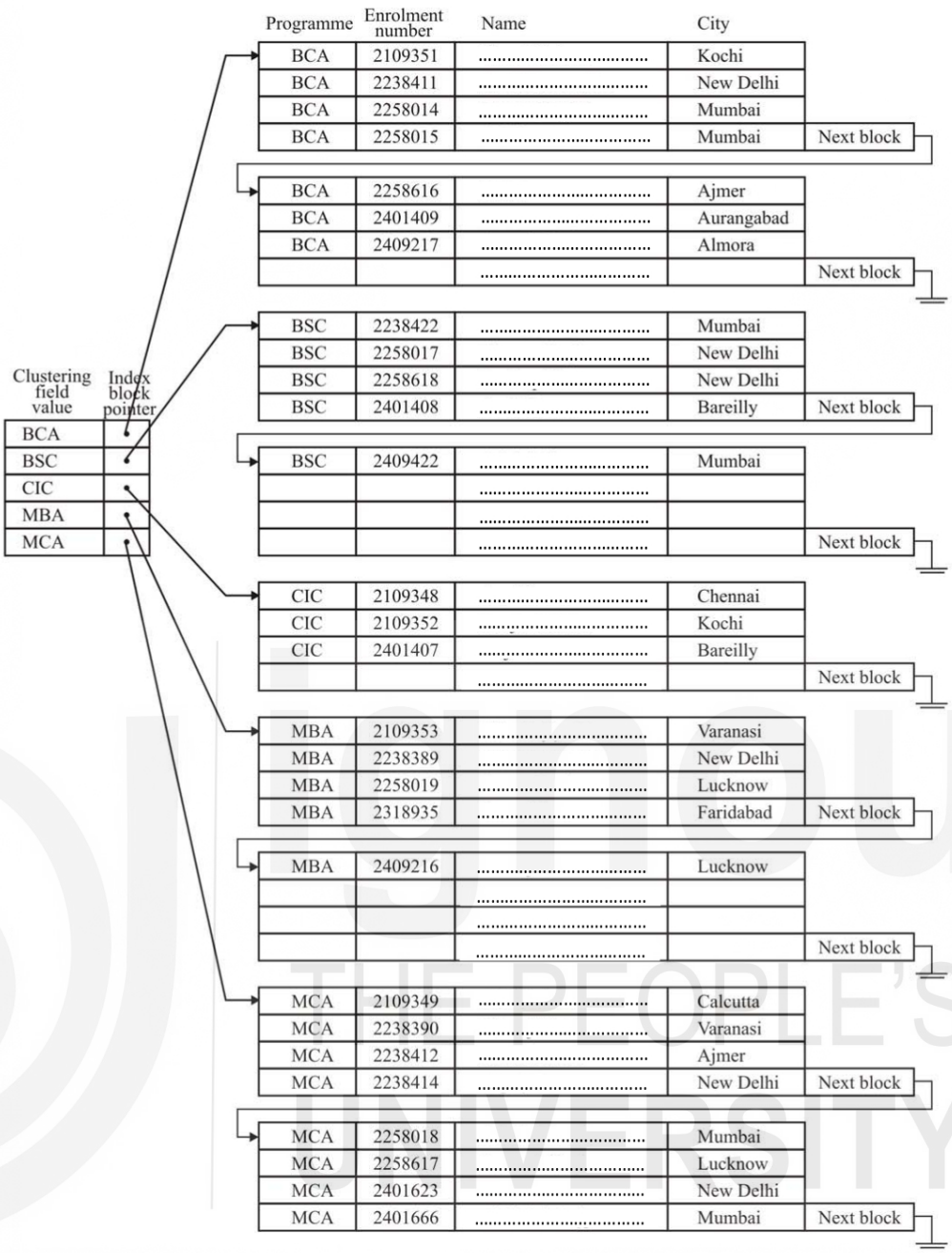


Figure 4.8: Clustering index with separate blocks for each clustering attribute value

Please note the following points for the above:

- Data insertion and deletion is easier than in the earlier clustering files, yet even now it is cumbersome.
- The blocks allocated for one index entry are in the form of linked list blocks.
- Clustering index is another example of a non-dense index as it has one entry for every distinct value of the clustering index attribute and not for every record in the file.

Secondary Indexes

Consider the student database and its primary and clustering index (only one will be applicable at a time). Now consider the situation when the database is to be searched or accessed in the alphabetical order of student names. Any search on a student name would require a sequential data file search, as there is no ordering on student names. Thus, searching for a student's name is going to be very time consuming. Such a search on an average would require reading half of the total number of blocks. Thus, you need secondary indices in database systems. A secondary index is a file that contains records containing a secondary index field value which is not the ordering field of the data file, and a pointer to the block that contains the data record. Please

note that although a data file can have only one primary index (as there can be only one ordering of a database file), it can have many secondary indices.

Secondary indexes can be defined on an alternate key or non-key attributes. A secondary index that is defined on the alternate key will be dense, while a secondary index on non-key attributes would require a bucket of pointers for one index entry. Let us explain them in more detail with the help of *Figure 4.9*.

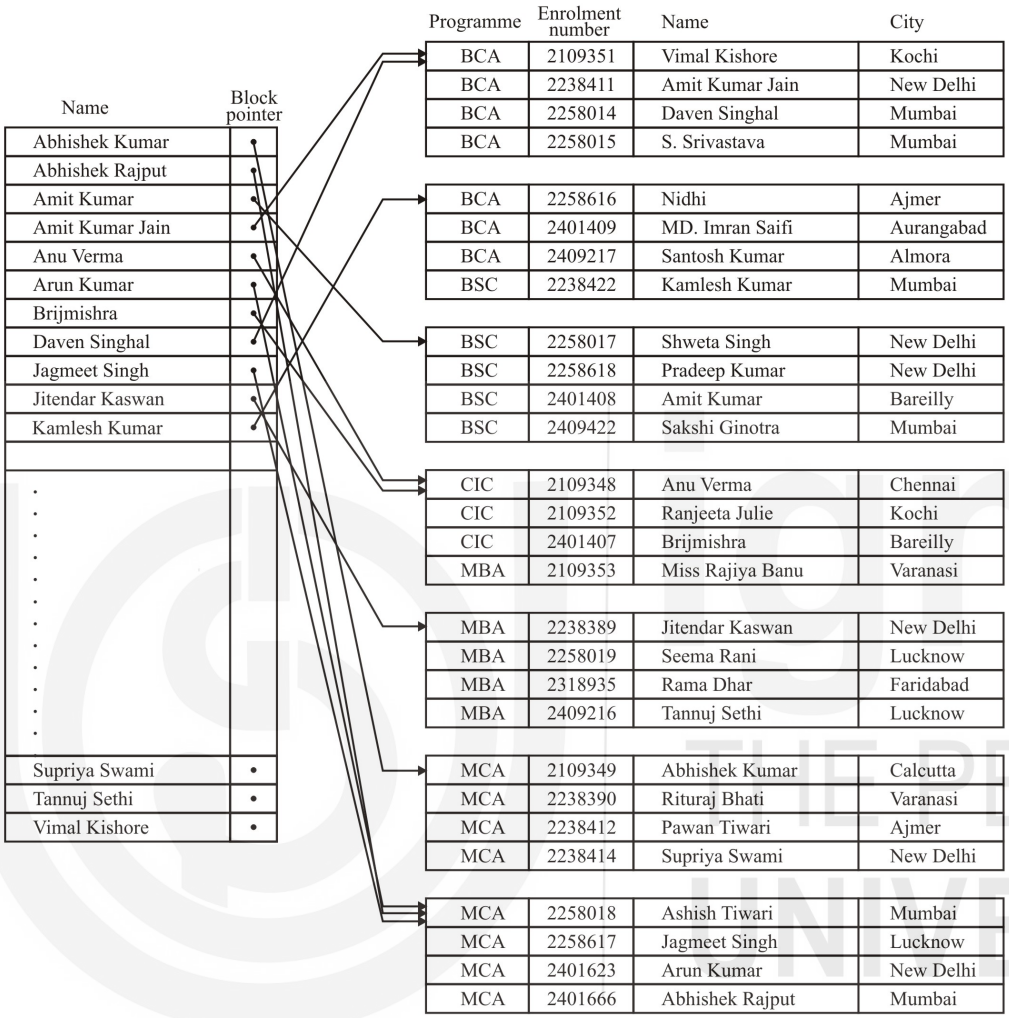


Figure 4.9: A Dense Secondary Index on a non-ordering key field of a file (The file is organised on the clustering field "Programme")

Please note the following in *Figure 4.9*.

- The names in the data file are unique and thus are being assumed as the alternate key. Each name therefore is appearing as the secondary index entry.
- The pointers are block pointers, thus are pointing to the beginning of the block and not a record. For simplicity, we have not shown all the pointers in Figure 4.9.
- This type of secondary index file is dense index as it contains one entry for each record/distinct value.
- The secondary index is larger than the Primary index as we cannot use block anchor values here as the secondary index attributes are not the ordering attribute of the data file.
- To search a value in a data file using name, first the index file is (binary) searched to determine the block, where the record having the desired key value

can be found. Then this block is transferred to the main memory where the desired record is searched and accessed.

- A secondary index file, usually, has a larger number of index entries than that of primary index. However, the secondary index improves the search time to a greater proportion than that of a primary index. This is due to the reason - If a primary index does not exist even then, you can perform binary search on the blocks of data records, as the records are ordered in the sequence of primary index value. However, if a secondary key does not exist, then you may need to search the records sequentially. This fact is demonstrated with the help of Example 2.

Example 2: Let us reconsider the problem of Example 1 with a few changes. An unordered student file, which is not ordered on a primary key, has 20,000 records stored on a disk having a Block size as 1 K. Assume that each student record is of 100 bytes, the secondary index field is of 8 bytes, and the block pointer is also of 8 bytes, find how many block accesses on average may be saved on using a secondary index on enrolment number.

Answer:

Number of accesses without using a Secondary Index:

Number of records in the file = 20000
Block size = 1024 bytes
Record size = 100 bytes
Number of records per block = integer value of [1024 / 100] = 10
Number of disk blocks acquired by the file
= [Number of records / records per block]
= [20000/10] = 2000
Since the file is un-ordered any search on an average will require about half of the above blocks to be accessed. Thus, average number of block accesses = 1000

Number of accesses with Secondary Index:

Size of an index entry = 8+8 = 16 bytes
Number of index entries that can be stored per block
= integer value of [1024 / 16] = 64
Number of index entries = number of records = 20000
Number of index blocks = ceiling of [20000/ 64] = 320
Number of index block transfers to find the value in index blocks
= ceiling of $[\log_2 320] = 9$
One block transfer will be required to get the data records using the index pointer after the required index value has been located. So total number of block transfers with secondary index = 9 + 1 = 10

Thus, the Secondary index would save about 1990 block transfers for the given case. This is a huge saving compared to a primary index. Please also compare the size of the secondary index to the primary index.

Let us now see an example of a secondary index that is on an attribute that is not an alternate key.

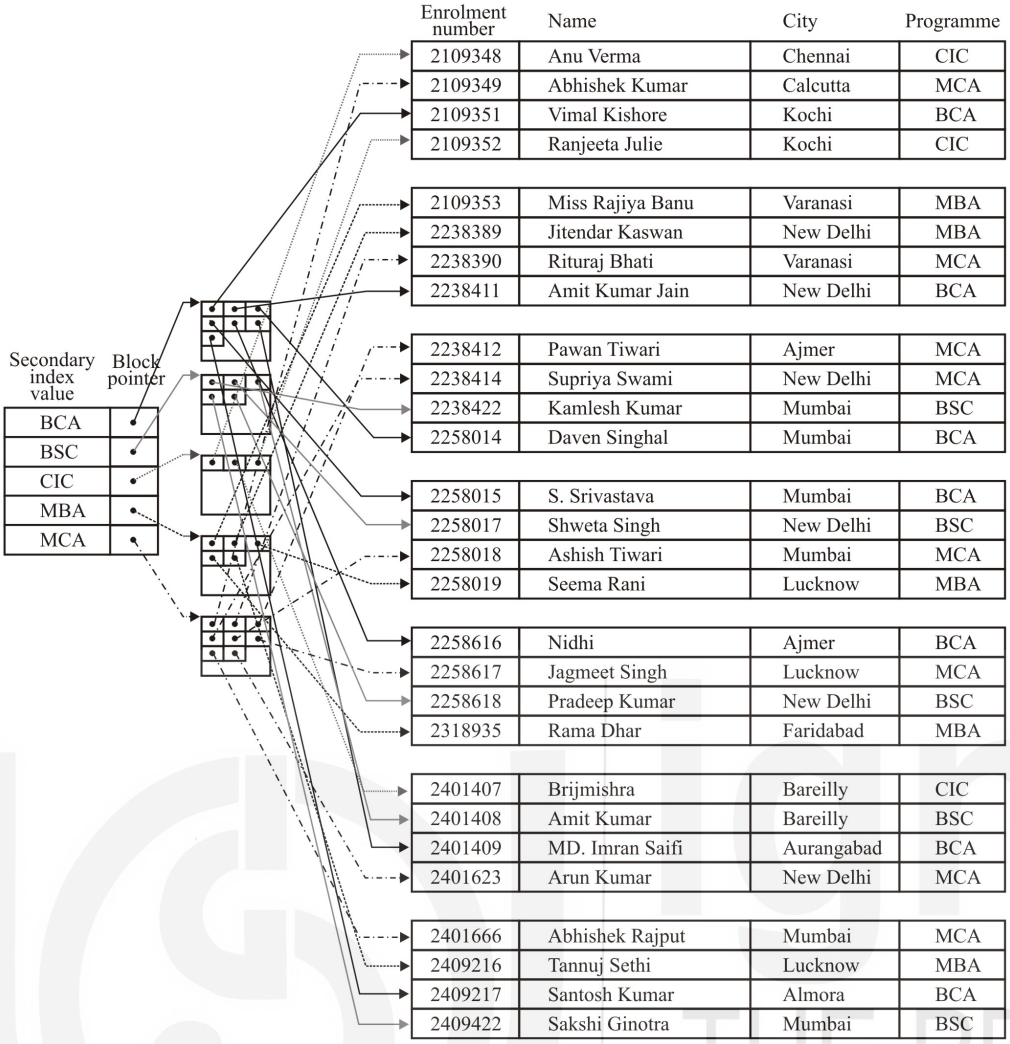


Figure 4.10: An example of Secondary Index with single level of indirection
(Index entries are of fixed length and have unique field values)
(The file is organised on the primary key)

A secondary index that needs to be created on a field that is not a candidate key can be implemented using several ways. We have shown here the way in which a block of pointer records is kept for implementing such an index. This method allows the index entries to be of fixed length. It also allows only a single entry for the value of the indexing attribute. In addition, the level of indirection allows multiple index pointes to be stored in a single block of data. The algorithms for searching the index, inserting and deleting new values into an index are very simple in such a scheme. Thus, this is the most popular scheme for implementing such secondary indexes.

Sparse and Dense Indexes

As discussed earlier, an index is defined as the ordered <*index value, address*> pair. These indexes in principle are the same as that of indexes used at the back of the book. The key ideas of the indexes are:

- They are sorted on the order of the index value (ascending or descending) as per the choice of the creator.
- The indexes are logically separate files (just like separate index pages of the book).
- An index is primarily created for fast access to information.
- The primary index is the index on the ordering field of the data file, whereas a secondary index is the index on any other field, thus, is more useful.

But what are sparse and dense indexes?

A dense index contains one index entry for every value of the indexing attributes, whereas a sparse index also called non-dense index contains few index entries out of the available indexing attribute values. For example, the primary index on enrolment number is sparse, while secondary index on student name is dense.

Multilevel Indexing Scheme

For small files, the indexing scheme keeps the address of the block file in each index entry. Such indices would be small and can be processed efficiently in the main memory. However, for a large file the size of the index can also be very large. In such a case, you can create indexes at several levels, with the last level pointing to the data records. Figure 4.11 shows this scheme.

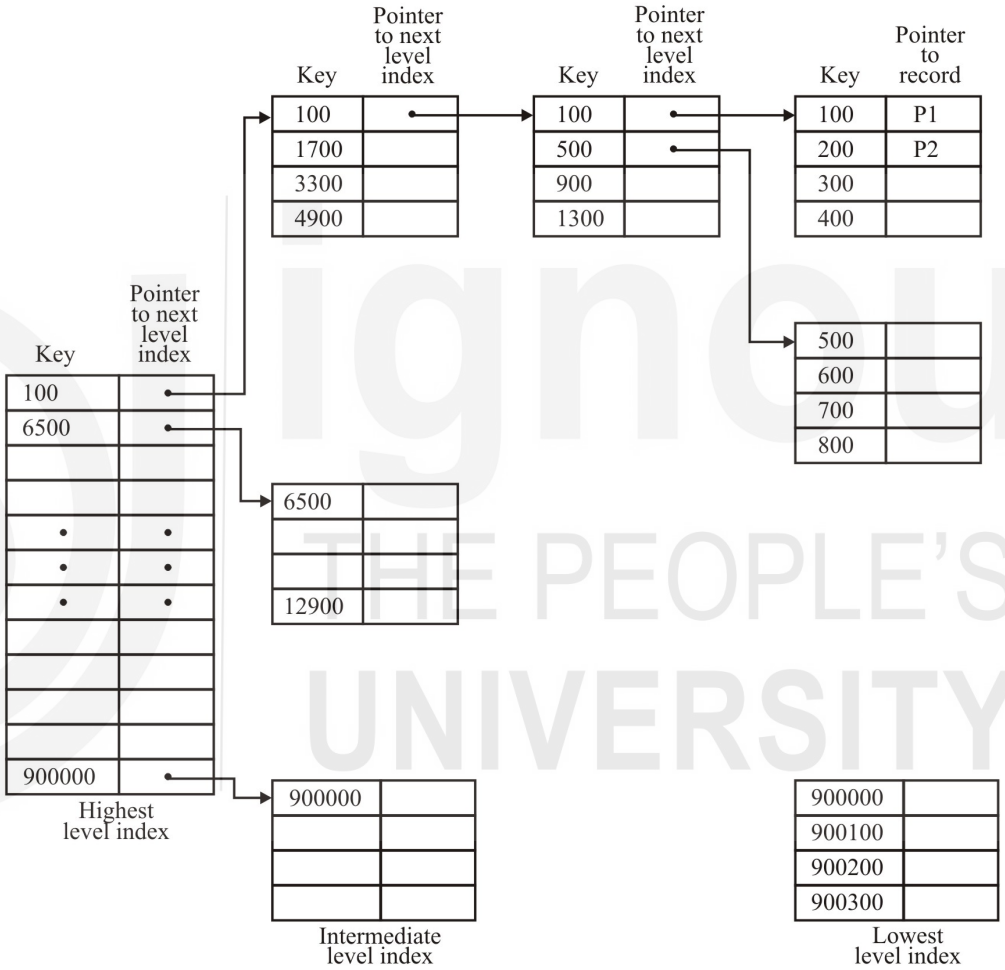


Figure 4.11: Hierarchy of Indexes

Please note the following relating to multi-level indexes:

- The lowest level index, as shown in Figure 4.11, has one entry for each record, though in a different index block. Thus, the lowest level index occupies the maximum space.
- Thus, maintenance of the multi-level indexes is expensive, as multiple indexes are to be maintained for every insertion and deletion of records.

After discussing so much about the indexes, let us now turn our attention to how an index can be implemented in a database. The indexes are implemented through B-Tree. The following section examines the index implementation in more detail.

Check Your Progress 2

1) A file contains 40,000 student records of 200 bytes, each having the 1 KB as the size of a block. What would be the size of the primary index, if the size of the

primary key is 4 byte and the block address is 8 bytes? What would be the average number of block accesses to access a record using this Index?

……

……

2) What would be the size of secondary index for the student file as stated in question 1 if it has a secondary index on its alternate key of size 8 bytes. What would be the average number of block accesses to access a record using this secondary Index?

……

……

……

3) Which of the following indexes are dense indexes? Give reasons.
(i) Primary Index (ii) Clustering Index (iii) Secondary Index on alternative key
(iv) Secondary index on non-key attributes with unique attribute values

……

……

……

……

4.6 IMPLEMENTING INDEX USING TREE STRUCTURE

Let us discuss the data structure that is used for creating indexes.

Can we use Binary Search Tree (BST) as Indexes?

Let us first reconsider the binary search tree. A BST is a data structure that has a property that all the keys that are to the left of a node are smaller than the key value of the node and all the keys to the right are larger than the key value of the node.

To search a typical key value, you start from the root and move towards left or right depending on the value of the key that is being searched. Since an index is a <*value, address*> pair, thus while using BST, you need to use the value as the key and address field must also be specified in order to locate the records in the file that is stored on the secondary storage devices. The following figure demonstrates the use of BST index for a University where a dense index exists on the enrolment number field.

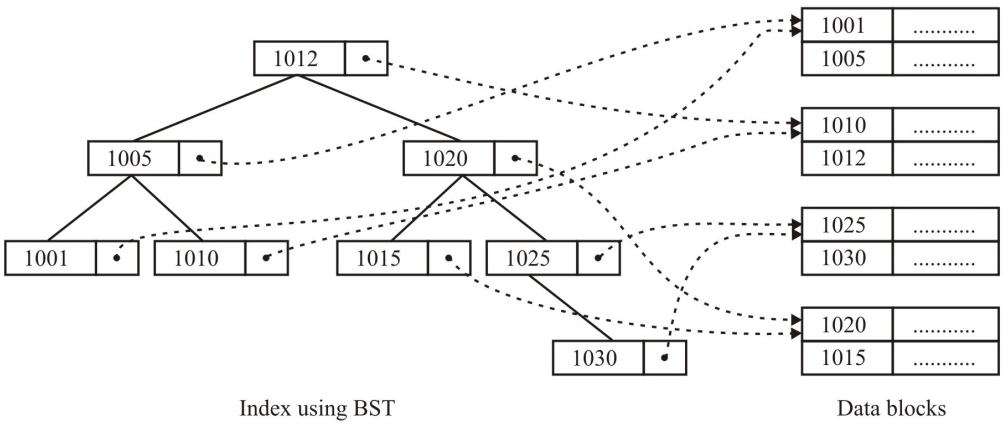


Figure 4.12: The Index structure using Binary Search Tree

Please note in Figure 4.12 that a key value is associated with a pointer to a record. A record consists of the key value and other information fields. Please note that a node on BST stores the <*key value, address*> pair.

Now, let us examine the suitability of BST as a data structure to implement indexes. A BST as a data structure is suitable for an index if the complete index is contained in the primary memory. However, indexes are quite large in nature and require a combination of primary and secondary storage. Therefore, you can use B-Tree data structure to implement the index.

A B-Tree as an index has two advantages:

- It is completely balanced.
- Each node of B-Tree can have a number of keys. An ideal node size of B-Tree would be equal to the block size of the secondary storage device that is being used for storing the index.

The question that needs to be answered here is: what should be the order of B-Tree for an index? The suggested order is from 80-200 depending on various index structures and block size.

B-tree is a data structure, which was proposed by R. Bayer and E. McCreight of Bell Scientific Research Labs in 1970. The B-Tree and its variants are secondary storage structures and have been found to be very useful for implementing indexes. An N order B-tree has:

- A node of B-tree of order N can have children/paths in the range - ceiling of [N/2] to N. However, the root node of the tree can have 2 to N children/paths.
- Each node can have one fewer key than the number of children/paths, but a maximum of N-1 keys can be stored in a node.
- The keys are normally arranged in a node in an increasing order.
- If a new key is inserted into a full node of order N (i.e. it already contains N-1 keys), then on addition of this new key value, the node would have N+1 paths (N keys). This node is split into two nodes and the median key value is moved to the parent of this node. In case the node that is being split is the root node, then it is split into two nodes and a new root node is created by using the median key of the node being split.
- B-tree does not allow any empty sub-tree, therefore, all the leaves of B-tree are at the same level. Therefore, a B-tree is a completely balanced tree.

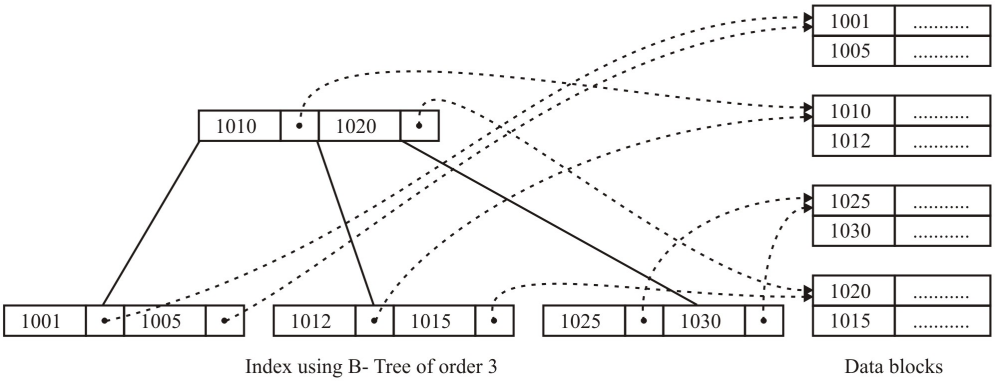


Figure 4.13: A B-Tree as an index

A B-Tree index is shown in Figure 4.13. The B-Tree has a very useful variant called B+Tree, which has all the key values at the leaf level also, in addition to the higher level. For example, the key value 1010 in *Figure 12* will also exist at leaf level. In

addition, these lowest level leaves are linked through pointers. Thus, the B+tree is a very useful structure for index-sequential organisation. You can refer to further readings for more details on these topics.

4.7 MULTI-KEY FILE ORGANISATIONS

Till now we have discussed file organisations having the single access key. But is it possible to have file organisations that allow access of records on more than one key field? This section discusses the two file organisations that allow multiple access paths, with each path having a different key. These are called multi-key file Organisations. These file organisations, in general, are part of a real database management system. Two of the commonest techniques for this Organisation are:

- Multi-list file Organisation
- Inverted file Organisation

Let us discuss these techniques in more detail. But first let us discuss the need for the Multiple access paths.

4.7.1 Multiple Access Paths

In practice, most of the online information systems require the support of multi-key files. For example, consider a banking database application having many kinds of users such as:

- Teller
- Loan officers
- Branch manager
- Account holders

All these users access the bank data however in a different way. Let us assume a sample data format for the Account relation in a bank as:

Account Relation:

Account Number	Account Holder Name	Branch Code	Account type	Balance	Permissible Loan Limit

A teller may access the record above to check the balance at the time of withdrawal. S/he needs to access the account based on branch code and account number. A loan approver may be interested in finding the potential customer by accessing the records in decreasing order of permissible loan limits. A branch manager may need to find the top ten most preferred customers in each category of account, so s/he may access the database in the order of account type and balance. The account holder may be interested in her/his own record. Thus, all these applications are trying to refer to the same data but using different key values. Thus, all the applications as above require the database file to be accessed in different format and order.

Multiple indexes can be used to access a data file through multiple access paths. In such a scheme only one copy of the data is kept, only the number of paths is added with the help of indexes. Let us discuss two important approaches, viz. multi-list file organisation and Inverted file organisation.

4.7.2 Multi-list file Organisation

This file organisation, as the name suggests, consists of multiple lists or indexes. The records in each list are linked from the index value. The linking of records, in general, is done in the sorted sequence of the key attribute to facilitate searching, insertion and deletion operations. The following example explains the multi-list file organisation.

A sample data of employees of an organisation is given in *Figure 4.14*. Assume that the Empid is the key attribute. You can create multiple index lists using this data.

Assumed Record Number	Employee id (Empid)	Employee Name	Job Title	Highest Qualification	Gender (Female F /Male M)	City of posting	Married - M/ Single - S	Salary per month
A	795	Praveen	Engineer	B. Tech.	M	Dehradun	S	16,200/-
B	495	Rohini	Manager	B. Tech.	F	Dehradun	M	19,000/-
C	905	Rishika	Manager	MCA	F	Jaipur	S	17,100/-
D	705	Gaurav	Engineer	B. Tech.	M	Jaipur	M	13,200/-
E	595	Dipti	Manager	MCA	F	Jaipur	S	14,100/-

Figure 4.14: Sample Employee Data

The primary link order (in the order of primary key Empid) would be:

B(495), E(595), D(705), A(795), C(905)

The primary index for this file would be:

>= 500 but < 700
> = 700 but < 900
>= 900 but < 1100

The index file for the example data as per Figure 4.14 is shown in *Figure 4.15*.

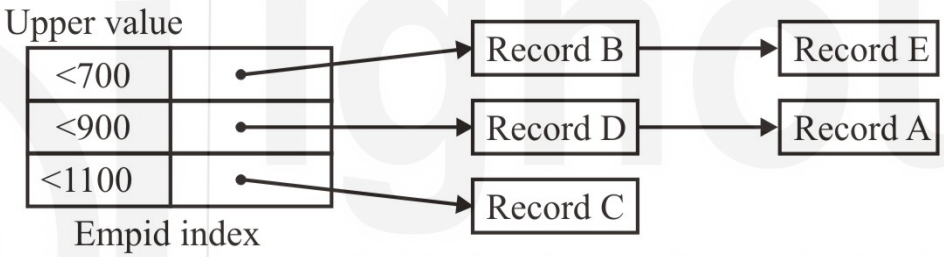


Figure 4.15: Linking together all the records in the same index value.

Please note that in the *Figure 4.15*, those records that fall in the same index value range of Empid are linked together. These lists are smaller than the total range, which will improve search performance.

This file can be supported by many more indexes that will enhance the search performance on various fields, thus, creating a multi-list file organisation. *Figure 4.16* shows various indexes and lists corresponding to those indexes. For simplicity we have just shown the links and not the complete record. Please note that nodes in the original file are assumed to be in the order of Empid's.

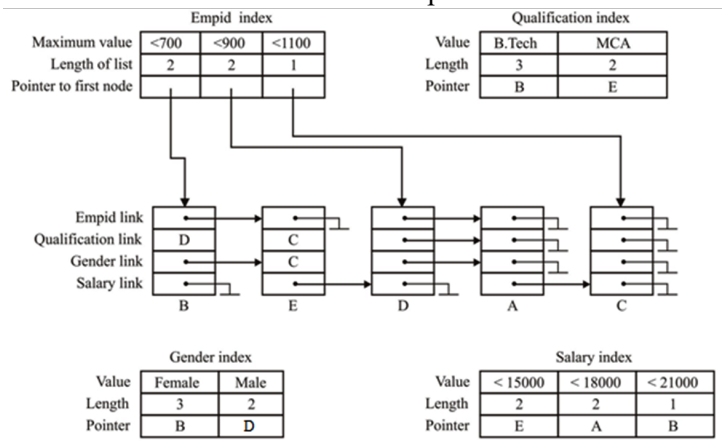


Figure 4.16: Multi-list representation for Figure 4.14

In Figure 4.16, the qualification index can be read as:

B. Tech. Starts at B → D → A.

MCA Starts at E → C.

Gender Index:

Female: Starts at B, → E, → C.

Male: Starts at B, → D, → A.

An interesting addition that can be done in the indexing scheme of multi-list organisation is that an index entry may contain the length of each sub-list and the index entry should have a pointer to the first record of that list. The length information is useful when the query contains a Boolean expression. For example, if you need to find the list of Female employees who have MCA qualifications, you can find the results in two ways. Either you go to the Gender index and search the Female index list for MCA qualification, or you search the qualification index to find MCA list and in MCA list search for Female candidates. Since the size of MCA list is 2 while the size of Female list is 3 so the preferable search will be through a smaller MCA index list. Thus, the information about the length of the list may help in reducing the search time in the complex queries. Performance of this structure is not good under heavy insertion and deletion of records. However, it is a good structure for searching records in case the appropriate index exists.

4.7.3 Inverted File Organisation

Inverted file organisation is one file organisation where the index structure is most important. In this organisation the basic structure of file records does not matter much. This file organisation is somewhat similar to that of multi-list file organisation with the key difference that in multi-list file organisation index points to a list, whereas in inverted file organisation the index itself contains the list. Thus, maintaining the proper index through proper structures is an important issue in the design of inverted file organisation. Let us show inverted file organisation with the help of data given in *Figure 4.14*.

Let us assume that the inverted file organisation for the data shown contains a dense index. *Figure 4.17* shows how the data can be represented using inverted file organisation.

Empid index

495	B
595	E
705	D
795	A
905	C

Qualification index

B.Tech	B, D, A
MCA	E, C

Gender index

Female	B, E, C
Male	D, A

Salary index

13200	D
14100	E
16200	A
17100	C
19000	B

Figure 4.17: Some of the indexes for fully inverted file

Please note the following points for the inverted file organisation:

- The index entries are of variable lengths as the number of records with the same key value is changing, thus, maintenance of index is more complex than that of multi-list file organisation.
- The queries that involve Boolean expressions require accesses only for those records that satisfy the query in addition to the block accesses needed for the indices. For example, the query about Female, MCA employees can be solved by the Gender and Qualification index. You just need to take the intersection of record numbers on the two indices. (Please refer to *Figure 4.17*). Thus, any complex query requiring Boolean expression can be handled easily through the help of indices.

4.8 IMPORTANCE OF FILE ORGANISATION IN DATABASES

To implement a database efficiently, there are several design tradeoffs needed. One of the most important ones is the file Organisation. For example, if there were to be an application that required only sequential batch processing, then the use of indexing techniques would be pointless and wasteful.

There are several important consequences of an inappropriate file Organisation being used in a database. The wrong file Organisation will result in:

- much larger processing time for retrieving or modifying the required record.
- undue disk access that could stress the hardware.

Needless to say, there could be many undesirable consequences at the user level, such as making some applications impractical.

Check Your Progress 3

1) What is the difference if indexes are implemented using binary search tree or using B-tree?

……

……

2) What are the advantages of using B+ tree index over B tree index if the supported file organisation is required to access the records sequentially too.

……

……

……

3) What is the need of a multi-list organisation? What is the advantage of storing the number of records in an index entry?

……

……

……

4.9 SUMMARY

In this unit, we discussed the physical database design issues in which we had addressed the file Organisation and file access method. The unit also discusses different types of file organization giving their advantages and their disadvantages.

An index is an important component of a database system, as one of the key requirements of DBMS is efficient access to data. This unit explains various types of indexes that may exist in database systems. Some of these are: Primary index, clustering index and secondary index. The secondary index results in better search performance but adds on the task of index updating. This unit also discusses two multi-key file organisations viz. multi-list and inverted file organisations. These are very useful for improving query performance.

4.10 SOLUTIONS / ANSWERS

Check Your Progress 1

1)

Operation	Comments
File Creation	It will be efficient if transaction records are ordered by record key
Record Location	As it follows the sequential approach it is inefficient. On an average, half of the records in the file must be processed to locate a record.
Record Creation	It will require you to browse through all the records to check if such a record already exists or not. Thus, the entire file must be read and written. Efficiency improves if a group of records are created together. This operation could be combined with deletion and modification transactions to achieve greater efficiency.
Record Deletion	The entire file must be read and written. Efficiency improves with greater number of deletions. This operation could be combined with addition and modification transactions to achieve greater efficiency.
Record Modification	Very efficient if the number of records to be modified is high and the records in the transaction file are ordered by the record key.

2) Direct-access systems do not search the entire file; rather they move directly to the record, which is to be accessed. To be able to achieve this, several strategies like relative addressing, hashing and indexing can be used.

3) It is a technique for physically arranging the records of a file on secondary storage devices. When choosing the file Organisation, you should consider the following factors:

1. Data retrieval speed
2. Data processing speed
3. Efficient use of storage space
4. Protection from failures and data loss
5. Scalability
6. Security

Check Your Progress 2

1) **Number of accesses without using Primary Index:**

Number of records in the file = 40000
Block size = 1024 bytes
Record size = 200 bytes
Number of records per block = integer value of [1024 / 200] = 05
Number of disk blocks acquired by the file
= [Number of records / records per block]
= [40000/05] = 8000
The file is in the order of primary index, so binary search can be employed to access the file. As $2^{13} = 8192$. Thus, about 13 Block accesses would be required to locate the block containing the desired record.

Number of accesses with Primary Index:

Size of an index entry = 4+8 = 12 bytes
Number of index entries that can be stored per block
= integer value of [1024 / 12] = 85
Number of index entries = number of disk Blocks of file = 8000
Number of index blocks = ceiling of [8000/ 85] = 94
Number of index block transfers to find the value in index blocks
= ceiling of [$\log_2$94] = 7
One block transfer will be required to get the data records using the index pointer after the required index value has been located. So total number of block transfers with secondary index = 7 + 1 = 8

Thus, the Primary index would save about 5 block transfers for the given case.

2) **Number of accesses without using Secondary Index:**

Number of disk Blocks = 8000 (as computed in Question 1)
The file is in the order of primary index, so search on alternative key would require on an average half of the Blocks to be searched.
Average number of block transfers (without secondary index) = 4000

Number of accesses with Secondary Index on alternate key (dense index):

Size of an index entry = 8+8 = 16 bytes
Number of index entries that can be stored per block
= integer value of [1024 / 16] = 64
Number of index entries = number of records = 40000
Number of index blocks = ceiling of [40000/ 64] = 625
Number of index block transfers to find the value in index blocks
= ceiling of [$\log_2$ 625] = 10
One block transfer will be required to get the data records using the index pointer after the required index value has been located. So total number of block transfers with secondary index = 10 + 1 = 11

Thus, the Secondary index would save a large number of block transfers.

3) (i) Sparse as only one entry per Block of data.
(ii) Dense, as one index entry for each attribute value, sparse if multi-level index is used.
(iii) Dense, as one index entry for each attribute value
(iv) Dense, as one index entry for each unique attribute

Check Your Progress 3

1) In a B+ tree the leaves are linked together to form a sequence set; interior nodes exist only for the purposes of indexing the sequence set (not to index into data/records). The insertion and deletion algorithms differ slightly.

2) Sequential access to the keys of a B-tree is much slower than sequential access to the keys of a B+ tree, since the latter have all the keys linked in sequential order in the leave.

3) The multi-list organisation enhances the query answering capability of databases on multiple key values. The number of records in an index entry determines the length of the index on a specific attribute value. This enhances the performance of searching when more than one search term are used, as you can select the smaller list and then apply second search term on it.