MASTER OF COMPUTER APPLICATION

MCA-11

DATABASE MANAGEMENT SYSTEM

Directorate of Distance Education
Guru Jambheshwar University of
Science & Technology
Hisar - 125001

CONTENTS

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	AUTHOR: DR. DEEPAK NANDAL
LESSON NO. 1	VETTER:
OVERVIEW OF FILE SYSTEM & DATABASE SYSTEMS	

STRUCTURE

1.0   LEARNING OBJECTIVE

- The objective of this chapter is to make the reader understand the concepts of data, information and knowledge. Detailed discussion about the traditional aspect of data and traditional file based system. To study the file oriented system and find out the drawbacks of it.

1.1 INTRODUCTION

Today, more than at any previous time, the success of an organization depends on its ability to acquire accurate and timely data about its operations, to manage this data effectively, and to use it to analyze and guide its activities. Phrases such as the *information superhighway* have become ubiquitous, and information processing is a rapidly growing multibillion dollar industry. The amount of information available to us is literally exploding, and the value of data as an organizational asset is widely recognized. Yet without the ability to manage this vast amount of data, and to quickly find the information that is relevant to a given question, as the amount of information increases, it tends to become a distraction and a liability, rather than an asset. This paradox drives the need for increasingly powerful and flexible data management systems. To get the most out of their large and complex datasets, users must have tools that simplify the tasks of managing the data and extracting useful information in a timely fashion. Otherwise, data can become a liability, with the cost of acquiring it and managing it far exceeding the value that is derived from it.

A historical perspective

"In the late 1960s, IBM developed the Information Management System (IMS) DBMS, used even today in many major installations. IMS formed the basis for an

alternative data representation framework called the *hierarchical data model*. The SABRE system for making airline reservations was jointly developed by American Airlines and IBM around the same time, and it allowed several people to access the same data through a computer network. Interestingly, today the same SABRE system is used to power popular Web-based travel services such as Travelocity!

In 1970, Edgar Codd, at IBM's San Jose Research Laboratory, proposed a new data representation framework called the *relational data model*. This proved to be a watershed in the development of database systems: it sparked rapid development of several DBMSs based on the relational model, along with a rich body of theoretical results that placed the field on a firm foundation. Codd won the 1981 Turing Award for his seminal work. Database systems matured as an academic discipline, and the popularity of relational DBMSs changed the commercial landscape. Their benefits were widely recognized, and the use of DBMSs for managing corporate data became standard practice.

In the 1980s, the relational model consolidated its position as the dominant DBMS paradigm, and database systems continued to gain widespread use. The SQL query language for relational databases, developed as part of IBM's System R project, is now the standard query language. SQL was standardized in the late 1980s, and the current standard, SQL-92, was adopted by the American National Standards Institute (ANSI) and International Standards Organization (ISO). Arguably, the most widely used form of concurrent programming is the concurrent execution of database programs (called *transactions*). Users write programs as if they are to be run by themselves, and the responsibility for running them concurrently is given to the DBMS. James Gray won the 1999 Turing award for his contributions to the field of transaction management in a DBMS. In the late 1980s and the 1990s, advances have been made in many areas of database systems. Considerable research has been carried out into more powerful query languages

and richer data models, and there has been a big emphasis on supporting complex analysis of data from all parts of an enterprise. Several vendors (e.g., IBM's DB2, Oracle 8, and Informix UDS) have extended their systems with the ability to store new data types such as images and text, and with the ability to ask more complex queries. Specialized systems have been developed by numerous vendors for creating *data warehouses*, consolidating data from several databases, and for carrying out specialized analysis.

An interesting phenomenon is the emergence of several *enterprise resource planning (ERP)* and *management resource planning (MRP)* packages, which add a substantial layer of application-oriented features on top of a DBMS. Widely used packages include systems from Baan, Oracle, PeopleSoft, SAP, and Siebel. These packages identify a set of common tasks (e.g., inventory management, human resources planning, financial analysis) encountered by a large number of organizations and provide a general application layer to carry out these tasks. The data is stored in a relational DBMS, and the application layer can be customized to different companies, leading to lower overall costs for the companies, compared to the cost of building the application layer from scratch.

Most significantly, perhaps, DBMSs have entered the Internet Age. While the first generation of Web sites stored their data exclusively in operating systems files, the use of a DBMS to store data that is accessed through a Web browser is becoming widespread. Queries are generated through Web-accessible forms and answers are formatted using a markup language such as HTML, in order to be easily displayed in a browser. All the database vendors are adding features to their DBMS aimed at making it more suitable for deployment over the Internet.

Database management continues to gain importance as more and more data is brought on-line, and made ever more accessible through computer networking. Today the field is being driven by exciting visions such as multimedia databases, interactive video, digital libraries,

and a host of scientific projects such as the human genome mapping effort and NASA's Earth Observation System project, and the desire of companies to consolidate their decision-making processes and *mine* their data repositories for useful information about their businesses. Commercially, database management systems represent one of the largest and most vigorous market segments. Thus the study of database systems could prove to be richly rewarding in more ways than one!

1.2 DEFINITION

Data is the fuel that drives the financial services industry. Without it, organizations would cease to function. It is data that ensures that every system and every process within the organization functions at an optimal level. Data is mission-critical because it:

- Influences every decision
- Powers risk management
- Offers insight into markets, products, services, customers and counterparties
- Pinpoints a company's positions, exposures, and available liquidity
- Is demanded by regulators and auditors

As the markets change, the volume of data to be managed is increasing, adding greater complexity to the process. Data management has for too long been regarded as an infrastructure problem for IT to solve. But this is changing. The reality is that data management is as central to successful, sustainable operations as risk management. Data management is not a technology or a tool – it is a business enabler. At Asset Control, we view data management as a critical process which ensures that essential business decisions are based on accurate, consistent and verifiable information.

Traditional File Based System- The traditional file system (TFS) is a method of storing and arranging computer files and the information in the file (data). Basically it organizes these files into a database for the storage, organization, manipulation, and retrieval by the computer's operating system.

1.3 BASIC CONCEPTS

In an organization, the data is the most basic resource. To run the organization efficiently, the proper organization and management of data is essential. The formal definition of the major terms used in databases and database systems is defined in this section.

1.3.1 DATA AND INFORMATION

- **Data:** The term data may be defined as known facts that could be recorded and stored on Computer Media. It is also defined as raw facts from which the required information is produced. Data represents unorganized and unprocessed facts.
  - Usually data is static in nature.
  - It can represent a set of discrete facts about events.
  - Data is a prerequisite to information.
  - An organization sometimes has to decide on the nature and volume of data that is required for creating the necessary information.
- **Information**: Data and information are closely related and are often used interchangeably. Information is nothing but refined data. In other way, we can say, information is processed, organized or summarized data. According to Burch et. al., "Information is data that have been put into a meaningful and useful content and communicated to a recipient who uses it to made decisions". Information consists of data, images, text, documents and voice, but always in a meaningful content. So we can say, that information is something more than mere data. Data are processed

to create information. The recipient receives the information and then makes a decision and takes an action, which may triggers other actions

- o Information can be considered as an aggregation of data (processed data) which makes decision making easier.
- o Information has usually got some meaning and purpose.

In these days, there is no lack of data, but there is lack of quality information. The quality information means information that is accurate, timely and relevant, which are the three major key attributes of information.

1. Accuracy: It means that the information is free from errors, and it clearly and accurately reflects the meaning of data on which it is based. It also means it is free from bias and conveys an accurate picture to the recipient.

2. Timeliness: It means that the recipients receive the information when they need it and within the required time frame.

3. Relevancy: It means the usefulness of the piece of information for the corresponding persons. It is a very subjective matter. Some information that is relevant for one person might not be relevant for another and vice versa e.g., the price of printer is irrelevant for a person who wants to purchase computer. So, organization that have good information system, which produce information that is accurate, timely and relevant will survive and those that do not realize the importance of information will soon be out of business.
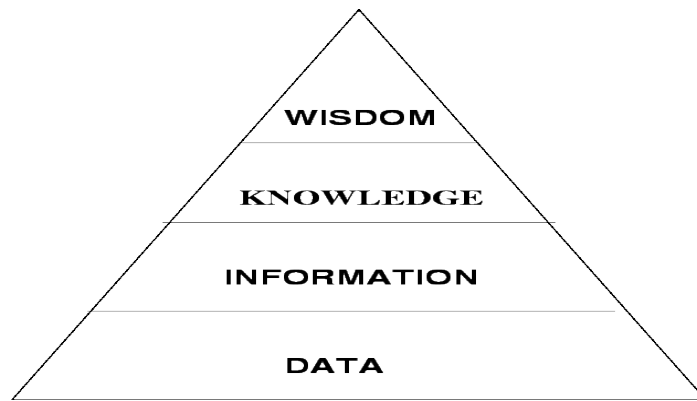
Figure1.1 Data, Information, Knowledge and Wisdom

1.3.2 FIELDS AND RECORDS:

Fields: It is the smallest unit of the data that has meaning to its users and is also called data item or data element. Name, Address and Telephone number are examples of fields. These are represented in the database by a value. A database field is a set of data values, of the same data type, in a table. It is also referred to as a column or an attribute.

Most databases also allow fields to hold complex data like pictures, entire files, and even movie clips. A field that allows the same data type does not mean it only has simple text values. Some databases allow the data to be stored as a file on the Operating System, while the field data only contains a pointer or link to the actual file. This is done to keep the database size manageable, given that smaller database sizes means less time for backups, as well as for searching data within the database.

A simple example is a table 1.1 that saves employee's job record. The fields in this table could be the following: Employee ID, Last Name, First Name, Position, Department, and Hire Date.

Employee ID	Last Name	First Name	Position	Department	Hire Date
00108	Doe	John	Assistant Manager	Human Resources	November 16, 2000

| 00109 | Parker | Anne | Supervisor | Financial Services | May 1, 2003 |

Table 1.1: Example of a table with fields

Records: Records provide a practical way to store and retrieve data from the database. Each record can have different kinds of data, and thus a single row could have several types of information. A customer record could contain an ID number, name, birth date, cell phone number, and email.

There is one exception to the above rule. A good database design should include a primary key for the table. This means that each record in the data set has one field that is unique among all records and that it can't be repeated. Tools like Microsoft Access let you easily set a field to be the primary key; this is usually a field that is auto-numbered (starting at 1) and keeps adding as you add rows/records to the table.

The term "record" can also be pronounced reCORD, meaning to save information/data. This is also a helpful means to identify a database record: It's a record, or the data, that has been reCORDed. A group of records can be called a file, data set, or table. Microsoft Access and other database tools refer to these objects as tables: This lesson will refer to the collective group of records as a table. A record is a group of data saved in a table. It is a set of fields, like an employee's job record as shown below in table 1.2.

Employee ID	Last Name	First Name	Position	Department	Hire Date
00108	Doe	John	Assistant Manager	Human Resources	November 16, 2000
00109	Parker	Anne	Supervisor	Financial Services	May 1, 2003

Table 1.2: Example of a Record

A record in a database is an object that can have one or more values. Groups of records are then saved in a table; the table determines the data that each record may have. Various tables hold various records in a database. A new record produces a new row in the table

that's why records are oftentimes labeled as rows. Separate fields are referred to as columns because they are identical for every record in the table. Record and row can be utilised mutually, but nearly all database management systems utilise row for error messages and queries. Records provide a practical way to save and pull out data from the database. Each record can have diverse types of data, and thus a single row could have several kinds of information. Records can be easily created, altered and erased without affecting other data in the database. An ideal database design should have a primary key for the table. A primary key is a unique field in each record in a database. In an employee's job record sample above, the Employee ID is the primary key. A group of records can be called a file, data set or table.

1.4 TRADITIONAL FILE BASED SYSTEM

A file processing system is a collection of files and programs that access/modify these files. Typically, new files and programs are added over time (by different programmers) as new information needs to be stored and new ways to access information are needed. Problems with file processing systems:

- Data redundancy and inconsistency
- Difficulty of accessing data
- Problems with concurrent access

Example: assume I'm paying for groceries with my Debit/Credit card at the same time my pay check is being deposited (and my bank uses a file processing system):

Withdrawal program	Deposit program
1. Read balance from checking account file as Rs. 50000.	1. Read balance from checking account file as Rs. 50000.
3. subtract Rs. 10000 (for groceries)	2. add Rs. 40000 (my salary)

4. update checking account file(new balance: Rs. 40000)	3. update checking account file (new balance: Rs. 90000)

It is difficult to prevent such problems unless programs (example: withdrawal and deposit) are coordinated or integrated.

• Atomicity problems - ensuring that a system failure during a database update does not leave the

database in an inconsistent state

• Security problems

    – not all users should have access to all data

    – example: bank payroll personnel shouldn't know my checking account balance

    – difficult to enforce security in an ad hoc system

• Integrity problems

    – data may need to satisfy certain conditions, called consistency constraints

    – difficult to enforce/add/change consistency constraints in a file processing system

DBMSs were developed to remedy these problems.

Traditional file processing systems include manual systems and also computer based file systems that were linked to particular application programs. This is the type of file processing that you used with your 3GL programming. They share a number of characteristics. Let's see how a traditional file system looks in figure 1.2.
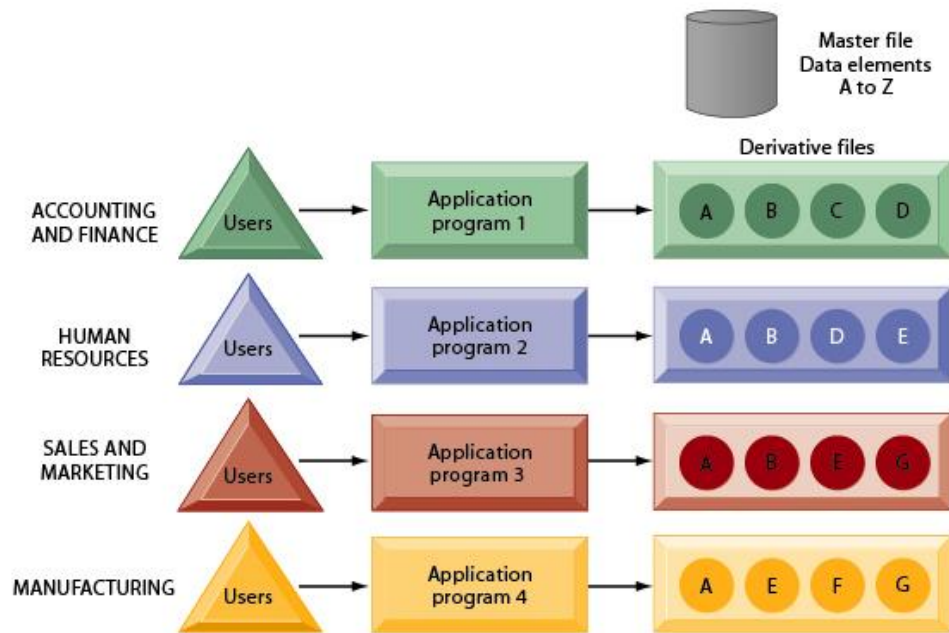
Figure 1.2: Traditional File System

1.4.1 DRAWBACKS OF TRADITIONAL FILE BASED SYSTEM

There are a number of disadvantages associated with traditional file processing systems.

1.   **Data duplication**

When files are duplicated and held in a number of locations situations can arise that will cause data to be inconsistent.

- Corrections or modifications made in one location may not be updated in another. For example, customer address files held by the Accounts Department may be updated while those held by Sales are not updated. For the customer this may mean that the account arrives but the goods do not.

- Modifications made to data files may also lead to less obvious discrepancies. For example a product name may be spelt differently in two locations eg. Bisleri, Bislery. A report generated calculating sales to customers by product may then include the same customers twice. This may not be obvious if the report is a summary style report.

2.   **Poor data control**

File systems have no centralised control of the data descriptions. Tables and field names may be used in different locations to mean different things. For example, the Sales department's files may list a customer as having a single Name field that is made up of customers Initial and Last name eg Rahul Gupta. The Accounts department may keep the customer's name in three separate fields; First name, Initial, Last Name. This may make it difficult to compare the data in the two files or at least require additional time in programming the comparison.

3.   Inadequate data manipulation capabilities

Data in traditional file systems is not easily related, particularly if the files have been developed for separate purposes. If the organisation requires information to be generated that accesses data from several unrelated files the task may prove difficult or require re-entry of data. For example, in a library the catalogue of books may be held in one file. Books on order for the library may be held in another file. When books are received the catalogue will need to be manually updated if the two files are not related.

4.   Program data dependence

File data is stored within each of the applications that use that data eg A sales transaction program may have several files relevant to it, Customer, Stock_in_hand, Sale_Info. These files are integrated into the program.

5.   Limited data sharing

The dependence of the data on the program means that the files are not necessarily suitable for a new program that is being developed. The new program may need its data in another form or require additional data that is not held.

6.   Lengthy development times

Each new application requires development of the program along with the development of the relevant files for that application. Although the data may be held elsewhere in the

organization the data will need to be imported or re-entered into the new files. This takes time. As organizations grow and change they need to change their internal applications quickly to meet new demands. Lengthy development times are a disadvantage.

7.   Program maintenance

File maintenance can be time consuming in traditional file processing systems. Changes to files mean changes to application programs

1.5 DBMS FUNCTIONS

Here are several functions that a DBMS performs to ensure data integrity and consistency of data in the database. The ten functions in the DBMS are: data dictionary management, data storage management, data transformation and presentation, security management, multiuser access control, backup and recovery management, data integrity management, database access languages and application programming interfaces, database communication interfaces, and transaction management.

1.   Data Dictionary Management

Data Dictionary is where the DBMS stores definitions of the data elements and their relationships (metadata). The DBMS uses this function to look up the required data component structures and relationships. When programs access data in a database they are basically going through the DBMS. This function removes structural and data dependency and provides the user with data abstraction. In turn, this makes things a lot easier on the end user. The Data Dictionary is often hidden from the user and is used by Database Administrators and Programmers.

2.   Data Storage Management

This particular function is used for the storage of data and any related data entry forms or screen definitions, report definitions, data validation rules, procedural code, and structures that can handle video and picture formats. Users do not need to know how data is stored or

manipulated. Also involved with this structure is a term called performance tuning that relates to a database's efficiency in relation to storage and access speed.

3. Data Transformation and Presentation

 [</span][link title](#)>   This function exists to transform any data entered into required data structures. By using the data transformation and presentation function the DBMS can determine the difference between logical and physical data formats.

4. Security Management

This is one of the most important functions in the DBMS. Security management sets rules that determine specific users that are allowed to access the database. Users are given a username and password or sometimes through biometric authentication (such as a fingerprint or retina scan) but these types of authentication tend to be more costly. This function also sets restraints on what specific data any user can see or manage.

5. Multiuser Access Control

 Data integrity and data consistency are the basis of this function. Multiuser access control is a very useful tool in a DBMS, it enables multiple users to access the database simultaneously without affecting the integrity of the database.

6. Backup and Recovery Management

 Backup and recovery is brought to mind whenever there is potential outside threats to a database. For example if there is a power outage, recovery management is how long it takes to recover the database after the outage. Backup management refers to the data safety and integrity; for example backing up all your mp3 files on a disk.

7. Data Integrity Management

The DBMS enforces these rules to reduce things such as data redundancy, which is when data is stored in more than one place unnecessarily, and maximizing data consistency, making sure database is returning correct/same answer each time for same question asked.

8.   Database Access Languages and Application Programming Interfaces

A query language is a nonprocedural language. An example of this is SQL (structured query language). SQL is the most common query language supported by the majority of DBMS vendors. The use of this language makes it easy for user to specify what they want done without the headache of explaining how to specifically do it.

9.   Database Communication Interfaces

This refers to how a DBMS can accept different end user requests through different network environments. An example of this can be easily related to the internet. A DBMS can provide access to the database using the Internet through Web Browsers (Mozilla Firefox, Internet Explorer, and Netscape).

1.6 CHECK YOUR PROGRESS

1. To create a file _____

    a. Allocate the space in the file system

    b. Make an entry for new file in directory

    c. Allocate the space in the file system and make an entry for new file in directory

    d. None of above

2. File system can be represented by _____.

3. Which file is a sequence of bytes organized into blocks understandable by the system's linker?

4. Mapping of file is managed by _____.

5. _____ is a unique tag, usually a number identifies the file in the file system.

1.7 SUMMARY

An organization must have accurate and reliable data (information) for effective decision making. Data (information) is the backbone and most critical resource of an organization that enables managers and organizations to gain a competitive edge. In this age of information explosion, where people are bombarded with data, getting the right information, in the right amount, at the right time is not an easy task. So, only those organizations will survive that successfully manage information.

A database system simplifies the tasks of managing the data and extracting useful information in a timely fashion. A database system is an integrated collection of related files, along with the details of the interpretation of the data. A Data Base Management System is a software system or program that allows access to data contained in a database. The objective of the DBMS is to provide a convenient and effective method of defining, storing, and retrieving the information stored in the database. The database and database management systems have become essential for managing business, governments, schools, universities, banks etc. File system is collection of data. In this system, user has to write procedures for managing database. It provides details of data representation and storage of data. In this –

- Data is stored in files.
- Each file has specific format.
- Programs that use these files depend on knowledge about that format.
- In earlier days, database applications were built on top of file systems.

This approach is mostly obsolete but –

- Understanding problems inherent in file based systems may prevent us from repeating these problems in our database system.

- Understanding how file system works is extremely useful when converting a file-based system to a database system.

Basically, it is a collection of application programs that performs services for end users such as production of reports. Each file defines and manages its own data. It doesn't have a crash mechanism i.e., if system crashes while entering some data, then content of file will be lost. This is disadvantage of traditional file based system. Also, it is very difficult to protect a file under the file system. This system can't efficiently store and retrieve data.

1.8 KEYWORDS

- **DATA-** Data are measured, collected and reported, and analysed, whereupon it can be visualized using graphs, images or other analysis tools. Data as a general concept refers to the fact that some existing information or knowledge is represented or coded in some form suitable for better usage or processing.
- **RECORD-** A record is a database entry that may contain one or more values. Groups of records are stored in a table, which defines what types of data each record may contain. Databases may contain multiple tables which may each contain multiple records.
- **FILE-** A file is a collection of records. For example, a telephone book is analogous to a file.
- **FIELD-** In computer science, data that has several parts, known as a record, can be divided into fields. Relational databases arrange data as sets of database records, so called rows. Each record consists of several fields; the fields of all records form the columns. Examples of fields: name, gender, hair colour.
- **TFS-** Traditional File System

1.9 SELF-ASSESSMENT TEST

1. What are data and information, and how are they related in a database?

2. Who is E.F. Codd, and why is he significant in the development of modern database systems?

3. What is Enterprise Resource Planning (ERP), and what kind of a database is used in an ERP application?

4. Discuss a traditional file system in detail.

5. How a traditional file system differ from Data base system, what are the drawbacks of it?

1.10 ANSWERS TO CHECK YOUR PROGRESS

1. C
2. File extension
3. Object file
4. File metadata
5. File indentifier

1.11 REFERENCES / SUGGESTED READINGS

- C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.
- Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.
- Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.
- Thomas Connolly Carolyn Begg, "Database System", 3/2, Pearson Education.
- https://www.geeksforgeeks.org/traditional-file-system/
- https://www.dlsweb.rmit.edu.au/Toolbox/Database/Certificate4_DB_Toolbox/content/dbsystems/file_process.htm
- https://whatisdbms.com/what-is-traditional-file-processing-system-and-its-characteristics/

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	AUTHOR: DR. DEEPAK NANDAL
LESSON NO. 2	VETTER:

DATABASE APPROACH- CHARACTERISTICS OF DATABASE APPROACH

2.0   LEARNING OBJECTIVE

- The objective of this chapter is to make the reader understand the procedural language for SQL. The architecture of PL/SQL in detailed will be studies and to get familiar with the loops that can be used in the procedural language for SQL.

2.1 INTRODUCTION

DBMS stands for Database Management System. We can break it like this DBMS = Database + Management System. Database is a collection of data and Management System is a set of programs to store and retrieve those data. Based on this we can define DBMS like this: DBMS is a collection of inter-related data and set of programs to store & access those data in an easy and effective manner. Database systems are basically developed for large amount of data. When dealing with huge amount of data, there are two things that require optimization: Storage of data and retrieval of data.

According to the principles of database systems, the data is stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed before storage. Let's take a layman example to understand this:

In a banking system, suppose a customer is having two accounts, one is saving account and another is salary account. Let's say bank stores saving account data at one place (these places are called tables we will learn them later) and salary account data at another place, in that case if the customer information such as customer name, address etc. are stored at both places then this is just a wastage of storage (redundancy/ duplication of data), to organize the data in a better way the information should be stored at one place and both the accounts should be linked to that information somehow. The same thing we achieve in DBMS.

Fast Retrieval of data: Along with storing the data in an optimized and systematic manner, it is also important that we retrieve the data quickly when needed. Database systems ensure that the data is retrieved as quickly as possible.

2.2 DEFINITION

The DBMS software together with the Database is called a database system. In other words, it can be defined as an organization of components that define and regulate the collection, storage, management and use of data in a database. Furthermore, it is a system whose overall purpose is to record and maintain information. A database system consists of fou major components as shown in Figure 2.1.
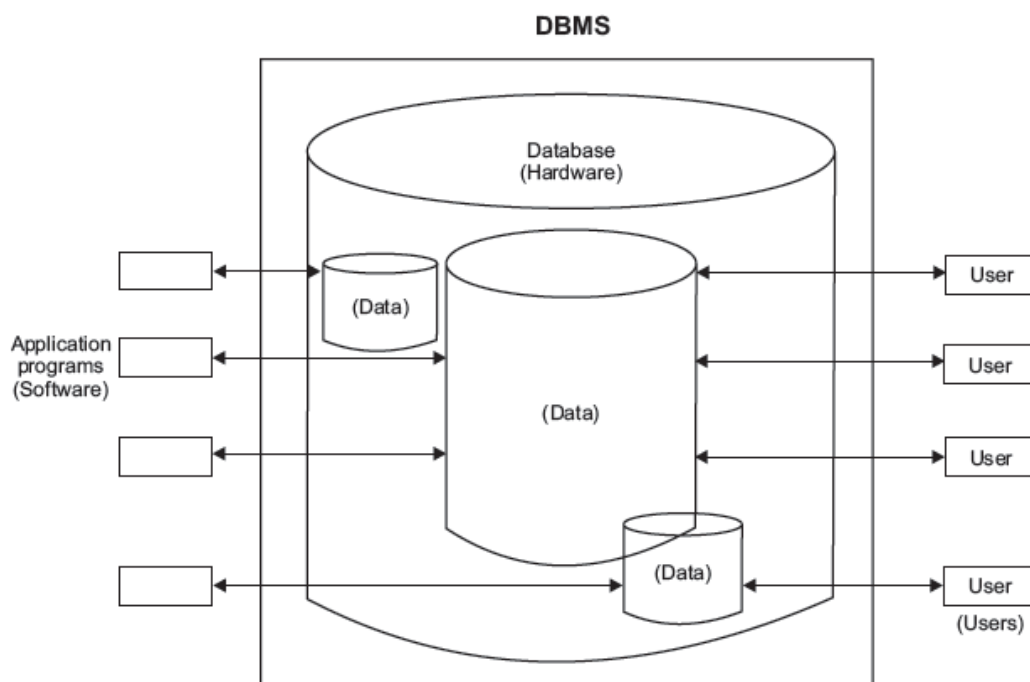

Figure 2.1: Database system

2.3 WHAT IS DBMS?

We have studied the concepts of data item (also called field), record and file in the previous chapter, we have seen information regarding all the employees being stored in a file called "employee-file". The file contained many records, one per employee. Each employee

record consisted of data items such as employee number, name and basic pay. Similar examples of files could be student-file, purchase-order-file, and invoice- file and so on. In typical business environments, it is always essential to be able to produce the right information at the right time with minimum efforts. Assume that a manufacturer of goods uses 10 such different files (one for suppliers, one for customers, one for accounts, etc.). It might not be very easy to answer a query such as: How many of our customers have credit balance with us for over a month now and whose average purchases from December last year to February this year have been above average? You can imagine the complexity involved in providing this information. It is not that such a report cannot be generated at all. It certainly can be produced, however, it will require a lot of effort. We shall see the reasons behind this and also study what better systems exist. More specifically, we shall study how a database is a better solution than a set of files. Also, a Database Management System (DBMS) scores over File Management System (FMS) on many counts.

The DBMS has evolved over the years from being a simple means of arranging data to a much more sophisticated organisation and retrieval of data as and when required, in real-time. We shall study the different types of databases and understand the differences between them. Relational Database Management Systems (RDBMS), about which we shall study too, have become the most popular of them all for many reasons. Our aim is also to see how we can access data from a RDBMS using the Structured Query Language (SQL). The four major components are:

1. **Data:** The whole data in the system is stored in a single database. This data in the database are both shared and integrated. Sharing of data means individual pieces of data in the database is shared among different users and every user can access the same piece of data but may be for different purposes. Integration of data means the

database can be function of several distinct files with redundancy controlled among the files.

2. **Hardware:** The hardware consists of the secondary storage devices like disks, drums and so on, where the database resides together with other devices. There is two types of hardware. The first one, i.e., processor and main memory that supports in running the DBMS. The second one is the secondary storage devices, i.e., hard disk, magnetic disk etc., that are used to hold the stored data.

3. **Software:** A layer or interface of software exists between the physical database and the users. This layer is called the DBMS. All requests from the users to access the database are handled by the DBMS. Thus, the DBMS shields the database users from hardware details. Furthermore, the DBMS provides the other facilities like accessing and updating the data in the files and adding and deleting files itself.

4. **Users**: The users are the people interacting with the database system in any way. There are four types of users interacting with the database systems. These are Application Programmers, online users, end users or naive users and finally the Database Administrator (DBA).

Although using files was a satisfactory approach for small organisations and businesses, it was not quite easy to work with for larger establishments. Hence, a need for storing information centrally and using it as and when needed was felt. This would take care of the problems with files. The most important change brought about by DBMS is that the programs no longer interact with the data files directly. Instead, they communicate with the DBMS, which acts as a middle agency. It controls the flow of information from and to the database, as shown in Fig. 2.2.

If we compare this figure with the earlier one, the initial reaction might be that an extra layer of complexity has been added. However, this extra layer is not a cause for worry as it

is completely transparent to the end user and, in fact, it helps. As the figure shows, the files are integrated. This means that there is no duplication of data. Instead, all the files are stored together. They are managed by DBMS. In fact, the user or programmer does not even know about the files used by DBMS. DBMS internally uses data structures such as chains, pointers and indexes.
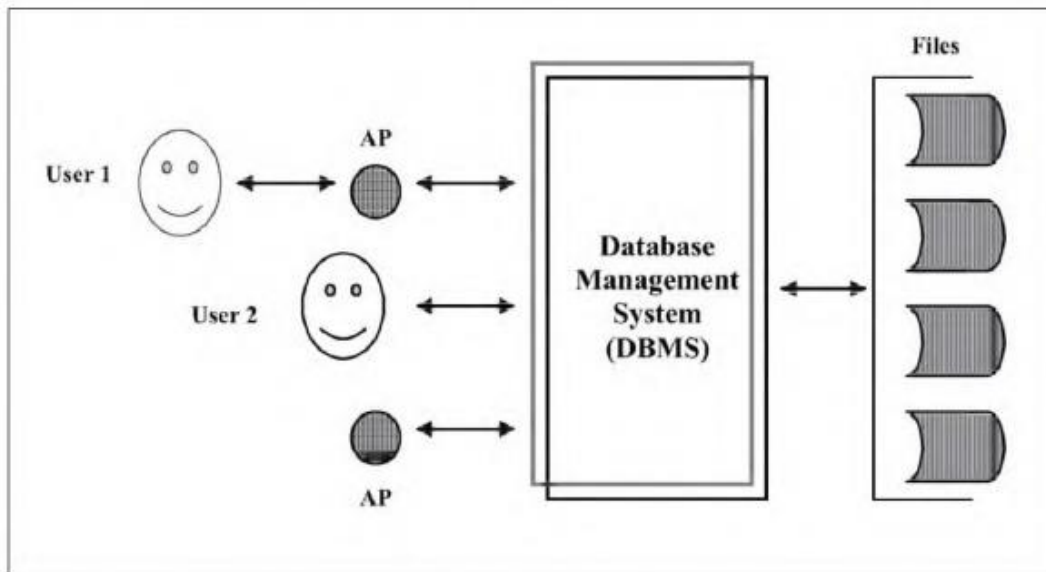

Figure 2.2: Database approach

However, the user or programmer need not worry about the internal details of how the data is stored such as whether it is on one disk or more, on which sectors, in a continuous pattern or in chunks, in what data structures (e.g. chains/indexes) and so on. If the user wants to find all the invoices in which the value is > $500, DBMS can produce the result. It may use the indexes on invoice value to achieve this, or it may go through the invoices record sequentially. The user need not worry. Only the category of people called Data Base Administrator (DBA) need to know the details of data storage. This is because they are concerned with the performance and security aspects of DBMS. This is how DBMS hides all the complexities involved in maintaining files and provides a common and simple interface. There is another interesting consequence illustrated in Fig. 2.2. In it,

User 2 is interacting directly with the database, without needing to use an application program. This is possible since DBMS provides a set of commands for interacting with and manipulating the database. At the same time, User 1 wants to access/manipulate the database, which is not directly possible by using its set of DBMS commands. Therefore, the user's interaction is through an application program. The third case is a batch program that executes without a user, sitting and interacting with the program database continuously through a terminal. The batch program executes on its own, once scheduled to run at a specific time. Thus, online (simple and complex) as well as batch data processing is easily handled by DBMS.

2.4 DATABASE APPROACH

In order to remove all limitations of the File Based Approach, a new approach was required that must be more effective known as Database approach. The Database is a shared collection of logically related data, designed to meet the information needs of an organization. A database is a computer based record keeping system whose over all purpose is to record and maintains information. The database is a single, large repository of data, which can be used simultaneously by many departments and users. Instead of disconnected files with redundant data, all data items are integrated with a minimum amount of duplication.

The database is no longer owned by one department but is a shared corporate resource. The database holds not only the organization's operational data but also a description of this data. For this reason, a database is also defined as a self-describing collection of integrated records. The description of the data is known as the Data Dictionary or Meta Data (the 'data about data'). It is the self-describing nature of a database that provides program-data independence.

A database implies separation of physical storage from use of the data by an application program to achieve program/data independence. Using a database system, the user or programmer or application specialist need not know the details of how the data are stored and such details are "transparent to the user". Changes (or updating) can be made to data without affecting other components of the system. These changes include, for example, change of data format or file structure or relocation from one device to another.

2.5 CHARACTERSTICS OF DATABASE APPROACH

There are number of characteristics that distinguish the database approach from the much older approach of programming with files.

- In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application.

For example, one user, the grade reporting office, may keep files on students and their grades. Programs to print a student's transcript and to enter new grades are implemented as part of the application. A second user, the accounting office, may keep track of students' fees and their payments. Although both users are interested in data about students, each user maintains separate files— and programs to manipulate these files— because each requires some data not available from the other user's files. This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.

- In the database approach, a single repository maintains data that is defined once and then accessed by various users. In file systems, each application is free to name data elements independently. In contrast, in a database, the names or

labels of data are defined once, and used repeatedly by queries, transactions, and applications.

The main characteristics of the database approach (feature of database approach) and how it differs from the traditional file system i.e file-processing approach:

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing

Self-describing nature of a database system:

1. A fundamental characteristics of database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called meta-data, and it describes the structure of the primary database.

2. The catalog is used by the DBMS software and also by database users who need information about the database structure. A general-purpose DBMS software package is not written for a specific database application. Therefore, it must refer to the catalog to know the structure of the files in a specific database, such as the type and format of data it will access. The DBMS software must work equally well with any number of database applications—for example, a university database, a banking database, or a company database—as long as the database definition is stored in the catalog.

29

3. In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs.

Insulation between programs and data, and data abstraction

1. In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property program-data independence.

2. For example, a file access program may be written in such a way that it can access only STUDENT records of the structure. If we want to add another piece of data to each STUDENT record, say the Birth_date , such a program will no longer work and must be changed. By contrast, in a DBMS environment, we only need to change the description of STUDENT records in the catalog to reflect the inclusion of the new data item Birth_date; no programs are changed.

3. The next time a DBMS program refers to the catalog, the new structure of STUDENT records will be accessed and used. In some types of database systems, such as object-oriented and object-relationa systems, users can define operations on data as part of the database definitions. An operation (also called a function or method) is specified in two parts. The interface (or signature) of an operation includes the operation name and the data types of its arguments (or parameters). The implementation (or method) of operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed program-operation independence.

4. The characteristic that allows program-data independence and program-operation independence is called data abstraction. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented. Informally, a data model is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model hides storage and implementation details that are not of interest to most database users.

2.6 ADVANTAGES OF DBMS

There are several advantages of Database management system over file system. Few of them are as follows:

- **No redundant data**: Redundancy removed by data normalization. No data duplication saves storage and improves access time.
- **Data Consistency and Integrity**: As we discussed earlier the root cause of data inconsistency is data redundancy, since data normalization takes care of the data redundancy, data inconsistency also been taken care of as part of it
- **Data Security**: It is easier to apply access constraints in database systems so that only authorized user is able to access the data. Each user has a different set of access thus data is secured from the issues such as identity theft, data leaks and misuse of data.
- **Privacy**: Limited access means privacy of data.
- **Easy access to data** – Database systems manages data in such a way so that the data is easily accessible with fast response times.

- **Easy recovery**: Since database systems keeps the backup of data, it is easier to do a full recovery of data in case of a failure.

- **Flexible**: Database systems are more flexible than file processing systems.

2.7 DISADVANTAGES OF DBMS

Here are some major disadvantages of DBMS

- DBMS implementation cost is high compared to the file system

- Complexity: Database systems are complex to understand

- Performance: Database systems are generic, making them suitable for various applications. However this feature affect their performance for some applications

2.8 CHECK YOUR PROGRESS

1. Which of the following is not a type of database management system?

(*a*) Hierarchical (*b*) Network

(*c*) Relational (*d*) Sequential.

2. A schema describes

(*a*) Data elements (*b*) Records and files

(*c*) Record relationship (*d*) All of the above.

3. Which data management language component enabled the DBA to define schema

components?

(*a*) DML (*b*) Subschema DLL

(*c*) Schema DLL (*d*) All of these.

4. Which statement is false regarding data independence?

(*a*) Hierarchical data model suffers from data independence.

(*b*) Network model suffers from data independence.

(*c*) Relational model suffers from logical data independence.

(*d*) Relational model suffers from physical data independence.

5. Databases may be more expensive to maintain than files because of

(*a*) backup and recovery needs

(*b*) the complexity of the database environment

(*c*) the need for specialized personnel

(*d*) all of the above.

6. Typically, a database consists _____ but can support mul _____.

(*a*) table, queries (*b*) information, data

(*c*) physical view, logical view (*d*) information view, data view.

2.9 SUMMARY

A DBMS is a software used to store and manage data. The DBMS was introduced during 1960's to store any data. It also offers manipulation of the data like insertion, deletion, and updating of the data. DBMS system also performs the functions like defining, creating, revising and controlling the database. It is specially designed to create and maintain data and enable the individual business application to extract the desired data. The main purpose of database systems is to manage the data. Consider a university that keeps the data of students, teachers, courses, books etc. To manage this data we need to store this data somewhere where we can add new data, delete unused data, update outdated data, retrieve data, and to perform these operations on data we need a Database management system that allows us to store the data in such a way so that all these operations can be performed on the data efficiently.

Simply put, a database system is a computerised record-keeping system with a lot of facilities. It is convenient to keep records and information in the form of computer

databases rather than in manual systems. Fig. 2.3 shows the three DBMS models; Hierarchical, Network and Relational.

The term model refers to the way data is organised in and accessible from DBMS. These three models differ in a number of ways, as we shall study later.
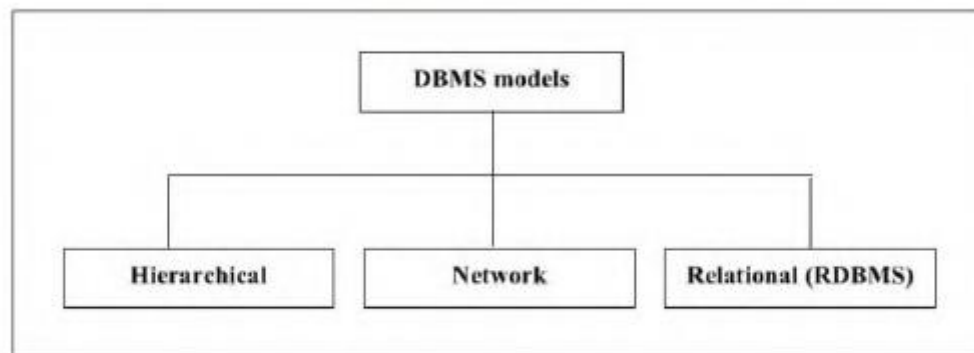

Figure 2.3: DBMS Models

We will concentrate on RDBMS and, as such, use an example from RDBMS to understand how users can access and manipulate databases. In RDBMS, data records (e.g. customer, student, book etc.) are stored on the hard disk by the operating system (O/S) such as UNIX and Windows 2000. The RDBMS interacts with the O/S and allows the user/programmer to view the records in the form of tables. Obviously, there is no such thing as a table on the hard disk.

2.10 KEYWORDS

- **DATA INTERGRITY-** Data integrity refers to the accuracy and consistency (validity) of data over its lifecycle. ... Data integrity can be compromised in several ways. Each time data is replicated or transferred, it should remain intact and unaltered between updates.
- **DATA REDUNDANCY-** Data redundancy is a condition created within a database or data storage technology in which the same piece of data is held in two separate

places. This can mean two different fields within a single database, or two different spots in multiple software environments or platforms.

- **ENDUSER-** An end user is the person that a software program or hardware device is designed for. The term is based on the idea that the "end goal" of a software or hardware product is to be useful to the consumer. The end user can be contrasted with the developers or programmers of the product.

- **DATA MODELS-** Data Models are fundamental entities to introduce abstraction in a DBMS. Data models define how data is connected to each other and how they are processed and stored inside the system. The very first data model could be flat data-models, where all the data used are to be kept in the same plane.

2.11 SELF-ASSESSMENT TEST

1. What is data?
2. What is Information?
3. What is the difference between data and information?
4. What is Metadata?
5. Explain various types of Metadata?
6. What is the difference between active and passive data dictionary?
7. What is data base?
8. What are the main characteristics of a database?
9. What are the capabilities of a database?

2.12 ANSWERS TO CHECK YOUR PROGRESS

1. D
2. D
3. D
4. D
5. D
6. C

2.13 REFERENCES / SUGGESTED READINGS

- C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.
- Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.
- Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.
- Thomas Connolly Carolyn Begg, "Database System", 3/2, Pearson Education.
- https://padakuu.com/article/237-characteristics-of-the-database-approach
- https://whatisdbms.com/characteristics-of-database-approach/
- https://ecomputernotes.com/fundamental/what-is-a-database/database-approach
- https://beginnersbook.com/2015/04/dbms-tutorial/

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	AUTHOR: DR. DEEPAK NANDAL
LESSON NO. 3	VETTER:
RESPONSIBILITY OF DATABASE ADMINISTRATOR & CLASSIFICATION OF DBMS	

3.0 LEARNING OBJECTIVE

- The objective of this chapter is to make the reader understand the responsibility of Database administrator and the meaning, as well as concept of centralized database and client server model of database. To know the about the classification of databases. To understand the interfaces and DBMS languages.

3.1 INTRODUCTION

In this chapter, we firstly focus on centralized as well as client server model of databases. The client-server model, or client-server architecture, is a distributed application framework dividing tasks between servers and clients, which either reside in the same system or communicate through a computer network or the Internet. The client relies on sending a request to another program in order to access a service made available by a server. The server runs one or more programs that share resources with and distribute work among clients. The client server relationship communicates in a request–response messaging pattern and must adhere to a common communications protocol, which formally defines the rules, language, and dialog patterns to be used. Client-server communication typically adheres to the TCP/IP protocol suite. TCP protocol maintains a connection until the client and server have completed the message exchange. TCP protocol determines the best way to distribute application data into packets that networks can deliver, transfers packets to and receives packets from the network, and manages flow control and retransmission of dropped or garbled packets. IP is a connectionless protocol in which each packet traveling through the Internet is an independent unit of data unrelated to any other data units.

Client requests are organized and prioritized in a scheduling system, which helps servers cope in the instance of receiving requests from many distinct clients in a short space of time. The client-server approach enables any general-purpose computer to expand its

capabilities by utilizing the shared resources of other hosts. Popular client-server applications include email, the World Wide Web, and network printing.

3.2 DEFINITION

Client- Server Database- Client-server denotes a relationship between cooperating programs in an application, composed of clients initiating requests for services and servers providing that function or service.

Centralized Database- A centralized database (sometimes abbreviated CDB) is a database that is located, stored, and maintained in a single location. This location is most often a central computer or database system, for example a desktop or server CPU, or a mainframe computer.

Distributed Database- A distributed database (DDB) is an integrated collection of databases that is physically distributed across sites in a computer network. A distributed database management system (DDBMS) is the software system that manages a distributed database such that the distribution aspects are transparent to the users.

Data Dictionary- A data dictionary is a centralized repository of metadata. Metadata is data about data. Some examples of what might be contained in an organization's data dictionary include: The names of fields contained in all of the organization's databases.

3.3 RESPONSIBILITY OF DATABASE ADMINISTRATOR

A database administrator's (DBA) primary job is to ensure that data is available, protected from loss and corruption, and easily accessible as needed. Below are some of the chief responsibilities that make up the day-to-day work of a DBA. DSP deliver an outsourced DBA service in the UK, providing Oracle Support and SQL Server Support; whilst mindset and toolset may be different, whether a database resides on premise or in a Public / Private Cloud, the role of the DBA is not that different.

1. Software installation and Maintenance

A DBA often collaborates on the initial installation and configuration of a new Oracle, SQL Server etc. database. The system administrator sets up hardware and deploys the operating system for the database server, then the DBA installs the database software and configures it for use. As updates and patches are required, the DBA handles this on-going maintenance.

And if a new server is needed, the DBA handles the transfer of data from the existing system to the new platform.

2. Data Extraction, Transformation, and Loading

Known as ETL, data extraction, transformation, and loading refers to efficiently importing large volumes of data that have been extracted from multiple systems into a data warehouse environment.

This external data is cleaned up and transformed to fit the desired format so that it can be imported into a central repository.

3. Specialized Data Handling

Today's databases can be massive and may contain unstructured data types such as images, documents, or sound and video files. Managing a very large database (VLDB) may require higher-level skills and additional monitoring and tuning to maintain efficiency.

4. Database Backup and Recovery

DBAs create backup and recovery plans and procedures based on industry best practices, then make sure that the necessary steps are followed. Backups cost time and money, so the DBA may have to persuade management to take necessary precautions to preserve data.

System admins or other personnel may actually create the backups, but it is the DBA's responsibility to make sure that everything is done on schedule.

In the case of a server failure or other form of data loss, the DBA will use existing backups to restore lost information to the system. Different types of failures may require different recovery strategies, and the DBA must be prepared for any eventuality. With technology change, it is becoming ever more typical for a DBA to backup databases to the cloud, Oracle Cloud for Oracle Databases and MS Azure for SQL Server.

5. Security

A DBA needs to know potential weaknesses of the database software and the company's overall system and work to minimise risks. No system is one hundred per cent immune to attacks, but implementing best practices can minimise risks.

In the case of a security breach or irregularity, the DBA can consult audit logs to see who has done what to the data. Audit trails are also important when working with regulated data.

6. Authentication

Setting up employee access is an important aspect of database security. DBAs control who has access and what type of access they are allowed. For instance, a user may have permission to see only certain pieces of information, or they may be denied the ability to make changes to the system.

7. Capacity Planning

The DBA needs to know how large the database currently is and how fast it is growing in order to make predictions about future needs. Storage refers to how much room the database takes up in server and backup space. Capacity refers to usage level.

If the company is growing quickly and adding many new users, the DBA will have to create the capacity to handle the extra workload.

8. Performance Monitoring

Monitoring databases for performance issues is part of the on-going system maintenance a DBA performs. If some part of the system is slowing down processing, the DBA may need to make configuration changes to the software or add additional hardware capacity. Many types of monitoring tools are available, and part of the DBA's job is to understand what they need to track to improve the system. 3rd party organisations can be ideal for outsourcing this aspect, but make sure they offer modern DBA support.

9. Database Tuning

Performance monitoring shows where the database should be tweaked to operate as efficiently as possible. The physical configuration, the way the database is indexed, and how queries are handled can all have a dramatic effect on database performance.

With effective monitoring, it is possible to proactively tune a system based on application and usage instead of waiting until a problem develops.

10. Troubleshooting

DBAs are on call for troubleshooting in case of any problems. Whether they need to quickly restore lost data or correct an issue to minimize damage, a DBA needs to quickly understand and respond to problems when they occur.

3.4 CLASSIFICATION OF DATABASE SYSTEMS

A Database Management System or DBMS is a single or set of computer programs that are responsible for creating, editing, deleting and generally maintaining a database or collection of data records. They type of database management system is determined by the database model. A database model is the manner in which the data collection is stored, managed and administered. The various database management systems based on these data models are:

3.4.1 Relational Database Management Systems

Relational database management systems are the most widely used database management systems today. They are relatively easy to use. Relational database management systems are named so because of the characteristic of normalizing the data which is usually stored in tables. The relational model relies on normalizing data within rows and columns in tables. The data can be related to other data in the same table or other tables which has to be correctly managed by joining one or more tables. Relational models may be somewhat less efficient than other models; however this may not be a problem with the processing power and memory found in modern computers. Data in this type of model is stored is fixed predefined structures and are usually manipulated using Structured Query Language (SQL). Relational database management systems include Oracle, Ms SQLServer, IBM DB2, mySQL, SQLite and PostgreSQL among others.

3.4.2 Flat File Based Database Management Systems

Flat File based database management systems are probably the simplest of them all. These are sometimes called Flat models. These come in human readable text formats as well as in binary formats. These are ideal for stand alone applications, holding software configuration and native format storage models. Flat files in a formatted row and column model rely on assumptions that every item in a particular model consists of the same data.

One common example of this type of database is the CSV (Comma Separated Values) and another is a spreadsheet such as Ms Excel.

3.4.3   Hierarchical Database Management Systems

Hierarchical database management systems operates on the parent child tree-like model. These normally have a 1:N relationship and are good for storing data with items describing attributes, features and so on. These could store a book with information on chapters and verses. They can also be used to store a database of songs, recipes, models of phones and anything that can be stored in a nested format. Hierarchical database management systems are not quite efficient for various real world operations. One such example of a Hierarchical database management system is a XML document.

3.4.4   Network Database Management Systems

A Network database management system uses a data model similar to Hierarchical database management systems. The major difference here is that the tree structure in the Network models can have a many parent to many child relational model. The Network model structure is based on records and sets and most of these databases use SQL for manipulation of their data. Network database management systems tend to be very flexible but are rarely used and was very quiet common in the1960s and 1970s. Searching for an item in this model requires the program to traverse the entire data set which is quit cumbersome. These have mainly been replaced by Relational database management systems in today's modern computing.

3.4.5   Object-oriented Database Management Systems

Object-oriented database management systems borrow from the model of the Object-oriented programming paradigm. In this database model, the Object and its data or attributes are seen as one ad accessed through pointers rather than stored in relational table

models. Object-oriented database models consist of diverse structures and is quite extensible. This data model was designed to work closely with programs built with Object-oriented programming languages thereby almost making the data and the program operate as one. With this model applications are able to treat the data as native code. There is little commercial implementation of this database model as it is still developing. Examples of Object-oriented database management systems include IBM DB4 and DTS/S1 from Obsidian Dynamics.

3.5 DBMS LANGUAGES AND INTERFACES

The DBMS must provide appropriate languages and interfaces for each category of users. In this section we discuss the types of languages and interfaces provided by a DBMS and the user categories targeted by each interface.

3.5.1   DBMS Languages

Once the design of a database is completed and a DBMS is chosen to implement the database, the first step is to specify conceptual and internal schemas for the database and any mappings between the two. In many DBMSs where no strict separation of levels is maintained, one language, called the data definition language (DDL), is used by the DBA and by database designers to define both schemas. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog. In DBMSs where a clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only.Another language, the storage definition language (SDL), is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages. For a true three-schema architecture, we would need a third language, the view definition language (VDL), to specify user views and their mappings to the conceptual schema, but in most DBMSs *the DDL is used to define*

both conceptual and external schemas Once the database schemas are compiled and the database is populated with data, users must have some means to manipulate the database. Typical manipulations include retrieval, insertion, deletion, and modification of the data. The DBMS provides a set of operations or a language called the data manipulation language (DML) for these purposes. There are two main types of DMLs. A high-level or nonprocedural DML can be used on its own to specify complex database operations concisely. Many DBMSs allow high-level DML statements either to be entered interactively from a display monitor or terminal or to be embedded in a general-purpose programming language. In the latter case, DML statements must be identified within the program so that they can be extracted by a precompile and processed by the DBMS. A low level or procedural DML *must* be embedded in a general-purpose programming language. This type of DML typically retrieves individual records or objects from the database and processes each separately. Therefore, it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records. Whenever DML commands, whether high level or low level, are embedded in a general-purpose programming language, that language is called the host language and the DML is called the data sublanguage. On the other hand, a high-level DML used in a standalone interactive manner is called a query language. In general, both retrieval and update commands of a high-level DML may be used interactively and are hence considered part of the query language.

3.5.2 DBMS Interfaces

User-friendly interfaces provided by a DBMS may include the following:

- **Menu-Based Interfaces for Web Clients or Browsing.** These interfaces present the user with lists of options (called **menus)** that lead the user through the formulation of a request. Menus do away with the need to memorize the

specific commands and syntax of a query language; rather, the query is composed step-by-step by picking options from a menu that is displayed by the system. Pull-down menus are a very popular technique in Web-based user interfaces. They are also often used in browsing interfaces, which allow a user to look through the contents of a database in an exploratory and unstructured manner.

- **Forms-Based Interfaces.** A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they can fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries. Forms are usually designed and programmed for naive users as interfaces to canned transactions. Many DBMSs have forms specification languages.

- **Graphical User Interfaces.** A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms. Most GUIs use a pointing device, such as a mouse, to select certain parts of the displayed schema diagram.

- **Natural Language Interfaces.** These interfaces accept requests written in English or some other language and attempt to *understand* them. A natural language interface usually has its own *schema*, which is similar to the database conceptual schema, as well as a dictionary of important words.

- **Speech Input and Output.** Limited use of speech as an input query and speech as an answer to a question or result of a request is becoming commonplace. Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure, and credit card account information are allowing speech

for input and output to enable customers to access this information. The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech takes place.

- **Interfaces for Parametric Users.** Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. For example, a teller is able to use single function keys to invoke routine and repetitive transactions such as account deposits or withdrawals, or balance inquiries.

- **Interfaces for the DBA.** Most database systems contain privileged commands that can be used only by the DBA staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

3.6 CLIENT-SERVER MODEL & CENTRALIZED MODEL OF DBMS

Client-Server Model-

The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client. Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc.

There are four main categories of client-server computing:

- One-Tier architecture: consists of a simple program running on a single computer without requiring access to the network. User requests don't manage any network

protocols, therefore the code is simple and the network is relieved of the extra traffic.

- Two-Tier architecture: consists of the client, the server, and the protocol that links the two tiers. The Graphical User Interface code resides on the client host and the domain logic resides on the server host. The client-server GUI is written in high-level languages such as C++ and Java.

- Three-Tier architecture: consists of a presentation tier, which is the User Interface layer, the application tier, which is the service layer that performs detailed processing, and the data tier, which consists of a database server that stores information.

- N-Tier architecture: divides an application into logical layers, which separate responsibilities and manage dependencies, and physical tiers, which run on separate machines, improve scalability, and add latency from the additional network communication. N-Tier architecture can be closed-layer, in which a layer can only communicate with the next layer down, or open-layer, in which a layer can communicate with any layers below it.

Here is a quick comparison between client-server and distributed system.

Sr No.	Client- Server	Distributed DBMS
1.	Client can access only one server at a time.	User can access many sites simultaneously.
2.	It is difficult to manage.	It is easy to manage.
3.	In this data is distributed across clients.	In this data is distributed across sites.

4.	Speed of accessing database is poor as compared to Distributed DBMS.	Speed of accessing database is much better than Client/Server Architecture.
5.	If somehow server crashes, the whole system stops.	The crash of one site does not stop the entire system.
6.	Accessing of data is easy to control.	Accessing of data is difficult to control.
7.	It is less expensive as compared to Distributed DBMS.	It is expensive.
8.	Maintenance cost is low.	Maintenance cost is high.

Centralized System-

A centralized database is stored at a single location such as a mainframe computer. It is maintained and modified from that location only and usually accessed using an internet connection such as a LAN or WAN. The centralized database is used by organizations such as colleges, companies, banks etc. As can be seen from the figure 3.1, all the information for the organization is stored in a single database and known as the centralized database.
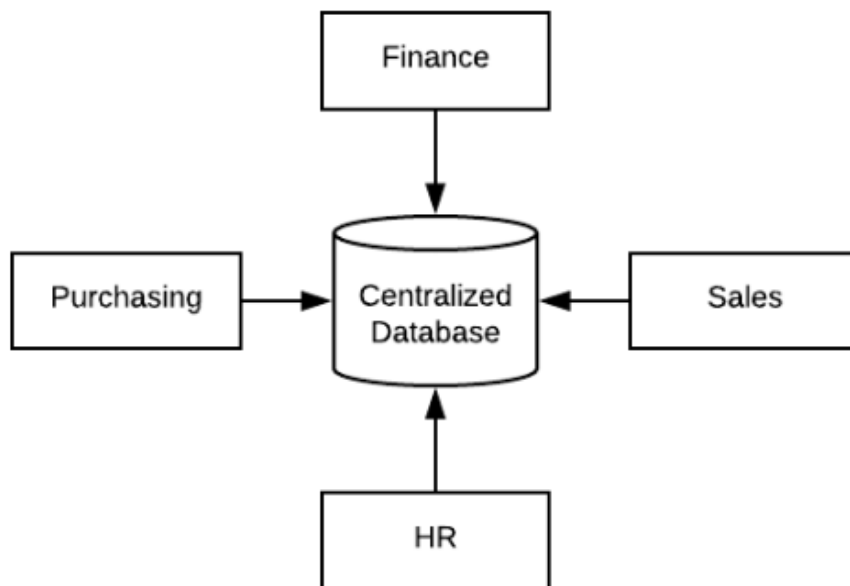

Figure 3.1: Centralized Model

Advantages

Some advantages of Centralized Database Management System are −

- The data integrity is maximized as the whole database is stored at a single physical location. This means that it is easier to coordinate the data and it is as accurate and consistent as possible.

- The data redundancy is minimal in the centralized database. All the data is stored together and not scattered across different locations. So, it is easier to make sure there is no redundant data available.

- Since all the data is in one place, there can be stronger security measures around it. So, the centralized database is much more secure.

- Data is easily portable because it is stored at the same place.

- The centralized database is cheaper than other types of databases as it requires less power and maintenance.

- All the information in the centralized database can be easily accessed from the same location and at the same time.

Disadvantages

Some disadvantages of Centralized Database Management System are −

- Since all the data is at one location, it takes more time to search and access it. If the network is slow, this process takes even more time.

- There is a lot of data access traffic for the centralized database. This may create a bottleneck situation.

- Since all the data is at the same location, if multiple users try to access it simultaneously it creates a problem. This may reduce the efficiency of the system.

- If there are no database recovery measures in place and a system failure occurs, then all the data in the database will be destroyed.

3.7 DATA DICTIONARY

A fundamental property of a database system is that it maintains a description of all the data that it contains. A relational DBMS maintains information about every relation and index that it contains. The DBMS also maintains information about views, for which no tuples are stored explicitly; rather, a definition of the view is stored and used to compute the tuples that belong in the view when the view is queried. This information is stored in a collection of relations, maintained by the system, called the catalog relations. The catalog relations are also called the system catalog, the *catalog*, or the data dictionary. The system catalog is sometimes referred to as metadata; that is, not data, but descriptive information about the data. The information in the system catalog is used extensively for query optimization. Many organizations now use data dictionary systems or information repositories, which are mini DBMSs that manage meta-data—that is, data that describes the database structure, constraints, applications, authorizations, users, and so on. These are often used as an integral tool for information resource management. A useful data dictionary system should store and manage the following types of information:

a) Descriptions of the schemas of the database system.

b) Detailed information on physical database design, such as storage structures, access paths, and file and record sizes.

c) Descriptions of the types of database users, their responsibilities, and their access rights.

d) High-level descriptions of the database transactions and applications and of the relationships of users to transactions.

e) The relationship between database transactions and the data items referenced by them. This is useful in determining which transactions are affected when certain data definitions are changed.

f) Usage statistics such as frequencies of queries and transactions and access counts to different portions of the database.

g) The history of any changes made to the database and applications, and documentation that describe the reasons for these changes. This is sometimes referred to as data provenance. This meta-data is available to DBAs, designers, and authorized users as online system documentation. This improves the control of DBAs over the information system as well as the users' understanding and use of the system.

3.8 CHECK YOUR PROGRESS

1. The database system which supports the majority of concurrent users is classified as _____.
2. The objects in DBMS belongs to same structure and behaves in the same way are considered as _____.
3. The type of legacy data model in which data is represented as record types and limited one to many relationships is called_____.
4. The DBMS in which the system involved are coupled together while having local autonomy is classified as _____.
5. The same class objects are arranged and organized in a way called_____.

3.9 SUMMARY

Major benefits of Client-server model of DBMS

- A single server hosting all the required data in a single place facilitates easy protection of data and management of user authorization and authentication.
- Resources such as network segments, servers, and computers can be added to a client-server network without any significant interruptions.

- Data can be accessed efficiently without requiring clients and the server to be in close proximity.
- All nodes in the client-server system are independent, requesting data only from the server, which facilitates easy upgrades, replacements, and relocation of the nodes.
- Data that is transferred through client-server protocols are platform-agnostic.

A Database can be in general defined as a collection of data that is organized efficiently so that the data can be retrieved and stored easily. A Database which is located and stored in a single location is called a Centralised Database. The centralized database's location is generally a server CPU or desktop or the mainframe computer which is accessed by the users through a computer network like LAN or WAN.

An organization may have several business processes running for various departments simultaneously. This may create issues if the organization wants to check on the data daily and the centralized database comes in handy in such cases. It is important for an organization to take decisions and without the presence of a centralized database, it becomes difficult. Because the organizations though have separate databases for different departments, they still need to maintain a centralized database where these separate databases are united to form a single database in order to provide the overall view of the complete data. For example, if the organization wants to find the details about a particular employee, they just need to access the centralized database to get all the details about the employee. So a centralized database not only helps in getting the information quicker with more ease but also helps in taking the business decisions as well. The usage of the Centralized Database ensures the security of the data, ease of accessing data from one place as well as providing a complete view of the data effectively reducing the extra layer of information.

There are several criteria based on which DBMS is classified. The classification and types of Database Management System (DBMS) is explained in a detailed manner based on the different factors.

- Relational Database
- Object Oriented Database
- Object Relational Database
- Hierarchal Database

3.10 KEYWORDS

- **Relational Databases-** A relational database is a collection of data items with pre-defined relationships between them. These items are organized as a set of tables with columns and rows. Tables are used to hold information about the objects to be represented in the database.

- **Object Oriented Databases-** An object database is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object databases are different from relational databases which are table-oriented. Object-relational databases are a hybrid of both approaches.

- **Network Databases-** A network database is a type of database model wherein multiple member records or files can be linked to multiple owner files and vice versa.

- **Hierarchical Databases-** In hierarchical model, data is organized into a tree like structure with each record is having one parent record and many children. The main drawback of this model is that, it can have only one to many relationships between nodes.

3.11 SELF-ASSESSMENT TEST

1. What do you understand by data redundancy?
2. Discuss the classification of database in detail. Giving an example of each type.
3. What are the various type's relationships in database? Define them.
4. Explain the data dictionary.
5. What is the importance of database management system interfaces?
6. How RDBMS stores its data?

3.12 ANSWERS TO CHECK YOUR PROGRESS

1. Multiuser System
2. Same class objects
3. Network model
4. Federated DBMS
5. Acyclic graphs and hierarchies

3.13 REFERENCES / SUGGESTED READINGS

*   C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.
*   Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.
*   Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.
*   Thomas Connolly Carolyn Begg, "Database System", 3/2, Pearson Education.
*   https://www.tutorialspoint.com/Centralized-Database-Management-System
*   https://www.omnisci.com/technical-glossary/client-server
*   https://www.geeksforgeeks.org/difference-between-client-server-and-distributed-dbms/

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	AUTHOR: DR. DEEPAK NANDAL
LESSON NO. 4	VETTER:
DATABASE SYSTEM ARCHITECTURE & DATA MODELS	

STRUCTURE

4.0 LEARNING OBJECTIVE

6. The objective of this chapter is to make the reader understand the architecture details of database management system, with three layer of architecture such as external, internal and conceptual. To know the various applications of database and types of database and to provide detailed view of the various data models that are available in DBMS.

4.1 INTRODUCTION

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent n modules, which can be independently modified, altered, changed, or replaced.

In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture. If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.

A 3-tier architecture as shown in figure 4.1 separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS. Multiple-tier database architecture is highly

modifiable, as almost all its components are independent and can be changed independently.

- **Database (Data) Tier** −At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.

- **Application (Middle) Tier** −At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.

- **User (Presentation) Tier** − End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier

4.2 DEFINITION

Database architecture focuses on the design, development, implementation and maintenance of computer programs that store and organize information for businesses, agencies and institutions. A database architect develops and implements software to meet the needs of users. The design of a DBMS depends on its architecture.

The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks. The client/server architecture consists of many PCs and a workstation which are connected via the network.
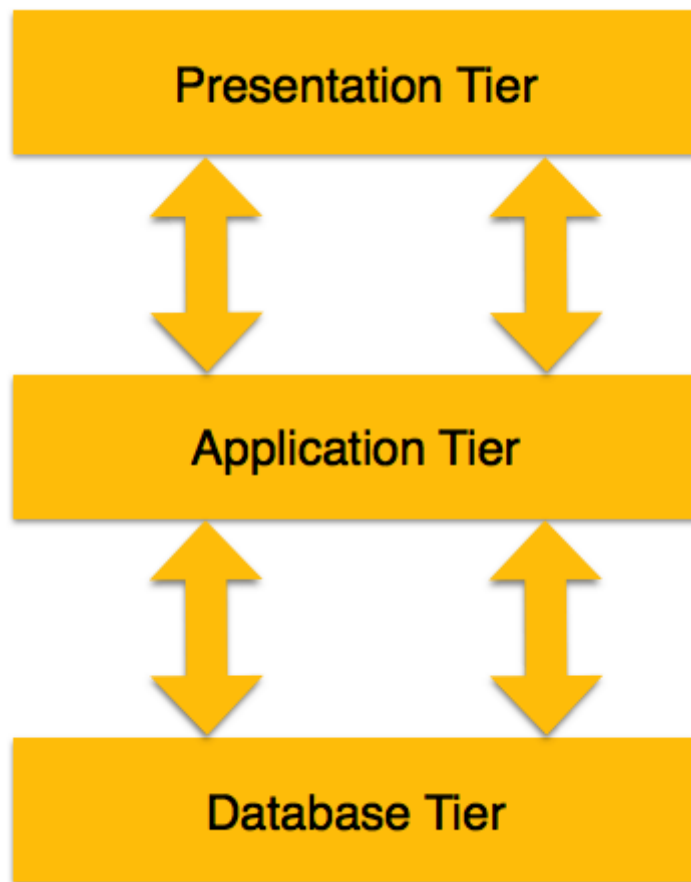
Figure 4.1: Outline of 3 tire architecture

A **data model**—a collection of concepts that can be used to describe the structure of a database—provides the necessary means to achieve this abstraction. By *structure of a database* we mean the data types, relationships, and constraints that apply to the data. Most data models also include a set of **basic operations** for specifying retrievals and updates on the database.

4.3 THREE LEVEL OF ARCHITECTURE

4.3.1 Single Level-

In this type of architecture, the database is readily available on the client machine, any request made by client doesn't require a network connection to perform the action on the database.

For example, let's say you want to fetch the records of employee from the database and the database is available on your computer system, so the request to fetch employee details will be done by your computer and the records will be fetched from the database by your computer as well. This type of system is generally referred as local database system.

4.3.2 Two Level-

In two-tier architecture, the Database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a reliable network as shown in the above diagram.

Whenever client machine makes a request to access the database present at server using a query language like SQL, the server perform the request on the database and returns the result back to the client. The application connection interface such as JDBC, ODBC are used for the interaction between server and client.

4.3.3 Three Level-

DBMS uses three-tier architecture to help achieve and visualize the characteristics discussed previously. The goal of the three-schema architecture, illustrated in Figure below, is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels as shown in figure 4.2:

1. The **internal level** has an **internal schema**, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

2. The **conceptual level** has a **conceptual schema**, which describes the structure of the whole database for a community of users. The conceptual schema hides the details

of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This *implementation conceptual schema* is often based on a *conceptual schema design* in a high-level data model.

3. The **external** or **view level** includes a number of **external schemas** or **user views**. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.
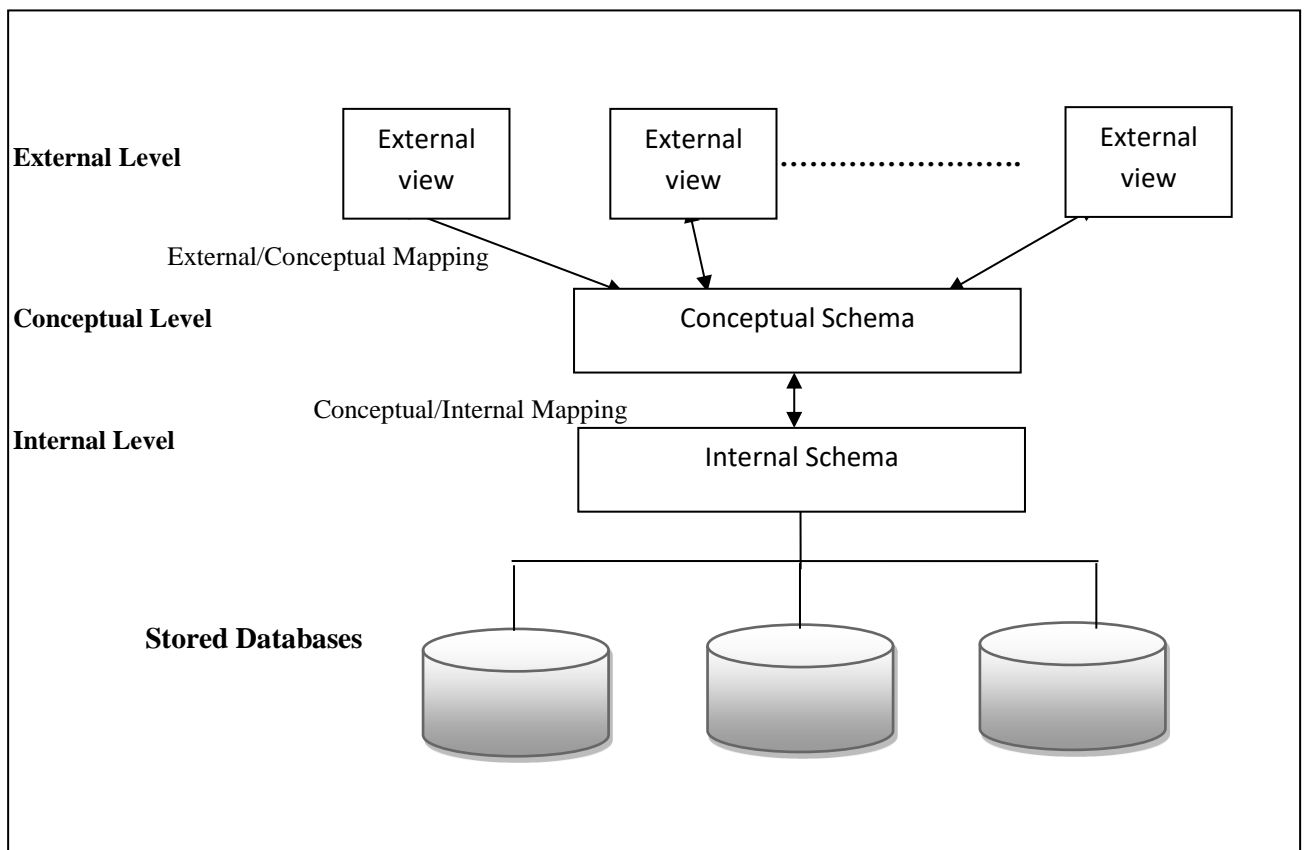

Figure 4.2: Three-Level Schema Architecture

Different Mappings in Three level Architecture of DBMS

The process of transforming requests and results between the three levels are called mappings. The database management system is responsible for this mapping between internal, external and conceptual schemas.

There are two types of mappings:

      1. Conceptual/Internal mapping.

      2. The External/Conceptual mapping.

1. The Conceptual/Internal Mapping: This mapping defines the correspondence or operations between the conceptual view and the physical view. It specifies how the data is retrieved from physical storage and shown at conceptual level and vice-versa. It specifies how conceptual records and fields are represented at the internal level. It also allows any differences in entity names, attribute names and their orders, data types etc., to be resolved.

2. The External/Conceptual Mapping: This mapping defines the correspondence between the conceptual view and the physical view. It specifies how the data is retrieved from conceptual level and shown at external level because at external level some part of database is hidden from a particular user and even names of data fields are changed etc. There could be one mapping between conceptual and internal level and several mappings between external and conceptual level. The physical data independence is achieved through conceptual/internal mapping while the logical data independence is achieved through external/ conceptual mapping. The information about the mapping requests among various schema levels are included in the system catalog of DBMS. When schema is changed at some level, the schema at the next higher level remains unchanged, only the mapping between the two levels is changed.

4.4 PHASES OF DATABASE DESIGN

Database designing for a real-world application starts from capturing the requirements to physical implementation using DBMS software which consists of following steps shown below in figure 5.3:

Conceptual Design: The requirements of database are captured using high level conceptual data model. For Example, the ER model is used for the conceptual design of the database.

Logical Design: Logical Design represents data in the form of relational model. ER diagram produced in the conceptual design phase is used to convert the data into the Relational Model.

Physical Design: In physical design, data in relational model is implemented using commercial DBMS like Oracle, DB2.
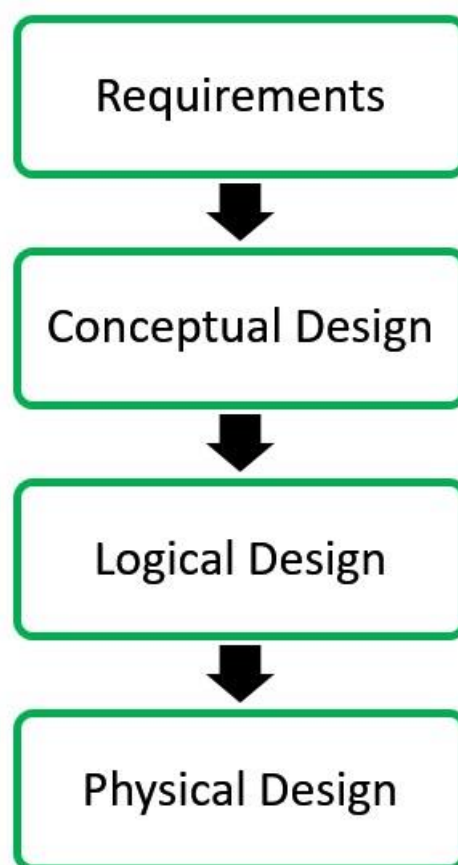

Figure 4.3: Phases of Database Design

4.5 APPLICATIONS OF DBMS

Database is a collection of related data and data is a collection of facts and figures that can be processed to produce information. Mostly data represents recordable facts. Data aids in producing information, which is based on facts. For example, if we have data about marks obtained by all students, we can then conclude about toppers and average marks. A database management system stores data in such a way that it becomes easier to retrieve, manipulate, and produce information. Following are the important characteristics and applications of DBMS.

- ACID Properties − DBMS follows the concepts of Atomicity, Consistency, Isolation, and Durability (normally shortened as ACID). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.

- Multiuser and Concurrent Access − DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.

- Multiple views − DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.

- Security − Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of

security features, which enables multiple users to have different views with different features. For example, a user in the Sales department cannot see the data that belongs to the Purchase department. Additionally, it can also be managed how much data of the Sales department should be displayed to the user. Since a DBMS is not saved on the disk as traditional file systems, it is very hard for miscreants to break the code.

5.6 DATA MODELING CONCEPT

Implementation data models that are closer to conceptual data models. A standard for object databases called the ODMG object model has been proposed by the Object Data Management Group Physical data models describe how data is stored as files in the computer by representing

Information such as record formats, record orderings, and access paths. An **access path** is a structure that makes the search for particular database records efficient. An **index** is an example of an access path that allows direct access to data using an index term or a keyword. It is similar to the index at the end of this book, except that it may be organized in a linear, hierarchical (tree-structured), or some other fashion

A data model organizes data (link is external) elements and standardizes how the data elements relate to one another. Since data elements document real life (link is external) people, places and things and the events between them, the data model represents reality. For example a house has many windows or a cat has two eyes. Data models are often used as an aid to communication between the business people defining the requirements (link is external) for a computer system (link is external) and the technical people defining the design in response to those requirements. They are used to show the data needed and created by business processes (link is external). A data model explicitly determines the structure of

data. Data models are specified in a data modeling (link is external) notation, which is often graphical in form. (Link is external)

A data model can be sometimes referred to as a data structure (link is external), especially in the context of programming languages (link is external). Data models are often complemented by function models (link is external).

The primary goal of using data model are:

- Ensures that all data objects required by the database are accurately represented. Omission of data will lead to creation of faulty reports and produce incorrect results.

- A data model helps design the database at the conceptual, physical and logical levels.

- Data Model structure helps to define the relational tables, primary and foreign keys and stored procedures.

- It provides a clear picture of the base data and can be used by database developers to create a physical database.

- It is also helpful to identify missing and redundant data.

- Though the initial creation of data model is labor and time consuming, in the long run, it makes your IT infrastructure upgrade and maintenance cheaper and faster.

Types of Data Model:

There are mainly three different types of data models: conceptual data models, logical data models, and physical data models, and each one has a specific purpose. The data models are used to represent the data and how it is stored in the database and to set the relationship between data items.

1. **Conceptual or Object Data Model:** This Data Model defines **WHAT** the system contains. This model is typically created by Business stakeholders and Data Architects. The purpose is to organize, scope and define business concepts and rules.

2. **Logical Data Model:** Defines **HOW** the system should be implemented regardless of the DBMS. This model is typically created by Data Architects and Business Analysts. The purpose is to developed technical map of rules and data structures.

3. **Physical Data Model**: This Data Model describes **HOW** the system will be implemented using a specific DBMS system. This model is typically created by DBA and developers. The purpose is actual implementation of the database.

4.7 OBJECT BASED DATA MODEL

This data model is another method of representing real world objects. It considers each object in the world as objects and isolates it from each other. It groups its related functionalities together and allows inheriting its functionality to other related sub-groups. Let us consider an Employee database to understand this model better. In this database we have different types of employees – Engineer, Accountant, Manager, Clark. But all these employees belong to Person group. Person can have different attributes like name, address, age and phone. What do we do if we want to get a person's address and phone number? We write two separate procedure sp_getAddress and sp_getPhone.

What about all the employees above? They too have all the attributes what a person has. In addition, they have their EMPLOYEE_ID, EMPLOYEE_TYPE and DEPARTMENT_ID attributes to identify them in the organization and their department. We have to retrieve their department details, and hence we sp_getDeptDetails procedure.

68

Currently, say we need to have only these attributes and functionality. Since all employees inherit the attributes and functionalities of Person, we can re-use those features in Employee. But do we do that? We group the features of person together into class. Hence a class has all the attributes and functionalities. For example, we would create a person class and it will have name, address, age and phone as its attribute, and sp_getAddress and sp_getPhone as procedures in it. The values for these attributes at any instance of time are object. i.e.; {John, Troy, 25, 2453545 : sp_getAddress (John), sp_getPhone (John)} forms on person object. {Mathew, Fraser Town, 28, 5645677: sp_getAddress (Mathew), sp_getPhone (Mathew} forms another person object.

Now, we will create another class called Employee which will inherit all the functionalities of Person class. In addition it will have attributes EMPLOYEE_ID, EMPLOYEE_TYPE and DEPARTMENT_ID, and sp_getDeptDetails procedure. Different objects of Employee class are Engineer, Accountant, Manager and Clerk.

Figure 4.4: Object Data Model Example

Here in the figure 4.4 we can observe that the features of Person are available only if other class is inherited from it. It would be a black box to any other classes. This feature of this model is called encapsulation. It binds the features in one class and hides it from other classes. It is only visible to its objects and any inherited classes.

4.8 PHYSICAL DATA MODEL

A **Physical Data Model** describes a database-specific implementation of the data model. It offers database abstraction and helps generate the schema. This is because of the richness of meta-data offered by a Physical Data Model. The physical data model also helps in visualizing database structure by replicating database column keys, constraints, indexes, triggers, and other RDBMS features as shown in figure 4.5.
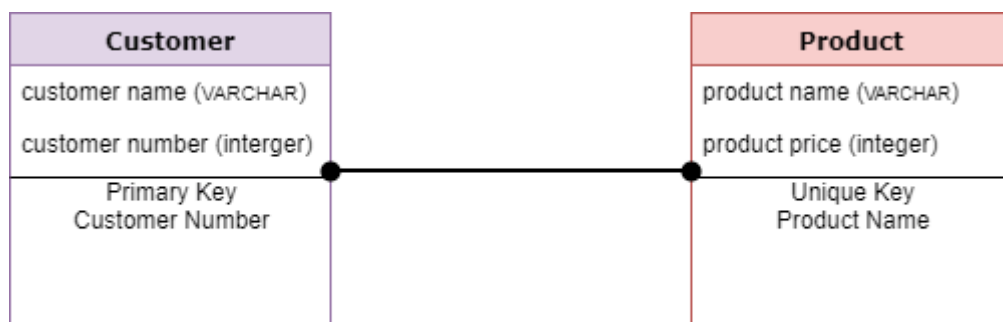

Figure 4.5: Physical data model example

Main Characteristics of a physical data model:

- The physical data model describes data need for a single project or application though it may be integrated with other physical data models based on project scope.
- Data Model contains relationships between tables that which addresses cardinality and null ability of the relationships.
- Developed for a specific version of a DBMS, location, data storage or technology to be used in the project.
- Columns should have exact datatypes, lengths assigned and default values.

- Primary and Foreign keys, views, indexes, access profiles, and authorizations, etc. are defined.

4.9 LOGICAL DATA MODEL

The Logical Data Model is used to define the structure of data elements and to set relationships between them. The logical data model adds further information to the conceptual data model elements. The advantage of using a Logical data model is to provide a foundation to form the base for the Physical model. However, the modeling structure remains generic as shown in figure 4.6.
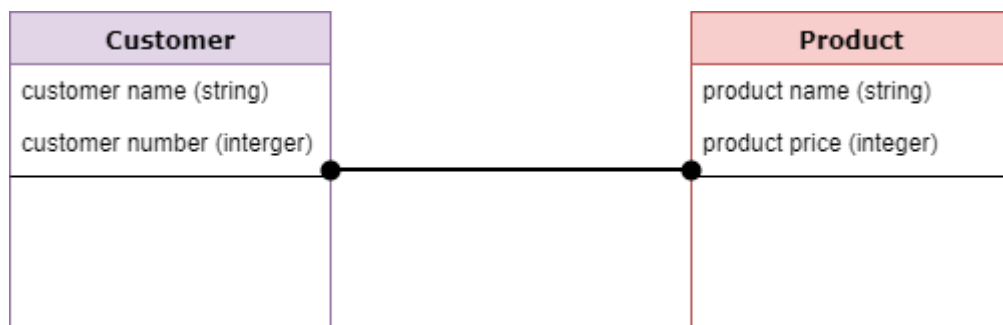

Figure 4.6: Logical data model example

4.10 CHECK YOUR PROGRESS

1. The level of data abstraction which describes how the data is actually stored is _____.
2. Collection of information stored in a database at a particular moment is _____.
3. In a hierarchical database, a hashing function is used to locate the _____.
4. The syntax of a user query is verified by _____.
5. The application server in a three-tier architecture, communicates with a database system to access _____.

4.11 SUMMARY

The three levels or views are discussed below:

(*i*) **Internal Level:** Internal level describes the actual physical storage of data or the way in which the data is actually stored in memory. This level is not relational because data is stored according to various coding schemes instead of tabular form (in tables). This is the low level representation of entire database. The internal view is described by means of an internal schema. The internal level is concerned with the following aspects:

– Storage space allocation

– Access paths

– Data compression and encryption techniques

– Record placement etc.

The internal level provides coverage to the data structures and file organizations used to store data on storage devices.

(*ii*) **Conceptual Level:** The conceptual level is also known as logical level which describes the overall logical structure of whole database for a community of users. This level is relational because data visible at this level will be relational tables and operators will be relational operators. This level represents entire contents of the database in an abstract form in comparison with physical level. Here conceptual schema is defined which hides the actual physical storage and concentrate on relational model of database.

(*iii*) **External Level:** The external level is concerned with individual users. This level describes the actual view of data seen by individual users. The external schema is defined by the DBA for every user. The remaining part of database is hidden from that user. This means user can only access data of its own interest. In other words, user can access only that part of database for which he is authorized by DBA. This level is also relational or very close to it.

4.12 KEYWORDS

- **CONCURRENCY CONTROL -** Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each another. Concurrent access is quite easy if all users are just reading data.

- **ACID-** The presence of four components — atomicity, consistency, isolation and durability — can ensure that a database transaction is completed in a timely manner. When databases possess these components, they are said to be ACID-compliant.

- **DATA ARCHITECT-** Data architects define how the data will be stored, consumed, integrated and managed by different data entities and IT systems, as well as any applications using or processing that data in some way.

- **SCHEMA-** The term "schema" refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases). The formal definition of a database schema is a set of formulas (sentences) called integrity constraints imposed on a database.

- **INSTANCE-** A database instance is a set of memory structures that manage database files. A database is a set of physical files on disk created by the CREATE DATABASE statement. The instance manages its associated data and serves the users of the database.

4.13 SELF-ASSESSMENT TEST

1. Explain the interfaces used for DBMS. Also discuss any special type of hardware/software requirements for using these interfaces?
2. How many types of architectures are there when we talk about databases?
3. What is n-types of DBMS architecture?
4. Discuss different types of applications of database systems.
5. Write short note the phases of database design.

4.14 ANSWERS TO CHECK YOUR PROGRESS

1. Physical level
2. Instance
3. Root
4. Parser

5. Data

4.15 REFERENCES / SUGGESTED READINGS

- C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.
- Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.
- Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.
- Thomas Connolly Carolyn Begg, "Database System", 3/2, Pearson Education.
- https://www.geeksforgeeks.org/introduction-of-3-tier-architecture-in-dbms-set-2/
- https://beginnersbook.com/2018/11/dbms-architecture/
- https://medium.com/oceanize-geeks/concepts-of-database-architecture-dfdc558a93e4
- https://www.tutorialspoint.com/dbms/index.htm

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	AUTHOR: DR. DEEPAK NANDAL
LESSON NO. 5	VETTER:
SCHEMAS AND DATA INDEPENDENCE	

STRUCTURE

5.0 LEARNING OBJECTIVE

- The objective of this chapter is to make the reader understand the DBMS schemas and how mapping of instances is done. To know the data independence in detail. To study types of data independence in databases.

5.1 INTRODUCTION

In the chapter we will firstly focus on the data Independence is defined as a property of DBMS that helps you to change the Database schema at one level of a database system without requiring to change the schema at the next higher level. Data independence helps you to keep data separated from all programs that make use of it. You can use this stored data for computing and presentation. In many systems, data independence is an essential function for components of the system. Database systems comprise of complex data structures. Thus, to make the system efficient for retrieval of data and reduce the complexity of the users, developers use the method of Data Abstraction.

There are mainly three levels of data abstraction:

1. Internal Level: Actual PHYSICAL storage structure and access paths.
2. Conceptual or Logical Level: Structure and constraints for the entire database
3. External or View level: Describes various user views

The term "database schema" can refer to a visual representation of a database, a set of rules that govern a database, or to the entire set of objects belonging to a particular user. A database schema represents the logical configuration of all or part of a relational database. It can exist both as a visual representation and as a set of formulas known as integrity constraints that govern a database. These formulas are expressed in a data definition language, such as SQL. As part of a data dictionary, a database schema indicates how the

entities that make up the database relate to one another, including tables, views, stored procedures, and more.

Typically, a database designer creates a database schema to help programmers whose software will interact with the database. The process of creating a database schema is called data modelling. When following the three-schema approach to database design, this step would follow the creation of a conceptual schema. Conceptual schemas focus on an organization's informational needs rather than the structure of a database.

There are two main kinds of database schema:

- A logical database schema conveys the logical constraints that apply to the stored data. It may define integrity constraints, views, and tables.
- A physical database schema lays out how data is stored physically on a storage system in terms of files and indices.

5.2 DEFINITION

Data Independence- Data Independence is defined as a property of DBMS that helps you to change the Database schema at one level of a database system without requiring to change the schema at the next higher level. Data independence helps you to keep data separated from all programs that make use of it.

Schema- The term "schema" refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases). The formal definition of a database schema is a set of formulas (sentences) called integrity constraints imposed on a database.

Sub schema- The subschema is the logical description of that section of the database which is relevant and available to an application. A subschema can, of course, be common to two or more different applications.

Instances- A database instance is a set of memory structures that manage database files. A database is a set of physical files on disk created by the CREATE DATABASE statement. The instance manages its associated data and serves the users of the database.

5.3 SCHEMAS

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema as shown in figure 5.1 defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.
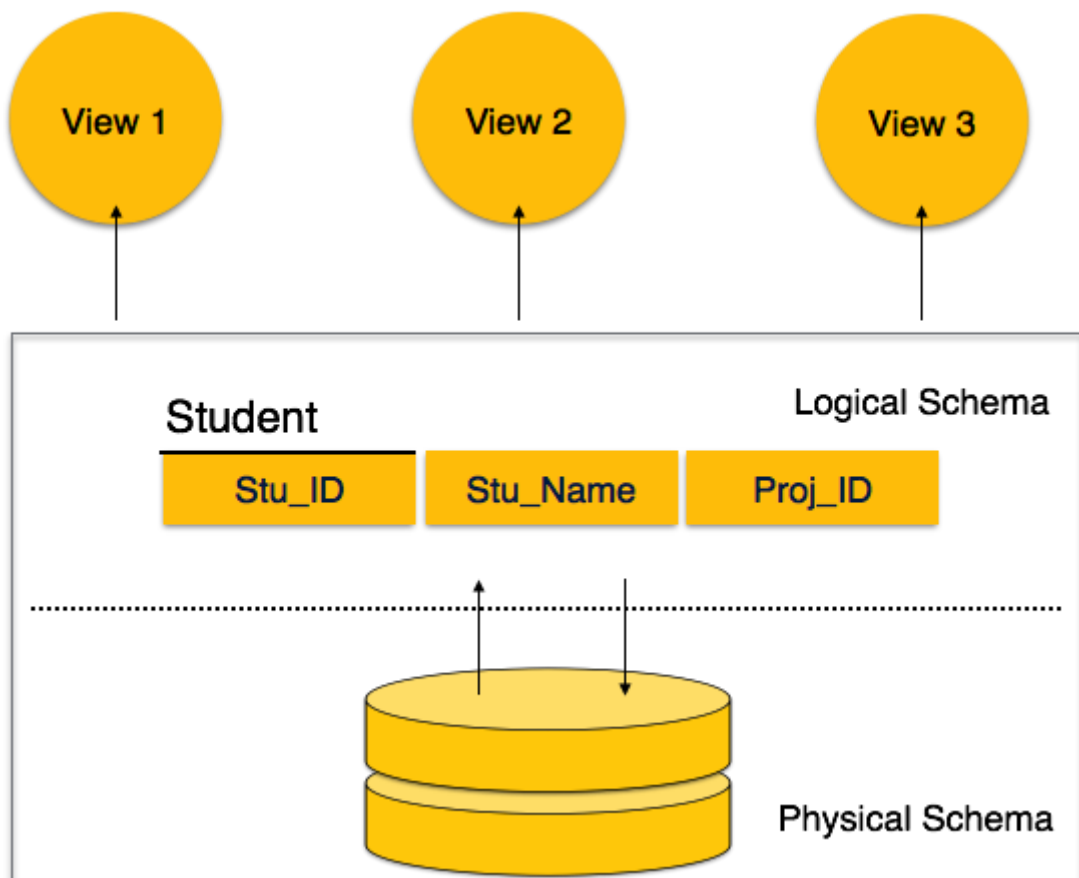

Figure 5.1: Schemas in DBMS

Design of database at logical level is called logical schema, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level).

Design of database at view level is called view schema. This generally describes end user interaction with database systems.

A database schema can be divided broadly into two categories −

1. Physical Database Schema − this schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
2. Logical Database Schema − this schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

5.4 MAPPING AND INSTANCES

Mapping- Process of transforming request and results between three levels it's called mapping. There are the two types of mappings:

1. Conceptual/Internal Mapping
2. External/Conceptual Mapping

1. Conceptual/Internal Mapping:

- The conceptual/internal mapping defines the correspondence between the conceptual view and the store database.
- It specifies how conceptual record and fields are represented at the internal level.
- It relates conceptual schema with internal schema.
- If structure of the store database is changed.

- If changed is made to the storage structure definition-then the conceptual/internal mapping must be changed accordingly, so that the conceptual schema can remain invariant.

- There could be one mapping between conceptual and internal levels.

2. External/Conceptual Mapping:

- The external/conceptual mapping defines the correspondence between a particular external view and conceptual view.

- It relates each external schema with conceptual schema.

- The differences that can exist between these two levels are analogous to those that can exist between the conceptual view and the stored database.

- Example: fields can have different data types; fields and record name can be changed; several conceptual fields can be combined into a single external field.

- Any number of external views can exist at the same time; any number of users can share a given external view: different external views can overlap.

- There could be several mapping between external and conceptual levels.

Database Instance- It is important that we distinguish these two terms individually. Database schema is the skeleton of database. It is designed when the database doesn't exist at all. Once the database is operational, it is very difficult to make any changes to it. A database schema does not contain any data or information. A database instance is a state of operational database with data at any given time. It contains a snapshot of the database. Database instances tend to change with time. A DBMS ensures that its every instance (state) is in a valid state, by diligently following all the validations, constraints, and conditions that the database designers have imposed.

The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

For example, let's say we have a single table student in the database, today the table has 100 records, and so today the instance of the database has 100 records. Let's say we are going to add another 100 records in this table by tomorrow so the instance of database tomorrow will have 200 records in table. In short, at a particular moment the data stored in database is called the instance that changes over time when we add or delete data from the database.

Key Difference between schemas and instances-

Schemas	Instances
It is the overall description of the database.	It is the collection of information stored in a database at a particular moment.
Schema is same for whole database.	Data in instances can be changed using addition, deletion, updation.
Does not change Frequently.	Changes Frequently.
Defines the basic structure of the database i.e how the data will be stored in the database.	It is the set of Information stored at a particular time.

5.5 DATA INDEPENDENCE

A database system normally contains a lot of data in addition to users' data. For example, it stores data about data, known as metadata, to locate and retrieve data easily. It is rather difficult to modify or update a set of metadata once it is stored in the database. But as a DBMS expands, it needs to change over time to satisfy the requirements of the users. If the entire data is dependent, it would become a tedious and highly complex job as shown in figure 5.2.
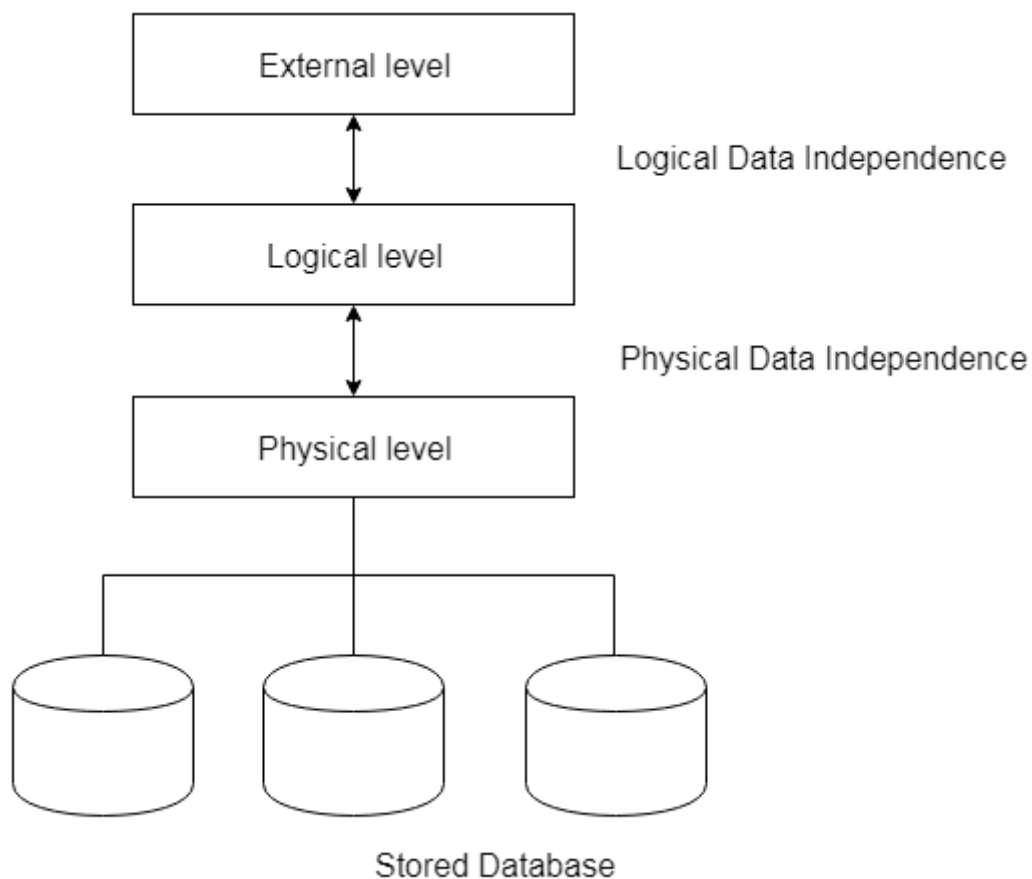

Figure 5.2: Data Independence

In DBMS there are two types of data independence

- Physical data independence
- Logical data independence.

5.5.1 LOGICAL DATA INDEPENDENCE

Physical Data Independence is defined as the ability to make changes in the structure of the lowest level of the Database Management System (DBMS) without affecting the higher-level schemas. Hence, modification in the Physical level should not result in any changes in the Logical or View levels. Logical data is data about database, that is, it stores information about how data is managed inside. For example, a table (relation) stored in the database and all its constraints, applied on that relation. Logical data independence is a kind of mechanism, which liberalizes itself from actual data stored on the disk see figure 6.3. If we do some changes on table format, it should not change the data residing on the disk.
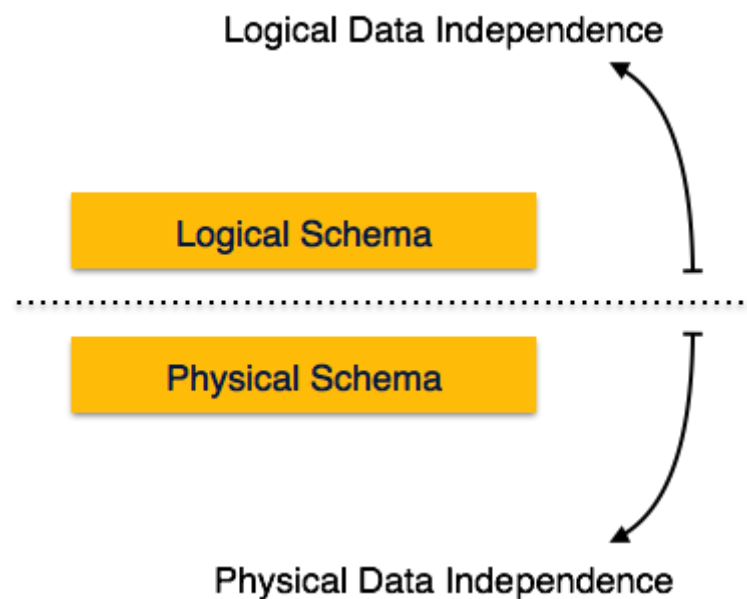

Figure 5.3: Data Independence abstract view

Example –

Changes in the lowest level (physical level) are: creating a new file, storing the new files in the system, creating a new index etc.

Instances of why we may want to do any sort of Data modification in the physical level-
We may want to alter or change the data in the physical level. This is because we may

want to add or remove files and indexes to enhance the performance of the database system and make it faster. Hence, in this way, the Physical Data Independence enables us to do Performance Tuning. Ideally, when we change the physical level, we would not want to alter the logical and view level.

How is Physical Data Independence achieved?

Physical Data Independence is achieved by modifying the physical layer to logical layer mapping (PL-LL mapping). We must ensure that the modification we have done is localized.

5.5.2 PHYSICAL DATA INDEPENDENCE

All the schemas are logical, and the actual data is stored in bit format on the disk. Physical data independence is the power to change the physical data without impacting the schema or logical data. For example, in case we want to change or upgrade the storage system itself − suppose we want to replace hard-disks with SSD − it should not have any impact on the logical data or schemas. Logical Data Independence can also be defined as the ability to make changes in the structure of the middle level of the Database Management System (DBMS) without affecting the highest-level schema or application programs. Hence, modification in the logical level should not result in any changes in the view levels or application programs.

Example –

Changes in the lowest level (physical level) are: adding new attributes to a relation, deleting existing attributes of the relation etc. Ideally, we would not want to change any application or programs that do not require to use the modified attribute.

How is Logical Data Independence achieved?

Logical Data Independence is achieved by modifying the view layer to logical layer mapping (VL-LL mapping)

5.6 CHECK YOUR PROGRESS

1. Which of the following is not a schema?
   a. Database Schema
   b. Physical Schema
   c. Critical Schema
   d. Logical Schema
2. Logical Design of database is called _____
3. Snapshot of the DTA in the database at a given instant of time is called _____.
4. Which of the following is the structure of the database?
   a. Table
   b. Schema
   c. Relation
   d. None of these
5. A logical description of some portion of database that is required by a user to perform task is called as_____.

5.7 SUMMARY

The plans of the database and data stored in the database are most important for an organization, since database is designed to provide information to the organization. The data stored in the database changes regularly but the plans remain static for longer periods of time. A schema is plan of the database that give the names of the entities and attributes and the relationship among them. A schema includes the definition of the database name, the record type and the components that make up the records. Alternatively, it is defined as a frame-work into which the values of the data items are fitted. The values fitted into the frame-work changes regularly but the format of schema remains the same *e.g.,* consider the database consisting of three files ITEM, CUSTOMER and SALES.   Generally,   a

schema can be partitioned into two categories, *i.e.,* (*i*) *Logical schema* and (*ii*) *Physical schema*.

(*i*) The *logical schema* is concerned with exploiting the data structures offered by the DBMS so that the schema becomes understandable to the computer. It is important as programs use it to construct applications.

(*ii*) The *physical schema* is concerned with the manner in which the conceptual database get represented in the computer as a stored database. It is hidden behind the logical schema and can usually be modified without affecting the application programs.

The DBMS's provide DDL and DSDL to specify both the logical and physical schema. When we talk about database we need to know about subschemas. A subschema is a subset of the schema having the same properties that a schema has. It identifies a subset of areas, sets, records, and data names defined in the database schema available to user sessions. The subschema allows the user to view only that part of the database that is of interest to him. The subschema defines the portion of the database as seen by the application programs and the application programs can have different view of data stored in the database. The different application programs can change their respective subschema without affecting other's subschema or view. The Subschema Definition Language (SDL) is used to specify a subschema in the DBMS. Whereas the data in the database at a particular moment of time is called an *instance* or a *database state*. In a given instance, each schema construct has its own current set of instances. Many instances or database states can be constructed to correspond to a particular database schema. Every time we update (*i.e.,* insert, delete or modify) the value of a data item in a record, one state of the database changes into another state.

Data independence can be explained using the three-schema architecture. Data independence refers characteristic of being able to modify the schema at one level of the

database system without altering the schema at the next higher level. There are two types of data independence:

- Logical Data Independence
- Physical Data Independence

5.8 KEYWORDS

- **Domain**
- **View-** Any set of tuples; a data report from the RDBMS in response to a query

5.9 SELF-ASSESSMENT TEST

1. What do you understand by data independence? What are its two types?
2. Define the relationship between view and data independence?
3. What is the role of Schema in databases?
4. What are the advantages and disadvantages of view in the databases?
5. Is there any other types of data independence other than logical and physical data independence?

5.10 ANSWERS TO CHECK YOUR PROGRESS

1. Critical Schema
2. Database Schema
3. Database Instance
4. Schema
5. User View

5.11 REFERENCES / SUGGESTED READINGS

- C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.
- Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.
- Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.

- Thomas Connolly Carolyn Begg, "Database System", 3/2, Pearson Education.
- https://www.geeksforgeeks.org/physical-and-logical-data-independence/
- https://mcqslearn.com/cs/dbms/data-models-categories-multiple-choice-questions.php
- https://tutorialink.com/dbms/data-independence.dbms
- https://beginnersbook.com/2015/04/instance-and-schema-in-dbms/

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	AUTHOR: DR. DEEPAK NANDAL
LESSON NO. 6	VETTER:
ENTITY-RELATION MODEL AND RELATIONSHIPS	

STRUCTURE

6.0 LEARNING OBJECTIVE

- The objective of this chapter is to make the reader understand the meaning, and concept of entity relation model in database management system, to know what are entities, attributes, entity sets and relationship instances as well in detail.

6.1 INTRODUCTION

ER model is represents real world situations using concepts, which are commonly used by people. It allows defining a representation of the real world at logical level.ER model has no facilities to describe machine-related aspects.

In ER model the logical structure of data is captured by indicating the grouping of data into entities. The ER model also supports a top-down approach by which details can be given in successive stages.

A relationship, in the context of databases, is a situation that exists between two relational database tables when one table has a foreign key that references the primary key of the other table. Relationships allow relational databases to split and store data in different tables, while linking disparate data items. For example, in a bank database a CUSTOMER_MASTER table stores customer data with a primary key column named CUSTOMER_ID; it also stores customer data in an ACCOUNTS_MASTER table, which holds information about various bank accounts and associated customers. To link these two tables and determine customer and bank account information, a corresponding CUSTOMER_ID column must be inserted in the ACCOUNTS_MASTER table, referencing existing customer IDs from the CUSTOMER_MASTER table. In this case, the ACCOUNTS_MASTER table's CUSTOMER_ID column is a foreign key that references a column with the same name in the CUSTOMER_MASTER table. This is an example of a relationship between the two tables.

The fundamental feature that differentiates relational databases from other database types (e.g., flat-files) is the ability to define relationships.

- **One-to-One Relationships**

A pair of tables bears a one-to-one relationship when a single record in the first table is related to only one record in the second table, and a single record in the second table is related to only one record in the first table. This can be shown as in figure 6.1.
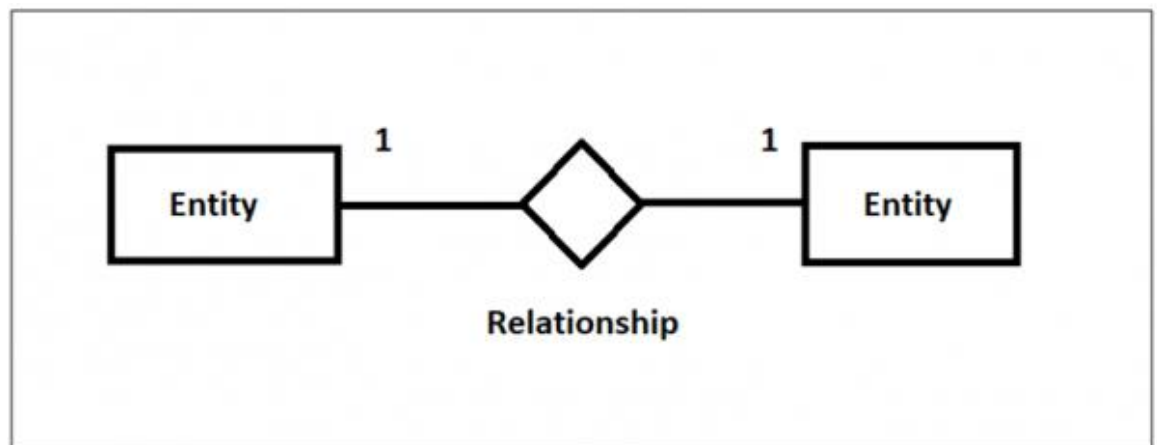
Figure 6.1: One to one relationship

- One to Many relationship

One-to-Many relationship in DBMS is a relationship between instances of an entity with more than one instance of another entity.

The relation can be shown in figure 6.2 –

Figure 6.2: One to Many Relationship

- Many to Many relationship

A pair of tables bears a many-to-many relationship when a single record in the first table can be related to one or more records in the second table and a single record in the second table can be related to one or more records in the first table.

Figure 6.3: Many to Many Relationship
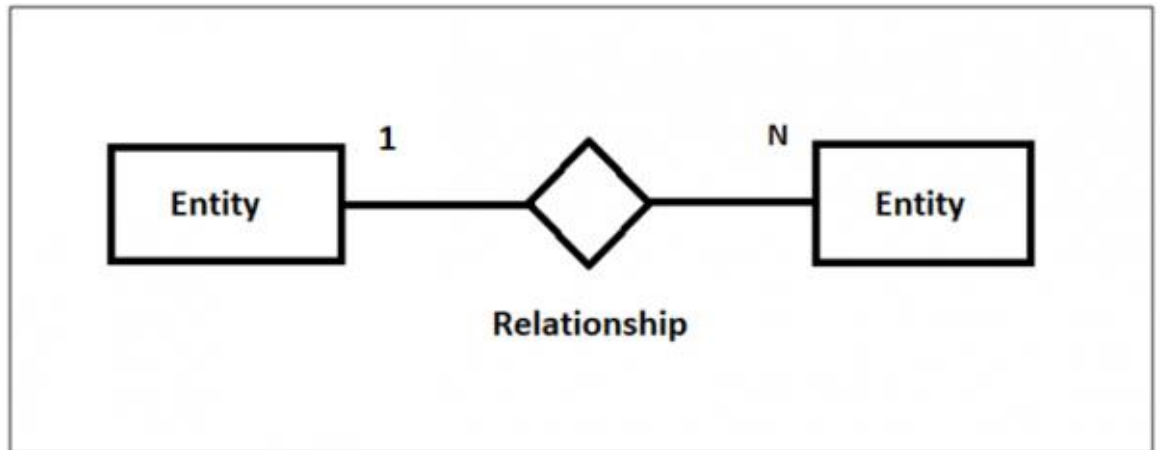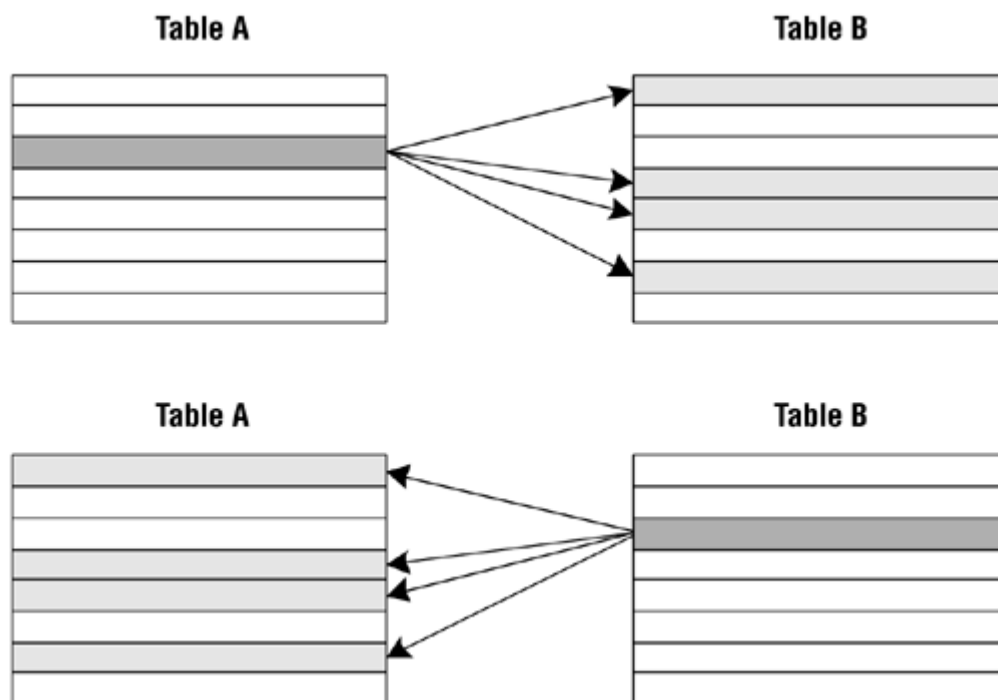
Assume once again that you're working with TABLE A and TABLE B and that there is a many-to-many relationship between them. Because of the relationship, a single record in TABLE A can be related to one or more records (but not necessarily all) in TABLE B. Conversely, a single record in the TABLE B can be related to one or more records (but not necessarily all) in TABLE A. Figure 6.3 shows the relationship from the perspective of each table.

6.2 DEFINITION

Entity- An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes- Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Entity Set and Keys- Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, the roll_number of a student makes him/her identifiable among students.

- **Super Key** − A set of attributes (one or more) that collectively identifies an entity in an entity set.

- **Candidate Key** − A minimal super key is called a candidate key. An entity set may have more than one candidate key.

- **Primary Key** − A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

Relationships- The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.

Database Relationship- Database relationships are associations between tables that are created using join statements to retrieve data. ... Both tables can have only one record on each side of the relationship. Each primary key value relates to none or only one record in the related table.

6.3 ENTITY-RELATIONSHIP MODEL

The *entity-relationship (ER) data model* allows us to describe the data involved in a real-world enterprise in terms of objects and their relationships and is widely used to develop an initial database design. The ER model is important primarily for its role in database design. It provides useful concepts that allow us to move from an informal description of what users want from their database to a more detailed, and precise description that can be implemented in a DBMS.

Entities, Attributes and Entity sets

An **entity** is an object in the real world that is distinguishable from other objects. Examples include the following: the Green Dragonzord toy, the toy department, the manager of the toy department, the home address of the manager of the toy department. It is often useful to identify a collection of similar entities. Such a collection is called an entity set. Note that entity sets need not be disjoint; the collection of toy department employees and the collection of appliance department employees may both contain employee Jai Parkash (who happens to work in both departments). We could also define an entity set called Employees that contains both the toy and appliance department employee sets.

An entity is described using a set of attributes. All entities in a given entity set have the same attributes; this is essentially what we mean by *similar*. Our choice of attributes reflects the level of detail at which we wish to represent information about entities. For example, the Employees entity set could use name, Adhar number (Adhar_No.), and parking lot (lot) as attributes. In this case we will store the name, social security number, and lot number for each employee. However, we will not store, say, an employee's address (or gender or age). For each attribute associated with an entity set, we must identify a domain of possible values. For example, the domain associated with the attribute *name* of Employees might be the set of 20-character strings. As another example, if the company rates employees on a scale of 1 to 10 and stores ratings in a field called *rating*, the associated domain consists of integers 1 through 10. In some cases, a particular entity may not have an applicable value for an attribute. For example, the Apartment_number attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes. Similarly, a College_degrees attribute applies only to people with college degrees. For such situations, a special value called NULL is created.

For each entity set, we choose a *key*. A key is a minimal set of attributes whose values uniquely identify an entity in the set. There could be more than one candidate key; if so, we designate one of them as the primary key. For now we will assume that each entity set contains at least one set of attributes that uniquely identifies an entity in the entity set; that is, the set of attributes contains a key. The following figure shows an Employee Entity with its attributes:
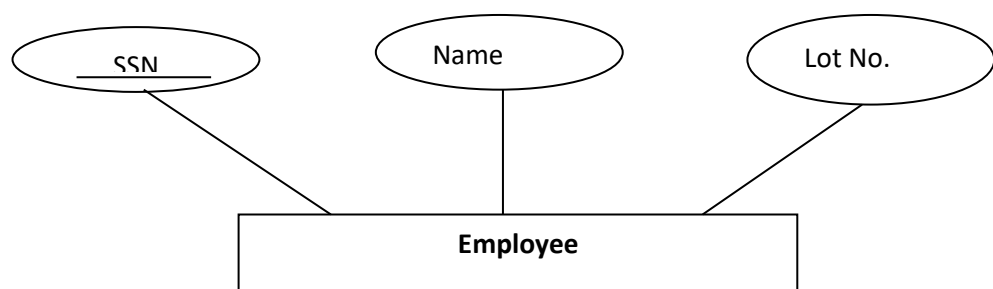

Figure 6.4   The Employee Entity Set

As shown in the figure 6.4 an entity set is represented by a rectangle, and an attribute is represented by an oval. Each attribute in the primary key is underlined. The domain information could be listed along with the attribute name, but we omit this to keep the figures compact. The key is *Adhar_No.*. Several types of attributes occur in the ER model: *simple* versus *composite*, *singlevalued* versus *multivalued*, and *stored* versus *derived*. First we define these attribute types and illustrate their use via examples

- **Composite versus Simple (Atomic) Attributes. Composite attributes** can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the Address attribute of the EMPLOYEE entity can be subdivided into Street_address, City, State, and Zip with the values 'KirtiNaragr', 'Sirsa', 'Haryana', and '125055.' Attributes that are not divisible are called **simple** or **atomic attributes**. Composite attributes can form a hierarchy; for

example, Street_address can be further subdivided into three simple component attributes: Number, Street, and Apartment_number. The value of a composite attribute is the concatenation of the values of its component simple attributes.

- **Single-Valued versus Multivalued Attributes.** Most attributes have a single value for a particular entity; such attributes are called single-valued. For example, Age is a single-valued attribute of a person. In some cases an attribute can have a set of values for the same entity—for instance, a Colors attribute for a car, or a College_degrees attribute for a person. Cars with one color have a single value, whereas two-tone cars have two color values. Similarly, one person may not have a college degree, another person may have one, and a third person may have two or more degrees; therefore, different people can have different numbers of values for the College_degrees attribute. Such attributes are called multivalued.

- **Stored versus Derived Attributes.** In some cases, two (or more) attribute values are related—for example, the Age and Birth_date attributes of a person. For a particular person entity, the value of Age can be determined from the current (today's) date and the value of that person's Birth_date. The Age attribute is hence called a derived attribute and is said to be derivable from the Birth_date attribute, which is called a stored attribute. Some attribute values can be derived from *related entities*; for example, an attribute Number_of_employees of a DEPARTMENT entity can be derived by counting the number of employees related to (working for) that department.

- **Complex Attributes.** Notice that, in general, composite and multivalued attributes can be nested arbitrarily. We can represent arbitrary nesting by grouping components of a composite attribute between parentheses () and separating the

components with commas, and by displaying multivalued attributes between braces
{ }. Such attributes are called complex attributes.

Relationship and Relationship Set

A relationship is an association among two or more entities. For example, we may
have the relationship that Ashok works in the pharmacy department. As with entities, we
may wish to collect a set of similar relationships into a relationship set. A relationship set
can be thought of as a set of n-tuples :

$$\{e_1, e_2\ldots\ldots\ldots e_n| e_1 \in E_1\ldots\ldots\ldots\ldots\ldots\ldots e_n \in E_n\}$$

In ER diagrams, relationship types are displayed as diamond-shaped boxes, which are
connected by straight lines to the rectangular boxes representing the participating entity
types. The relationship name is displayed in the diamond-shaped box.

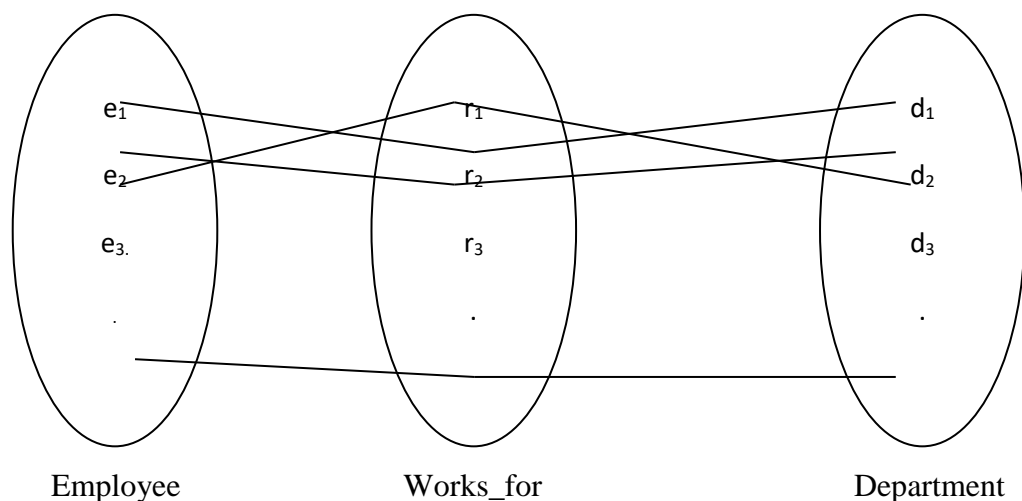*Figure 6.5 Works_for relationship between Employee and Department*

Degree of a Relationship Type. The **degree** of a relationship type is the number of
participating entity types. Hence, the WORKS_FOR relationship is of degree two as shown

in figure 6.5. A relationship type of degree two is called **binary**, and one of degree three is called **ternary**.

Role Names and Recursive Relationships. Each entity type that participates in a relationship type plays a particular role in the relationship. The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means. For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of *employee* or *worker* and DEPARTMENT plays the role of *department* or *employer*. Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name. However, in some cases the *same* entity type participates more than once in a relationship type in *different roles*. In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships.** The SUPERVISION relationship type relates an employee to a supervisor, where both employee and supervisor entities are members of the same

EMPLOYEE entity set. Hence, the EMPLOYEE entity type *participates twice* in SUPERVISION: once in the role of *supervisor* (or *boss*), and once in the role of *supervisee* (or *subordinate*)

Constraints on Binary Relationship Types

- **Cardinality Ratios for Binary Relationships.** The **cardinality ratio** for a binary relationship specifies the *maximum* number of relationship instances that an entity can participate in. For example, in the WORKS_FOR binary relationship type, DEPARTMENT: EMPLOYEE is of cardinality ratio 1: N, meaning that each department can be related to (that is, employs) any number of employees, but an employee can be related to (work for) only one department.

- **Participation Constraints and Existence Dependencies.** The participation constraint specifies 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 and partial—that we illustrate by example. If a company policy states that *every* employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS_FOR relationship instance. Thus, the participation of EMPLOYEE in WORKS_FOR is called total participation, meaning that every entity in *the total set* of employee entities must be related to a department entity via WORKS_FOR. Total participation is also called existence dependency. In *manage* relationship we do not expect every employee to manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is partial, meaning that *some* or *part of the set of* employee entities are related to some department entity via MANAGES, but not necessarily all. In ER diagrams, total participation (or existence dependency) is displayed as a *double line* connecting the participating entity type to the relationship, whereas partial participation is represented by a *single line.*

Weak Entity Types

Entity types that do not have key attributes of their own are called weak entity types. In contrast, regular entity types that do have a key attribute—which include all the examples discussed so far—are called strong entity types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other entity type the identifying or owner entity type, and we call the relationship type that relates a weak entity type to its owner the identifying relationship of the weak entity type.

The following table 9.1 shows the basic notations for ER diagram
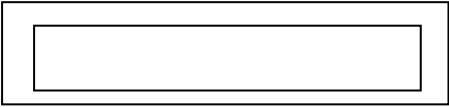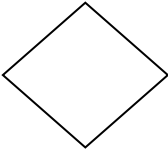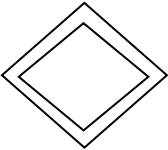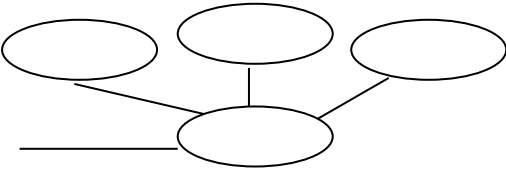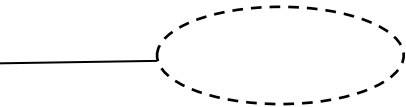
Table 6.1 Basic notation in E-R Model

6.4 ENTITY RELATION DIAGRAMS

In the following figure 6.6 we have two entities Student and College and their relationship.

1. ER Diagram of Student and College - The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time. Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.
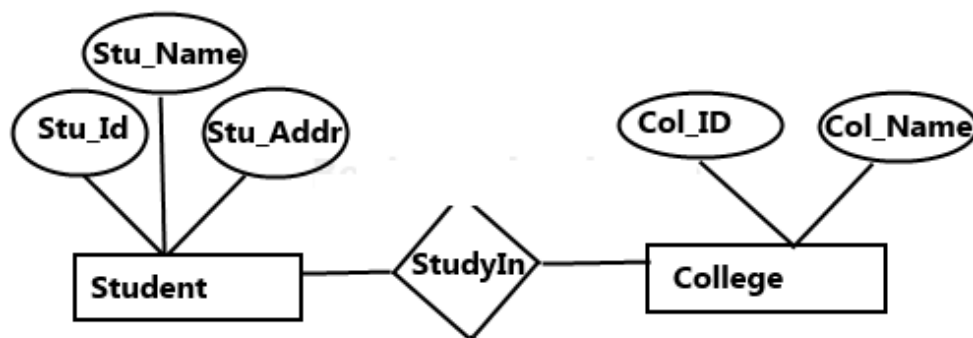
Figure 6.6: ER diagram of Student and College relationship

2. ER Diagram of University Database
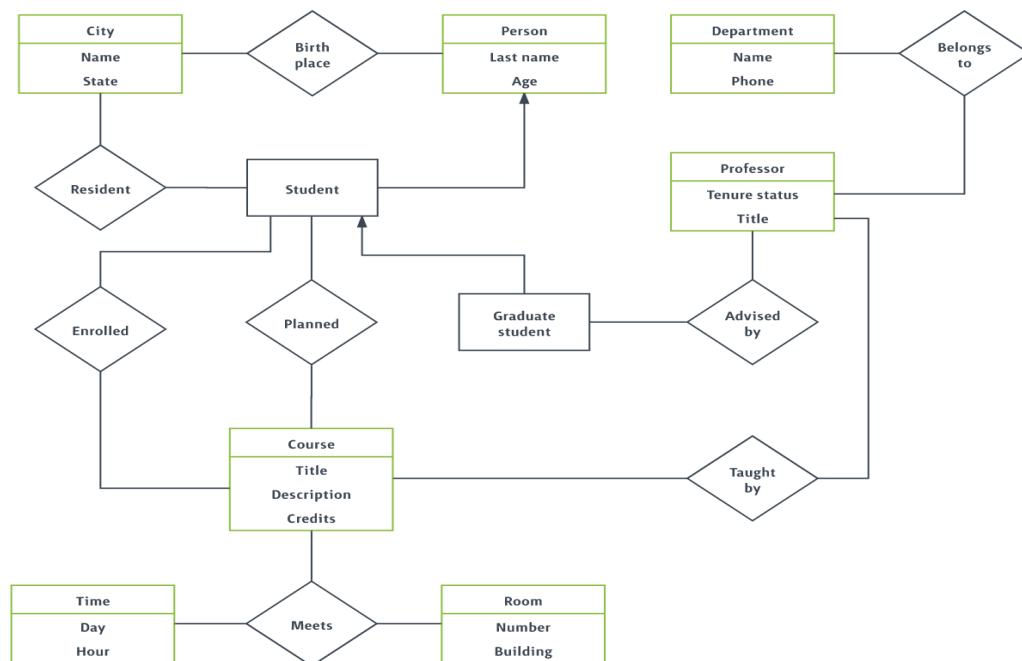
Figure 6.7: ER Diagram of University Database System

ER Diagram of Library Database
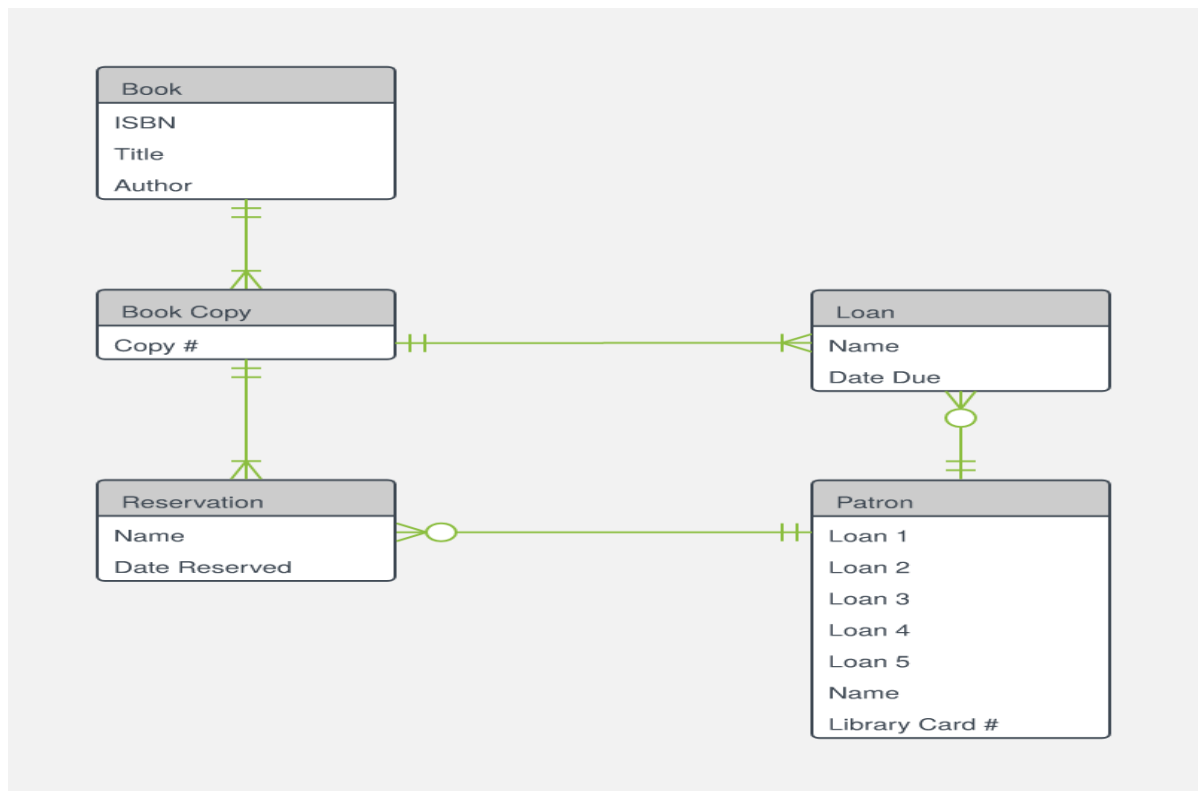
Figure 6.8: ER Diagram of Library Database

Figure 6.7 and figure 6.8 are the examples of some ER diagrams.

6.5 CHECK YOUR PROGRESS

1. A many to many relationship between two entities usually results in how many tables?
2. An _____ is a set of entities of the same type that share the same properties, or attributes.
3. Entity is a _____
4. Every weak entity set can be converted into a strong entity set by:
5. E-R modelling technique is a _____.

6.6 SUMMARY

ER Model is used to model the logical view of the system from data perspective which consists of these components as shown in figure 6.9:

Entity, Entity Type, Entity Set –

An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

An Entity is an object of Entity Type and set of all entities is called as entity set. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set

Attribute(s):

Attributes are the properties which define the entity type. For example, Roll_No, Name, DOB, Age, Address, Mobile_No are the attributes which defines entity type Student.
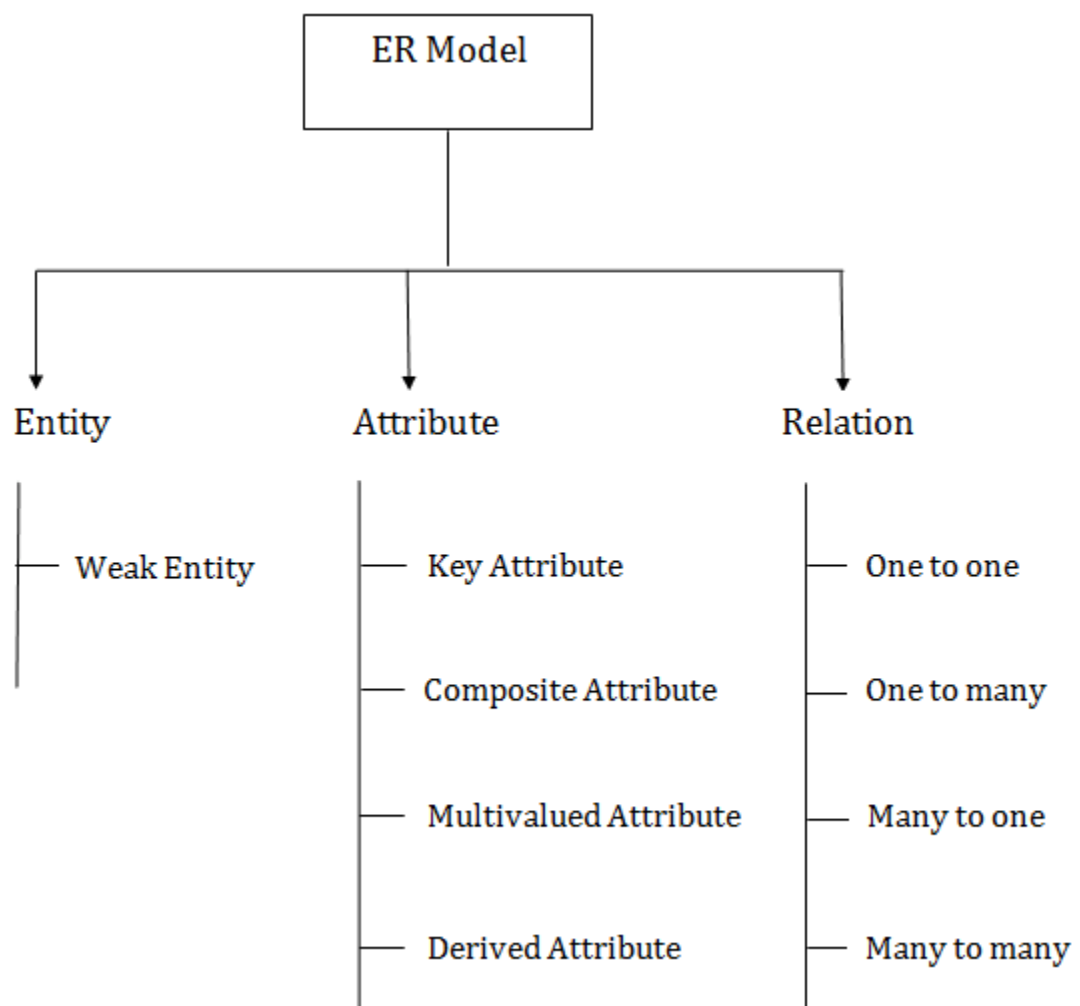
Figure 6.9: Components of ER diagram

Relationship-

A relationship type represents the association between entity types. For example,'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.

Degree of Relationship set-

The number of different entity sets participating in a relationship set is called as degree of a relationship set.

1. UnaryRelationship– When there is only ONE entity set participating in a relation, the relationship is called as unary relationship. For example, one person is married to only one person.

2. BinaryRelationship– When there are TWO entities set participating in a relation, the relationship is called as binary relationship.For example, Student is enrolled in Course.

3. n-aryRelationship– When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

One to one –A row in table A can have only one matching row in table B, and vice versa. This is not a common relationship type, as the data stored in table B could just have easily been stored in table A. However, there are some valid reasons for using this relationship type. A one-to-one relationship can be used for security purposes, to divide a large table, and various other specific purposes.

One to Many- This is the most common relationship type. In this type of relationship, a row in table A can have many matching rows in table B, but a row in table B can have only one matching row in table A.

Many to many- In a many-to-many relationship, a row in table. A can have many matching rows in table B, and vice versa. A many-to-many relationship could be thought of as two one-to-many relationships, linked by an intermediary table. The intermediary table is typically referred to as a "junction table" (also as a "cross-reference table"). This table is used to link the other two tables together. It does this by having two fields that reference the primary key of each of the other two tables.

6.7 KEYWORDS

- **ERD-** An Entity Relationship Diagram (ERD) is a snapshot of data structures. An Entity Relationship Diagram shows entities (tables) in a database and relationships between tables within that database.
- **DATA MODEL-** A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities

6.8 SELF-ASSESSMENT TEST

1. When is the concept of a weak entity used in data modeling? Define the terms *owner entity type, weak entity type, identifying relationship type,* and *Partial key*.
2. Discuss the conventions for displaying an ER schema as an ER diagram.
3. Discuss the naming conventions used for ER schema diagrams.
4. What is a network data model? Discuss with example.
5. Describe the hierarchical model with the constraints and structure of database.

6.9 ANSWERS TO CHECK YOUR PROGRESS

1. Three
2. Entity Set
3. Real World thing

4. Adding appropriate attributes

5. Top-down appraoch

6.10 REFERENCES / SUGGESTED READINGS

- C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.

- Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.

- Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.

- Thomas Connolly Carolyn Begg, "Database System", 3/2, Pearson Education.

- https://www.tutorialspoint.com/dbms/er_model_basic_concepts.htm

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	**AUTHOR: DR. DEEPAK NANDAL**
LESSON NO. 7	**VETTER:**
RELATIONAL MODEL AND QUERY LANGUAGE	

7.0 LEARNING OBJECTIVE

- The objective of this chapter is to understand the concepts of relational database, to know the difference between DBMS and RDBMS. To understand the parameters of RDBMS and components of RDBMS in detail such as the concepts and notations of the relational model. This chapter is helps the reader to understand the most popular and widely used query language SQL. This chapter presents the main features of the SQL standard for *commercial* relational DBMSs. The main characteristics of SQL, SQL data types and SQL literals.

7.1 INTRODUCTION

The relational model was first introduced by Ted Codd of IBM Research in 1970 in a classic paper (Codd 1970), and attracted immediate attention due to its simplicity and mathematical foundation. The model uses the concept of a *mathematical* relation-which looks somewhat like a table of values-as its basic building block, and has its theoretical basis in set theory and first-order predicate logic. In this chapter we discuss the basic characteristics of the model and its constraints. The first commercial implementations of the relational model became available in the early *1980s,* such as the Oracle DBMS and the SQL/DS system on the MVS operating system by IBM. Since then, the model has been implemented in a large number of commercial systems. Current popular relational DBMSs (RDBMSs) include DB2 and lnformix Dynamic Server (from IBM), Oracle and Rdb (from Oracle), and SQL Server and Access (from Microsoft). Most of the problems faced at the time of implementation of any system are outcome of a poor database design. In many cases it happens that system has to be continuously modified in multiple respects due to changing requirements of users. It is very important that a proper planning has to be done. A relation in a relational database is based on a relational schema, which consists of number of

attributes. A relational database is made up of a number of relations and corresponding relational database schema. The goal of a relational database design is to generate a set of relation schema that allows us to store information without unnecessary redundancy and also to retrieve information easily. One approach to design schemas that are in an appropriate normal form. The normal forms are used to ensure that various types of anomalies and inconsistencies are not introduced into the database.

Database management systems (DBMS) must have a query language so that the users can access the data stored in the database. **Relational algebra (RA)** is considered as a *procedural query language* where the user tells the system to carry out a set of operations to obtain the desired results. i.e. The user tells what data should be retrieved from the database and how to retrieve it. In this article, I will give a brief introduction to relational algebra and go through a few operations with examples and PostgreSQL commands.

Relational Calculus, which is a non-procedural query language. In this chapter, you will learn about the relational calculus and its concept about the database management system. A certain arrangement is explicitly stated in relational algebra expression, and a plan for assessing the query is implied. In the relational calculus, there is no description and depiction of how to assess a query; instead, a relational calculus query focuses on what is to retrieve rather than how to retrieve it. It uses mathematical predicate calculus. The relational calculus is not the same as that of differential and integral calculus in mathematics but takes its name from a branch of symbolic logic termed as predicate calculus. When applied to databases, it is found in two forms. These are

- Tuple relational calculus which was originally proposed by Codd in the year 1972 and

- Domain relational calculus which was proposed by Lacroix and Pirotte in the year 1977

A calculus expression specifies what is to be retrieved rather than how to retrieve it. Therefore, the relational calculus is considered to be a nonprocedural language. This differs from relational algebra, where we must write a sequence of operations to specify a retrieval request; hence, it can be considered as a procedural way of stating a query. It is possible to nest algebra operations to form a single expression; however, a certain order among the operations is always explicitly specified in a relational algebra expression. This order also influences the strategy for evaluating the query. A calculus expression may be written in different ways, but the way it is written has no bearing on how a query should be evaluated.

The SQL language may be considered one of the major reasons for the commercial success of relational databases. Because it became a standard for relational databases, users were less concerned about migrating their database applications from other types of database systems—for example, network or hierarchical systems—to relational systems. This is because even if the users became dissatisfied with the particular relational DBMS product they were using, converting to another relational DBMS product was not expected to be too expensive and time-consuming because both systems followed the same language standards. However, the relational algebra operations are considered to be too technical for most commercial DBMS users because a query in relational algebra is written as a sequence of operations that, when executed, produces the required result. Hence, the user must specify how—that is, *in what order*—to execute the query operations. On the other hand, the SQL language provides a higher-level *declarative* language interface, so the user only specifies *what* the result is to be, leaving the actual optimization and decisions on how to execute the query to the DBMS. Although SQL includes some features from relational algebra, it is based to a greater extent on the *tuple relational calculus*.

111

The name SQL is presently expanded as Structured Query Language. Originally, SQL was called SEQUEL (Structured English QUEry Language) and was designed and implemented at IBM Research as the interface for an experimental relational database system called SYSTEM R. SQL is now the standard language for commercial relational DBMSs. A joint effort by the American National Standards Institute (ANSI) and the International Standards Organization (ISO) has led to a standard version of SQL (ANSI 1986), called SQL-86 or SQL1. A revised and much expanded standard called SQL-92 (also referred to as SQL2) was subsequently developed. The next standard that is well-recognized is SQL:1999, which started out as SQL3. Two later updates to the standard are SQL:2003 and SQL:2006, which added XML features among other updates to the language. Another update in 2008 incorporated more object database features in SQL. .SQL is a comprehensive database language: It has statements for data definitions, queries, and updates. Hence, it is both a DDL *and* a DML. In addition, it has facilities for defining views on the database, for specifying security and authorization, for defining integrity constraints, and for specifying transaction controls. It also has rules for embedding SQL statements into a general-purpose programming language such as Java, COBOL, or C/C++.

7.2 DEFINITION

Relational Algebra- Relational algebra is a procedural query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, and delete on the data. When I say that relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved. On the other hand relational calculus is a non-procedural query language, which means it tells what data to be retrieved but doesn't tell how to retrieve it. We will discuss relational calculus in a separate tutorial. Relational algebra is a

procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

What is Relational Calculus?

Relational calculus is a non-procedural query language that tells the system what data to be retrieved but doesn't tell how to retrieve it. Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results. The relational calculus tells what to do but never explains how to do. Contrary to Relational Algebra which is a procedural query language to fetch data and which also explains how it is done, Relational Calculus in non-procedural query language and has no description about how the query will work or the data will be fetched. It only focusses on what to do, and not on how to do it.

Relational Calculus exists in two forms as shown in figure 4.1:

1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)
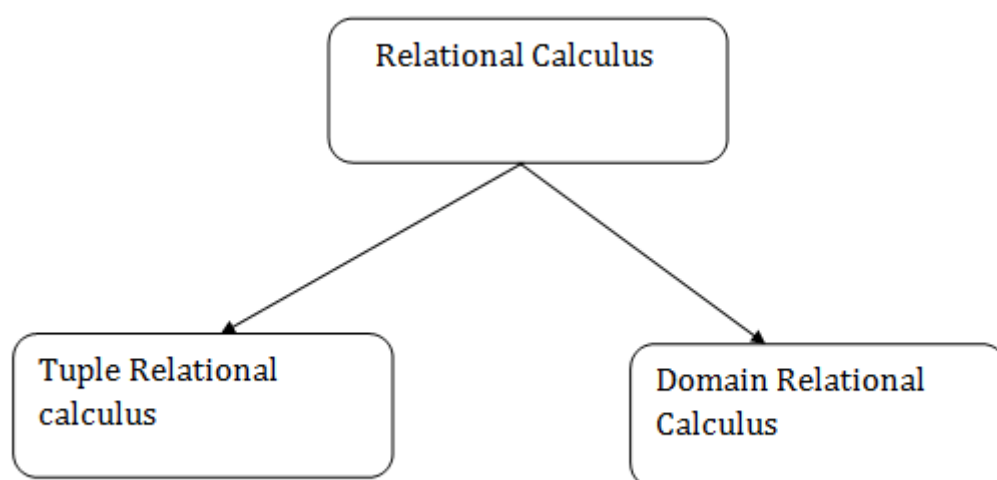
Figure 7.1: Types of relational calculus

SQL- SQL uses the terms **table**, **row**, and **column** for the formal relational model terms *relation*, *tuple*, and *attribute*, respectively. Unlike most programming languages, SQL is unique in that it is not procedural but declarative in nature. This means that when using this language one states what data is desired and not how to get that data. A component within the database server known as the *optimizer* will automatically determine how to get the data most efficiently. Therefore the user may concentrate solely on what data is desired and then allow the database to automatically select the optimum method by which to retrieve that data.

7.3 WHAT IS RDBMS?

RDBMS stands for Relational Database Management System. RDBMS data is structured in database tables, fields and records. Each RDBMS table consists of database table rows. Each database table row consists of one or more database table fields. RDBMS store the data into collection of tables, which might be related by common fields (database table columns). RDBMS also provide relational operators to manipulate the data stored into the database tables. Most RDBMS use SQL as database query language. The most popular RDBMS are MS SQL Server, DB2, Oracle and MySQL. The relational model is an example of record-based model. Record based models are so named because the database is structured in fixed format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record types. The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model. The relational model was designed by the IBM research scientist and mathematician, Dr. E.F.Codd. Many modern DBMS do not conform to the Codd's definition of a RDBMS, but nonetheless they are still considered to be RDBMS.

Two of Dr.Codd's main focal points when designing the relational model were to further reduce data redundancy and to improve data integrity within database systems.

The relational model originated from a paper authored by Dr.codd entitled "A Relational Model of Data for Large Shared Data Banks", written in 1970. This paper included the following concepts that apply to database management systems for relational databases. The relation is the only data structure used in the relational data model to represent both entities and relationships between them. Rows of the relation are referred to as tuples of the relation and columns are its attributes. Each attribute of the column are drawn from the set of values known as domain. The domain of an attribute contains the set of values that the attribute may assume. From the historical perspective, the relational data model is relatively new .The first database systems were based on either network or hierarchical models .The relational data model has established itself as the primary data model for commercial data processing applications. Its success in this domain has led to its applications outside data processing in systems for computer aided design and other environments. A relational database management system (RDBMS) is a collection of programs and capabilities that enable IT teams and others to create, update, administer and otherwise interact with a relational database. RDBMS store data in the form of tables, with most commercial relational database management systems using Structured Query Language (SQL) to access the database. However, since SQL was invented after the initial development of the relational model, it is not necessary for RDBMS use.

7.4 DIFFERENCE BETWEEN DBMS AND RDBMS

A DBMS has to be persistent, that is it should be accessible when the program created the data ceases to exist or even the application that created the data restarted. A DBMS also has to provide some uniform methods independent of a specific application for accessing the information that is stored. RDBMS is a Relational Data Base Management System

Relational DBMS. This adds the additional condition that the system supports a tabular structure for the data, with enforced relationships between the tables. This excludes the databases that don't support a tabular structure or don't enforce relationships between tables. You can say DBMS does not impose any constraints or security with regard to data manipulation it is user or the programmer responsibility to ensure the ACID PROPERTY of the database whereas the RDBMS is more with this regard because RDBMS define the integrity constraint for the purpose of holding ACID PROPERTY.

In general, databases store sets of data that can be queried for use in other applications. A database management system supports the development, administration and use of database platforms. An RDBMS is a type of database management system (DBMS) that stores data in a row-based table structure which connects related data elements. An RDBMS includes functions that maintain the security, accuracy, integrity and consistency of the data. This is different than the file storage used in a DBMS. Other differences between database management systems and relational database management systems include:

- **Number of allowed users-** While a DBMS can only accept one user at a time, an RDBMS can operate with multiple users.

- **Hardware and software requirements-** A DBMS needs less software and hardware than an RDBMS.

- **Amount of data**- RDBMS can handle any amount of data, from small to large, while a DBMS can only manage small amounts.

- **Database structure-** In a DBMS, data is kept in a hierarchical form, whereas an RDBMS utilizes a table where the headers are used as column names and the rows contain the corresponding values.

- **ACID implementation-** DBMS do not use the atomicity, consistency, isolation and durability (ACID) model for storing data. On the other hand, RDBMS base the structure of their data on the ACID model to ensure consistency.

- **Distributed databases-** While an RDBMS offers complete support for distributed databases, a DBMS will not provide support.

- **Types of programs managed-** While an RDBMS helps manage the relationships between its incorporated tables of data, a DBMS focuses on maintaining databases that are present within the computer network and system hard disks.

- **Support of database normalization-** An RDBMS can be normalized, but a DBMS cannot.

DBMS vs RDBMS using different parameters

Parameter	DBMS	RDBMS
Storage	DBMS stores data as a file.	Data is stored in the form of tables.
Database structure	DBMS system, stores data in either a navigational or hierarchical form.	RDBMS uses a tabular structure Where the headers are the column names, and the rows contain corresponding values
Number of Users	DBMS supports single user only.	It supports multiple users.
ACID	In a regular database, the data may not be	Relational databases are harder to construct, but they are consistent and

Parameter	DBMS	RDBMS
	stored following the ACID model. This can develop inconsistencies in the database.	well structured. They obey ACID (Atomicity, Consistency, Isolation, Durability).
Type of program	It is the program for managing the databases on the computer networks and the system hard disks.	It is the database systems which are used for maintaining the relationships among the tables.
Hardware and software needs.	Low software and hardware needs.	Higher hardware and software need.
Integrity constraints	DBMS does not support the integrity constants. The integrity constants are not imposed at the file level.	RDBMS supports the integrity constraints at the schema level. Values beyond a defined range cannot be stored into the particular RDMS column.
Normalization	DBMS does not support Normalization	RDBMS can be Normalized.
Distributed Databases	DBMS does not support distributed database.	RBMS offers support for distributed databases.

Parameter	DBMS	RDBMS
Ideally suited for	DBMS system mainly deals with small quantity of data.	RDMS is designed to handle a large amount of data.

7.6 RELATIONAL ALGEBRA

The relational algebra is often considered to be an integral part of the relational data model, and its operations can be divided into two groups. One group includes set operations from mathematical set theory; these are applicable because each relation is defined to be a set of tuples in the formal relational model. Set operations include UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT. The other group consists of operations developed specifically for relational databases-these include SELECT PROJECT, and JOIN, among others. This chapter firstly discuss the SELECT and POJECT operations because they are unary operations that operate on single relations. Then the chapter discusses the JOIN and other complex binary operations, which operate on two tables. Some common database requests cannot be performed with the original relational algebra operations, so additional operations were created to express these requests. These include aggregate functions, which are operations that can summarize data from the tables, as well as additional types of JOIN and UNION operations. These operations were added to the original relational algebra because of their importance to many database applications. As, the chapter ends with the discussion of relational algebra, the subsequent chapter will focus on describing the other main formal language for relational databases and relational calculus.

The relational algebra is a theoretical procedural query language which takes an instance of relations and does operations that work on one or more relations to describe another relation without altering the original relation(s). Thus, both the operands and the outputs are relations. So the output from one operation can turn into the input to another operation, which allows expressions to be nested in the relational algebra, just as you nest arithmetic operations. This property is called closure: relations are closed under the algebra, just as numbers are closed under arithmetic operations.

The relational algebra is a relation-at-a-time (or set) language where all tuples are controlled in one statement without the use of a loop. There are several variations of syntax for relational algebra commands, and you use a common symbolic notation for the commands and present it informally.

The primary operations of relational algebra are as follows:

- Select

- Project

- Union

- Set different

- Cartesian product

- Rename

Relational algebra is a family of algebras with a well-founded semantics used for modelling the data stored in relational databases, and defining. It takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate

results are also considered relations. Relational algebra collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform this action. SQL Relational algebra query operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.

The figure 7.2 shows how we use relational algebra to fetch information or data from a bigger dataset or table. In relational algebra, input is a relation (table from which data has to be accessed) and output is also a relation (a temporary table holding the data asked for by the user).
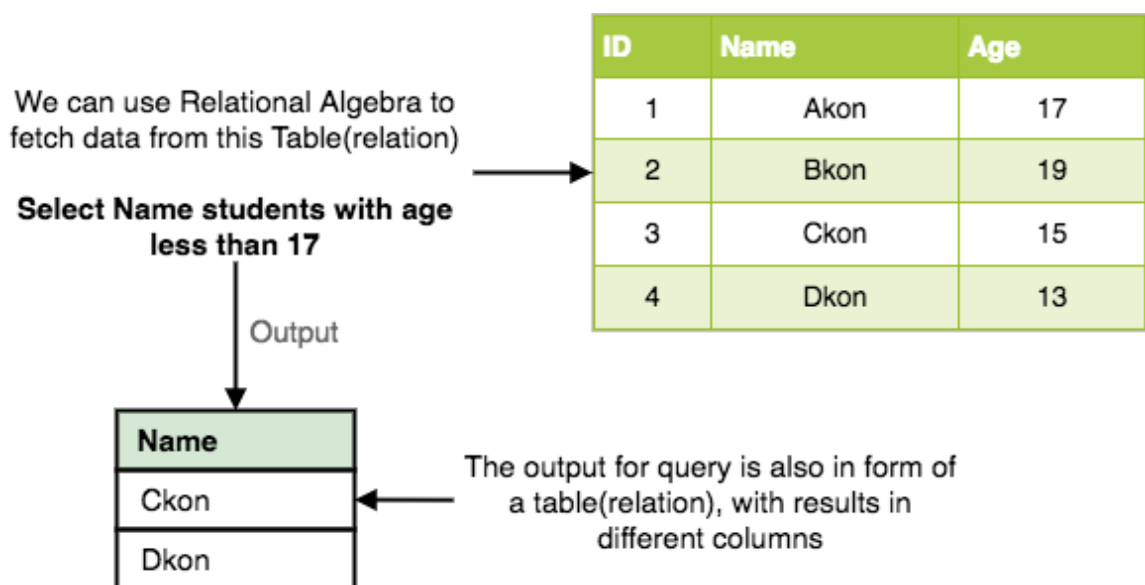
Figure 7.2: Use of Relational Algebra

Relational Algebra works on the whole table at once, so we do not have to use loops etc. to iterate over all the rows (tuples) of data one by one. All we have to do is specify the table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data for you.

7.7 RELATIONAL CALCULUS

Relational calculus is a non-procedural query language that tells the system what data to be retrieved but doesn't tell how to retrieve it. Relational calculus is a non-procedural query language. Relational Calculus exists in two forms.

1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)

Tuple Relational Calculus

In the tuple relational calculus, you will have to find tuples for which a predicate is true. The calculus is dependent on the use of tuple variables. A tuple variable is a variable that 'ranges over' a named relation: i.e., a variable who's only permitted values are tuples of the relation. The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation. The result of the relation can have one or more tuples. Tuple Relational Calculus is a non-procedural query language unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it.

Domain Relational Calculus

In contrast to tuple relational calculus, domain relational calculus uses list of attribute to be selected from the relation based on the condition. It is same as TRC, but differs by selecting the attributes rather than selecting whole tuples. In the tuple relational calculus, you have use variables that have a series of tuples in a relation. In the domain relational calculus, you will also use variables, but in this case, the variables take their values from domains of attributes rather than tuples of relations. In domain relational calculus, filtering is done

based on the domain of the attributes and not based on the tuple values. The second form of relation is known as Domain relational calculus.

- In domain relational calculus, filtering variable uses the domain of attributes. Domain relational calculus uses the same operators as tuple calculus.

- It uses logical connectives ∧ (and), ∨ (or) and ┐ (not).

- It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable.

7.7 CHARACTERISTICS OF SQL

SQL is both an easy-to-understand language and a comprehensive tool for managing data. Here are some of the major features of SQL and the market forces that have made it successful:

a) **Vendor Independence**

A SQL-based database and the programs that use it can be moved from one DBMS to another vendor's DBMS with minimal conversion effort and little retraining of personnel.

b) **SQL Standards**

In 1986, the American National Standards Institute (ANSI) and the International Standards Organization (ISO) published the first official standard for SQL which was expanded in 1989, 1992 and 1999. The evolving standards serve as an official stamp of approval for SQL and have speeded its market acceptance.

c) **Portability across Computer Systems**

SQL databases run on various computer systems, ranging from mainframes to stand-alone computers. SQL-based applications that begin on single-user or departmental server systems can be moved to larger server systems as they grow.

d) Relational Foundation

We already know that SQL is a language for relational databases. The relational database model and row/column structure make SQL simple and easy to understand. he relational model also has a strong theoretical foundation that has guided the evolution and implementation of relational databases.

e) Programmatic Database Access

SQL is also a database language used by programmers to write applications that access a database. The same SQL statements are used for both interactive and programmatic access, so the database access parts of a program can be tested first with interactive SQL and then embedded into the program.

7.8 SQL DATA TYPES

The basic **data types** available for attributes include numeric, character string, bit string, Boolean, date, and time.

■ **Numeric** data types include integer numbers of various sizes (INTEGER or INT, and SMALLINT) and floating-point (real) numbers of various precision (FLOAT or REAL, and DOUBLE PRECISION). Formatted numbers can be declared by using DECIMAL(i,j)—or DEC(i,j) or NUMERIC(i,j)—where i, the *precision*, is the total number of decimal digits and j, the *scale*, is the number of digits after the decimal point. The default for scale is zero, and the default for precision is implementation-defined.

■ **Character-string** data types are either fixed length—CHAR(n) or CHARACTER(n), where n is the number of characters—or varying length— VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n), where n is the maximum number of characters. When specifying a literal string value, it is placed between single quotation marks (apostrophes), and it is *case sensitive* (a distinction is made between uppercase and

lowercase). For fixedlength strings, a shorter string is padded with blank characters to the right. For example, if the value 'Sudha' is for an attribute of type CHAR(10), it is padded with five blank characters to become 'Sudha ' if needed. Padded blanks are generally ignored when strings are compared

■ **Bit-string** data types are either of fixed length n—BIT(n)—or varying length—BIT VARYING(n), where n is the maximum number of bits. The default for n, the length of a character string or bit string, is 1. Literal bit strings are placed between single quotes but preceded by a B to distinguish them from character strings; for example, B'10101'.5 Another variable-length bitstring data type called BINARY LARGE OBJECT or BLOB is also available

to specify columns that have large binary values, such as images. As for CLOB, the maximum length of a BLOB can be specified in kilobits (K), megabits (M), or gigabits (G). For example, BLOB(30G) specifies a maximum length of 30 gigabits.

■ A **Boolean** data type has the traditional values of TRUE or FALSE. In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is UNKNOWN.

■ The **DATE** data type has ten positions, and its components are YEAR, MONTH, and DAY in the form YYYY-MM-DD. The TIME data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form HH:MM:SS. Only valid dates and times should be allowed by the SQL implementation. This implies that months should be between 1 and 12 and dates must be between 1 and 31; furthermore, a date should be a valid date for the corresponding month. The < (less than) comparison can be used with dates or times—an *earlier* date is considered to be smaller than a later date, and similarly with time.

Some additional data types are discussed below. The list of types discussed here is not exhaustive; different implementations have added more data types to SQL.

■ A **timestamp** data type (TIMESTAMP) includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier. Literal values are represented by single quoted strings preceded by the keyword TIMESTAMP, with a blank space between data and time; for example, TIMESTAMP '2008-09-27 09:12:47.648302'.

■ Another data type related to DATE, TIME, and TIMESTAMP is the INTERVAL data type. This specifies an **interval**—a *relative value* that can be used to increment or decrement an absolute value of a date, time, or timestamp. Intervals are qualified to be either YEAR/MONTH intervals or DAY/TIME intervals. The format of DATE, TIME, and TIMESTAMP can be considered as a special type of string. Hence, they can generally be used in string comparisons by being **cast** (or **coerced** or converted) into the equivalent strings.

7.9 SQL LITERAL

Data Literal: A program source element that represents a data value. Data literals can be divided into multiple groups depending upon the type of the data it is representing and how it   is   representing.

1. Character String Literals are used to construct character strings, exact numbers, approximate numbers and data and time values. The syntax rules of character string literals are pretty simple:

- A character string literal is a sequence of characters enclosed by quote characters.
- The quote character is the single quote character "'".
- If "'" is part of the sequence, it needs to be doubled it as "''".

126

Examples of character string literals:

      'Hello'

      'world!'

      'Loews

      '123'

2. Hex String Literals are used to construct character strings and exact numbers. Hexadecimal literals consist of 0 to 62000 hexadecimal digits delimited by a matching pair of single quotes, where a hexadecimal digit is a character from 0 to 9, a to f, or A to F. The syntax rules for hex string literals are also very simple:

- A hex string literal is a sequence of hex digits enclosed by quote characters and prefixed with "x".

- The quote character is the single quote character "'".

Examples of hex string literals:

x '41423534'

x ' 57664873'

3. Numeric Literals are used to construct exact numbers and approximate numbers. A numeric literal is a string of 1 to 40 characters selected from the following:

• plus sign

• minus sign

• digits 0 through 9

• decimal point

Numeric literals are also referred to as numeric constants. Syntax rules of numeric literals are:

- A numeric literal can be written in signed integer form, signed real numbers without exponents, or real numbers with exponents.

127

Examples of numeric literals:

1

22.33

-345

4. Date and Time Literals are used to construct date and time values. The syntax of date and time literals are:

- A date literal is written in the form of "DATE 'yyyy-mm-dd'".

- A time literal is written in the form of "TIMESTAMP 'yyyy-mm-dd hh:mm:ss'".

Examples of data and time literals:

DATE '2013-07-15'

TIMESTAMP '2013-07-15 01:02:03'

7.10 SQL CONSTRAINTS

Constraints are the rules enforced on data columns on a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database. Constraints can either be column level or table level. Column level constraints are applied only to one column whereas, table level constraints are applied to the entire table.

Following are some of the most commonly used constraints available in SQL:

• NOT NULL Constraint: Ensures that a column cannot have a NULL value.

• DEFAULT Constraint: Provides a default value for a column when none is specified.

• UNIQUE Constraint: Ensures that all the values in a column are different.

• PRIMARY Key: Uniquely identifies each row/record in a database table.

• FOREIGN Key: Uniquely identifies a row/record in any another database table.

• CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.

• INDEX: Used to create and retrieve data from the database very quickly.

7.11 CHECK YOUR PROGRESS

1. A relation in a relational database is based on a relational schema, which consists of number of ………………… .

2. …………………is a Relational Data Base Management System.

3. Rows of the relation are referred to as ………………… of the relation.

4. The relational model was designed by the IBM research scientist and mathematician, Dr. …………………..

5. The ………………… is the only data structure used in the relational data model to represent both entities and relationships between them.

6. Does the normal forms never removes anomalies?

7. Is each attribute of the column are drawn from the set of values known as domain?

8. Rename operator is represented by_____.

9. The union operator comes under which type? Unary or binary.

10. Rename operator comes under which category? Unary or binary.

11. TRC stands for_____?

12. SQL is a combination of a _____ language and a _____ language.

13. SQL stands for_____.

14. SQL was developed by _____ in the late 1970's.

7.12 SUMMARY

A DBMS is a software used to store and manage data. The DBMS was introduced during 1960's to store any data. It also offers manipulation of the data like insertion, deletion, and updating of the data. DBMS system also performs the functions like defining, creating, revising and controlling the database. It is specially designed to create and maintain data and enable the individual business application to extract the desired data.

Relational Database Management System (RDBMS) is an advanced version of a DBMS system. It came into existence during 1970's. RDBMS system also allows the organization to access data more efficiently then DBMS. RDBMS is a software system which is used to store only data which need to be stored in the form of tables. In this kind of system, data is managed and stored in rows and columns which is known as tuples and attributes. RDBMS is a powerful data management system and is widely used across the world. The goal of a relational database design is to generate a set of relation schema that allows us to store information without unnecessary redundancy and also to retrieve information easily. A database system is an integrated collection of related files, along with details of interpretation of the data contained therein. DBMS is a software that allows access to data contained in a database. The objective of the DBMS is to provide a convenient and effective method of defining, storing and retrieving the information contained in the database. The DBMS interfaces with application programs so that the data contained in the database can be used by multiple applications and users. The DBMS allows these users to access and manipulate the data contained in the database in a convenient and effective manner. In addition the DBMS exerts centralized control of the database, prevents unauthorized users from accessing the data and ensures privacy of data.

In Relational database model, a table is a collection of data elements organised in terms of rows and columns. A table is also considered as a convenient representation of relations. But a table can have duplicate row of data while a true relation cannot have duplicate data. Table is the simplest form of data storage. All data stored in the tables are provided by an RDBMS. Ensures that all data stored are in the form of rows and columns. Facilitates primary key, which helps in unique identification of the rows. Index creation for retrieving data at a higher speed. Facilitates a common column to be shared amid two or more tables. Major components of RDBMS are Table, Record or Tuple, Field, Domain, Instance, Schema, Keys. Relational database stores data in tables. Tables are organized into columns, and each column stores one type of data (integer, real number, character strings, date). The data for a single "instance" of a table is stored as a row. Many relational database systems have an option of using the SQL (Structured Query Language) for querying and maintaining the database.

In this chapter we presented two formal languages for the relational model of data. They are used to manipulate relations and produce new relations as answers to queries. We discussed the relational algebra and its operations, which are used to specify a sequence of operations to specify a query. Then we introduced two types of relational calculi called tuple calculus and domain calculus; they are declarative in that they specify the result of a query without specifying how to produce the query result. The data for a single "instance" of a table is stored as a row. Many relational database systems have an option of using the SQL (Structured Query Language) for querying and maintaining the database.

Relational calculus is a non-procedural query language. It uses mathematical predicate calculus instead of algebra. It provides the description about the query to get the result whereas relational algebra gives the method to get the result. It informs the system what to

do with the relation, but does not inform how to perform it. For example, steps involved in listing all the students who attend 'Database' Course in relational algebra would be

- SELECT the tuples from COURSE relation with COURSE_NAME = 'DATABASE'

- PROJECT the COURSE_ID from above result

- SELECT the tuples from STUDENT relation with COUSE_ID resulted above.

Whereas, SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database. This tutorial will provide you with the instruction on the basics of each of these commands as well as allow you to put them to practice using the SQL Interpreter.

7.13 KEYWORDS

- **Domain-** A domain describes the set of possible values for a given attribute, and can be considered a constraint on the value of the attribute. Mathematically, attaching a domain to an attribute means that any value for the attribute must be an

element of the specified set. The character string *"ABC"*, for instance, is not in the integer domain, but the integer value *123* is.

- **Constraints-** Constraints make it possible to further restrict the domain of an attribute. For instance, a constraint can restrict a given integer attribute to values between 1 and 10.

- **Tuple-** A data set representing a single item.

- **Column-** A labeled element of a tuple, e.g. "Address" or "Date of birth"

- **Table-** A set of tuples sharing the same attributes; a set of columns and rows

- **View-** Any set of tuples; a data report from the RDBMS in response to a query

7.14 SELF-ASSESSMENT TEST

1. Explain the following terms

   i)   Domain

   ii)   Tuple

   iii) Relation

   iv)   Attribute

2. Explain difference between DBMS and RDBMS.

3. Why relational data model is so popular?

4. What are record based models?

5. How RDBMS stores its data?

6. Explain the role of relational algebra in relational database.

7. What are the different type of relational algebra? Discuss in detail

8. List the data types that are allowed for SQL attributes.

9. How does SQL allow implementation of the entity integrity and referential integrity constraints described in Chapter 3? What about referential triggered actions?

10. How do the relations (tables) in SQL differ from the relations defined formally in relation algebra? Discuss the other differences in terminology. Why does SQL allow duplicate tuples in a table or in a query result?

7.15 ANSWERS TO CHECK YOUR PROGRESS

1. Attributes

2. RDBMS

3. Tuples

4. E.F Codd

5. Relation

6. False

7. True

8. ρ

9. Binary

10. Unary

11. Tuple Relational Calculus

12. Data manipulation

13. Structured Query language

14. IBM

7.16 REFERENCES / SUGGESTED READINGS

- C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.

- Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.

- Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.

- https://www.geeksforgeeks.org/difference-between-rdbms-and-dbms/

- https://en.wikipedia.org/wiki/Relational_database

- https://www.javatpoint.com/what-is-rdbms

- https://searchdatamanagement.techtarget.com/definition/RDBMS-relational-database-management-system

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	AUTHOR: DR. DEEPAK NANDAL
LESSON NO. 8	VETTER:
RELATIONAL DATABASE DESIGN	

STRUCTURE

8.0 LEARNING OBJECTIVE

- To understand the concepts of Functional Dependency, Normalization.

- To learn the purpose of functional dependency.

- To discuss and Update the anomalies in functional dependency.

- To understand the data redundancy in Functional Dependency.

8.1 INTRODUCTION

Each relation schema consists of a number of attributes, and the relational database schema consists of a number of relation schemas. So far, we have assumed that attributes are grouped to form a relation schema by using the common sense of the database designer or by mapping a database schema design from a conceptual data model such as the ER or enhanced ER (EER) or some other conceptual data model. These models make the designer identify entity types and relationship types and their respective attributes, which leads to a natural and logical grouping of the attributes into relations when the mapping procedures. We have not developed any measure of appropriateness or "goodness" to measure the quality of the design, other than the intuition of the designer. In this chapter we discuss some of the theory that has been developed with the goal of evaluating relational schemas for design quality-that is, to measure formally why one set of groupings of attributes into relation schemas is better than another.

There are two levels at which we can discuss the "goodness" of relation schemas. The first is the logical (or conceptual) level-how users interpret the relation schemas and the meaning of their attributes. Having good relation schemas at this level enables users to understand clearly the meaning of the data in the relations, and hence to formulate their queries correctly. The second is the implementation (or storage) level-how the tuples in a

base relation are stored and updated. This level applies only to schemas of base relations-which will be physically stored as files-whereas at the logical level we are interested in schemas of both base relations and views (virtual relations). The relational database design theory developed in this chapter applies mainly to *base relations,* although some criteria of appropriateness also apply to views. As with many design problems, database design may be performed using two approaches: bottom-up or top-down. A bottom-up design methodology (also called *design by synthesis)* considers the basic relationships *among individual attributes* as the starting point and uses those to construct relation schemas. This approach is not very popular in practice. Because it suffers from the problem of having to collect a large number of binary relationships among attributes as the starting point. In contrast, a top-down design methodology (also called *design by analysis)* starts with a number of groupings of attributes into relations that exist together naturally, for example, on an invoice, a form, or a report. The relations are then analysed individually and collectively, leading to further decomposition until all desirable properties are met. The theory described in this chapter is applicable to both the top-down and bottom-up design approaches, but is more practical when used with the top-down approach. We define the concept of *functional dependency,* a formal constraint among attributes that is the main tool for formally measuring the appropriateness of attribute groupings into relation schemas. Properties of functional dependencies are also studied and analysed. Then properties of functional dependencies are also studied and analysed. Then we will discuss the how functional dependencies can be used to group attributes into relation schemas that are in a *normal form.* A relation schema is in a normal form when it satisfies certain desirable properties. The process of *normalization* consists of analysing relations to meet increasingly more stringent normal forms leading to progressively better groupings of

attributes. Normal forms are specified in terms of functional dependencies-which are identified by the database designer-and key attributes of relation schemas.

When developing the schema of a relational database, one of the most important aspects to be taken into account is to ensure that the duplication is minimized. This is done for 2 purposes:

- Reducing the amount of storage needed to store the data.

- Avoiding unnecessary data conflicts that may creep in because of multiple copies of the same data getting stored.

8.2 DEFINITION

Functional Dependency: A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has n attributes AI, A2, ……, An; let us think of the whole database as being described by a single universal relation schema R = {A1, A2, A3…….., An). 6We do not imply that we will actually store the database as a single universal table; we use this concept only in developing the formal theory of data dependencies

Definition: A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has n attributes A1, A2, ..., An. If we think of the whole database as being described by a single universal relation schema R = {A1, A2, ... , An}. A functional dependency (FD) is a relationship between two attributes, typically between the PK and other non-key attributes within a table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y.

It determines the relation of one attribute to another attribute in a database management system (DBMS) system. Functional dependency helps you to maintain the quality of data in the database. A functional dependency is denoted by an arrow →. The functional dependency of X on Y is represented by X → Y. Functional Dependency plays a vital role to find the difference between good and bad database design.

Example:

Employee number	Employee Name	Salary	City
1	Dana	50000	San Francisco
2	Francis	38000	London
3	Andrew	25000	Tokyo

In this example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc. By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.

Definition of Normalization:

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant (useless) data.

- Ensuring data dependencies make sense i.e. data is logically stored.

8.3 PURPOSE OF FUNCTIONAL DEPENDANCY

A functional dependency, denoted by X $\longrightarrow$ Y, between two sets of attributes X and Y that are subsets of R, such that any two tuples t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[Y].

This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; we say that the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component. We say that there is a functional dependency from X to Y, or that Y is functionally dependent on X.

Functional dependency is represented as FD or f.d. The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side.

X functionally determines Y in a relation schema R if, and only if, whenever two tuples of r(R) agree on their X-value, they must necessarily agree on their Y-value. If a constraint on R states that there cannot be more than one tuple with a given X-value in any relation instance r(R)—that is, X is a candidate key of R— this implies that X $\longrightarrow$ Y for any subset of attributes Y of R.

If X is a candidate key of R, then $X \rightarrow R.$

If $X \rightarrow Y$ in R, this does not imply that $Y \rightarrow X$ in R.

A functional dependency is a property of the semantics or meaning of the attributes. Whenever the semantics of two sets of attributes in R indicate that a functional dependency should hold, we specify the dependency as a constraint.

Legal Relation States:

Relation extensions r(R) that satisfy the functional dependency constraints are called legal relation states (or legal extensions) of R. Functional dependencies are used to describe further a relation schema R by specifying constraints on its attributes that must hold at all times. Certain FDs can be specified without referring to a specific relation, but as a property of those attributes given their commonly understood meaning.

For example, {State, Driver_license_number} $\rightarrow$ Ssn should hold for any adult in the United States and hence should hold whenever these attributes appear in a relation. Consider the relation schema EMP_PROJ from the semantics of the attributes and the relation, we know that the following functional dependencies should hold:

a. Ssn$\rightarrow$Ename

b. Pnumber $\rightarrow$ {Pname, Plocation}

c. {Ssn, Pnumber}$\rightarrow$Hours

A functional dependency is a property of the relation schema R, not of a particular legal relation state r of R. Therefore, an FD cannot be inferred automatically from a given relation extension r but must be defined explicitly by someone who knows the semantics of the attributes of R.

Example 1: For the relation Student(studentID, name, DateOfBirth, phoneNumber), assuming

each student has only one name, then the following functional dependency holds

{studentID} →{name, DateOfBirth}

However, assuming a student may have multiple phone numbers, then the FD

{studentID} →   {phoneNumber}

does not hold for the table.

By convention, we often omit the curly braces { } for the set, and write the first functional

dependency in Example 1 as

studentID→name, DateOfBirth.

Note that the above FD can also be written equivalently into the two FDs below:

studentID →   name

studentID→DateOfBirth

8.4 DATA REDUNDANCY IN FUNCTIONAL DEPENDENCY

Data redundancy is a condition created within a database or data storage technology in which the same piece of data is held in two separate places. This can mean two different fields within a single database, or two different spots in multiple software environments or platforms. Data redundancy occurs when the same piece of data is stored in two or more separate places and is a common occurrence in many businesses. As more companies are moving away from siloed data to using a central repository to store information, they are finding that their database is filled with inconsistent duplicates of the same entry. Although it can be challenging to reconcile — or even benefit from — duplicate data entries, understanding how to reduce and track data redundancy efficiently can help mitigate long-term inconsistency issues for your business.

Sometimes data redundancy happens by accident while other times it is intentional. Accidental data redundancy can be the result of a complex process or inefficient coding while intentional data redundancy can be used to protect data and ensure consistency — simply by leveraging the multiple occurrences of data for disaster recovery and quality checks. If data redundancy is intentional, it's important to have a central field or space for the data. This allows you to easily update all records of redundant data when necessary.

Four major advantages of Data Redundancy:

Although data redundancy sounds like a negative event, there are many organizations that can benefit from this process when it's intentionally built into daily operations.

1. Alternative data backup method

Backing up data involves creating compressed and encrypted versions of data and storing it in a computer system or the cloud. Data redundancy offers an extra layer of protection and reinforces the backup by replicating data to an additional system. It's often an advantage when companies incorporate data redundancy into their disaster recovery plans.

2. Better data security

Data security relates to protecting data, in a database or a file storage system, from unwanted activities such as cyberattacks or data breaches. Having the same data stored in two or more separate places can protect an organization in the event of a cyberattack or breach — an event which can result in lost time and money, as well as a damaged reputation.

3. Faster data access and updates

When data is redundant, employees enjoy fast access and quick updates because the necessary information is available on multiple systems. This is particularly important for customer service-based organizations whose customers expect promptness and efficiency.

4. Improved data reliability

Data that is reliable is complete and accurate. Organizations can use data redundancy to double check data and confirm it's correct and completed in full — a necessity when interacting with customers, vendors, internal staff, and others.

Although there are noteworthy advantages of intentional data redundancy, there are also several significant drawbacks when organizations are unaware of its presence.

Possible data inconsistency

Data redundancy occurs when the same piece of data exists in multiple places, whereas data inconsistency is when the same data exists in different formats in multiple tables. Unfortunately, data redundancy can cause data inconsistency, which can provide a company with unreliable and/or meaningless information.

Increase in data corruption

Data corruption is when data becomes damaged as a result of errors in writing, reading, storage, or processing. When the same data fields are repeated in a database or file storage system, data corruption arises. If a file gets corrupted, for example, and an employee tries to open it, they may get an error message and not be able to complete their task.

Increase in database size

Data redundancy may increase the size and complexity of a database — making it more of a challenge to maintain. A larger database can also lead to longer load times and a great deal of headaches and frustrations for employees as they'll need to spend more time completing daily tasks.

Increase in cost

When more data is created due to data redundancy, storage costs suddenly increase. This can be a serious issue for organizations who are trying to keep costs low in order to increase

profits and meet their goals. In addition, implementing a database system can become more expensive.

There are four informal measures of quality for relation schema design.

- Semantics of the attributes.

- Reducing the redundant values in tuples.

- Reducing the null values in tuples.

- Disallowing the possibility of generating spurious tuples.

Semantics of the Relation Attributes- The easier it is to explain the semantics of the relation, the better the relation schema design will be.

GUIDELINE 1: Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation. Intuitively, if a relation schema corresponds to one entity type or one relationship type, the meaning tends to be clear. Otherwise, the relation corresponds to a mixture of multiple entities and relationships and hence becomes semantically unclear.

Example: A relation involves two entities- poor design.

EMP DEPT

ENAME	SSN	BDATE	ADDREESS	DNUMBER	DNAME	DMGRSSN

8.5 UPDATE ANOMALIES

Consider the two relation schemas EMP_LOCS and EMP_PROJl in Figure 8.1 a, A tuple in EMP_LOCS means that the employee whose name is ENAME works on some project whose location is PLOCATION.

EMP_LOCS

ENAME	PLOCATION

p.k.

EMP_PROJ1

SSN	PNUMBER	HOURS	PNAME	PLOCATION

p.k.
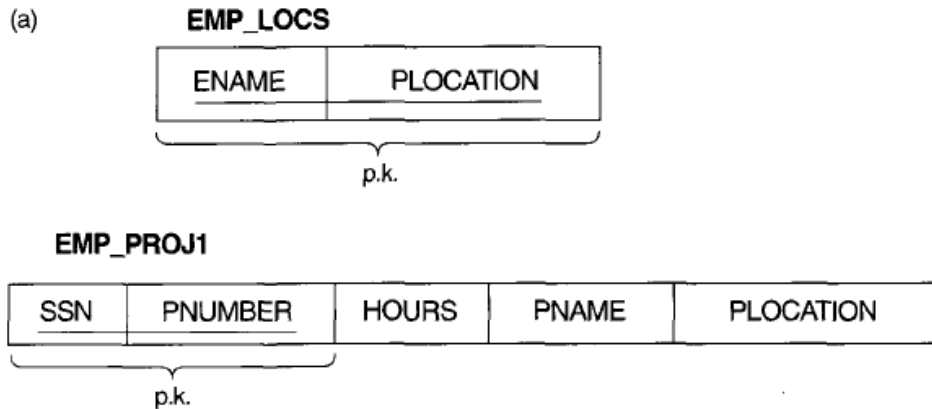
Figure 8.1 (a): The two relation schemas EMP_LOCS and EMP_PROJ1

(b)

EMP_LOCS

ENAME	PLOCATION
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Borg, James E.	Houston

EMP_PROJ1

SSN	PNUMBER	HOURS	PNAME	PLOCATION
123456789	1	32.5	Product X	Bellaire
123456789	2	7.5	Product Y	Sugarland
666884444	3	40.0	Product Z	Houston
453453453	1	20.0	Product X	Bellaire
453453453	2	20.0	Product Y	Sugarland
333445555	2	10.0	Product Y	Sugarland
333445555	3	10.0	Product Z	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston
999887777	30	30.0	Newbenefits	Stafford
999887777	10	10.0	Computerization	Stafford
987987987	10	35.0	Computerization	Stafford
987987987	30	5.0	Newbenefits	Stafford
987654321	30	20.0	Newbenefits	Stafford
987654321	20	15.0	Reorganization	Houston
888665555	20	null	Reorganization	Houston

147

Figure 8.1 (b) The result of projecting the extension of EMP_PROJ form Figure 8.1(a) on the relations EMP_LOCS and EMP_PROJ1

Update anomalies for base relations EMP DEPT and EMP PROJ in Figure 8.1

- ➢ Insertion anomalies: For EMP DEPT relation in Figure 8.1

  - To insert a new employee tuple, we need to make sure that the values of attributes DNUMBER, DNAME, and DMGRSSN are consistent to other employees (tuples) in EMP DEPT.

  - It is difficult to insert a new department that has no employees as yet in the EMP DEPT relation.

- ➢ Deletion anomalies: If we delete from EMP DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database.

- ➢ Modification anomalies: If we update the value of MGRSSN in a particular department, we must to update the tuples of all employees who work in that department; otherwise, the database will become inconsistent.

GUIDELINE 2: Design the base relation schemas so that no insertion, deletion, or Modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure the programs that update the database will operate correctly. It is advisable to use anomaly-free base relations and to specify views that include the JOINs for placing together the attributes frequently referenced to improve the performance.

8.6 CHECK YOUR PROGRESS

1. We can use the following three rules to find logically implied functional dependencies. This collection of rules is called

a) Axioms

b) Armstrong's axioms

c) Armstrong

d) Closure

2. Which of the following is not Armstrong's Axiom?

a) Reflexivity rule

b) Transitivity rule

c) Pseudo transitivity rule

d) Augmentation rule

3. The relation employee(ID,name,street,Credit,street,city,salary) is decomposed into

```
employee1 (ID, name)
employee2 (name, street, city, salary)
```

This type of decomposition is called

a) Lossless decomposition

b) Lossless-join decomposition

c) All of the mentioned

d) None of the mentioned

4. Inst_dept (ID, name, salary, dept name, building, budget) is decomposed into

```
instructor (ID, name, dept name, salary)
department (dept name, building, budget)
```

This comes under

a) Lossy-join decomposition

b) Lossy decomposition

c) Lossless-join decomposition

d) Both Lossy and Lossy-join decomposition

5. Suppose relation R(A,B,C,D,E) has the following functional dependencies:

```
A -> B

B -> C

BC -> A

A -> D

E -> A

D -> E
```

Which of the following is not a key?

a) A

b) E

c) B, C

d) D

8.7 SUMMARY

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn. Functional dependency is represented by an arrow sign ($\rightarrow$) that is, X$\rightarrow$Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side. Database normalization is the process of efficiently organizing data in a database so that redundant data is eliminated. This process can ensure that all of a company's data looks and reads similarly across all records. By implementing data normalization, an organization standardizes data fields such as customer names, addresses, and phone numbers. Normalizing data involves organizing the columns and tables of a database to make sure their dependencies are enforced correctly. The "normal form" refers to the set of rules or normalizing data, and a database is known as "normalized" if it's free of delete, update, and insert anomalies. When it comes to normalizing data, each company

has their own unique set of criteria. Therefore, what one organization believes to be "normal," may not be "normal" for another organization. For instance, one company may want to normalize the state or province field with two digits, while another may prefer the full name. Regardless, database normalization can be the key to reducing data redundancy across any company.

Efficient data redundancy is possible. Many organizations like home improvement companies, real estate agencies, and companies focused on customer interactions have customer relationship management (CRM) systems. When a CRM system is integrated with another business software like an accounting software that combines customer and financial data, redundant manual data is eliminated, leading to more insightful reports and improved customer service. Database management systems are also used in a variety of organizations. They receive direction from a database administrator (DBA) and allow the system to load, retrieve, or change existing data from the systems. Database management systems adhere to the rules of normalization, which reduces data redundancy. Hospitals, nursing homes, and other healthcare entities use database management systems to generate reports that provide useful information for physicians and other employees. When data redundancy is efficient and does not lead to data inconsistency, these systems can alert healthcare providers of rises in denial claim rates, how successful a certain medication is, and other important pieces of information.

8.8 KEYWORDS

- **AXIOM -** Axioms is a set of inference rules used to infer all the functional dependencies on a relational database.

- **DECOMPOSITION-** It is a rule that suggests if you have a table that appears to contain two entities which are determined by the same primary key then you should consider breaking them up into two different tables.

- **DEPENDENT -** It is displayed on the right side of the functional dependency diagram.

- **UNION -** It suggests that if two tables are separate, and the PK is the same, you should consider putting them. Together.

- **DETERMINANT -** It is displayed on the left side of the functional dependency Diagram.

5.9 SELF-ASSESSMENT TEST

1. Explain the Functional Dependency in detail.
2. Discuss how to Insert and Update anomaly in functional dependency.
3. What is the key role of Normalization?
4. How normalization and functional dependency are related to each other?
5. Discuss with example the redundancy in functional dependency.

8.10 ANSWERS TO CHECK YOUR PROGRESS

1. B
2. C
3. D
4. D
5. C

8.11 REFERENCES / SUGGESTED READINGS

- C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.

- Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.

- Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.

- https://opentextbc.ca/dbdesign01/chapter/chapter-11-functional-dependencies/

- https://hackr.io/blog/dbms-normalization

- https://www.guru99.com/database-normalization.html

- https://www.javatpoint.com/dbms-normalization

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	AUTHOR: DR. DEEPAK NANDAL
LESSON NO. 9	VETTER:
NORMAL FORMS	

STRUCTURE

9.0 LEARNING OBJECTIVE

- To understand the concepts of Anomalies in Database

- Learn how to update, insert and Delete anomalies

- To understand the concept of Normalization in removing anomalies in database

- Study and learn decomposition methods and different forms of Normalization

9.1 INTRODUCTION

NORMALIZATION is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically. The inventor of the relational model Edgar Codd proposed the theory of normalization with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this in table 9.1:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

Table 9.1: Un-Normalized Data in a table

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data.

9.2   DEFINTION OF NORMALIZATION

Database Normalization is a technique that helps in designing the schema of the database in an optimal manner so as to ensure the above points. The core idea of database normalization is to divide the tables into smaller sub tables and store pointers to data rather than replicating it. For a better understanding of what we just said, here is a simple DBMS Normalization example:

To understand (RDBMS) normalization in the database with example tables, let's assume that we are supposed to store the details of courses and instructors in a university. Here is what a sample database could look like:

Course code	Course venue	Instructor Name	Instructor's phone number
CS101	Lecture Hall 20	Prof. George	+91 6514821924
CS152	Lecture Hall 21	Prof. Atkins	+91 6519272918
CS154	CS Auditorium	Prof. George	+91 6514821924

Here, the data basically stores the course code, course venue, instructor name, and instructor's phone number. At first, this design seems to be good. However, issues start to develop once we need to modify information. For instance, suppose, if Prof. George changed his mobile number. In such a situation, we will have to make edits in 2 places.

What if someone just edited the mobile number against CS101, but forgot to edit it for CS154? This will lead to stale/wrong information in the database.

This problem, however, can be easily tackled by dividing our table into 2 simpler tables:

Table 1 (Instructor):

1. Instructor ID
2. Instructor Name
3. Instructor mobile number

Table 2 (Course):

- Course code
- Course venue
- Instructor ID

Now, our data will look like the following:

Table 1 (Instructor):

Insturctor's ID	Instructor's name	Instructor's number
1	Prof. George	+1 6514821924
2	Prof. Atkins	+1 6519272918

Table 2 (Course):

Course code	Course venue	Instructor ID
CS101	Lecture Hall 20	1

CS152	Lecture Hall 21	2
CS154	CS Auditorium	1

Basically, we store the instructors separately and in the course table, we do not store the entire data of the instructor. We rather store the ID of the instructor. Now, if someone wants to know the mobile number of the instructor, he/she can simply look up the instructor table. Also, if we were to change the mobile number of Prof. George, it can be done in exactly one place. This avoids the stale/wrong data problem.

Further, if you observe, the mobile number now need not be stored 2 times. We have stored it at just 1 place. This also saves storage. This may not be obvious in the above simple example. However, think about the case when there are hundreds of courses and instructors and for each instructor, we have to store not just the mobile number, but also other details like office address, email address, specialization, availability, etc. In such a situation, replicating so much data will increase the storage requirement unnecessarily. The above is a simplified example of how database normalization works. We will now more formally study it.

- Normalization is the process of organizing the data in the database.

- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.

- Normalization divides the larger table into the smaller table and links them using relationship.

- The normal form is used to reduce redundancy from the database table.

Normalization rules are divided into the following normal forms:

1. First Normal Form

2. Second Normal Form

3. Third Normal Form

4. BCNF

5. Fourth Normal Form

9.3 DECOMPOSITION

Definition. The **normal form** of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized. Normal forms, when considered *in isolation* from other factors, do not guarantee a good database design. It is generally not sufficient to check separately that each relation schema in the database is, say, in BCNF or 3NF. Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:

- The **non-additive join or lossless join property**, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition.

- The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting after decomposition.

In fact Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties stated previously. The practical utility of these normal forms becomes questionable when the constraints on which they are based are rare,

and hard to understand or to detect by the database designers and users who must discover these constraints. Thus, database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF. Another point worth noting is that the database designers *need not* normalize to the highest possible normal form. Relations may be left in a lower normalization status, such as 2NF.

9.4 FIRST NORMAL FORM (1NF)

First normal form (1NF) is now considered to be part of the formal definition of a relation in the basic (flat) relational model; historically, it was defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domain of an attribute must include only *atomic* (simple, indivisible) *values* and that the value of any attribute in a tuple must be a *single value* from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a *single tuple*. In other words, 1NF disallows *relations within relations* or *relations as attribute values within tuples*. The only attribute values permitted by 1NF are single **atomic** (or **indivisible**) **values**.

Consider the DEPARTMENT relation schema shown in Figure below, whose primary key is Dnumber, and suppose that we extend it by including the Dlocations attribute as shown in Figure. We assume that each department can have *a number of* locations.. As we can see, this is not in 1NF because Dlocations is not an atomic attribute.

DEPARTMENT

Dnumber	Dname	Dmgr_SSN	DLocation
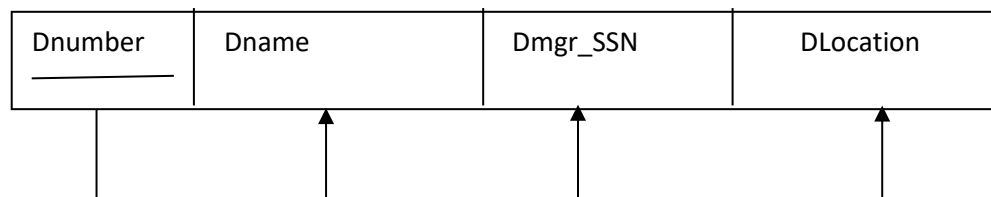
There are three main techniques to achieve first normal form for such a relation:

- Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary key of this relation is the combination {Dnumber, Dlocation},

- Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT,

- If a *maximum number of values* is known for the attribute—for example, if it is known that *at most three locations* can exist for a department—replace the Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3. This solution has the disadvantage of introducing *NULL values* if most departments have fewer than three locations.

Of the three solutions above, the first is generally considered best because it does not suffer from redundancy and it is completely general, having no limit placed on a maximum number of values.

Example:

The First normal form simply says that each cell of a table should contain exactly one value. Let us take an example. Suppose we are storing the courses that a particular instructor takes, we can store it like this:

Instructor's name	Course code
Prof. George	(CS101, CS154)
Prof. Atkins	(CS152)

Here, the issue is that in the first row, we are storing 2 courses against Prof. George. This isn't the optimal way since that's now how SQL databases are designed to be used. A better method would be to store the courses separately. For instance:

Instructor's name	Course code
Prof. George	CS101
Prof. George	CS154
Prof. Atkins	CS152

This way, if we want to edit some information related to CS101, we do not have to touch the data corresponding to CS154. Also, observe that each row stores unique information. There is no repetition. This is the First Normal Form.

9.5 Second Normal Form (2NF)

Second normal form (2NF) is based on the concept of *full functional dependency.* A functional dependency $X \rightarrow Y$ is a **full functional dependency** if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute $A \varepsilon X$, $(X - \{A\})$ does *not* functionally determine Y. A functional dependency $X \rightarrow Y$ is a **partial dependency** if some attribute $A \varepsilon X$ can be removed from X and the dependency still holds; that is, for some $A \varepsilon X$, $(X - \{A\}) \rightarrow Y$.

Definition. A relation schema R is in 2NF if every nonprime attribute A in R is *fully functionally dependent* on the primary key of R. The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains a single attribute, the test need not be applied at all.

General Definition of 2NF

A relation schema R is in **second normal form (2NF)** if every nonprime attribute A in R is not partially dependent on *any* key of R.

If a relation schema is not in 2NF, it can be *second normalized* or *2NF normalized* into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent. The following example shows how we can decompose a relation not in 2NF into three relations which are now in 2NF. For a table to be in second normal form, the following 2 conditions are to be met:

1. The table should be in the first normal form.

2. The primary key of the table should compose of exactly 1 column.

The first point is obviously straightforward since we just studied 1NF. Let us understand the first point - 1 column primary key. Well, a primary key is a set of columns that uniquely identifies a row. Basically, no 2 rows have the same primary keys.

(a) Relation not in 2NF

SSN	Pnumber	Hours	Ename	Pname	PLocation

FD1

FD2

FD3
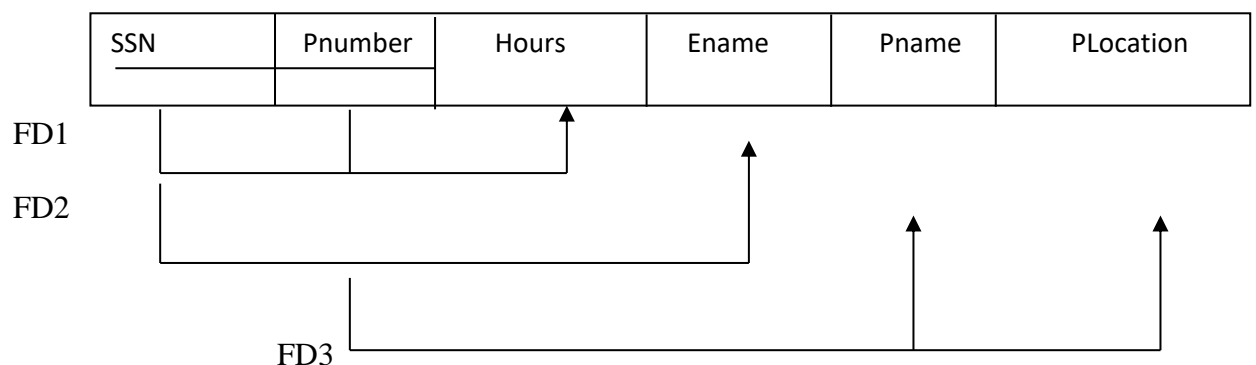
(b)   Relation decomposed in 2NF

SSN	Pnumber	Hours

SSN	Ename

Pnumber	Pname	PLocation

FD1   FD2   FD3
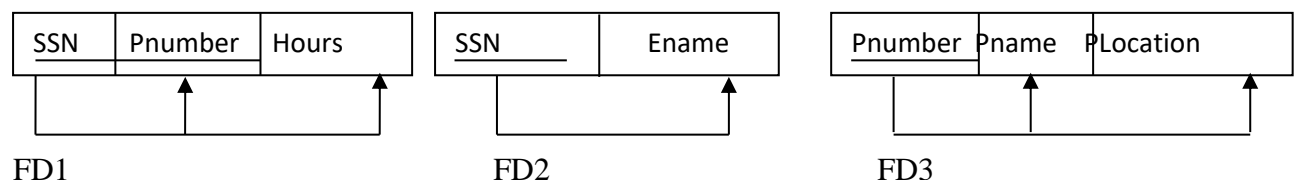
Example:

Course code	Course venue	Instructor Name	Instructor's phone number
CS101	Lecture Hall 20	Prof. George	+91 6514821924
CS152	Lecture Hall 21	Prof. Atkins	+91 6519272918
CS154	CS Auditorium	Prof. George	+91 6514821924

Here, in this table, the course code is unique. So, that becomes our primary key. Let us take another example of storing student enrollment in various courses. Each student may enroll in multiple courses. Similarly, each course may have multiple enrollments. A sample table may look like this (student name and course code):

Student name	Course code
Rahul	CS152
Rajat	CS101
Rahul	CS154
Raman	CS101

Here, the first column is the student name and the second column is the course taken by the student. Clearly, the student name column isn't unique as we can see that there are 2 entries

corresponding to the name 'Rahul' in row 1 and row 3. Similarly, the course code column is not unique as we can see that there are 2 entries corresponding to course code CS101 in row 2 and row 4. However, the tuple (student name, course code) is unique since a student cannot enroll in the same course more than once. So, these 2 columns when combined form the primary key for the database. As per the second normal form definition, our enrollment table above isn't in the second normal form. To achieve the same (1NF to 2NF), we can rather break it into 2 tables:

Students:

Student name	Enrolment number
Rahul	1
Rajat	2
Raman	3

Here the second column is unique and it indicates the enrollment number for the student. Clearly, the enrollment number is unique. Now, we can attach each of these enrollment numbers with course codes**.**

Courses:

Course code	Enrolment number
CS101	2
CS101	3
CS152	1
CS154	1

These 2 tables together provide us with the exact same information as our original table.
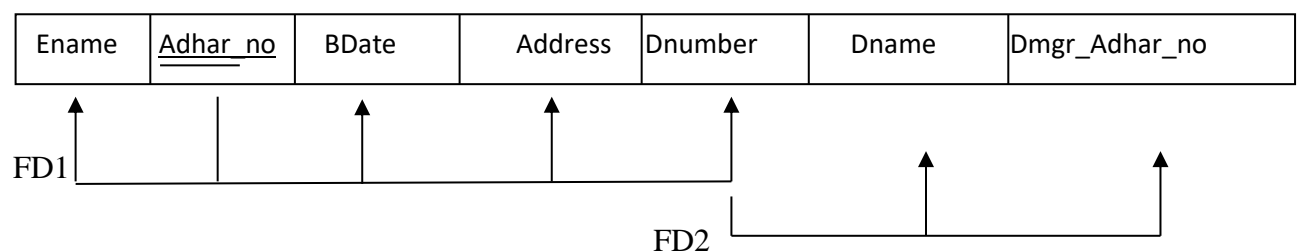
9.6 Third Normal Form (3NF)

Third normal form (3NF) is based on the concept of *transitive dependency*. A functional dependency $X{\rightarrow}Y$ in a relation schema R is a **transitive dependency** if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R, and both $X{\rightarrow}Z$ and $Z{\rightarrow}Y$ hold.

Definition. According to Codd's original definition, a relation schema R is in **3NF** if it satisfies 2NF *and* no nonprime attribute of R is transitively dependent on the primary key.

The relation schema EMP_DEPT in Figure (a) below is in 2NF, since no partial dependencies

on a key exist. However, EMP_DEPT is not in 3NF because of the transitive dependency of Dmgr_Adhar_No. (and also Dname) on Adhar_No. via Dnumber. We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas shown in Figure (b). Intuitively, we see that the two relations represent independent entity facts about employees and departments :

(a)

(b)

167
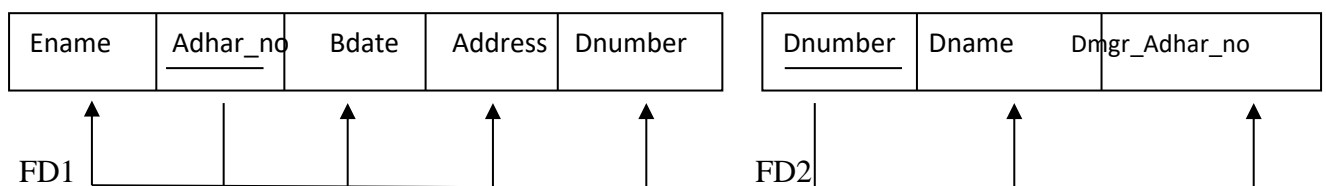
General Definition

Definition. A relation schema R is in **third normal form (3NF)** if, whenever a *nontrivial* functional dependency $X{\rightarrow}A$ holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R. A relation schema R violates the general definition of 3NF if a functional dependency $X \rightarrow A$ holds in R that does not meet either condition—meaning that it violates *both* conditions (a) and (b) of 3NF. This can occur due to two types of problematic functional dependencies:

■ A nonprime attribute determines another nonprime attribute. Here we typically have a transitive dependency that violates 3NF.

■ A proper subset of a key of R functionally determines a nonprime attribute.

Here we have a partial dependency that violates 3NF (and also 2NF). Therefore, we can state a **general alternative definition of 3NF** as follows:

Alternative Definition. A relation schema R is in 3NF if every nonprime attribute of R meets both of the following conditions:

■ It is fully functionally dependent on every key of R.

■ It is nontransitively dependent on every key of R.

Example:

➢ Before we delve into details of third normal form, let us again understand the concept of a functional dependency on a table. Column A is said to be functionally dependent on column B if changing the value of A may require a change in the value of B. As an example, consider the following table:

Course code	Course venue	Instructor's name	Department
MA214	Lecture Hall 18	Prof. George	CS Department
ME112	Auditorium building	Prof. John	Electronics Department

Here, the department column is dependent on the professor name column. This is because if in a particular row, we change the name of the professor, we will also have to change the department value. As an example, suppose MA214 is now taken by Prof. Ronald who happens to be from the Mathematics department, the table will look like this:

Course code	Course venue	Instructor's name	Department
MA214	Lecture Hall 18	Prof. Ronald	Mathematics Department
ME112	Auditorium building	Prof. John	Electronics Department

Here, when we changed the name of the professor, we also had to change the department column. This is not desirable since someone who is updating the database may remember to change the name of the professor, but may forget updating the department value. This can cause inconsistency in the database.

Third normal form avoids this by breaking this into separate tables:

Course code	Course venue	Instructor's ID
MA214	Lecture Hall 18	1
ME112	Auditorium building,	2

Here, the third column is the ID of the professor who's taking the course.

Instructor's ID	Instructor's Name	Department
1	Prof. Ronald	Mathematics Department
2	Prof. John	Electronics Department

Here, in the above table, we store the details of the professor against his/her ID. This way, whenever we want to reference the professor somewhere, we don't have to put the other details of the professor in that table again. We can simply use the ID.

Therefore, in the third normal form, the following conditions are required:

- The table should be in the second normal form.

- There should not be any functional dependency.

9.7 Boyce-Codd Normal Form (BCNF)

Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF. That is, every relation in BCNF is also in 3NF; however, a relation in 3NF is *not necessarily* in BCNF.

Definition. A relation schema R is in **BCNF** if whenever a *nontrivial* functional dependency $X \rightarrow A$ holds in R, then X is a superkey of R. The formal definition of BCNF differs from the definition of 3NF in that condition (b) of 3NF, which allows A to be prime, is absent from BCNF. That makes BCNF a stronger normal form compared to 3NF. In

practice, most relation schemas that are in 3NF are also in BCNF. Only if $X \rightarrow A$ holds in a relation schema R with X not being a superkey *and A* being a prime attribute will R be in 3NF but not in BCNF. Consider an example which shows a relation TEACH with the following dependencies:

FD1:   {Student, Course} → Teacher

FD2:   Teacher→ Course

Student	Course	Teacher
Neeraj	DBMS	H K Lal
Saroj	Operating System	P K Sharma
Saroj	DBMS	Radhe Shyam
Saroj	Autometa	M K Gupta
Shikha	DBMS	H K Lal
Shikha	Operating System	RajNath
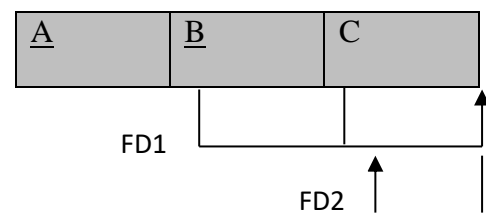
The relation is in 3NF but not in BCNF

Boyce-Codd Normal form is a stronger generalization of third normal form. A table is in Boyce-Codd Normal form if and only if at least one of the following conditions are met for each functional dependency A → B:

- A is a superkey

- It is a trivial functional dependency.

Let us first understand what a superkey means. To understand BCNF in DBMS, consider the following BCNF example table:

Course code	Course venue	Instructor Name	Instructor's phone number

CS101	Lecture Hall 20	Prof. George	+91 6514821924
CS152	Lecture Hall 21	Prof. Atkins	+91 6519272918
CS154	CS Auditorium	Prof. George	+91 6514821924

Here, the first column (course code) is unique across various rows. So, it is a superkey. Consider the combination of columns (course code, professor name). It is also unique across various rows. So, it is also a superkey. A superkey is basically a set of columns such that the value of that set of columns is unique across various rows. That is, no 2 rows have the same set of values for those columns. Some of the superkeys for the table above are:

- Course code

- Course code, professor name

- Course code, professor mobile number

A superkey whose size (number of columns) is the smallest is called as a candidate key. For instance, the first superkey above has just 1 column. The second one and the last one have 2 columns. So, the first superkey (Course code) is a candidate key.

Boyce-Codd Normal Form says that if there is a functional dependency $A \rightarrow B$, then either A is a superkey or it is a trivial functional dependency. A trivial functional dependency means that all columns of B are contained in the columns of A. For instance, (course code, professor name) $\rightarrow$ (course code) is a trivial functional dependency because when we know the value of course code and professor name, we do know the value of course code and so, the dependency becomes trivial.

Let us understand what's going on:

A is a superkey: this means that only and only on a superkey column should it be the case that there is a dependency of other columns. Basically, if a set of columns (B) can be determined knowing some other set of columns (A), then A should be a superkey. Superkey basically determines each row uniquely.

It is a trivial functional dependency: this means that there should be no non-trivial dependency. For instance, we saw how the professor's department was dependent on the professor's name. This may create integrity issues since someone may edit the professor's name without changing the department. This may lead to an inconsistent database. There are also 2 other normal forms:

9.8 CHECK YOUR PROGRESS

1. If F is a set of functional dependencies, then the closure of F is denoted by?

   a) F*

   b) Fo

   c) F+

   d) F

2. In the_____ normal form, a composite attribute is converted to individual attributes.

   A. First

   B. Second

   C. Third

   D. Fourth

3. Table in 2NF eliminated _____.

4. Functional dependencies are the types of constraints that are based on_____.

5. _____ is the bottom up approach to database design that design by examining the relationship between attributes.

9.9 SUMMARY

Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of (1) minimizing redundancy and (2) minimizing the insertion, deletion, and update anomalies. It can be considered as a "filtering" or "purification" process to make the design have successively better quality. Unsatisfactory relation schemas that do not meet certain conditions—the **normal form tests**—are decomposed into smaller relation schemas that meet the tests and hence possess the

desirable properties. Thus, the normalization procedure provides database designers with the following:

■ A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.

■ A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be **normalized** to any desired degree.

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables. Normalization is used for mainly two purposes,

- Eliminating redundant(useless) data.

- Ensuring data dependencies make sense i.e data is logically stored.

9.10 KEYWORDS

- **SUPERKEY: A** superkey is a set of attributes within a table whose values can be used to uniquely identify a tuple. A candidate key is a minimal set of attributes necessary to identify a tuple; this is also called a minimal superkey.

- **ANOMALY:** Anomalies are problems that can occur in poorly planned, un-normalised databases where all the data is stored in one table (a flat-file database).

- **CANDIDATE KEY:** Primary Key is a unique and non-null key which identify a record uniquely in table. A table can have only one primary key. Candidate key is also a unique key to identify a record uniquely in a table but a table can have multiple candidate keys

- **4NF:** Fourth normal form (4NF): Fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF and 3NF) and the Boyce-Codd Normal Form (BCNF).

- **5NF:** Fifth normal form (5NF), also known as project-join normal form (PJ/NF), is a level of database normalization designed to reduce redundancy in relational databases recording multi-valued facts by isolating semantically related multiple relationships.

9.11 SELF-ASSESSMENT TEST

1. Explain why normalization is needed?

2. What are anomalies in a database? How we handle them?

3. Discuss 3NF in detail.

4. Which forms has a relation that possesses data about an individual entity? Explain

5. Which forms are based on the concept of functional dependency?

9.12 ANSWERS TO CHECK YOUR PROGRESS

1. C

2. A

3. All hidden dependencies

4. Key

5. Normalization

9.13 REFERENCES / SUGGESTED READINGS

- C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.

- Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.

- Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.

- https://www.tutorialspoint.com/dbms/database_normalization.htm

- https://www.guru99.com/database-normalization.html

- https://www.studytonight.com/dbms/database-normalization.php

- https://www.javatpoint.com/dbms-normalization

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	**AUTHOR: DR. DEEPAK NANDAL**
LESSON NO. 10	**VETTER:**
CONCURRENCY CONTROL TECHNIQUES	

STRUCTURE

10.0 LEARNING OBJECTIVE

- The objective of this chapter is to make the reader understand the meaning, and concept of Concurrency Control Techniques. To know the transaction states and properties of all the transactions states as well as the recovery methods in transactions.

10.1 INTRODUCTION

Concurrency Control deals with interleaved execution of more than one transaction. In the next article, we will see what serializability is and how to find whether a schedule is serializable or not. Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical Database, it would have a mix of READ and WRITE operations and hence the concurrency is a challenge.

DBMS Concurrency Control is used to address such conflicts, which mostly occur with a multi-user system. Therefore, Concurrency Control is the most important element for proper functioning of a Database Management System where two or more database transactions are executed simultaneously, which require access to the same data.

Reasons for using Concurrency control method is DBMS:

- To apply Isolation through mutual exclusion between conflicting transactions
- To resolve read-write and write-write conflict issues
- To preserve database consistency through constantly preserving execution obstructions
- The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
- Concurrency control helps to ensure serializability

For Example, Assume that two people who go to electronic kiosks at the same time to buy a movie ticket for the same movie and the same show time.

However, there is only one seat left in for the movie show in that particular theatre. Without concurrency control in DBMS, it is possible that both moviegoers will end up purchasing a ticket. However, concurrency control method does not allow this to happen. Both moviegoers can still access information written in the movie seating database. But concurrency control only provides a ticket to the buyer who has completed the transaction process first.

10.2 DEFINITION

Concurrency Control- Concurrency Control in Database Management System is a procedure of managing simultaneous operations without conflicting with each other. It ensures that Database transactions are performed concurrently and accurately to produce correct results without violating data integrity of the respective Database.

Transaction- A set of logically related operations is known as transaction.

10.3 OVERVIEW OF DATABASE TRANSACTIONS

Transaction is a logical unit of work that represents real-world events of any organisation or an enterprise whereas concurrency control is the management of concurrent transaction execution. Transaction processing systems execute database transactions with large databases and hundreds of concurrent users, for example, railway and air reservations systems, banking system, credit card processing, stock market monitoring, super market inventory and checkouts and so on.

A transaction is a logical unit of work of database processing that includes one or more database access operations.

A transaction can be defined as an action or series of actions that is carried out by a single user or application program to perform operations for accessing the contents of the database. The operations can include retrieval, (Read), insertion (Write), deletion and modification. A transaction must be either completed or aborted.

It can either be embedded within an application program or can be specified interactively via a high-level query language such as SQL. Its execution preserves the consistency of the database. Each transaction should access shared data without interfering with the other transactions and whenever a transaction successfully completes its execution; its effect should be permanent. This basic abstraction frees the database application programmer from the following concerns:

- Inconsistencies caused by conflicting updates from concurrent users.
- Partially completed transactions in the event of systems failure.
- User-directed undoing of transactions.

10.4 TRANSACTION STATES

A transaction is a sequence of READ and WRITE actions that are grouped together to from a database access. A transaction may consist of a simple SELECT operation to generate a list of table contents, or it may consist of a series of related UPDATE command sequences. A transaction can include the following basic database access operations:

Operations	Descriptions
Retrive	To retrive data stored ina database.
Insert	To store new data in database.

Delete	To delete existing data from database.
Update	To modify existing data in database.
Commit	To save the work done permanently.
Rollback	To undo the work done.

Transaction that changes the contents of the database must alter the database from one **consistent** state to another. A consistent database state is one in which all data integrity constraints are satisfied. To ensure database consistency, every transaction must begin with the database in a known consistent state.

Transaction Execution and Problems:

A transaction which successfully completes its execution is said to have been committed. Otherwise, the transaction is aborted. Thus, if a committed transaction performs any update operation on the database, its effect must be reflected on the database even if there is a failure.

10.4 TRANSACTION STATES

States through which a transaction goes during its lifetime. These are the states which tell about the current state of the Transaction and also tell how we will further do processing we will do on the transactions. These states govern the rules which decide the fate of the transaction whether it will commit or abort figure 10.1.

A transaction can be in one of the following states:

| State	Description
Active state	A transaction goes into an active state immediately after it starts execution, where it can issue READ and WRITE operations. A transaction may be aborted when the transaction itself detects an error during execution which it cannot recover from, for example, a transaction trying to debit loan amount of an employee from his insufficient gross salary. A transaction may also be aborted before it has been committed due to system failure or any other circumstances beyond its control.
Partially committed	When the transaction ends, it moves to the partially committed state.When the last state is reached. To this point, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently. Once this check is successful, the transaction is said to have reached its commit point and enters the committed state.
Aborted	When the normal execution can no longer be performed. Failed or aborted transactions may be restarted later, either automatically or after being resubmitted by the user as new transactions.
Committed	After successful completion of transaction. A transaction is said to be in a committed state if it has partially committed and it can be ensured that it will never be aborted.
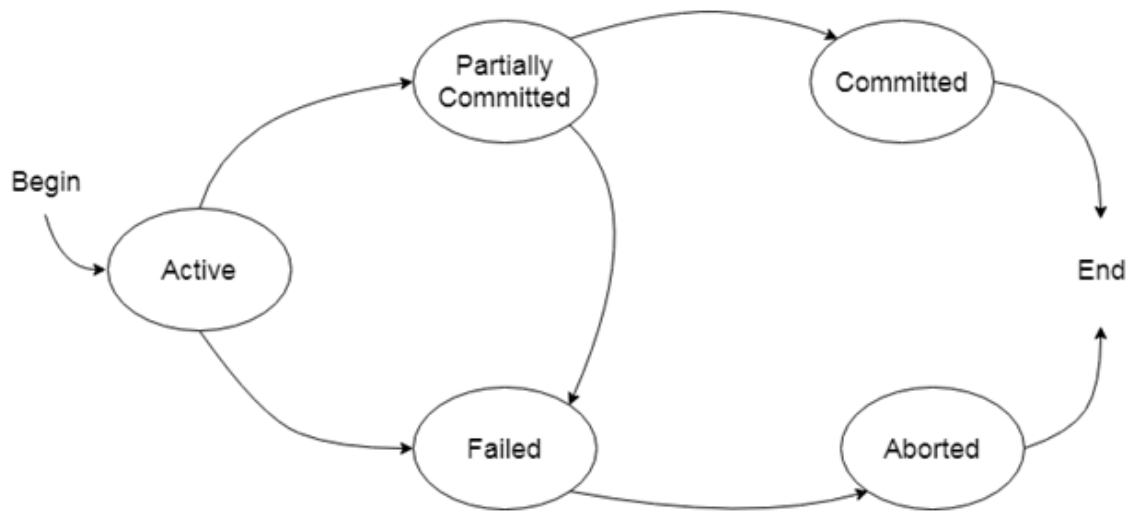
Figure 10.1: Transaction States

10.5 ACID PROPERTIES OF A TRANSACTION

A transaction is a single logical unit of work which accesses and possibly modifies the contents of a database. Transactions access data using read and write operations. In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called ACID properties.

- **Atomicity-**

  By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.

  o Abort: If a transaction aborts, changes made to database are not visible.

  o Commit: If a transaction commits, changes made are visible.

- **Consistency-**

  This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database.

Referring to the example above, the total amount before and after the transaction must be maintained.

- **Isolation-**

  This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

- **Durability-**

  This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

The **ACID** properties, in totality, provide a mechanism to ensure correctness and consistency of a database in a way such that each transaction is a group of operations that acts a single unit, produces consistent results, acts in isolation from other operations and updates that it makes are durably stored.

One of the easiest ways to describe a database transaction is that it is any change in a database, any "transaction" between the database components and the data fields that they contain.

However, the terminology becomes confusing, because in enterprise as a whole, people are so used to referring to financial transactions as simply "transactions." That sets up a central conflict in tech-speak versus the terminology of the average person.

A database "transaction" is any change that happens. To talk about handling financial transactions in database environments, the word "financial" should be used explicitly. Otherwise, confusion can easily crop up. Database systems will need specific features, such as PCI compliance features, in order to handle financial transactions specifically.

As databases have evolved, transaction handling systems have also evolved. A new kind of database called NoSQL is one that does not depend on the traditional relational database data relationships to operate.

While many NoSQL systems offer ACID compliance, others utilize processes like snapshot isolation or may sacrifice some consistency for other goals. Experts sometimes talk about a trade-off between consistency and availability, or similar scenarios where consistently may be treated differently by modern database environments. This type of question is changing how stakeholders look at database systems, beyond the traditional relational database paradigms.

10.6 TRANSACTION RECOVERY

Database systems, like any other computer system, are subject to failures but the data stored in it must be available as and when required. When a database fails it must possess the facilities for fast recovery. It must also have atomicity i.e. either transactions are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database.

There are both automatic and non-automatic ways for both, backing up of data and recovery from any failure situations. The techniques used to recover the lost data due to system crash, transaction errors, viruses, catastrophic failure, incorrect commands execution etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred update and immediate update or backing up data can be used.

Recovery techniques are heavily dependent upon the existence of a special file known as a **system log**. It contains information about the start and end of each transaction and any updates which occur in the **transaction**. The log keeps track of all transaction operations that affect the values of database items. This information is needed to recover from transaction failure.

- The log is kept on disk start_transaction(T): This log entry records that transaction T starts the execution.

- read_item(T, X): This log entry records that transaction T reads the value of database item X.

- write_item(T, X, old_value, new_value): This log entry records that transaction T changes the value of the database item X from old_value to new_value. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X.

- commit(T): This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database.

- abort(T): This records that transaction T has been aborted.

- checkpoint: Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point

before which the DBMS was in consistent state, and all the transactions were committed.

A transaction T reaches its **commit** point when all its operations that access the database have been executed successfully i.e. the transaction has reached the point at which it will not **abort** (terminate without completing). Once committed, the transaction is permanently recorded in the database. Commitment always involves writing a commit entry to the log and writing the log to disk. At the time of a system crash, item is searched back in the log for all transactions T that have written a start_transaction(T) entry into the log but have not written a commit(T) entry yet; these transactions may have to be rolled back to undo their effect on the database during the recovery process

- **Undoing –** If a transaction crashes, then the recovery manager may undo transactions i.e. reverse the operations of a transaction. This involves examining a transaction for the log entry write_item(T, x, old_value, new_value) and setting the value of item x in the database to old-value.There are two major techniques for recovery from non-catastrophic transaction failures: deferred updates and immediate updates.

- **Deferred update –** This technique does not physically update the database on disk until a transaction has reached its commit point. Before reaching commit, all transaction updates are recorded in the local transaction workspace. If a transaction fails before reaching its commit point, it will not have changed the database in any way so UNDO is not needed. It may be necessary to REDO the effect of the operations that are recorded in the local transaction workspace, because their effect may not yet have been written in the database. Hence, a deferred update is also known as the **No-undo/redo algorithm**

- **Immediate update –** In the immediate update, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However,

these operations are recorded in a log on disk before they are applied to the database, making recovery still possible. If a transaction fails to reach its commit point, the effect of its operation must be undone i.e. the transaction must be rolled back hence we require both undo and redo. This technique is known as **undo/redo algorithm.**

- **Caching/Buffering –** In this one or more disk pages that include data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk. A collection of in-memory buffers called the DBMS cache is kept under control of DBMS for holding these buffers. A directory is used to keep track of which database items are in the buffer. A dirty bit is associated with each buffer, which is 0 if the buffer is not modified else 1 if modified.

- **Shadow paging –** It provides atomicity and durability. A directory with n entries is constructed, where the ith entry points to the ith database page on the link. When a transaction began executing the current directory is copied into a shadow directory. When a page is to be modified, a shadow page is allocated in which changes are made and when it is ready to become durable, all pages that refer to original are updated to refer new replacement page.

10.7 CHECK YOUR PROGRESS

1. A _____ consists of a sequence of query and update statements.
2. Which of the following makes the transaction permanent in the database?
   A. View
   B. Commit
   C. Rollback
   D. Flashback
3. In case of any shut down during transaction before commit _____ is done automatically.
4. In order to maintain the consistency during transactions database provides_____>

5. A transaction completes its execution is said to be _____.

10.8 SUMMARY

Concurrency control is a database management systems (DBMS) concept that is used to address occur with a multi-user system. Concurrency control, when applied to a DBMS, is meant to coordinate simultaneous transactions while preserving data integrity. The Concurrency is about to control the multi-user access of Database. When more than one transactions are running simultaneously there are chances of a conflict to occur which can leave database to an inconsistent state. To handle these conflicts we need concurrency control in DBMS, which allows transactions to run simultaneously but handles them in such a way so that the integrity of data remains intact.

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose. Following are the Concurrency Control techniques in DBMS:

- Lock-Based Protocols

- Two Phase Locking Protocol

- Timestamp-Based Protocols

- Validation-Based Protocols

10.9 KEYWORDS

- **Failed state-** If a transaction is executing and a failure occurs, either a hardware failure or a software failure then the transaction goes into failed state from the active state.

- **Transaction-** Transaction is a set of statements which performs tasks like accessing the data or probably update it, within the DBMS.

- **Abort**: If a transaction aborts, changes made to database are not visible.

- **Commit**: If a transaction commits, changes made are visible.

- **Starvation**- Starvation or Livelock is the situation when a transaction has to wait for a indefinite period of time to acquire a lock.

10.10 SELF-ASSESSMENT TEST

1. Explain the concurrency control techniques in DBMS?

2. Explain the term Transaction in DBMS.

3. What are the different transaction states?

4. Discuss the ACID Properties in detail.

5. What are the feasible threats to a Database? Discuss the importance and need of recovery during transactions.

10.11 ANSWERS TO CHECK YOUR PROGRESS

1. Transaction

2. C

3. Rollback

4. Atomic

5. Commited

10.12 REFERENCES / SUGGESTED READINGS

- C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.
- Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.
- Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.
- Thomas Connolly Carolyn Begg, "Database System", 3/2, Pearson Education.
- https://tutorialink.com/dbms/introduction-to-transaction-concepts.dbms

- https://www.techopedia.com/definition/16455/transaction-databases
- https://www.geeksforgeeks.org/transaction-states-in-dbms/
- https://www.javatpoint.com/dbms-states-of-transaction
- https://beginnersbook.com/2018/12/dbms-transaction-states/

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	AUTHOR: DR. DEEPAK NANDAL
LESSON NO. 11	VETTER:
LOCKING AND RECOVERY TECHNIQUES IN CENTRALIZED DBMS	

STRUCTURE

11.0 LEARNING OBJECTIVE

- The objective of this chapter is to make the reader understand the Locking methods of concurrency control, to know the importance of timestamp ordering. Discuss the deadlock in the systems and how to recover from it.

11.1 INTRODUCTION

Transaction processing systems usually allow multiple transactions to run concurrently. By allowing multiple transactions to run concurrently will improve the performance of the system in terms of increased throughout or improved response time, but this allows causes several complications with consistency of the data. Ensuring consistency in spite of concurrent execution of transaction require extra work, which is performed by the concurrency controller system of DBMS.

What is Lock?

A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it. Generally, there is one lock for each data item in the database. Locks are used as a means of synchronizing the access by concurrent transactions to the database item.

Types of Locks

Several types of locks are used in concurrency control. To introduce locking concepts gradually, we first discuss binary locks, which are simple but restrictive and so are not used in practice. We then discuss shared/exclusive locks, which provide more general locking capabilities and are used in practical database locking schemes.

- **Binary Locks**

A binary lock can have two states or values: locked and unlocked. A distinct lock is associated with each database item *A*. If the value of the lock on *A* is 1, item *A* cannot be accessed by a database operation that requests the item. If the value of the lock on *A* is 0 then item can be accessed when requested. We refer to the current value of the lock associated with item *A* as *LOCK (A)*. There are two operations, lock item and unlock item are used with binary locking A transaction requests access to an item *A* by first issuing a lock *item (A)* operation. If LOCK (A) = 1, the transaction is forced to wait. If LOCK (A) = 0 it is set to 1 (the transaction locks the item) and the transaction is allowed to access item *A*. When the transaction is through using the item, it issues an unlock *item (A)* operation, which sets *LOCK (A)* to 0 (unlocks the item) so that *A* may be accessed by other transactions. Hence binary lock enforces mutual exclusiol1 on the data item.

Rules of Binary Locks

If the simple binary locking scheme described here is used, every transaction must obey the following rules:

1. A transaction must issue the operation lock_item (A) before any read_item (A) or write, item operations are performed in T.

2. A transaction T must issue the operation unlock_item (A) after all read_item (A) and write_item (A) operations are completed in T.

3. A transaction T will not issue a lock _*item (A)* operation if it already holds the lock on Item *A*.

4. A transaction T will not issue an unlock _*item (A)* operation unless it already holds the lock on item *A*.

5. The lock manager module of the DBMS can enforce these rules. Between the Lock_*item (A)* and unlock_*item (A)* operations in transaction T, is said to hold the lock on item *A*. At most one transaction can hold the lock on a particular item. Thus no two transactions can access the' same item concurrently.

Disadvantages of Binary Locks

As discussed earlier, binary locking scheme is too restrictive for database items, because at most one transaction can hold a lock on a given item. So, binary locking system cannot be used for practical purpose.

Share/Exclusive (for Read/Write) Locks

We should allow several transactions to access the same item *A* if they all access *A'* for reading purposes only. However, if a transaction is to write an item A, it must have exclusive access to *A*. For this purpose, a different type of lock called a multiple-mode lock is used. In this scheme there are shared/exclusive or read/write locks are used.

Locking operations

There are three locking operations called read_lock(A), write_lock(A) and unlock(A) represented as lock-S(A), lock-X(A), unlock(A) (Here, S indicates shared lock, X indicates exclusive lock)can be performed on a data item. A lock associated with an item A, LOCK (A), now has three possible states: "read-locked", "write-locked," or "unlocked." A read-locked item is also called share-locked item because other transactions are allowed to read the item, whereas a write-locked item is caused exclusive-locked, because a single transaction exclusively holds the lock on the item.

Compatibility of Locks

Suppose that there are A and B two different locking modes. If a transaction T1 requests a lock of mode on item Q on which transaction T2 currently hold a lock of mode B. If transaction can be granted lock, in spite of the presence of the mode B lock, then we say mode A is compatible with mode B. Such a function is shown in one matrix as shown below:

	S	X
S	true	false
X	false	false

Compatibility Graph

The graphs shows that if two transactions only read the same data object they do not conflict, but if one transaction writes a data object and another either read or write the same data object, then they conflict with each other. A transaction requests a shared lock on data item Q by executing the lock-S(Q) instruction. Similarly, an exclusive lock is requested through the lock- X(Q) instruction. A data item Q can be unlocked via the unlock(Q) instruction.

To access a data item, transaction T1 must first lock that item. If the data item is already locked by another transaction in an incompatible mode, the concurrency control manager will not grant the lock until all incompatible locks held by other transactions have been released. Thus, T1 is made to wait until all incompatible locks held by other transactions have been released.

11.2 DEFINITION

Concurrency Control- Concurrency control is provided in a database to:

1. Enforce isolation among transactions.

2. Preserve database consistency through consistency preserving execution of transactions.

3. Resolve read-write and write-read conflicts.

Various concurrency control techniques are:

1. Two-phase locking Protocol

2. Time stamp ordering Protocol

3. Multi version concurrency control

4. Validation concurrency control

11.3 LOCKING METHODS OF CONCURRENECY CONTROL

Locking is an operation which secures: permission to read, OR permission to write a data item. Two phase locking is a process used to gain ownership of shared resources without creating the possibility of deadlock.

The 3 activities taking place in the two phase update algorithm are:

1. Lock Acquisition
2. Modification of Data
3. Release Lock

Two phase locking prevents deadlock from occurring in distributed systems by releasing all the resources it has acquired, if it is not possible to acquire all the resources required without waiting for another process to finish using a lock. This means that no process is ever in a state where it is holding some shared resources, and waiting for another process

to release a shared resource which it requires. This means that deadlock cannot occur due to resource contention.

A transaction in the Two Phase Locking Protocol can assume one of the 2 phases:

(i) Growing Phase:

In this phase a transaction can only acquire locks but cannot release any lock. The point when a transaction acquires all the locks it needs is called the Lock Point.

(ii) Shrinking Phase:

In this phase a transaction can only release locks but cannot acquire any.

Basically locking in DBMS can be defined as:

"A lock is a variable, associated with the data item, which controls the access of that data item."

Locking is the most widely used form of the concurrency control. Locks are further divided into three fields:

1. Lock Granularity
2. Lock Types
3. Deadlocks

1. Lock Granularity:

A database is basically represented as a collection of named data items. The size of the data item chosen as the unit of protection by a concurrency control program is called GRANULARITY. Locking can take place at the following level:

- Database level.
- Table level.
- Page level.
- Row (Tuple) level.
- Attributes (fields) level.

i) Database level Locking :

At database level locking, the entire database is locked. Thus, it prevents the use of any tables in the database by transaction T2 while transaction T1 is being executed. Database level of locking is suitable for batch processes. Being very slow, it is unsuitable for on-line multi-user DBMSs.

ii) Table level Locking :

At table level locking, the entire table is locked. Thus, it prevents the access to any row (tuple) by transaction T2 while transaction T1 is using the table. if a transaction requires access to several tables, each table may be locked. However, two transactions can access the same database as long as they access different tables. Table level locking is less restrictive than database level. Table level locks are not suitable for multi-user DBMS

iii) Page level Locking :

At page level locking, the entire disk-page (or disk-block) is locked. A page has a fixed size such as 4 K, 8 K, 16 K, 32 K and so on. A table can span several pages, and a page can contain several rows (tuples) of one or more tables. Page level of locking is most suitable for multi-user DBMSs.

iv) Row (Tuple) level Locking :

At row level locking, particular row (or tuple) is locked. A lock exists for each row in each table of the database. The DBMS allows concurrent transactions to access different rows of the same table, even if the rows are located on the same page. The row level lock is much less restrictive than database level, table level, or page level locks. The row level locking improves the availability of data. However, the management of row level locking requires high overhead cost.

v) Attributes (fields) level Locking :

At attribute level locking, particular attribute (or field) is locked. Attribute level locking allows concurrent transactions to access the same row, as long as they require the use of different attributes within the row. The attribute level lock yields the most flexible multi-user data access. It requires a high level of computer overhead.

2. Lock Types :

The DBMS mainly uses following types of locking techniques.

- Binary Locking
- Shared / Exclusive Locking
- Two - Phase Locking (2PL)

a. Binary Locking:

A binary lock can have two states or values: locked and unlocked (or 1 and 0, for simplicity). A distinct lock is associated with each database item X.

If the value of the lock on X is 1, item X cannot be accessed by a database operation that requests the item. If the value of the lock on X is 0, the item can be accessed when requested. We refer to the current value (or state) of the lock associated with item X as LOCK(X).

Two operations, lock_item and unlock_item, are used with binary locking.

Lock_item(X):

A transaction requests access to an item X by first issuing a lock_item(X) operation. If LOCK(X) = 1, the transaction is forced to wait. If LOCK(X) = 0, it is set to 1 (the transaction locks the item) and the transaction is allowed to access item X.

Unlock_item (X):

When the transaction is through using the item, it issues an unlock_item(X) operation, which sets LOCK(X) to 0 (unlocks the item) so that X may be accessed by other transactions. Hence, a binary lock enforces mutual exclusion on the data item; i.e., at a time only one transaction can hold a lock.

b. Shared / Exclusive Locking:

Shared lock:

These locks are reffered as read locks, and denoted by 'S'.

If a transaction T has obtained Shared-lock on data item X, then T can read X, but cannot write X. Multiple Shared lock can be placed simultaneously on a data item.

Exclusive lock:

These Locks are referred as Write locks, and denoted by 'X'.

If a transaction T has obtained Exclusive lock on data item X, then T can be read as well as write X. Only one Exclusive lock can be placed on a data item at a time. This means multipls transactions does not modify the same data simultaneously.

c. Two-Phase Locking (2PL):

Two-phase locking (also called 2PL) is a method or a protocol of controlling concurrent processing in which all locking operations precede the first unlocking operation.   Thus, a transaction is said to follow the two-phase locking protocol if all locking operations (such as read_Lock, write_Lock) precede the first unlock operation in the transaction.   Two-phase locking is the standard protocol used to maintain level 3 consistency 2PL defines how transactions acquire and relinquish locks. The essential discipline is that after a transaction has released a lock it may not obtain any further locks. 2PL has the following two phases:

A **growing** phase, in which a transaction acquires all the required locks without unlocking any data. Once all locks have been acquired, the transaction is in its locked point.

A **shrinking** phase, in which a transaction releases all locks and cannot obtain any new

lock.

A transaction shows Two-Phase Locking technique.

Time	Transaction	Remarks
t0	Lock - X (A)	acquire Exclusive lock on A.
t1	Read A	read original value of A
t2	A = A - 100	subtract 100 from A
t3	Write A	write new value of A
t4	Lock - X (B)	acquire Exclusive lock on B.
t5	Read B	read original value of B
t6	B = B + 100	add 100 to B
t7	Write B	write new value of B
t8	Unlock (A)	release lock on A
t9	Unock (B)	release lock on B

11.4 TIMESTAMP ORDERING

Concurrency Control can be implemented in different ways. One way to implement it is by

using Locks. Now, lets discuss about Time Stamp Ordering Protocol.

As earlier introduced, **Timestamp** is a unique identifier created by the DBMS to identify a

transaction. They are usually assigned in the order in which they are submitted to the system.

Refer to the timestamp of a transaction T as **$TS(T)$**. For basics of Timestamp you may refer here.

Timestamp Ordering Protocol –

The main idea for this protocol is to order the transactions based on their Timestamps. A schedule in which the transactions participate is then serializable and the only *equivalent serial schedule permitted* has the transactions in the order of their Timestamp Values. Stating simply, the schedule is equivalent to the particular *Serial Order* corresponding to the *order of the Transaction timestamps*. Algorithm must ensure that, for each items accessed by *Conflicting Operations* in the schedule, the order in which the item is accessed does not violate the ordering. To ensure this, use two Timestamp Values relating to each database item **X**.

- **W_TS(X)** is the largest timestamp of any transaction that executed **write(X)** successfully.
- **R_TS(X)** is the largest timestamp of any transaction that executed **read(X)** successfully.

Basic Timestamp Ordering –

Every transaction is issued a timestamp based on when it enters the system. Suppose, if an old transaction T_i has timestamp $TS(T_i)$, a new transaction T_j is assigned timestamp $TS(T_j)$ such that **$TS(T_i) < TS(T_j)$**. The protocol manages concurrent execution such that the timestamps determine the serializability order. The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. Whenever some Transaction T tries to issue a R_item(X) or a W_item(X), the Basic TO algorithm compares the timestamp of T with **R_TS(X) & W_TS(X)** to ensure that the Timestamp order is not violated. This describe the Basic TO protocol in following two cases.

1. Whenever a Transaction *T* issues a **W_item(X)** operation, check the following conditions:

   - If $R\_TS(X) > TS(T)$ or if $W\_TS(X) > TS(T)$, then abort and rollback T and reject the operation. else,
   - Execute W_item(X) operation of T and set W_TS(X) to TS(T).

2. Whenever a Transaction *T* issues a **R_item(X)** operation, check the following conditions:

   - If $W\_TS(X) > TS(T)$, then abort and reject T and reject the operation, else
   - If W_TS(X) <= TS(T), then execute the R_item(X) operation of T and set R_TS(X) to the larger of TS(T) and current R_TS(X).

Whenever the Basic TO algorithm detects twp conflicting operation that occur in incorrect order, it rejects the later of the two operation by aborting the Transaction that issued it. Schedules produced by Basic TO are guaranteed to be *conflict serializable*. Already discussed that using Timestamp, can ensure that our schedule will be *deadlock free*.

One drawback of Basic TO protocol is that it **Cascading Rollback** is still possible. Suppose we have a Transaction T_1 and T_2 has used a value written by T_1. If T_1 is aborted and resubmitted to the system then, T must also be aborted and rolled back. So the problem of Cascading aborts still prevails.

Let's gist the Advantages and Disadvantages of Basic TO protocol:

   - Timestamp Ordering protocol ensures serializablity since the precedence graph will be of the form as shown in figure 11.1:
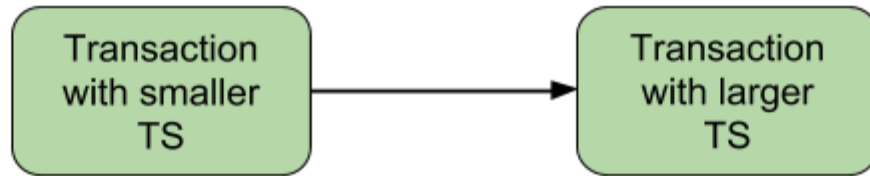
Figure 11.1: Precedence Graph for Timestamp Ordering

- Timestamp protocol ensures freedom from deadlock as no transaction ever waits.

- But the schedule may *not be cascade free*, and may not even be recoverable.

11.5 MULTI VERSION TECHNIQUES

MVCC provides concurrent access to the database without locking the data. This feature improves the performance of database applications in a multiuser environment. Applications will no longer hang because a read cannot acquire a lock.

MVCC provides each user connected to the database with a "snapshot" of the data to work with. The data is consistent with a point in time. Other users of the database see no changes until the transaction is committed. The snapshot can be taken at the start of a transaction, or at the start of each statement, as determined by the isolation level setting.

- This release provides full MVCC support, in which readers do not block writers, and writers do not block readers.

- The user invokes MVCC protocols for a session or table with the SQL statement:

- SET LOCKMODE *session* | ON *table_name* WHERE LEVEL = MVCC

- The alterdb command has two new options, -disable_mvcc and -enable_mvcc, which disable and enable MVCC, respectively. By default, MVCC is enabled for all existing and newly created databases.

- Using MVCC is optional. Your existing applications that do not use MVCC will execute in the same manner they worked previously. The overhead of MVCC is the cost of maintaining multiple versions of database pages.

- For the system administrator, MVCC may require additional buffer manager memory because Consistent Read pages occupy cache space that otherwise might be used by database pages.

- The MVCC feature changes the format of many log records, which means that after running upgradedb, previous journals and checkpoints will be invalid.

  For details about this feature, see the following:

  •The chapters "Understanding the Locking System" and "Understanding Multiversion Concurrency Control" in the *Database Administrator Guide*

  •The SET LOCKMODE and SET SESSION ISOLATION LEVEL statements in the *SQL Reference Guide*

  •The alterdb command in the *Command Reference Guide*

11.6 DEADLOCKS

In a database, a deadlock is an unwanted situation in which two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as it brings the whole system to a Halt.

Example – let us understand the concept of Deadlock with an example: Suppose, Transaction T1 holds a lock on some rows in the Students table and **needs to update** some rows in the Grades table. Simultaneously, Transaction **T2 holds** locks on those very rows (Which T1 needs to update) in the Grades table **but needs** to update the rows in the Student table **held by Transaction T1**.

Now, the main problem arises. Transaction T1 will wait for transaction T2 to give up lock, and similarly transaction T2 will wait for transaction T1 to give up lock. As a consequence, All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions as shown in figure 11.2.
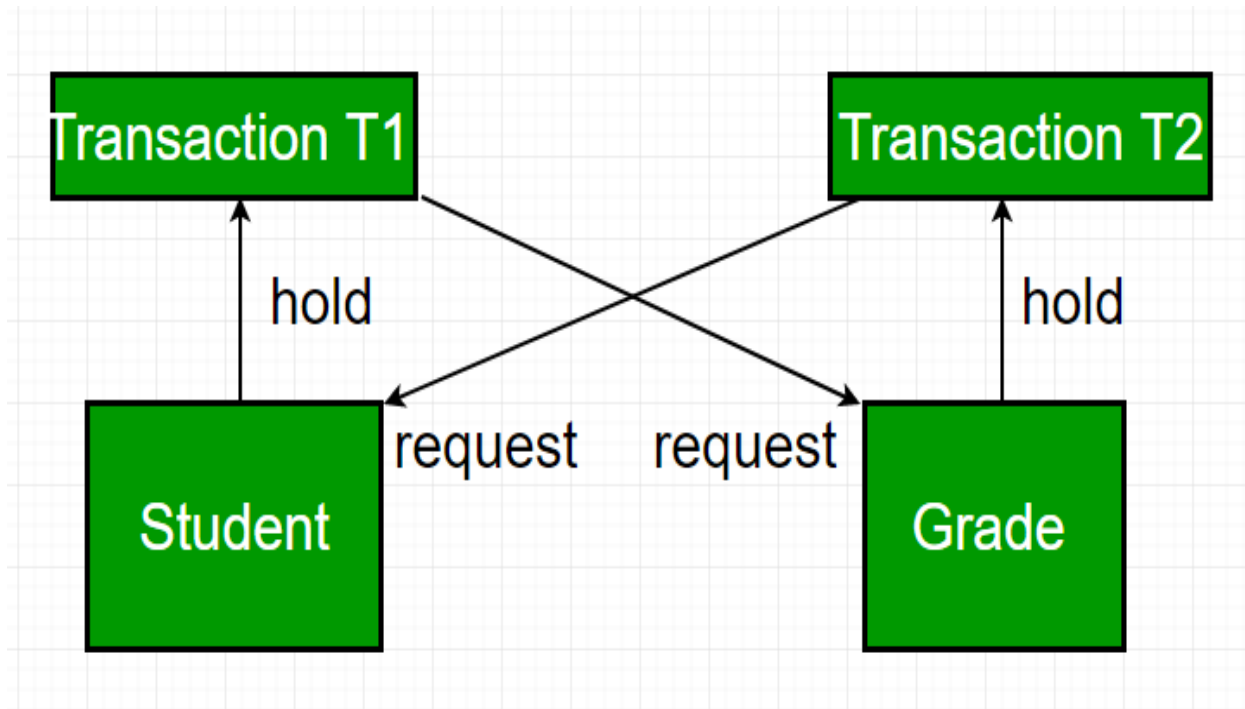

Figure 11.2: Deadlock in DBMS

11.7 RECOVERY IN DBMS

Basically, whenever a transaction is submitted to a DBMS for execution, the operating system is responsible for making sure or to be confirmed that all the operation which need to be in performed in the transaction have completed successfully and their effect is either recorded in the database or the transaction doesn't affect the database or any other transactions.

The DBMS must not permit some operation of the transaction T to be applied to the database while other operations of T is not. This basically may happen if a transaction fails after executing some of its operations but before executing all of them.

Types of failures –

There are basically following types of failures that may occur and leads to failure of the transaction such as:

1. Transaction failure

2. System failure

3. Media failure and so on.

Let us try to understand the different types of failures that may occur during the transaction.

1. **System crash –**A hardware, software or network error occurs comes under this category this types of failures basically occurs during the execution of the transaction. Hardware failures are basically considered as Hardware failure.

2. **System error –** Some operation that is performed during the transaction is the reason for this type of error to occur, such as integer or divide by zero. This type of failures is also known as the transaction which may also occur because of erroneous parameter values or because of a logical programming error. In addition to this user may also interrupt the execution during execution which may lead to failure in the transaction.

3. **Local error –** This basically happens when we are doing the transaction but certain conditions may occur that may lead to cancellation of the transaction. This type of error is basically coming under Local error. The simple example of this is that data for the transaction may not found. When we want to debit money from an insufficient balance account which leads to the cancellation of our request or transaction. And this exception should be programmed in the transaction itself so that it wouldn't be considered as a failure.

4. **Concurrency control enforcement** – The concurrency control method may decide to abort the transaction, to start again because it basically violates serializability or we can say that several processes are in a deadlock.

5. **Disk failure** – This type of failure basically occur when some disk loses their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read /write operation of the transaction.

6. **Castropher- –**These are also known as physical problems it basically refers to the endless list of problems that include power failure or air-conditioning failure, fire, theft sabotage overwriting disk or tapes by mistake and mounting of the wrong tape by the operator.

11.8 CHECK YOUR PROGRESS

1. Locks placed by command are called _____.
2. Which of the following locks the item from change but not from read?
a) implicit locks
b) explicit lock
c) exclusive locks
d) shared locks
3. The advantage of optimistic locking is that:
   a) The lock is obtained only after the transaction has processed.
   b) The lock is obtained before the transaction has processed.
   c) The lock never needs to be obtained.
   d) Transactions that are best suited are those with a lot of activity.
4. Which of the following refers to a cursor type that when the cursor is opened, a primary key value is saved for each row in the recordset; when the application accesses a row, the key is used to fetch the current values of the row?
a) Forward only
b) Static
c) Keyset

d) Dynamic

11.9 SUMMARY

Locking mechanisms are a way for databases to produce sequential data output without the sequential steps. The locks provide a method for securing the data that is being used so no anomalies can occur like lost data or additional data that can be added because of the loss of a transaction.

1. Two-Phase Locking Protocol:

Locking is an operation which secures: permission to read, OR permission to write a data item. Two phase locking is a process used to gain ownership of shared resources without creating the possibility of deadlock.

The 3 activities taking place in the two phase update algorithm are:

1. Lock Acquisition

2. Modification of Data

3. Release Lock

Two phase locking prevents deadlock from occurring in distributed systems by releasing all the resources it has acquired, if it is not possible to acquire all the resources required without waiting for another process to finish using a lock. This means that no process is ever in a state where it is holding some shared resources, and waiting for another process to release a shared resource which it requires. This means that deadlock cannot occur due to resource contention.

A transaction in the Two Phase Locking Protocol can assume one of the 2 phases:

- **(i) Growing Phase:**

   In this phase a transaction can only acquire locks but cannot release any lock. The point when a transaction acquires all the locks it needs is called the Lock Point.

- **(ii) Shrinking Phase:**

In this phase a transaction can only release locks but cannot acquire any.

2. <u>Time Stamp Ordering Protocol</u>:

A timestamp is a tag that can be attached to any transaction or any data item, which denotes a specific time on which the transaction or the data item had been used in any way. A timestamp can be implemented in 2 ways. One is to directly assign the current value of the clock to the transaction or data item. The other is to attach the value of a logical counter that keeps increment as new timestamps are required.

The timestamp of a data item can be of 2 types:

- **(i) W-timestamp(X):**

This means the latest time when the data item X has been written into.

- **(ii) R-timestamp(X):**

This means the latest time when the data item X has been read from. These 2 timestamps are updated each time a successful read/write operation is performed on the data item X.

3. Multiversion Concurrency Control:

Multiversion schemes keep old versions of data item to increase concurrency.

Multiversion 2 phase locking:

Each successful write results in the creation of a new version of the data item written.

Timestamps are used to label the versions. When a read(X) operation is issued, select an appropriate version of X based on the timestamp of the transaction.

11.10 KEYWORDS

- **Two- phase locking-** The Two Phase Commit is designed to coordinate the transactions of the requests to the system. The idea behind the protocol is to produce serialized results from a non-serialized system.
- **View-** Any set of tuples; a data report from the RDBMS in response to a query.

- **Live lock-** A transaction is in a state of live lock if it cannot proceed for an indefinite period while other transactions in the system continue normally.

11.11 SELF-ASSESSMENT TEST

1. What are different types of locking techniques in DBMS?

2. What is a deadlock? How deadlock occur in a database?

3. How to recover a database management system from a deadlock?

4. What is timestamp ordering?

5. What is multiversion2 phase locking?

11.12 ANSWERS TO CHECK YOUR PROGRESS

1. B
2. Shared locks
3. A
4. C

11.13 REFERENCES / SUGGESTED READINGS

- C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.
- Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.
- Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.
- Thomas Connolly Carolyn Begg, "Database System", 3/2, Pearson Education.
- https://www.geeksforgeeks.org/concurrency-control-in-dbms/
- https://www.geeksforgeeks.org/deadlock-in-dbms/?ref=lbp
- https://www.javatpoint.com/dbms-recoverability-of-schedule
- https://www.slideshare.net/rajvardhan7/multiversion-concurrency-control-techniques

SUBJECT: DATABASE MANAGEMENT SYSTEM	
COURSE CODE: MCA-11	AUTHOR: DR. DEEPAK NANDAL
LESSON NO. 12	VETTER:
DDBMS DESIGN	

STRUCTURE

12.0 LEARNING OBJECTIVE

- The objective of this chapter is to make the reader understand the distributed database systems, to know the difference between dbms and ddbms. To know the types of the distributed database systems.

12.1 INTRODUCTION

In today's organizations, there is a need for a well-maintained database to maintain its functionality. Earlier, databases used to be centralized in nature. But, with the boost in globalization, organizations inclined to diversify throughout the globe. These days, instead of opting for a central database, many organizations choose to distribute data over local servers. This distribution of data over various servers is generally known as distributed databases. A distributed database system is the collection of logically interrelated data, distributed across various locations that communicate via a computer network.

To ensure a successful database management system, it is vital to carefully work out a strategy to align the data requirements and business agenda of your organization. Hiring the services of a database development company can be of great help in creating and implementing database solutions. A professional company can help you in determining the best database management system and test database programs. Many database development companies these days provide custom database solutions provider inclined to client's needs to help efficient management and security of crucial business data.

Since its introduction, Distributed database systems have eliminated many shortcomings of the centralized database systems and fit more in the decentralized

structures of many organizations. Here are some of the key benefits of a distributed database system over the centralized database system, have a look:

- **Reliability and Availability**: Distributed database systems are more reliable as compared to a centralized database system. In the case of database failures, the whole system of centralized databases comes to an end. Whereas, in the case of distributed database systems, if a component fails, the performance of the system continues may be at a slower rate.

- **Modular Development**: In the case of centralized database systems, if the system requires to be extended to new locations or new units, the action needs substantial efforts and interruption in the existing functioning. But, in a distributed database system, the expansion work simply requires adding new computers and local data to the new site and finally connecting them to the distributed system without disturbing the existing functionality.

- **Quick Response**: When data is distributed in an efficient manner, the user requests or queries can be met from local data itself, thus offering a quick response. In the case of centralized database systems, all kinds of queries need to pass through the central computer for processing, which may lead to delay in response.

- **Lower Communication Overhead:** When data is positioned locally where it is frequently used, then the communication costs for data management can be minimized. However, in the case of centralized database systems, the communication costs are quite high.

- **Secured Management of Distributed Data:** A number of transparencies such as fragmented transparency, network transparency, and replication transparency are implemented to cover the actual implementation details of the entire

distributed system. Thus, distributed database systems provide more security of data as compared to centralized database systems.

12.2 DEFINITION

Data Replication- Data replication is the storage of data copies at multiple sites on the network.

Client- Server Architecture- Implementation of a distributed database system must be carefully managed within a client server architecture. Typically, the server provides the resources for the client to use. The client receives the request from the user and the request is passed to the server. The server receives, schedules and executes the requests, selecting only what the client requires. The request is sent only when the client requests it.

Data Fragmentation- Data fragmentation is a technique used to break up objects. In designing a distributed database, you must decide which portion of the database is to be stored where.

12.3 DISTRIBTUED DATABASE SYSTEM

A distributed database is basically a database that is not limited to one system, it is spread over different sites, i.e. on multiple computers or over a network of computers. A distributed database system is located on various sited that don't share physical components. This may be required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database.

Types:

1. Homogeneous Database:

In a homogeneous database, all different sites store database identically. The operating system, database management system and the data structures used – all are same at all sites. Hence, they're easy to manage.

2. Heterogeneous Database:

In a heterogeneous distributed database, different sites can use different schema and software that can lead to problems in query processing and transactions. Also, a particular site might be completely unaware of the other sites. Different computers may use a different operating system, different database application. They may even use different data models for the database. Hence, translations are required for different sites to communicate.

12.4 DATA REPLICATION

Data replication is the process of storing separate copies of the database at two or more sites. It is a popular fault tolerance technique of distributed databases. Data replication is the storage of data copies at multiple sites on the network. Fragment copies can be stored at several site, thus enhancing data availability and response time. Replicated data is subject to a mutual consistency rule. This rule requires that all copies of the data fragments must be identical and to ensure data consistency among all of the replications. Although data replication is beneficial in terms of availability and response times, the maintenance of the replications can become complex. For example, if data is replicated over multiple sites, the DDBMS must decide which copy to access. For a query operation, the nearest copy is all that is required to satisfy a transaction. However, if the operation is an update, then all copies must be selected and updated to satisfy the mutual consistency rule.

Advantages of Data Replication

- **Reliability** − In case of failure of any site, the database system continues to work since a copy is available at another site(s).

- **Reduction in Network Load** − Since local copies of data are available, query processing can be done with reduced network usage, particularly during prime hours. Data updating can be done at non-prime hours.

- **Quicker Response** − Availability of local copies of data ensures quick query processing and consequently quick response time.

- **Simpler Transactions** − Transactions require less number of joins of tables located at different sites and minimal coordination across the network. Thus, they become simpler in nature.

Disadvantages of Data Replication

- **Increased Storage Requirements** − Maintaining multiple copies of data is associated with increased storage costs. The storage space required is in multiples of the storage required for a centralized system.

- **Increased Cost and Complexity of Data Updating** − Each time a data item is updated, the update needs to be reflected in all the copies of the data at the different sites. This requires complex synchronization techniques and protocols.

- **Undesirable Application – Database coupling** − If complex update mechanisms are not used, removing data inconsistency requires complex co-ordination at application level. This results in undesirable application – database coupling.

Some commonly used replication techniques are −

- Snapshot replication

- Near-real-time replication

- Pull replication

12.5 FRAGMENTATION TECHNIQUES

Data fragmentation is a technique used to break up objects. In designing a distributed database, you must decide which portion of the database is to be stored where. One technique used to break up the database into logical units called fragments. Fragmentation information is stored in a distributed data catalogue which the processing computer uses to process a user's request. As a point of discussion, we can look at data fragmentation in terms of relations or tables. The following matrix describes the different types of fragmentation that can be used.

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called **fragments**. Fragmentation can be of three types: horizontal, vertical, and hybrid (combination of horizontal and vertical). Horizontal fragmentation can further be classified into two techniques: primary horizontal fragmentation and derived horizontal fragmentation.

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called "re-constructiveness."

Advantages of Fragmentation

- Since data is stored close to the site of usage, efficiency of the database system is increased.

- Local query optimization techniques are sufficient for most queries since data is locally available.

- Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages of Fragmentation

- When data from different fragments are required, the access speeds may be very high.

- In case of recursive fragmentations, the job of reconstruction will need expensive techniques.

- Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

VERTICAL FRAGMENTATION

In vertical fragmentation, the fields or columns of a table are grouped into fragments. In order to maintain re-constructiveness, each fragment should contain the primary key field(s) of the table. Vertical fragmentation can be used to enforce privacy of data.

HORIZONTAL FRAGMENTATION

Horizontal fragmentation groups the tuples of a table in accordance to values of one or more fields. Horizontal fragmentation should also confirm to the rule of re-constructiveness. Each horizontal fragment must have all columns of the original base table.

HYBRID FRAGMENTATION

In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques are used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task.

12.6 CHECK YOUR PROGRESS

1. Global wait-for graph is used for _____ in distributed database.

2. Which of the following is not a promise of distributed database?

   a. Network transparency

   b. Replication Transparency

   c. Fragmentation Transparency

   d. None of the above

3. The real use of the two phase commit protocol is _____

4. A distributed transaction can be _____ if queries are issued at one or more nodes.

5. Depending on the solution each node in the distributed database system can act as_____

12.7 SUMMARY

Whether the database is centralized or distributed, the design principles and concepts are same. However, the design of a distributed database introduces three new issues:

- How to partition the database into fragments.
- Which fragments to replicate.
- Where to locate those fragments and replicas.

Data fragmentation and data replication deal with the first two issues and data allocation deals with the third issue.

Data Fragmentation:

Data fragmentation allows you to break a single object into two or more segments, or fragments. The object might be a user's database, a system database, or a table. Each fragment can be stored at any site over a computer network. Information about data fragmentation is stored in the distributed data catalog (DDC), from which it is accessed by the TP to process user requests. Data fragmentation strategies, as discussed here, are based at the table level and consist of dividing a table into logical fragments. You will explore three types of data fragmentation strategies: horizontal, vertical, and mixed.

Horizontal fragmentation refers to the division of a relation into subsets (fragments) of tuples (rows). Each fragment is stored at a different node, and each fragment has unique rows. However, the unique rows all have the same attributes (columns). In short, each fragment represents the equivalent of a SELECT statement, with the WHERE clause on a single attribute.

Vertical fragmentation refers to the division of a relation into attribute (column) subsets. Each subset (fragment) is stored at a different node, and each fragment has unique columns—with the exception of the key column, which is common to all fragments. This is the equivalent of the PROJECT statement in SQL.

Mixed fragmentation refers to a combination of horizontal and vertical strategies. In other words, a table may be divided into several horizontal subsets (rows), each one having a subset of the attributes (columns).

12.8 KEYWORDS

- **MYSQL-** Cluster is the distributed database combining linear scalability and high availability. It provides in-memory real-time access with transactional consistency

across partitioned and distributed datasets. It is designed for mission critical applications.

- **DDBMS-** A distributed database management system (DDBMS) is the software system that manages a distributed database such that the distribution aspects are transparent to the users.

- **Distributed database-** A distributed database (DDB) is an integrated collection of databases that is physically distributed across sites in a computer network.

12.9 SELF-ASSESSMENT TEST

1. What is Distributed Database Management System?
2. What are the major differences between DBMS and DDBMS?
3. What is Fragmentation?
4. What is Data Replication?
5. Discuss some design issues in DDBMS.

12.10 ANSWERS TO CHECK YOUR PROGRESS

1. Handling deadlocks
2. D
3. Atomicity i.e. all or nothing commits at all sites
4. Partially read only

12.11 REFERENCES / SUGGESTED READINGS

- C.J Date, "An Introduction to Database Systems", 8th edition, Addison Wesley N. Delhi.
- Ivan Bayross, "SQL, PL/SQL-The Programming Language of ORACLE", BPB Publication 3rd edition.
- Elmasri and Navathe, "Fundamentals of Database Systems", 5th edition, Pearson Education.
- Thomas Connolly Carolyn Begg, "Database System", 3/2, Pearson Education.
- https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_design_strategies.htm

- https://www.dlsweb.rmit.edu.au/Toolbox/knowmang/content/distributed_sys/ddms_design.htm
- http://www.myreadingroom.co.in/notes-and-studymaterial/65-dbms/559-database-design-concepts.html
- https://www.smartsight.in/technology/a-detailed-guide-about-data-allocation-in-distributed-database-design/
- https://www.geeksforgeeks.org/distributed-database-system/
- https://www.geeksforgeeks.org/concurrency-control-in-dbms/