DATABASE MANAGEMENT SYSTEM

BISHWO PRAKASH POKHAREL

Introduction

1.1 Definition of database and database system

Data: The word data is used to refer to a fact about the person, place, object, event or concept. Data and information is not the same thing. Data arranged in certain order and form which is useful to us is called information. Data is the raw material to generate information i.e. data is to be processed to produce information. Data is the known facts and figures about a person, place, event or thing which can be recorded in computer in the form of number, text, picture, audio, video etc. For example, '100', 'Bibek Kandel', 'Class-XII' etc. is some of the example of data. It is the basic element or raw fact of database as it provides the information after processing.

Information: Information is the processed value which we get after processing by the computer. Information is very meaningful and useful to us and it enables to make right decision at right time. Database system provides us the right information by processing the collected data in the database. When we process data and convert it into a form that is useful and meaningful to the decision maker, it becomes information. Human beings apply facts, principles, knowledge, experience, and intuition to convert data into information. Only then does it become useful for making decisions. Note, however, that it is difficult to place a dollar value on information; also, information is time-dependent, since its value and usefulness often decrease with time.

Database: A database is an organized collection of facts. In other words we can say that it is a collection of information arranged and presented to serve an assigned purpose. An example of a database is a dictionary, where words are arranged alphabetically. Another example is a telephone directory, where subscriber names are listed in an alphabetic order. Similarly, when we think of a box of cards with names and addresses written as a mailing, list, then the box and its contents form a mailing database contains cards with mailing addresses. All these cards are placed in alphabetic order of names and the collection of these cards would be called a database.

Key Terms used in database system

Fields: Fields contain three important terms used in database. Fields contain one piece of information of entry. E.g., in an address book each entry has fields name, last name, address, phone number, email, birthdates etc. Each unique type of information is stored in its own field.

Record: One full set of fields i.e. all the related information about one person or object is called the record for example , for an address book all the information for first person is one record and the information for the second person is called another record and so on. A single record is called Tuple.

Table: A complete collection of record is called a table. A table contains row and columns where column represent fields and each row represents record.

Database Management System (DBMS): DBMS is a suite of programs which typically manage large structured sets of persistent data, offering ad hoc query facilities to many users. They are widely used in business applications. A database management system (DBMS) can be an extremely complex set of software programs that controls the organization, storage and retrieval of data (fields, records and files) in a database. It also controls the security and integrity of the database. The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data.

When a DBMS is used, information systems can be changed much more easily as the organization's information requirements change. New categories of data can be added to the database without disruption to the existing system. Primary goal of DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. DBMS allows us to define structure for storage of information and also provides mechanism to manipulate this information. DBMS also provides safety for the information stored despite system crashes or attempts of authorized access.

Objectives of DBMS:

1. Provide for mass storage device of relevant data
2. Making access to the data easy for the user
3. Providing prompt response to user request for data
4. Making the latest modification to the database available immediately
5. Eliminate redundant data
6. Allow multiple users to be active at one time
7. Allow the growth of database system
8. Protect the data from physical harm and unauthorized access

Functions of DBMS:

The major functions of DBMS are as follows:

i. Creating database file
ii. Entering database record
iii. Sorting the database records
iv. Deleting records
v. Updating records
vi. Searching records
vii. Merging database file
viii. Copying records
ix. Printing reports
x. Backup database

xi. Provide security to data.

Different views of Database Management Systems:

A database management system provides the ability for many different users to share data and process resources but as there can be many different users, there are many different database needs. The question is how can a single, unified database meets the differing requirement of so many users?

A DBMS minimizes these problems by providing two views of the database: a physical view and a logical view.

1. **Physical View:** The physical view deals with the actual, physical arrangement and location of data in the direct access storage devices (DASDs). Database specialists use the physical view to make efficient use of storage and processing resources. Users, however, may wish to see data differently from how they are stored, and they do not want to know all the technical details of physical storage. After all, a business user is primarily interested in using the information, not in how it is stored.
2. **Logical View/User's view:** The logical view of a database program represents data in a format that is meaningful to user and to the software programs that process those data. That is, the logical view tells the user, in user terms, what is in the database. One strength of a DBMS is that while there is only one physical view of the data, there can be an endless number of different logical views. This feature allows users to see database information in a more business-related way rather than from a technical, processing viewpoint. Thus, the logical view refers to the way user views data.

Data Redundancy:

In non-database systems each application has its own private files. This can often lead to redundancy in stored data, with resultant waste in storage space. In a database the data is integrated.

The database may be thought of as a unification of several otherwise distinct data files, with any redundancy among those files partially or wholly eliminated.

Data integration is generally regarded as an important characteristic of a database. The avoidance of redundancy should be an aim, however, the vigour with which this aim should be pursued is open to question.

- Data redundancy can lead to inconsistency in the database unless controlled.
- The system should be aware of any data duplication – the system is responsible for ensuring updates are carried out correctly.
- A DB with uncontrolled redundancy can be in an inconsistent state – it can supply incorrect or conflicting information.
- A given fact represented by a single entry cannot result in inconsistency – few systems are capable of propagating updates i.e. most systems do not support controlled redundancy.

Data Integrity:

This describes the problem of ensuring that the data in the database is accurate.

- Inconsistencies between two entries representing the same 'fact' give an example of lack of integrity (caused by redundancy in the database).

- Integrity constraints can be viewed as a set of assertions to be obeyed when updating a DB to preserve an error-free state.
- Even if redundancy is eliminated, the DB may still contain incorrect data.
- Integrity checks which are important data items and record types in database.

Integrity checks on data items can be divided into 4 groups:

- **Type checks:** Ensuring a numeric field is numeric and not a character – this check should be performed automatically by the DBMS.
- **Redundancy checks:** Direct or indirect – this check is not automatic in most cases.
- **Range checks:** To ensure a data item value falls within a specified range of values, such as checking dates. For example (age>0 AND age<100)
- **Comparison checks:** In this check a function of a set of data items values is compared against a function of another set of data item values. For example, the max salary for a given set of employees must be less than the min salary for the set of employees on a higher salary scale.

Components of database environment:

1. **Computer aided software engineering (CASE) tools:** They are the automated tools used to design database and application program. For e.g. Ms-Access, SQL Server etc.
2. **Repository:** It is centralized knowledge base for all data definition, data relationship, screen and report formats and other system components. It contains an extended set of metadata important for managing databases as well as other components of an information system.
3. **DBMS:** It is used to define, create, maintain and provide controlled access to the database and also the repository.
4. **Application program:** Computer program that are used to create and maintain the database and provide information to users.
5. **Database:** An organized collection of logically related data, usually designed to meet the information needs of multiple users in an organization. It contains the occurrences of data.
6. **User interface:** Languages, menu and other facilities by which user interact with various system components such as case tools, application program, DBMS, and the repository.
7. **Data administrators:** This person is responsible for the overall information resources of an organization. This person must be technically qualified. S/He uses CASE tools to improve the productivity of database planning and design.
8. **System developers:** The persons such as system analysts and programmers who design new application programs. System developers often use CASE tools for system requirements analysis and program design. He/She is responsible to write a program to access data by using specific programming language.
9. **End users:** Person who performs only queries. He/She can add, delete, and modify data in the database and who request or receive information from it.

1.2 Characteristics of database approach

The database approach has proven far better than the traditional file management system. Database Approach has many characteristics that make it more robust in nature. The main Characteristics of Database Approaches are as follows:

Characteristics of Database Approach

1. **Represent Some Aspects of real world applications**
   A database represents some features of real world applications. Any change in the real world is reflected in the database. If we have some changes in our real applications like railway reservation system then it will be reflected in database too.

   For example, let us take railway reservation system; we have in our mind some certain applications of maintaining records of attendance, waiting list, train arrival and departure time, certain day etc. related to each train.
2. **Manages Information**
   A database always takes care of its information because information is always helpful for whatever work we do. It manages all the information that is required to us. By managing information using a database, we become more deliberated user of our data.
3. **Easy Operation implementation**
   All the operations like insert, delete, update, search etc. are carried out in a flexible and easy way. Database makes it very simple to implement these operations. A user with little knowledge can perform these operations. This characteristic of database makes it more powerful.
4. **Multiple views of database**
   Basically, a view is a subset of the database. A view is defined and devoted for a particular user of the system. Different users of the system may have different views of the same system.

   Every view contains only the data of interest to a user or a group of users. It is the responsibility of users to be aware of how and where the data of their interest is stored.
5. **Data for specific purpose**
   A database is designed for data of specific purpose. **For example**, a database of student management system is designed to maintain the record of student's marks, fees and attendance etc. This data has a specific purpose of maintaining student record.
6. **It has Users of Specific interest**
   A database always has some indented group of users and applications in which these user groups are interested.

   For example, in a library system, there are three users, official administration of the college, the librarian, and the students.
7. **Self Describing nature**
   A database is of self describing nature; it always describes and narrates itself. It contains the description of the whole data structure, the constraints and the variables.

It makes it different from traditional file management system in which definition was not the part of application program. These definitions are used by the users and DBMS software when needed.

8. **Logical relationship between records and data**
   A database gives a logical relationship between its records and data. So a user can access various records depending upon the logical conditions by a single query from the database.
9. **Shelter between program and data**
   In traditional file management system, if any user makes changes in the structure of a file then all the programs accessed by that file needed to be changed. The structure of data files is defined by the application programs.
   But in database approach the data structure is carried in system catalog not in the program. So user doesn't need to make changes in all the programs.

1.3 Database system versus traditional file processing system

Traditional File Processing System

File processing systems was an early attempt to computerize the manual filing system that we are all familiar with. A file system is a method for storing and organizing computer files and the data they contain to make it easy to find and access them. File systems may use a storage device such as a hard disk or CD-ROM and involve maintaining the physical location of the files.

In our own home, we probably have some sort of filing system, which contains receipts, guarantees, invoices, bank statements, and such like. When we need to look something up, we go to the filing system and search through the system starting from the first entry until we find what we want. Alternatively, we may have an indexing system that helps to locate what we want more quickly. For example we may have divisions in the filing system or separate folders for different types of item that are in some way logically related.

The manual filing system works well when the number of items to be stored is small. It even works quite adequately when there are large numbers of items and we have only to store and retrieve them. However, the manual filing system breaks down when we have to cross-reference or process the information in the files. For example, a typical real estate agent's office might have a separate file for each property for sale or rent, each potential buyer and renter, and each member of staff.

Clearly the manual system is inadequate for this' type of work. The file based system was developed in response to the needs of industry for more efficient data access. In early processing systems, an organization's information was stored as groups of records in separate files.

In the traditional approach, we used to store information in flat files which are maintained by the file system under the operating system's control. Here, flat files are files containing records having no structured relationship among them.

Limitations of the File Processing System / File-Based Approach

There are following problems associated with the File Based Approach:

1. Separated and Isolated Data: To make a decision, a user might need data from two separate files. First, the files were evaluated by analysts and programmers to determine the specific data required from each file and the relationships between the data and then applications could be written in a programming language to process and extract the needed data. Imagine the work involved if data from several files was needed.

2. Duplication of data: Often the same information is stored in more than one file. Uncontrolled duplication of data is not required for several reasons, such as:

- Duplication is wasteful. It costs time and money to enter the data more than once
- It takes up additional storage space, again with associated costs.
- Duplication can lead to loss of data integrity; in other words the data is no longer consistent. For example, consider the duplication of data between the Payroll and Personnel departments. If a member of staff moves to new house and the change of address are communicated only to Personnel and not to Payroll, the person's pay slip will be sent to the wrong address. A more serious problem occurs if an employee is promoted with an associated increase in salary. Again, the change is notified to Personnel but the change does not filter through to Payroll. Now, the employee is receiving the wrong salary. When this error is detected, it will take time and effort to resolve. Both these examples, illustrate inconsistencies that may result from the duplication of data. As there is no automatic way for Personnel to update the data in the Payroll files, it is difficult to foresee such inconsistencies arising. Even if Payroll is notified of the changes, it is possible that the data will be entered incorrectly.

3. Data Dependence: In file processing systems, files and records were described by specific physical formats that were coded into the application program by programmers. If the format of a certain record was changed, the code in each file containing that format must be updated. Furthermore, instructions for data storage and access were written into the application's code. Therefore, .changes in storage structure or access methods could greatly affect the processing or results of an application.

In other words, in file based approach application programs are data dependent. It means that, with the change in the physical representation (how the data is physically represented in disk) or access technique (how it is physically accessed) of data, application programs are also affected and needs modification. In other words application programs are dependent on the how the data is physically stored and accessed.

If for example, if the physical format of the master/transaction file is changed, by making the modification in the delimiter of the field or record, it necessitates that the application programs which depend on it must be modified.

4. Difficulty in representing data from the user's view: To create useful applications for the user, often data from various files must be combined. In file processing it was difficult to determine relationships between isolated data in order to meet user requirements.

5. Data Inflexibility: Program-data interdependency and data isolation, limited the flexibility of file processing systems in providing users with ad-hoc information requests.

6. Incompatible file formats: As the structure of files is embedded in the application programs, the structures are dependent on the application programming language. For example, the structure of a file generated by a COBOL program may be different from the structure of a file generated by a 'C' program. The direct incompatibility of such files makes them difficult to process jointly.

7. Data Security. The security of data is low in file based system because, the data is maintained in the flat file(s) is easily accessible. For Example: Consider the Banking System. The Customer Transaction file has details about the total available balance of all customers. A Customer wants information about his account balance. In a file system it is difficult to give the Customer access to only his data in the· file. Thus enforcing security constraints for the entire file or for certain data items are difficult.

8. Transactional Problems. The File based system approach does not satisfy transaction properties like Atomicity, Consistency, Isolation and Durability properties commonly known as ACID properties.

For example: Suppose, in a banking system, a transaction that transfers Rs. 1000 from account A to account B with initial values' of A and B being Rs. 5000 and Rs. 10000 respectively. If a system crash occurred after the withdrawal of Rs. 1000 from account A, but before depositing of amount in account B, it will result an inconsistent state of the system. It means that the transactions should not execute partially but wholly. This concept is known as Atomicity of a transaction (either 0% or 100% of transaction). It is difficult to achieve this property in a file based system.

9. Concurrency problems. When multiple users access the same piece of data at same interval of time then it is called as concurrency of the system. When two or more users read the data simultaneously there is problem, but when they like to update a file simultaneously, it may result in a problem.

10. Poor data modeling of real world. The file based system is not able to represent the complex data and interfile relationships, which results poor data modeling properties.

Database Approach:

A database is a collection of interrelated data's stored in a database server; these data will be stored in the form of tables. The primary aim of database is to provide a way to store and retrieve database information in fast and efficient manner.

The fundamental characteristic of database approach is that the database system not only contains data's but it contains a complete definition or description of the database structure and constraints. These definitions are stored in a system catalog, which contains the

information about the structure and definitions of the database. Hence, this approach will work on any type of database. For example, insurance database, airlines, banking database, finance details, and enterprise information database. But in traditional file processing system the application is developed for a specific purpose and they will access specific database only.

Database Management Systems (DBMS) have replaced the traditional file based data storage systems. This is because they are:

- More powerful
- Easier to manage
- Can store large volumes of data
- Manipulation and fetching of data from a DBMS is many times easier than doing so from a file based data storage system.
- Faster

Facilities of Database Approach:

The facilities offered by DBMS vary a great deal, depending on their level of sophistication. In general, however, a good DBMS should provide the following advantages over a conventional system:

- **Independence of data and program:** This is a prime advantage of a database. Both the database and the user program can be altered independently of each other thus saving time and money which would be required to retain consistency.
- **Data share-ability and non-redundant of data:** The ideal situation is to enable applications to share an integrated database containing all the data needed by the applications and thus eliminate as much as possible the need to store data redundantly.
- **Integrity:** With many different users sharing various portions of the database, it is impossible for each user to be responsible for the consistency of the value in the database and for maintaining the relationships of the user data items to all other data items, some of which may be unknown or even prohibited for the user to access.
- **Centralized Control:** With central control of the database, the DBA can ensure that standards are followed in the representation of data.
- **Security:** Having control over the database the DBA can ensure that access to the database is through proper channels and can define the access rights of any user to any data items or defined subset of the database. The security system must prevent corruption of the existing data either accidently or maliciously.
- **Performance and efficiency:** In view of the size of database and of demanding database accessing requirements, good performance and efficiency are major requirements. Knowing the overall requirements of the organization, as opposed to the requirements of any individual user, the DBA can structure the database system to provide an overall service that is 'best for the enterprise'.

1.4 Advantages and limitations of using DBMS

Advantages of DBMS

The database management system has promising potential advantages, which are explained below:

1. **Controlling Redundancy:** In file system, each application has its own private files, which cannot be shared between multiple applications. This can often lead to considerable redundancy in the stored data, which results in wastage of storage space. By having centralized database most of this can be avoided. It is not possible that all redundancy should be eliminated. Sometimes there are sound business and technical reasons for· maintaining multiple copies of the same data. In a database system, however this redundancy can be controlled.
2. **Integrity can be enforced:** Integrity of data means that data in database is always accurate, such that incorrect information cannot be stored in database. In order to maintain the integrity of data, some integrity constraints are enforced on the database. A DBMS should provide capabilities for defining and enforcing the constraints.
3. **Inconsistency can be avoided**: When the same data is duplicated and changes are made at one site, which is not propagated to the other site, it gives rise to inconsistency and the two entries regarding the same data will not agree. At such times the data is said to be inconsistent. So, if the redundancy is removed chances of having inconsistent data is also removed.

   An inconsistent database is capable of supplying incorrect or conflicting information. So there should be no inconsistency in database. It can be clearly shown that inconsistency can be avoided in centralized system very well as compared to file system.
4. **Data can be shared:** As explained earlier, the data about Name, Class, Father__name etc. of General_Office is shared by multiple applications in centralized DBMS as compared to file system so now applications can be developed to operate against the same stored data. The applications may be developed without having to create any new stored files.
5. **Standards can be enforced**: Since DBMS is a central system, so standard can be enforced easily may be at Company level, Department level, National level or International level. The standardized data is very helpful during migration or interchanging of data. The file system is an independent system so standard cannot be easily enforced on multiple independent applications.
6. **Restricting unauthorized access:** When multiple users share a database, it is likely that some users will not be authorized to access all information in the database. For example, account office data is often considered confidential, and hence only authorized persons are allowed to access such data. In addition, some users may be permitted only to retrieve data, whereas others are allowed both to retrieve and to update. Hence, the type of access operation retrieval or update must also be controlled. Typically, users or user groups are given account numbers protected by passwords, which they can use to gain access to the database. A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions. The DBMS should then enforce these restrictions automatically.

7. **Solving Enterprise Requirement than Individual Requirement:** Since many types of users with varying level of technical knowledge use a database, a DBMS should provide a variety of user interface. The overall requirements of the enterprise are more important than the individual user requirements. So, the DBA can structure the database system to provide an overall service that is "best for the enterprise".
   For example: A representation can be chosen for the data in storage that gives fast access for the most important application at the cost of poor performance in some other application. But, the file system favors the individual requirements than the enterprise requirements
8. **Providing Backup and Recovery:** A DBMS must provide facilities for recovering from hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update program, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the program started executing.
9. **Cost of developing and maintaining system is lower:** It is much easier to respond to unanticipated requests when data is centralized in a database than when it is stored in a conventional file system. Although the initial cost of setting up of a database can be large, but the cost of developing and maintaining application programs to be far lower than for similar service using conventional systems. The productivity of programmers can be higher in using non-procedural languages that have been developed with DBMS than using procedural languages.
10. **Data Model can be developed:** The centralized system is able to represent the complex data and interfile relationships, which results better data modeling properties. The data madding properties of relational model is based on Entity and their Relationship.
11. **Concurrency Control:** DBMS systems provide mechanisms to provide concurrent access of data to multiple users.

Disadvantages of DBMS

The disadvantages of the database approach are summarized as follows:

1. **Complexity:** The provision of the functionality that is expected of a good DBMS makes the DBMS an extremely complex piece of software. Database designers, developers, database administrators and end-users must understand this functionality to take full advantage of it. Failure to understand the system can lead to bad design decisions, which can have serious consequences for an organization.
2. **Size:** The complexity and breadth of functionality makes the DBMS an extremely large piece of software, occupying many megabytes of disk space and requiring substantial amounts of memory to run efficiently.
3. **Performance:** Typically, a File Based system is written for a specific application, such as invoicing. As result, performance is generally very good. However, the DBMS is written to be more general, to cater for many applications rather than just one. The effect is that some applications may not run as fast as they used to.
4. **Higher impact of a failure:** The centralization of resources increases the vulnerability of the system. Since all users and applications rely on the availability of the DBMS, the failure of any component can bring operations to a halt.

5. **Cost of DBMS:** The cost of DBMS varies significantly, depending on the environment and functionality provided. There is also the recurrent annual maintenance cost.
6. **Additional Hardware costs:** The disk storage requirements for the DBMS and the database may necessitate the purchase of additional storage space. Furthermore, to achieve the required performance it may be necessary to purchase a larger machine, perhaps even a machine dedicated to running the DBMS. The procurement of additional hardware results in further expenditure.
7. **Cost of Conversion:** In some situations, the cost of the DBMS and extra hardware may be insignificant compared with the cost of converting existing applications to run on the new DBMS and hardware. This cost also includes the cost of training staff to use these new systems and possibly the employment of specialist staff to help with conversion and running of the system. This cost is one of the main reasons why some organizations feel tied to their current systems and cannot switch to modern database technology.

Database States:

1. **Online:** The database is available for read and writes queries. This is the default state after creating a new database. This is the only state in which a database can be a replication master. A database cannot be a replication slave in this state.
2. **Provisioning:** Not available for read or write queries. This is the first state after running REPLICATE DATABASE dbName FROM ……. During this state, the database is a replication slave and is currently downloading and replying a snapshot file from the replication master. Once the snapshot has been successfully downloaded and replayed, the database will transition to the replicating state.
3. **Replicating:** Available for read but not write queries. The database is a replication slave and is continually downloading and replaying committed transactions from the replication master. To find the lag between master and slave we can use SHOW REPLICATION STATUS.
4. **Replication paused:** Available for read but not write queries. The database is a replication slave, but replication has been paused with PAUSE REPLICATION dbName. From this state, replication can be continued from same or a different master using REPLICATE DATABASE dbName FROM …….
5. **Recovering:** Not available for read or write queries. This is the immediate state after starting MemSQL. The database is currently recovering from snapshot and log files on disk. Once all data is recovered, the database will transition to replicating (if it's a replication slave) or online (if it's not a replication slave). If the end of the transaction log is corrupted (e.g. due to power loss) and the database is not a replication slave, then the database will transition to offline.
6. **Offline:** Not available for read or write queries. We make it online through some command only then we can use it.
7. **Unrecoverable:** Not available for read or write queries. The database enters this state if recovery of the snapshot fails. Such a failure occurs if MemSQL runs out memory while loading the snapshot or if the snapshot file is damaged.

Database System Concepts and Architecture

Types of Database:

There are two main types of database:

1. **Flat File:** The flat file style of database are ideal for small amounts of data that needs to be human readable or edited by hand. Essentially all they are made up of is a set of strings in one or more files that can be parsed to get the information they store; great for storing simple lists and data values, but can get complicated when you try to replicate more complex data structures. That's not to say that it is impossible to store complex data in a flat-file database; just that doing so can be more costly in time and processing power compared to a relational database. The methods used for storing the more complex data types, are also likely to render the file unreadable and un-editable to anyone looking after the database.

   The typical flat file database is split up using a common delimiter. If the data is simple enough, this could be a comma, but more complex strings are usually split up using tabs, new lines or a combination of characters not likely to be found in the record itself.
2. **Relational database:** The relational databases such as MYSQL, Microsoft SQL Server and Oracle, have a much more logical structure in the way that it stores data. Tables can be used to represent real world objects, with each field acting like an attribute. For example, a table called books could have the columns title, author and ISBN, which describe the details of each book where each row in the table is a new book.

   The 'relation' comes from the fact that the tables can be linked to each other, for example the author of a book could be cross-referenced with the authors table (assuming there was one) to provide more information about the author. These kind of relations can be quite complex in nature, and would be hard to replicate in the standard flat-file format.

Applications of Database System:

Database is used in wide variety of application. None of the organization can be run without database. Every organization, individual, and others are critically depends on their database system. It is the foundation stone over which an organization depends.

1. **Banking:** For customer information, accounts, loans and banking transaction.

2. **Airlines:** Airlines were among the first to use database in a geographically distributed manner terminals situated around the world-access the central database system through phone lines and other data network.
3. **Universities:** For student information, course-registration and grades.
4. **Credit Card Transactions:** For purchase on credit card and generation of monthly statements.
5. **Telecommunication:** For keeping records of calls made, generating monthly bills, maintaining balance on prepaid calling cards, and storing information about the communication networks.
6. **Finance:** For storing information about holding, sales, and purchase of financial instruments such as stock and bonds.
7. **Sales:** For customer, product and purchase information.

2.1 Data models

DBMS ARCHITECTURE

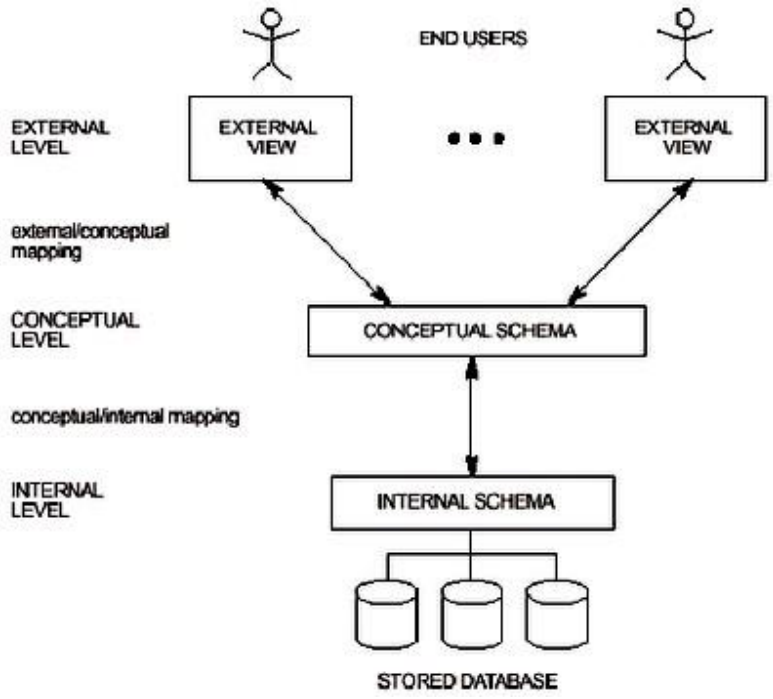


Fig: Three-Schema DBMS Architecture

The goal of the three-schema architecture, illustrated in above Figure, is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels:

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. A high-level data model or an implementation data model can be used at this level.

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. A high-level data model or an implementation data model can be used at this level.

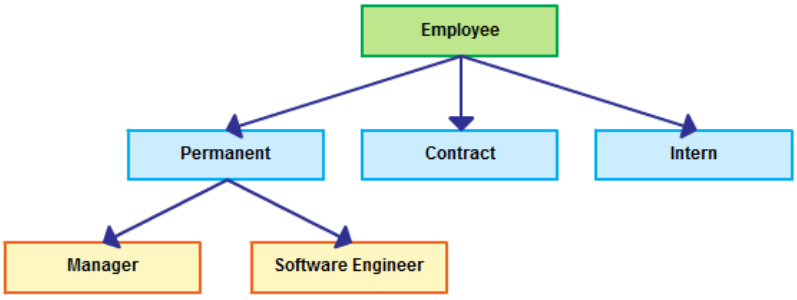
The three-schema architecture is a convenient tool for the user to visualize the schema levels in a database system. In most DBMSs that support user views, external schemas are specified in the same data model that describes the conceptual-level information. Some DBMSs allow different data models to be used at the conceptual and external levels. Notice that the three schemas are only descriptions of data; the only data that actually exists is at the physical level. In a DBMS based on the three schema architecture, each user group refers only to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests and results between levels are called mappings. These mappings may be time-consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views. Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

Database Models:

Database Model:

The description of database is called database schema and it is specified during database design. The process of designing the database schema is called data modelling. The data modelling describes the structure of database such as no of tables, data types, relationship etc. that should hold on the data. Following are the commonly used database model:

a) **Hierarchical Model:** It is one of the oldest database models. In a hierarchical model, data is organized into a tree-like structure, implying a single upward link in each record to describe the nesting, and a sort field to keep the records in a particular order in each same-level list. Hierarchical structures were widely used in the early mainframe database management systems, such as the Information Management System (IMS) by IBM, and now describe the structure of XML documents.



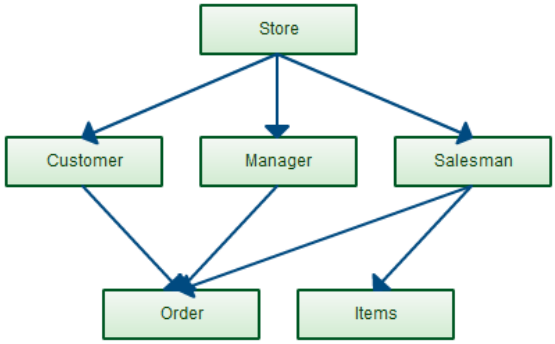
Advantages:

i. It is the easiest model of database
ii. Searching is fast and easy if parent is known.
iii. This model is very efficient in handling 'one-to-many' relationship.

Disadvantages:

i. It is old and outdated database model.
ii. Modification and addition of the child node is very hard. Hence, it is non-flexible database model.
iii. It can't handle 'many-to-many' relationship.

b) Network Model: The network model organizes data using two fundamental concepts, called *records* and *sets*. Records contain fields which may be organized hierarchically. Sets (not to be confused with mathematical sets) define one-to-many relationships between records: one owner, many members. A record may be an owner in any number of sets, and a member in any number of sets. The network model is a variation on the hierarchical model, to the extent that it is built on the concept of multiple branches (lower-level structures) emanating from one or more nodes (higher-level structures), while the model differs from the hierarchical model in that branches can be connected to multiple nodes. The network model is able to represent redundancy in data more efficiently than in the hierarchical model.



Advantages:

i. More flexible than hierarchical model.
ii. Reduces data redundancy because similar data is not stored in more than one file.
iii. Searching is faster because of multidimensional pointers.

Disadvantages:

i. It is very complex to design.
ii. Needs long program to handle the relationship.
iii. Pointers, needed in the database, model increases overhead of database storage.
iv. Less security model because data can be accessed from any parents.

c) Relational Model: It is the best and the most common database model developed by E.F. Codd. The relational database model basically defines the structure or organization of data and a set of operations on the data. It is a simple model in which database is represented as a collection of 'relation', where relation is represented by a two dimensional table.

Advantages:

i. Since, one table is linked to other tables with some common fields; rules implemented on one table can easily be implemented to another table.
ii. Referential integrity can be easily implemented.
iii. The database has very less data redundancy.
iv. Normalization of the database is possible.
v. Rapid database processing and searching is possible.

Disadvantages:

i. It is more complex than other models due to relationship with other tables.
ii. Too many rules make the database not very user-friendly.

d) Object Oriented Model: It can be seen as extending the E-R model with notion of encapsulation, method, and object identity. It combines the feature of object oriented data model and relational data model. In this model, attributes and methods that operate on those attributes are encapsulated in structures called object classes. Relationships between object classes are shown in party by nesting or encapsulating one object class within another. New object classes are defined from more general object classes. A major advantage of this data model is that complex data types like graphics, video and sound are supported as easily as simpler data types. It support complex data and event driven programming is appropriate for the application.

e) Operational Database: Operational database (OLTP: Online Transaction Processing) on the other hand, are used to manage more dynamic bits of data. These types of databases allow us to do more than simply view archived data. Operational databases allow us to modify that data (add, change or delete data).

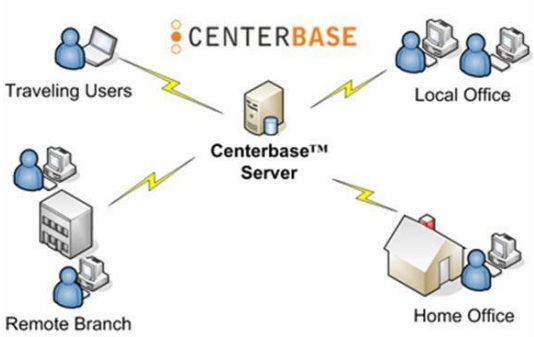
These types of databases are usually used to track real-time information. For example, a company might have an operational database used to track warehouse/stock quantities. As customers order products from an online web store, an operational database can be used to keep track of how many items have been sold and when the company will need to reorder stock.

2.2 DBMS architecture and Data independence

Database Architecture:

1. Centralized Database:

The centralized database has one central computer, called server, to store all the data and files and it provides services to all the clients in the networks. Only the central computer or database server is responsible for processing the data. Data retrieve is not very difficult and security also is not so crucial part as a DBA is there as a controller of the whole database.



Advantages:

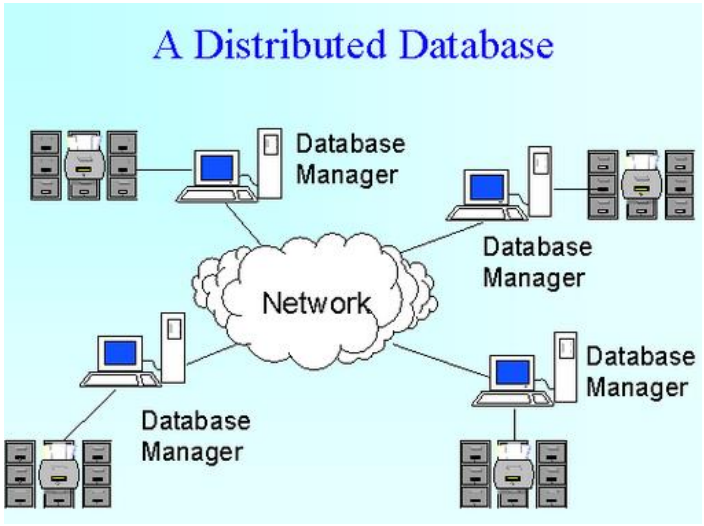
i. Low cost to set up
ii. High performance
iii. Centralization of all data in a single computer called server.
iv. Easier to manage and manipulate data and database.
v. High security
vi. Suitable for small organization.
vii. Easier data access.

Disadvantages:

i. Cannot cover large area and not suitable for large organization.
ii. Database is location dependent, cannot be accessed from other places.
iii. It doesn't support globalized connection.

2. Distributed Database:

It is a set of database stored on multiple computers that appears to applications as a single database. The user can simultaneously access and modify the data in several databases in a network. The computers in a distributed system communicate with each other through various communication media, such as high speed buses or telephone line. The computers neither share the main memory nor a clock cycle of processor, although to work properly many applications on different computers might have to synchronize their clicks.



Advantages:

i. Data sharing and distributed control all over the world.
ii. Improved reliability for users.
iii. Improved availability of data.

iv. Economy on operating and data sharing.
v. Modular growth can support.

Disadvantages:

i. Higher software development cost.
ii. Greater potential for bugs and hacked.
iii. Increased processing overhead for client and server computers.
iv. More complexity in database design.
v. Less security model because data may travel continent to continent.
vi. More difficult for general integrity.

3. Client – Server: This brings out the so called client/server topology that has functionality split between a server and multiple clients. The client/server topology can be grouped into multiple client/single servers and multiple client/multiple server configurations. Functionality and processing capability of the client processors and communication speed between the client and server also distinguishes two classes of the client/server topology, namely, transaction server (thin client) and data server (fat client). The client/server topology is one step towards distributed processing. It offers a user-friendly environment, simplicity of implementation, and high degree of hardware utilization at the server side.

4. Parallel (Multiprocessor): A parallel database system seeks to improve performance through parallelization of various operations, such as loading data, building indexes and evaluating queries. Although data may be stored in a distributed fashion, the distribution is governed solely by performance considerations. Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel. In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially.

Data Independence

We can define two types of data independence:

1. **Logical data independence**:

It is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), or to reduce the database (by removing a record type or data item). In the latter case, external schemas that refer only to the remaining data should not be affected. Only the view definition and the mappings need be changed in a DBMS that supports logical data independence. Application programs that reference the external schema constructs must work as before, after the conceptual schema undergoes a logical reorganization. Changes to constraints can be applied also to the conceptual schema without affecting the external schemas or application programs.

2. **Physical data independence**:

It is the capacity to change the internal schema without having to change the conceptual (or external) schemas. Changes to the internal schema may be needed because some physical files had to be reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema. Whenever

we have a multiple level DBMS, its catalogue must be expanded to include information on how to map requests and data among the various levels. The DBMS uses additional software to accomplish these mappings by referring to the mapping information in the catalogue. Data independence is accomplished because, when the schema is changed at some level, the schema at the next higher level remains unchanged; only the mapping between the two levels is changed. Hence, application programs referring to the higher-level schema need not be changed.

2.3 Database Languages

Database language provides a database to specify its schema and a data manipulation language to express database queries and updates.

1. **Data Definition Language (DDL):** A database schema can be expressed by a set of definition called DDL. These commands are used to create, alter (change), drop tables and establishing constraints. These commands will primarily be used by database administrators during the setup and removal phases of a database project. The term was first introduced in relation to the Codasyl database model, where the schema of the database was written in a Data Definition Language describing the records, fields, and "sets" making up the user Data Model. Initially it referred to a subset of SQL, but is now used in a generic sense to refer to any formal language for describing data or information structures, like XML schemas. Some DDL commands are create, alter, drop etc. such as:
    - **CREATE:** To create objects in the database.
    - **ALTER:** Alters the structure of the database.
    - **DROP:** Delete objects from the database.

    This language contains the update of special set of tables called data dictionary or data directory. A data dictionary contains metadata – data about data. The database system consults the data dictionary before reading and modifying actual data.
2. **Data Manipulation Language (DML):** It is a language that enables users to access or manipulate data is organized by a special data model:
    Procedural DML: It requires a user to specify what data are needed and how to get those data.
    Declarative DML: It requires a user to specify what data are needed without specifying how to get those data.

    DML statements are used to work with the data in tables. These are the core commands of SQL. These commands are used for updating, inserting, modifying and querying the data in the database. They may be issued interactively, so that a result is returned immediately following the execution of the statement or they may be included within programs written in programming language. Some DML commands are as follows:
    - **SELECT:** Retrieve data from the database.
    - **INSERT:** Insert data into a table.
    - **UPDATE:** Updates existing data within a table.
    - **DELETE:** Deletes all records from a table, the space for the records remain.

2.4 Database users and database administrators

Database users are of 4 different types:

1) Naive users:

These are the unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

E.g. consider a user who checks for account balance information over the World Wide Web. Such a user access a form, enters the account number and password etc. And the application program on the internet then retrieves the account balance using given account information which s passed to the user.

2) Application programmers:

These are computer professionals who write application programs, used to develop user interfaces. The application programmer uses Rapid Application Development (RAD) toolkit or special type of programming languages which include special features to facilitate generation of forms and display of date on screen.

3) Sophisticated users:

These users interact with the database using database query language. They submit their query to the query processor. Then Data Manipulation Language (DML) functions are performed on the database to retrieve the data. Tools used by these users are OLAP (Online Analytical Processing) and data mining tools.

4) Specialized users:

These users write specialized database applications to retrieve data. These applications can be used to retrieve data with complex data types e.g. graphics data and audio data.

Database Administrator (DBA):

A DBA is a person who is responsible for maintaining the RDBMS in an organization. In any organization, a chief administrator is needed to oversee and manage the resources of database because the same data resources of the database are shared by all the staffs and users. A DBA is such a chief person, who is assigned to take care of database and DBMS software.

Qualities of a good DBA:

i. The DBA should have depth knowledge of OS in which database server is running.
ii. They should have sound knowledge of SQL.
iii. They should have sound knowledge of good database design.
iv. They should have sound understanding of network architecture.
v. Good knowledge of database server.

DBA Responsibilities:

- **Installing and upgrading a database server:** The DBA is responsible for installing and upgrading database server. In the case of upgrading database server, the DBA is responsible for ensuring that if the upgrade is not successful, the server can be rolled back to an earlier release until the upgrade issues can be resolved.
- **Using storage properly:** Database server enables us to automatically grow the size of your database and transaction logs, or we can choose to select a fixed size for the database transaction log. Maintaining the proper use of storage means monitoring space requirement and adding new storage space.
- **Performing backup and recovery duties:** It include the aspects like establishing standards and schedules for database backups, developing recovery procedure for each database and making sure that the backup schedules meet the recovery requirements.
- **Managing database users and security:** With any database servers, the DBA works tightly with the network administrator to add network users to the database or the DBA adds user logins. The DBA is also responsible for assigning users to the database and determining the proper security level for each user. Within each database the DBA is responsible for assigning permission to the various database objects such as tables, views, and procedures.
- **Working with developers:** It is important for the DBA to work closely with development terms to assist in overall database design, such as creating normalized databases, helping developers tune queries.
- **Establishing and enforcing standards:** The DBA should establish naming conventions and standards for the database server and databases and make sure that everyone sticks to them.
- **Scheduling events:** The DBA is responsible for setting up and scheduling various events to aid in performing many tasks such as backups and replication.
- **Providing 24-hour access:** The database server must stay up, and the database must always be protected and online. Be prepared to perform some maintenance and upgrades after hours. If the database server should go down, be ready to get the server up and running.
- **Learning constantly:** To be a good DBA, we must continue to study and practice our mission critical procedures such as testing our backups by recovering to a test database. In this business, technology changes very fast, so we must continue learning database server, available client/servers, and database design tools. It is never ending process.

2.5 E-R model:

Features of E-R model:

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

- It describes data as a collection of entities, relationships and attributes.

Entities

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

A collection of a similar kind of entities is called an **Entity Set or entity type.**
An entity type usually has an attribute whose values are distinct for each individual entity in the collection. Such an attribute is called a key attribute and its values can be used to identify each entity uniquely.
Strong entity set: The entity types containing a key attribute are called strong entity type or regular entity types.

Attributes

An attribute is a property used to describe the specific feature of the entity. So to describe an entity entirely, a set of attributes is used.
For example, a student entity may be described by the student's name, age, address, course etc.
An entity will have a value for each of its attributes. For example, for a particular student the following values can be assigned:
Roll No: 45
Name: Preeti
Age: 18
Address: Bharatpur-11, Chitwan
Course: BBA

Domains:

- Each simple attribute of an entity type contains a possible set of values that can be attached to it. This is called the domain of an attribute. An attribute cannot contain a value outside this domain.
- For example: For PERSON entity PERSON_ID has a specific domain, integer values say from 1 to 100.

Types of attributes:
Attributes attached to an entity can be of various types:

1. **Simple:** The attribute that cannot be further divided into smaller parts and represents the basic meaning is called a simple attribute. For example: The 'First name', 'Last name', age attributes of a person entity represent a simple attribute.
2. **Composite:** Attribute that can be further divided into smaller units and each individual unit contains a specific meaning. For example: The NAME attribute of an employee entity can be sub-divided into First name, Last name and Middle name.

3. **Single valued:** Attributes having a single value for a particular entity. For example: Age is a single valued attribute of a student entity.
4. **Multivalued:** Attributes that have more than one value for a particular entity is called a multivalued attribute. Different entities may have different number of values for these kinds of attributes. For multivalued attributes we must also specify the minimum and maximum number of values that can be attached. For example: Phone number for a person entity is a multivalued attribute.
5. **Stored:** Attributes that are directly stored in the database. For example, 'Birth date' attribute of a person.
6. **Derived:** Attributes that are not stored directly but can be derived from stored attributes are called derived attributes. For example, the years of services of a 'person' entity can be determined from the current date and the date of joining of the person. Similarly, total salary of a 'person' can be calculated from 'basic salary' attribute of a 'person'.

Relationships

A relationship can be defined as:

- A connection or set of associations, or
- A rule for communication among entities.

For example, in college the database, the association between student and course entity, i.e., "student opts course" is an example of a relationship.

Relationship sets: A relationship set is a set of relationships of the same type. For example, consider the relationship between two entities sets *student* and course. Collection of all the instances of relationship opts forms a relationship set called relationship type.

Degree: The degree of a relationship type is the number of participating entity types. The relationship between two entities is called binary relationship. A relationship among three entities is called ternary relationship. Similarly, relationship among n entities is called n-ry relationship.

Keys

One or more columns in a database table that is used to sort and/or identify rows in a table. For e.g. if you were sorting people by the field salary then the salary field is the key.

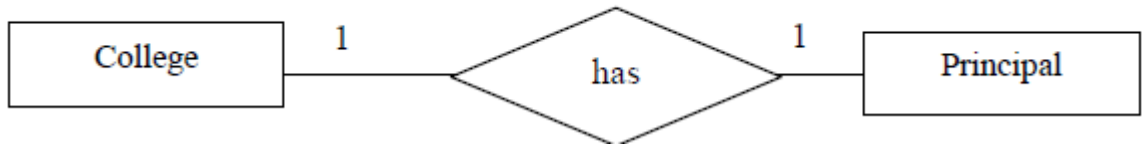
- **Primary key:** A primary key is a one or more fields that uniquely identifies a row in a table. The primary key cannot be null (blank). The primary key is indexed.
- **Foreign key:** A foreign key is a relationship between columns in two database tables (one of which is indexed) designed to insure consistency of data. For e.g. each record in a CUSTOMER table contains the ID of the account manager for that customer. In the ACCOUNT_MANAGER table the ID would typically be the primary key (indexed, unique, not null). The ID field in the CUSTOMER table is the foreign key; only values for ACCOUNT_MANAGER.ID will be allowed in the CUSTOMER.ID field.
- **Composite key:** A primary key composed of one or more columns e.g. a staff table STAFF contains the fields FNAME and LNAME for first and last names respectively.

- **Surrogate key:** A primary key which is internally generated (typically auto-incremental integer value) that does not exist in the real world i.e. ID=1 for Customer A and ID=2 for Customer B serves to uniquely identify the record but has no bearing the customer themselves and is an attribute they will never (need to) be aware of.
- **Candidate key:** A candidate key is a column or group of columns that can uniquely identify a row in the table without referring to any other source. In a table which has multiple candidate keys one is selected to be the primary key. For e.g. you could have an EMPLOYEE table with a candidate key using FULL_NAME and another using DATE_OF_BIRTH.
- **Compound key:** A composite key consisting of two or more fields that uniquely describe a row in a table. The difference between compound and candidate is that all of the fields in the compound key are foreign keys; in the candidate key one or more of the fields may be foreign keys (but it is not mandatory). You could have an EMPLOYEE table with a candidate key using PASSPORT_NUMBER and another using SOCIAL_SECURITY_NUMBER. In exclusion both can uniquely identify a row. Either can be used as a primary key (but not both since a table can have only one primary key).

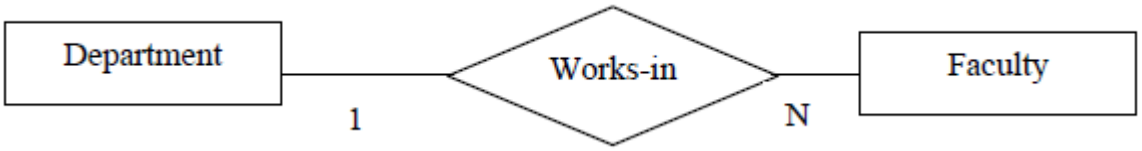
Cardinalities

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

1. **One – to – One:** An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.
   For example: Relationship between college and principal.

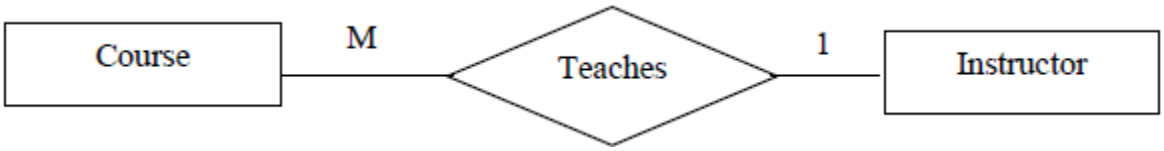


   One college can have at the most one principal and one principal can be assigned to only one college. Similarly, we can define the relationship between university and vice-chancellor.
2. **One – to – many:** An entity in A is associated with any number of entities in B. An entity in B is associated with the most one entity in A.
   For example: Relationship between department and faculty.

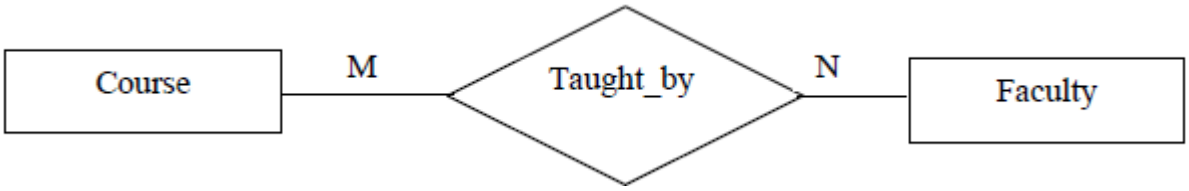


   One department can appoint any number of faculty members but a faculty member is assigned to only one department.
3. **Many – to – one:** An entity in A is associated with at most one entity in B. An entity B is associated with any number in A.

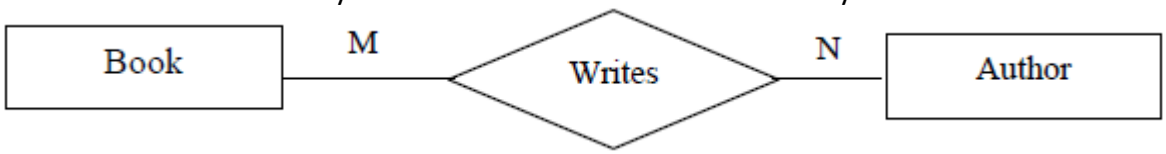
For example: Relationship between course and instructor. An instructor can teach various courses but a course can be taught only by one instructor. Please note this is an assumption.



4. **Many – to – many:** Entities in A and B are associated with any number of entities from each other.
   For example: Taught_by relationship between course and faculty.
   One faculty member can be assigned to teach many courses and one course may be taught by many faculty members.



Relationship between book and author.
One author can write many books and one book can be written by more than one.



Participation constraints

The participation constraints specify whether the existence of an entity depends on its being related to another entity via the relationship type. There are 2 types of participation constraints:

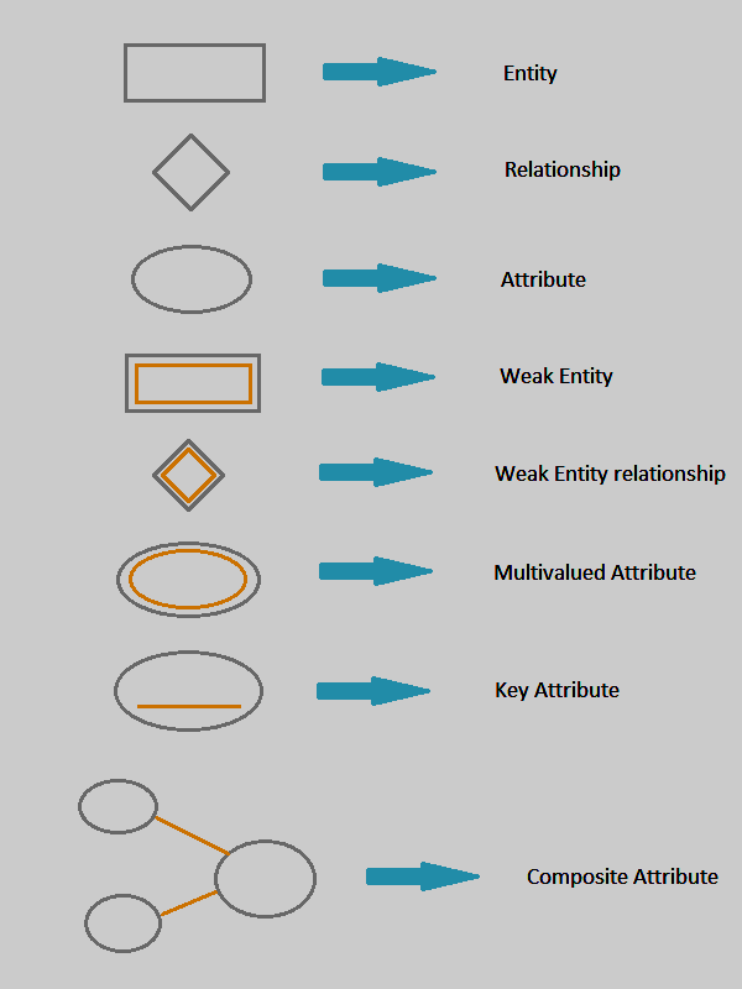
Total: When all the entities from an entity set participate in a relationship type, is called total participation, for example, the participation of the entity set student in the relationship set must 'opts' is said to be total because every student enrolled must opt for a course.

Partial: When it is not necessary for all the entities from an entity set to participate in a relationship type, it is called partial participation. For example, the participation of the entity set student in 'represents' is partial, since not every student in a class is a class representative.

E-R diagram

- An E-R diagram can express the overall logical structure of a database graphically. E-R diagrams are simple and clear— qualities that may well account in large part for the widespread use of the E-R model. Such a diagram consists of the following major components:
- **Rectangles**, which represent entity sets
- **Ellipses**, which represent attributes
- **Diamonds**, which represent relationship sets
- **Lines**, which link attributes to entity sets and entity sets to relationship sets

- **Double ellipses**, which represent multi valued attributes
- **Dashed ellipses**, which denote derived attributes
- **Double lines**, which indicate total participation of an entity in a relationship set
- **Double rectangles**, which represent weak entity sets



Consider the entity-relationship diagram Figure below, which consists of two entity sets, customer and loan, related through a binary relationship set borrower. The attributes associated with customer are customer-id, customer-name, customer-street, and customer-city. The attributes associated with loan are loan-number and amount. In the Figure,attributes of an entity set that are members of the primary key are underlined. The relationship set borrower may be many-to-many, one-to-many, many-to-one, or one-to-one. To distinguish among these types, we draw either a directed line (→) or an undirected line (—) between the relationship set and the entity set in question.

A directed line from the relationship set borrower to the entity set loan specifies that borrower is a one-to-one or many-to-one relationship set, from customer to loan; borrower cannot be a many-to-many or a one-to-many relationship set from customer to loan.

An undirected line from the relationship set borrower to the entity set loan specifies that borrower is either a many-to-many or one-to-many relationship set from customer to loan.

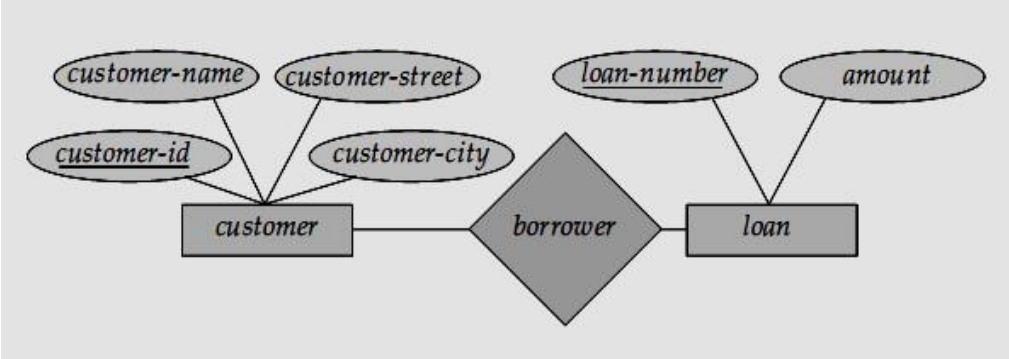


Figure: E-R diagram corresponding to customers and loans.

If a relationship set has also some attributes associated with it, then we link these attributes to that relationship set. Following figure shows how composite attributes can be represented in the E-R notation.

Here, a composite attribute name, with component attributes first-name, middle-initial, and last-name replaces the simple attribute customer-name of customer. Also, a composite attribute address, whose component attributes are street, city, state, and zip-code replaces the attributes customer-street and customer-city of customer. The attribute street is itself a composite attribute whose component attributes are street-number, street-name, and apartment number.

Figure also illustrates a multi valued attribute phone-number, depicted by a double ellipse, and a derived attribute age, depicted by a dashed ellipse.

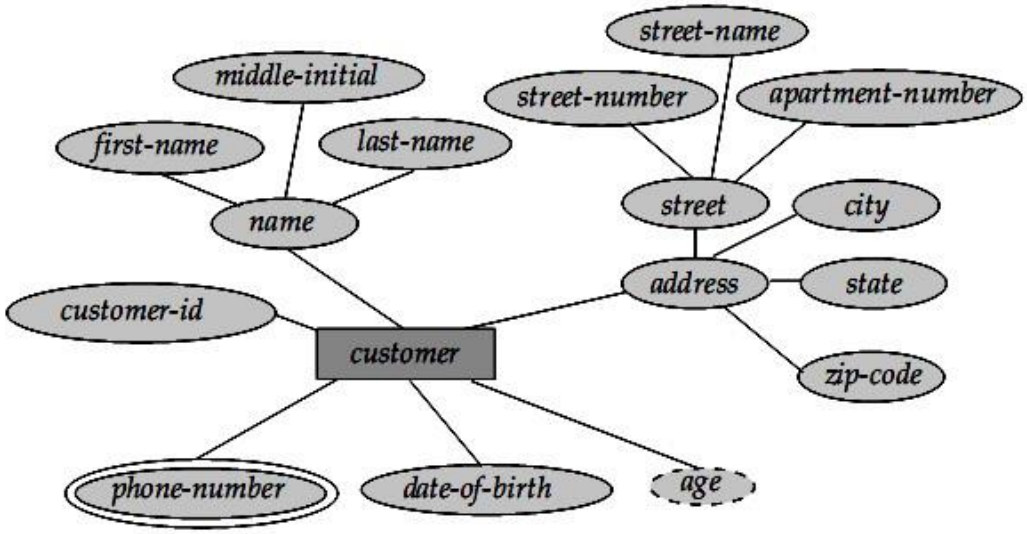
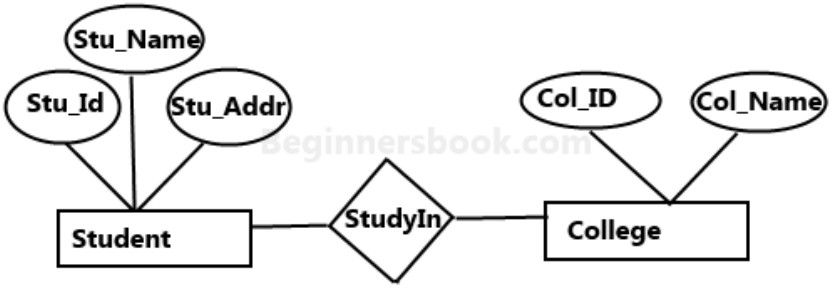
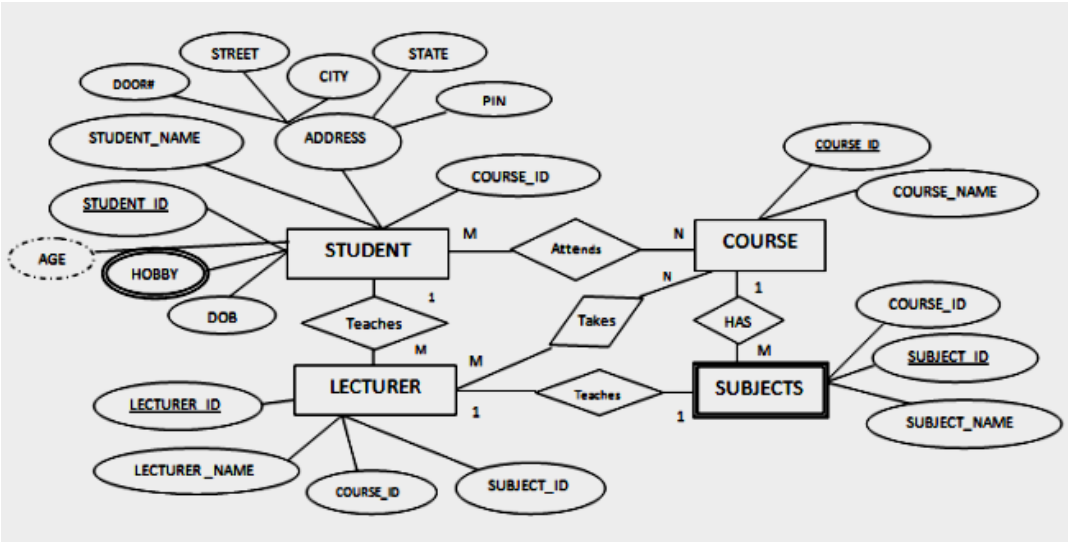


Figure: E-R diagram with composite, multi valued, and derived attributes.

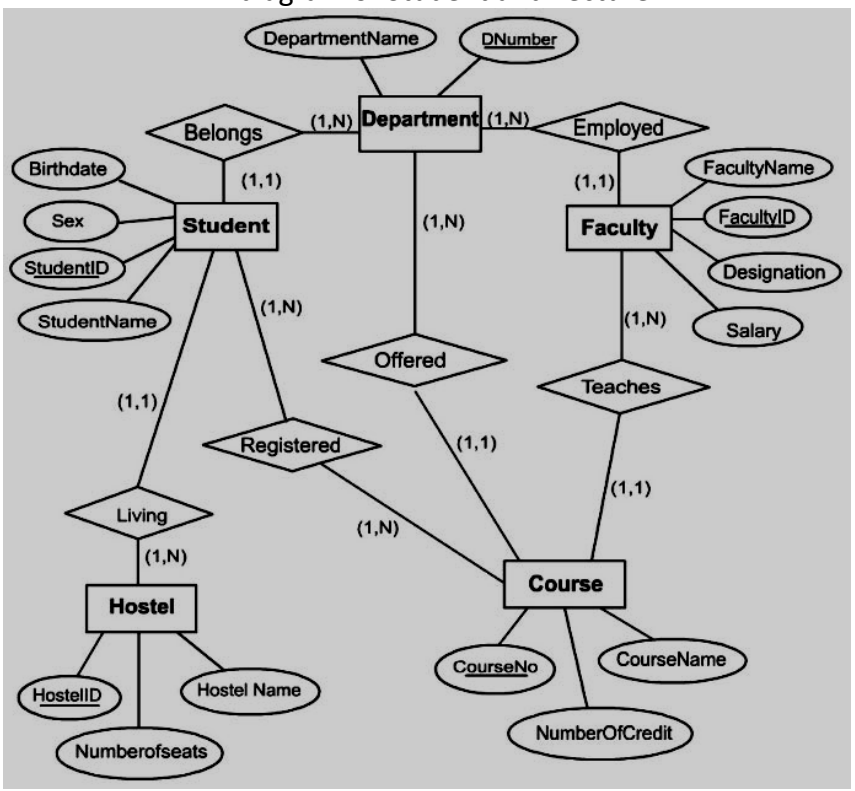
Some Examples of E-R diagram:

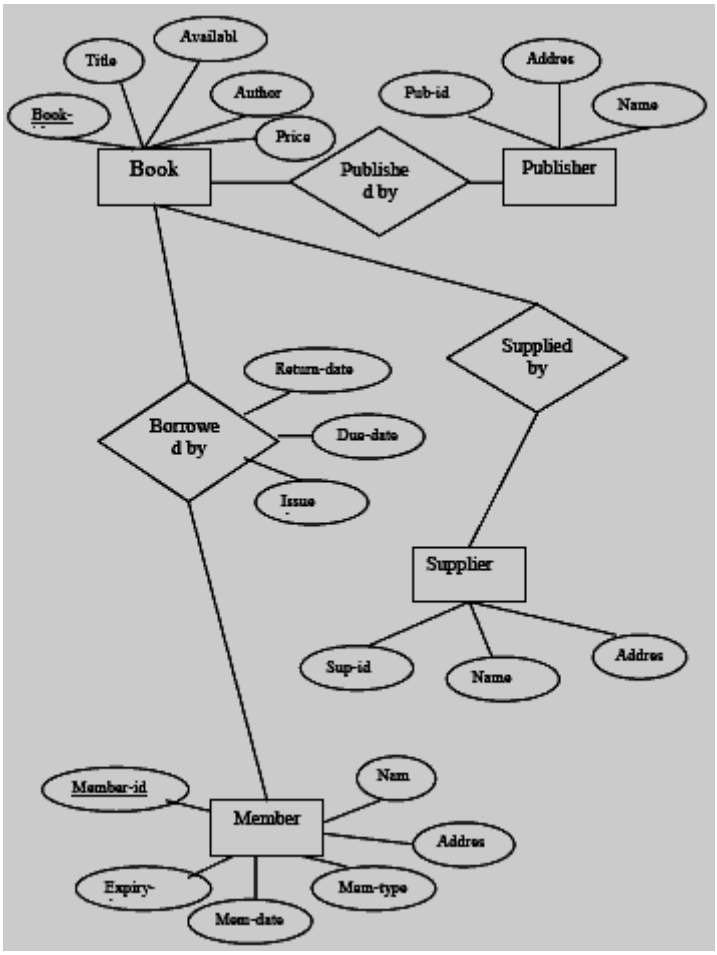


E-R diagram of student and college

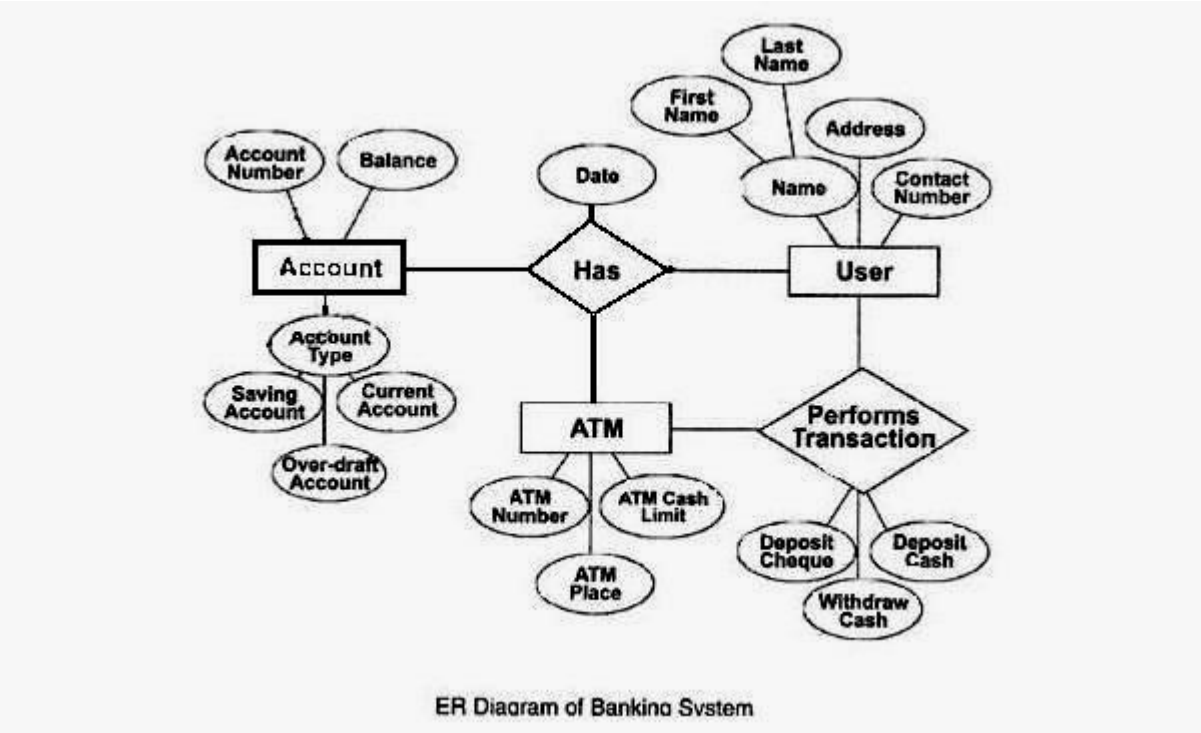


E-R diagram of Student and Lecturer





E-R diagram of Library system



ER Diagram of Banking System

2.6 Data dictionary

A Data Dictionary stores information about the structure of the database. It is used heavily. Hence a good data dictionary should have a good design and efficient implementation. It is seen that when a program becomes somewhat large in size, keeping track of all the available names that are used and the purpose for which they were used becomes more and more difficult. After a significant time if the same or another programmer has to modify the program, it becomes extremely difficult.

The problem becomes even more difficult when the number of data types that an organisation has in its database increases. The data of an organisation is a valuable corporate resource and therefore some kind of inventory and catalogue of it must be maintained so as to assist in both the utilisation and management of the resource. It is for this purpose that a data dictionary or dictionary/directory is emerging as a major tool. A dictionary provides definitions of things. A directory tells you where to find them. A data dictionary/directory contains information (or data) about the data.

An ideal data dictionary should include everything a DBA wants to know about the database.

1. External, conceptual and internal database descriptions.
2. Descriptions of entities (record types), attributes (fields), as well as cross-references, origin and meaning of data elements.
3. Synonyms, authorisation and security codes.
4. Which external schemas are used by which programs, which the users are, and what their authorisations are.
5. Statistics about database and its usage including number of records, etc.

A data dictionary is implemented as a database so that users can query its contents. The cost effectiveness of a data dictionary increases as the complexity of an information system increases. A data dictionary can be a great asset not only to the DBA for database design, implementation and maintenance, but also to managers or end users in their project planning.

Relational Model

A model in database system basically defines the structure or organisation of data and a set of operations on that data. Relational model is a simple model in which database is represented as a collection of "Relations", where each relation is represented by a two dimensional table. Thus, because of its simplicity it is most commonly used. The following table represents a simple relation:

Person_ID	Name	Age	Address
1	Ram Gupta	35	Bharatpur, Chitwan
2	Hari Prasad Dhungana	30	Pokhara, Kaski
3	Nisha Devkota	36	Gorkha Bazar, Gorkha

Following are some of the advantages of relational model:

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

3.1 Properties of relation

- Each row represents an n-tuple of R
- Ordering of rows is immaterial
- All rows are distinct
- Ordering of columns is significant
  - Because two columns can have same domain.
  - But columns are labeled
  - Applications need not worry about order
  - They can simply use the names
- Domain of each column is a primitive type
- Relation consists of a relation schema and instance

3.2 Schemas, Tuples, Domains and Schema diagram

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

Tuple:

Each row in a table represents a record and is called a tuple. A table containing 'n' attributes in a record is called n-tuple.

Attribute:

The name of each column in a table is used to interpret its meaning and is called an attribute. Each table is called a relation.

For example: Following table represents a relation PERSON. The column PERSON_ID, NAME, AGE and ADDRESS are the attributes of PERSON and each row in the table represents a separate tuple (record).

Relation name: PERSON

PERSON_ID	NAME	AGE	ADDRESS	TELEPHONE
1	Ram Gupta	35	Bharatpur, Chitwan	056-560112
2	Hari Prasad Dhungana	30	Pokhara, Kaski	9846067897
3	Nisha Devkota	36	Gorkha Bazar, Gorkha	9846039897

Domain:

A domain is a set of permissible values that can be given to an attribute. So every attribute in a table has a specific domain. Values to these attributes cannot be assigned outside their domains.

In the example above if domain of PERSON_ID is a set of integer values from 1 to 1000 than a value outside this range will not be valid. Some other common domains may be age between 1 and 150. The domain can be defined by assigning a type or a format or a range to an attribute. For example, a domain for a number 501 to 999 can be specified by having a 3-digit number format having a range of values between 501 and 999. However, please note the domains can also be non-contiguous.

Relation:

A relation consists of:

- Relational schema
- Relation instance

Relational Schema:A relational schema specifies the relation's name, its attributes and the domain of each attribute. If R is the name of a relation and A_1, A_2, …… A_n is a list of attributes representing R then R(A_1, A_2, ……, A_n) is called a relational schema. Each attribute in this relational schema takes a value from some specific domain called Domain(A_i).

For example, the relational schema for relation PERSON as in first table will be:
PERSON(PERSON_ID: Integer, NAME: string, AGE: integer, ADDRESS: string)

Total number of attributes in a relation denotes the degree of a relation. Since the PERSON relation contains four attributes, so this relation is of degree 4.

Relation Instance or Relation State: A relation instance denoted as **r** is a collection of tuples for a given relational schema at a specific point of time.
A relation state **r** of the relation schema $R(A_1, A_2, \ldots\ldots\ldots\ldots, A_N)$, also denoted by r(R) is a set of n-tuples.
$r = \{t_1, t_2, \ldots\ldots. t_m\}$
Where, each n-tuple is an ordered list of n values.
$t = <V_1, v_2, \ldots\ldots\ldots, v_n>$
Where, each v_i belongs to domain (A_i) or contains null values.
The relation schema is also called *'intension'* and relation state is also called *'extension'*.
Let us elaborate the definitions above with the help of examples:

Example 1: RELATION SCHEMA For STUDENT:
STUDENT (RollNo: String, Name: string, login: string, age: integer)
RELATION INSTANCE

	ROLLNO	NAME	LOGIN	AGE
t_1	3467	Shikha	Noorie_jan@yahoo	20
t_2	4677	Kanu	Golgin_atat@yahoo	20

Where t_1 = (3467, shikha, Noorie-jan@yahoo.com, 20) for this relation instance,
m = 2 and n = 4

Example 2: RELATIONAL SCHEMA FOR PERSON:
PERSON(PERSON_ID: integer, NAME: string, AGE: integer, ADDRESS: string)
RELATION INSTANCE

Person_ID	Name	Age	Address
1	Ram Gupta	35	Bharatpur, Chitwan
2	Hari Prasad Dhungana	30	Pokhara, Kaski
3	Nisha Devkota	36	Gorkha Bazar, Gorkha

In this instance, m = 3 and n = 4

Thus current relation state reflects only the valid tuples that represent a particular state of the real world. However, Null values can be assigned for the cases where the values are unknown or missing.

3.3 Relational Algebra:

Relational algebra is a set of basic operations used to manipulate the data in relational model. These operations enable the user to specify basic retrieval request. The result of retrieval is a new relation, formed from one or more relations. These operations can be classified in two categories:

Basic Set Operations:

1. UNION
2. INTERSECTION
3. SET DIFFERENCE
4. CARTISIAN PRODUCT

Relational Operations:

1. SELECT
2. PROJECT
3. JOIN
4. DIVISION

Basic Set Operation:

These are the binary operations; i.e., each is applied to two sets or relations. These two relations should be union compatible except in case of *Cartesian Product.* Two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots\ldots, B_n)$ are said to be union compatible if they have the ***same degree n*** and domains of the corresponding attributes are also the same. i.e. ***Domain (A_i) = Domain(B_i) for 1 < = i < = n.***

1. UNION:

If R_1 and R_2 are two union compatible relations then R3 = R1 ∪ R2 is the relation containing tuples that are either in R1 or in R2 or in both.

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

For Example:

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R3 = R1 ∪ R2 is

Q

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

Note: 1) Union is a commutative operation, i.e. R ∪ S = S ∪ R

2) Union is an associative operation, i.e. R ∪(S ∪ T) = (R ∪ S) ∪ T

2. Intersection:

If R1 and R2 are two compatible functions or relations, then the result of R3 = R1 ∩ R2 is the relation that includes all tuples that are in both the relations. In other words, R3 will have tuples such that R3 = {t | R1 ∋ t ∧ R2 ∋ t}.

For Example:

R1

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R2

A	B
A1	B1
A2	B2
A3	B3
A4	B4

$R3 = R1 \cap R2$ is

A	B
A1	B1
A2	B2
A4	B4

Note: 1) Intersection is a commutative operation, i.e. $R1 \cap R2 = R2 \cap R1$.
2) Intersection is an associative operation, i.e., $R1 \cap (R2 \cap R3) = (R1 \cap R2) \cap R3$

3. Set Difference:

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

For Example:

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R1-R2 =

A	B
A3	B3

R2-R1=

A	B
A7	B7

Note: 1) Difference operation is not commutative, i.e., **R1 – R2 ≠ R2 – R1**
2) Difference operation is not associative, i. e., **R1 – (R2 – R3) ≠(R1 – R2) – R3**

4. Cartesian Product:

If R1 and R2 are two functions or relations, then the result of R3 = R1 × R2 is the combination of tuples that are in R1 and R2. The product is commutative and associative.
Degree (R3) =Degree of (R1) + Degree (R2).

In other words, R3 will have tuples such that R3 = {t_1 | t_2 | ∋ R1 ∧t_1 |R2 ∋t_2}.

For Example:

R1

C
C1
C2

R2

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R3 = R1 × R2 is

A	B	C
A1	B1	C1
A1	B1	C2
A2	B2	C1
A2	B2	C2
A3	B3	C1
A3	B3	C2
A4	B4	C1
A4	B4	C2

Relational Operations:

Let us now discuss the relational operations:

1. SELECT:

The select operation is used to select some specific records from the database based on some criteria. This is a unary operation mathematically denoted as σ.

Syntax: $\sigma_{<\text{Selection condition}>}$(Relation)

The Boolean expression is specified in <Select condition> is made of a number of clauses of the form:

<attribute name><comparison operator><constant value>

or

<attribute name><comparison operator><attribute name>

Comparison operators in the set {≤, ≥, ≠, =, <, >} apply to the attributes whose domains are ordered value like integer.

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

$$\sigma_{AGE <= 30} (PERSON)$$

The resultant relation will be as follows:

Person_ID	Name	Age	Address
2	Hari Prasad Dhungana	30	Pokhara, Kaski

Note:

1. Select operation is commutative; i.e.
   $\sigma_{<condition1>}(\sigma_{<condition2>} (R)) = \sigma_{<condition2>} (\sigma_{<conditioin1>}(R))$
   Hence, sequence of select can be applied in any order.
2. More than one condition can be applied using Boolean operators AND, OR etc.

2. The PROJECT Operation:
The project operation is used to select the records with specified attributes while discarding the others based on some specific criteria. This is denoted as Π.
Syntax: $\Pi_{\text{List of attribute for project}}$ (Relation)

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

$$\Pi_{Name} (PERSON)$$

The resultant relation will be as follows:

Name
Ram Gupta
Hari Prasad Dhungana
Nisha Devkota

Note: $\Pi_{<List1>}(\Pi_{<List2>} (R)) = \Pi_{<List1>} (R)$
As long as<list2> contains attributes in <list1>.

3. The JOIN Operation:
The JOIN operation is applied on two relations. When we want to select related tuples from two given relation join is used. This is denoted as '.'. The join operation requires that both the joined relations must have at least one domain compatible attributes.
Syntax: R1 $_{<\text{join condition}>}$ R2 is used to combine related tuples from two relations R1 and R2 into a single tuple.
<join condition> is of the form:
<condition> AND <condition> AND …….. AND <condition>
Degree of relation:
Degree (R1 $_{<\text{join condition}>}$R2) < = Degree (R1) + Degree (R2)
There are three types of joins:

a) Theta Join: When each condition is of the form A θ B, A is an attribute of R1 and B is an attribute of R2 and have the same domain, and θ is one of the comparison operators {$\leq$, $\geq$, $\neq$, =, <, >}.
b) Equijoin: When each condition appears with equality condition (=) only.

c) **Natural join (denoted by R * S):** When two join attributes have the same name in both relations. (That attribute is called Join attribute), only one of the two attributes is retained in the join relation. The join condition in such a case is = for the join attribute. The condition is not shown in the natural join.

Let us show these operations with the help of the following example:
Consider the following relations:

STUDENT

ROLLNO	NAME	ADDRESS	COURSE_ID
100	Ramjee	Bhaktapur	CS01
101	Rupa	Lalitpur	CS02
102	Arpita	Kathmandu	CS04

COURSE

COURSE ID	COURSE_NAME	DURATION
CS01	MCA	3 yrs
CS02	BCA	3 yrs
CS03	M.Sc.	2 yrs
CS04	B.Sc.	3 yrs
CS05	MBA	2 yrs

If we want to display name of all the students along with their course details then natural join is used in the following way:

STUDENT . COURSE
Resultant relation will be as follows:

STUDENT

ROLLNO	NAME	ADDRESS	COURSE_ID	COURSE_NAME	DURATION
100	Ramjee	Bhaktapur	CS01	MCA	3 yrs
101	Rupa	Lalitpur	CS02	BCA	3 yrs
102	Arpita	Kathmandu	CS04	B.Sc.	3 yrs

Outer join

The outer-join operation is extension to natural join. It has a capability to deal with missing information. There are three form of outer-join operation

(a) Left outer-join (⟕)
- Takes all tuples in the left relation. If there are any tuples in right relation that does not match with tuple in left relation, simply pad these right relation tuples with null.
- Add them to the result of the left outer-join.

(b) Right outer-join(⟖)
- Takes all tuples in the right relation. If there are any tuples in the left relation that does not match with tuple in right relation, simply pad left relation tuples with null.
- Add them to the result of the left outer-join.

(c) Full outer-join (⟗)
- Pad tuples from the left relation that that did not match any from the right relation
- Pad tuples from the right relation that that did not match any from the left relation
- Add them to the result of full outer-join.

Example:

Consider relations

loan_number	branch_name	amount
L01	B1	500
L02	B2	600
L05	B1	700

Relation loan

customer_name	loan_number
X	L01
Y	L02
Z	L07

Relation borrower

Natural join (Inner join)

Loan ⋈ borrower

loan_number	branch_name	amount	customer_name
L01	B1	500	X
L02	B2	600	Y

Left outer-join

loan ⟕ borrower

loan_number	branch_name	amount	customer_name
L01	B1	500	X
L02	B2	600	Y
L05	B1	700	null

Right outer-join

Loan ⟖ borrower

loan_number	branch_name	amount	customer_name
L01	B1	500	X
L02	B2	600	Y
L07	null	null	Z

Full outer-join

Loan ⟗ borrower

loan_number	branch_name	amount	customer_name
L01	B1	500	X
L02	B2	600	Y
L05	B1	700	null
L07	null	null	Z

4. The DIVISION operation:

To perform the division operation R1 ÷ R2, R2 should be a proper subset of R1. In the following example R1 contains attributes A and B and R2 contains only attribute B so R2 is a proper subset of R1. If we perform R1÷ R2 than the resultant relation will contain those values of A from R1 that are related to all values of B present in R2.

Example:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ϵ	6
ϵ	1
β	2

Relation r

B
1
2

Relation s

r ÷ s :

A
α
β

Assignment Operator:

The assignment operation provides convenient way to express complex query. The assignment operation denoted by ←, works like assignment in programming language. The evaluation of an assignment does not result any relation being displayed to the user. But the result of the expression to the right of the ← is assigned to the relation variable. This relation variable may used in subsequent expressions. With the assignment expression, a query can be written as a sequential program consisting a series of assignments followed by an expression whose value is displayed as the result of the query.

For example:

Find all customer who taken loan from bank as well as he/she has bank account.

$$Temp1 \leftarrow \Pi_{customer\_name} (borrower)$$
$$Temp2 \leftarrow \Pi_{customer\_name} (depositor)$$
$$result \leftarrow Temp1 \cap Temp2$$

Rename Operation:

The result of relational-algebra expression does not have a name to refer it. It is better to give name to result relation. The rename operator is denoted by lower case Greek letter rho (ρ). Rename operation in relation-algebra expressed as

$$\rho\ x(E)$$

where E is a relational algebra expression and x is name for result relation. It returns the result of expression E under the name x.

Since a relation r is itself a relational-algebra expression thus, the rename operation can also apply to rename the relation r (i.e. to get same relation under a new name). Rename operation can also used to rename attributes of relation. Assume a relational algebra expression E has arity n. Then expression

$$\rho_{x(A1,A2,\ldots,An)}(E)$$

returns the result of expression E under the name x and it renames attributes to A1,A2, ..., An.

Some solved examples:

1. Select those tuples of loan relation where the branch is Kathmandu.

$$\sigma_{branch\_name = "Kathmandu"} (loan)$$

2. Find all tuples in loan relation in which amount loan is more than 5000.

$$\sigma_{amount>5000} (loan)$$

3. Find all tuples in loan relation where amount is more than 5000 and branch is Kathmandu.

$$\sigma_{branch\_name = "Kathmandu" \wedge amount>5000} (loan)$$

4. Find account number and their balance from account relation.

$$\Pi_{account\_number,\ balance} (account)$$

5. Find those customers who stay in Kathmandu.

$$\Pi_{customer\_name} (\sigma_{customer\_city = "Kathmandu"} (customer))$$

6. Find all customers with either account or loan.

$$\Pi_{customer\_name} (depositor) \cup \Pi_{customer\_name} (borrower)$$

7. Find all customer of the bank who have account but not loan.

$$\Pi_{customer\_name} (depositor) - \Pi_{customer\_name} (borrower)$$

8. Find all customer who taken loan from branch "B1".

$$\Pi_{customer\_name} (\sigma_{borrower.loan\_number = loan.loan\_number} (\sigma_{branch\_name = "B1"} (borrower \times loan)))$$

9. Find the largest account balance in the bank.

$$\Pi_{balance} (account) - \Pi_{account.balance} (\sigma_{account.balance < d.balance} (account \times \rho_d (account)))$$

Process:

account_number	balance
A1	500
A2	600
A3	700

Relation account

Account_number	balance
A1	500
A2	600
A3	700

Relation d

$$account \times \rho_d (account)$$

account.account_number	account.balance	d.balance
A1	500	500
A1	500	600
A1	500	700
A2	600	500
A2	600	600
A2	600	700
A3	700	500
A3	700	600
A3	700	700

$$\Pi_{account.balance} (\sigma_{account.balance<d.balance} (account \times \rho_d (account)))$$

$\Pi_{balance}(account) - \Pi_{account.balance}(\sigma_{account.balance<d.balance}(account \times \rho_{d}(account)))$

balance
500
600
700

account.balance
500
600

Output:

balance
700

10. Find all customer who have both loan and account.

$\Pi_{customer\_name}$ (borrower) $\cap \Pi_{customer\_name}$ (depositor)

Difference between relational algebra and SQL

S.N.	Relational Algebra	SQL
1	It is closed (the result of every expression is a relation)	It is a superset of relational algebra
2	It has a rigorous foundation	It has convenient formatting features.
3	It has simple semantics.	It has complicated function.
4	It is used for reasoning, query optimisation etc.	It is an end-user language.

Structured Query Language

4.1 Introduction to SQL, DDL and DML

Structure Query Language (SQL) is a programming language used for storing and managing data in RDBMS. SQL was the first commercial language introduced fro E.F. Codd's Relational model. Today almost all RDMBS (MySql, Oracle, Infomix, Sybase, MS Access) uses SQL as the standard database language. SQL is used to perform all type of data operations in RDBMS.

SQL Command:
SQL defines following data languages to manipulate data of RDBMS.

DDL : Data Definition Language:
All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

Command	Description
create	to create new table or database
alter	for alteration
truncate	delete data from table
drop	to drop a table
rename	to rename a table

DML: Data Manipulation Language
DML commands are not auto-committed. It means changes are not permanent to database, they can be rolled back.

Command	Description
insert	to insert a new row
update	to update existing row
delete	to delete a row
merge	merging two rows or two tables.

4.2 Basic structure of SQL statements

SQL (pronounced "ess-que-el") is the acronym for Structured Query Language. SQL is the standard language for communicating with relational database management systems. SQL statements are used to retrieve data from the database as well as perform tasks such as adding, updating and deleting the data.

Basic Query Structure:
SELECT field1 [, "field2", etc]
FROM table
[WHERE "condition"]
[GROUP BY "field"]
[ORDER BY "field"]

[ ] = optional

The **SELECT** statement is used to query the database and retrieve the fields that we specify. We can select as many fields (column names) as we want, or use the asterisk symbol (*) to select all fields.

The **FROM** statement specifies the table names that will be queried to retrieve the desired data.

The **WHERE** clause (optional) specifies which data values or rows will be returned or displayed, based on the criteria we specify.

The **GROUP BY** clause (optional) organizes data into groups.

The **ORDER BY** clause (optional) sorts the data by the field specified.

=	Equal
>	Greater than
<	Less than
> =	Greater than or equal to
< =	Less than or equal to
<>	Not equal to
Like	String comparison test.

4.3 SQL Queries:

SQL SELECT Statement:
The SELECT statement is used to select data from a database. The result is stored in a result table, called the result-set.
Syntax:
SELECT column_name, column_name
FROM table_name;

*SELECT * FROM table_name;*

For Example:
SELECT CustomerName, City FROM Customers;
The above statement selects the "CustomerName" and "City" columns from the "Customers" table.
SELECT * FROM Customers
The above statement selects all the columns from the "Customers" table.

SQL SELECT DISTINCT Statement:
The SELECT DISTINCT statement is used to return only distinct (different) values.
In a table, a column may contain many duplicate values; and sometimes we only want to list the different (distinct) values.
The DISTINCT keyword can be used to return only distinct (different) values.
Syntax:
SELECT DISTINCT column_name, column_name
FROM table_name;

For Example:
SELECT DISTINCT City FROM Customers;

SQL WHERE Clause:
The where clause is used to filter the records.
The WHERE clause is used to extract only those records that fulfil a specified criterion.

Syntax:
SELECT column_name, column_name
FROM table_name
WHERE column_name operator value;

For Example:
SELECT * FROM Customers
WHERE Country = 'Nepal';

Text Fields Vs. Numeric Fields:
SQL requires single quotes around text values (most database systems will also allow double quotes).
However, numeric fields should not be enclosed in quotes:
For Example:
SELECT * FROM Customers
WHERE CustomerID = 1;

Operators in the WHERE Clause:
The following operators can be used in the WHERE clause:

Operator	Description
=	Equal
<>	Not equal (In some versions of SQL this operator may be written as !=)
>	Greater than
<	Less than
> =	Greater than or equal
< =	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

SQL AND & OR Operators:
The AND & OR operators are used to filter records based on more than one condition.
The AND operator displays a record if both the first condition and the second condition are true.

The OR operator displays a record if either the first condition or the second condition is true.

For Example:
SELECT * FROM Customers
WHERE Country = 'Nepal'
AND City = 'Bharatpur';

The above SQL statement selects all customers from the country "Nepal" and the city "Bharatpur" in the "Customer" table.

For Example:
The following SQL statement selects all customers from the city "Bharatpur" or "Tandi", in the "Customer" table:
SELECT * FROM Customers
WHERE City='Bharatpur'
OR City="Tandi";

Combining AND & OR
We can also combine AND and OR (use parenthesis to form complex expressions).
The following SQL statement selects all customers from the country "Nepal" and the city must be equal to "Bharatpur" or "Tandi", in the "Customers" table:
SELECT * FROM Customers
WHERE Country = 'Nepal'
AND (City='Bharatpur' OR City='Tandi');

SQL ORDER BY Keyword:
The ORDER BY keyword is used to sort the result-set by one or more columns. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, we can use the DESC keyword.
Syntax:
SELECT column_name, column_name
FROM table_name
ORDER BY column_name ASC|DESC, column_name ASC|DESC;

For Example:
The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:
SELECT * FROM Customers
ORDER BY Country;

ORDER BY DESC Example:
The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:
SELECT * FROM Customers
ORDER BY Country DESC;

ORDER BY Several columns:
The following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column:
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;

SQL INSERT INTO Statement:
The INSERT INTO statement is used to insert new records in a table. It is possible to write the INSERT INTO statement in two forms:
The first form does not specify the column names where the data will be inserted, only their values:
INSERT INTO table_name
VALUES (value1, value2, value3, ……..);

The second form specifies both the column names and the values to be inserted:
INSERT INTO table_name (column1, column2, column3, …)
VALUES (value1, value2, value3, …);

For Example:
Assume we wish to insert a new row in the "Customers" table.
We can use the following SQL Statement:
INSERT INTO Customers (CustomerName, Address, City, Country)
VALUES ('Namita', 'Bharatpur-11', 'Chitwan', 'Nepal');

SQL UPDATE Statement:
The UPDATE statement is used to update existing records in a table.

Syntax:
UPDATE table_name
SET column1=value1, column2=value2, …
WHERE some_column=some_value;

UPDATE Multiple Columns:
To update more than one column, use a comma as separator.
Assume we wish to update the customer "Namita" with a new Address and City, we use the following SQL statement:
UPDATE Customers
SET Address = 'Pokhara – 2', City='Kaski'
WHERE CustomerID=1;

UPDATE Multiple Records:

In an update statement, it is the WHERE clause that determines how many records which will be updated.
The WHERE clause: WHERE Country = 'Nepal' will update all records which have the value "Nepal" in the field "Country".
For Example:
UPDATE Customers
SET Address='Tandi'
WHERE Country='Nepal';

SQL DELETE Statement
The DELETE statement is used to delete rows in a table.
Syntax:
DELETE FROM table_name
WHERE some_column = some_value;

For Example:
Assume we wish to delete the customer "Namita" from the "Customers" table, we use the following SQL statement:
DELETE FROM Customers
WHERE CustomerName = 'Namita' AND City = 'Tandi';

DELETE All Data:
It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:
Syntax:
DELETE FROM table_name;
or
*DELETE * FROM table_name;*

SQL LIKE Operator:
The LIKE operator is used to search for a specified pattern in a column.
Syntax:
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;

For Example:
The following SQL statement selects all customers with a city starting with the letter "N":
SELECT * FROM Customers
WHERE City LIKE 'N%';
Tips: The "%" sign is used to define wildcards (missing letters) both before and after the pattern. The following SQL statement selects all customers with a City ending with the letter "L":
SELECT * FROM Customers
WHERE City LIKE '%s';

The following SQL statement selects all customers with a country containing the pattern "land":
SELECT * FROM Customers
WHERE Country LIKE '%land%';

Using NOT keyword allows us to select records that do not match the pattern.
The following SQL statement selects all customers with Country not containing the pattern "land".
SELECT * FROM Customers
WHERE Country NOT LIKE '%land%'

SQL IN Operator:
The IN operator allows us to specify multiple values in a WHERE clause.
Syntax:
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, …);

For Example:
The following SQL statement selects all customers with a City of "Paris" or "London".
SELECT * FROM Customers
WHERE City IN ('Paris', 'London');

SQL BETWEEN Operator:
The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.
Syntax:
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;

For Example:
The following SQL statement selects all products with a price between 10 and 20:
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;

BETWEEN Operator with Date value example:
The following SQL statement selects all orders with and OrderDate BETWEEN '04-July-2016' and '09-July-2016':
SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/04/2016# AND #07/09/2016#;

Create Table statement: In a relational database, data is stored in tables. Given that there is no way for the database vendor to know ahead of time what our data storage needs are, we will for sure need to create tables that fit our needs in the database. Therefore, the **CREATE TABLE** statement is one of the most fundamental components of SQL.

Syntax:
CREATE TABLE "table_name"
("column1" "data type for column1" [column1 constraint(s)],
"column2" "data type for column2" [column2 constraint(s)],
…
[table constraint(s)]);

Six types of constraints can be placed when creating a table:

- NOT NULL Constraint: Ensures that a column cannot have NULL value.
- DEFAULT Constraint: Provides a default value for a column when none is specified.
- UNIQUE Constraint: Ensures that all values in a column are different.
- CHECK Constraint: Makes sure that all values in a column satisfy certain criteria.
- Primary Key Constraint: Used to uniquely identify a row in the table.
- Foreign Key Constraint: Used to ensure referential integrity for the data.

In the CREATE TABLE it is necessary to specify the data type for each column when we create a tuple. We can define the following data types:

Numeric: This type of data stores numerical values. Data types that fall in this category include integer, float, real, numeric or decimal. Common functions that operate on this type of data include COUNT, SUM, MAX, MIN and AVG.

Character String: This type of data stores character values. The two common types are CHAR(n) and VARCHAR(n). The CHAR(n) data type holds n characters, padded with spaces at the end if needed. VARCHAR stands for varying char, meaning that the length of the field can vary. For example, a VARCHAR(10) data type holds up to 10 characters. But if data is only 8 character long, then it will only store 8 characters.

Binary: This type of data allows us to store binary objects in a database table. Data types that fall in this category include Blob, Binary, and Raw. Note that a field of binary data type cannot be used as keys, and one cannot build a table index using a binary column.

Date/Datetime: This type of data allows us to store date or datetime in a database table. Different databases have very different implementations of the date/datetime data type.

To create a customer table with the fields First_Name, Last_Name, Address, City, Country and Birth_Date we would type in:

CREATE TABLE Customer
(
First_Name char(50),
Last_Name char(50),
Address char(50),

City char(25),
Birth_Date datetime
);

SQL NOT NULL Constraint:
The NOT NULL constraint enforces a column to NOT accept NULL values.
The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL enforces the "P_ID" column and the "LastName" column to not accept NULL values:
Example:
CREATE TABLE PersonsNOTNull
(
P_ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varcher(255),
Address varchar(255),
City varchar(255)
)

SQL UNIQUE Constraint:
The UNIQUE constraint uniquely identifies each record in a database table. The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.
For Example:
CREATE TABLE Persons
(
P_ID int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)

The PRIMARY KEY Constraint
The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain UNIQUE values. A primary key column cannot contain NULL values.
Most tables should have a primary key, and each table can have only ONE Primary key.

For Example:
CREATE TABLE Persons
(
P_ID int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,

FirstName varchar(255),
Address varchar(255),
City varchar(255),
);

FOREIGN KEY Constraint:
A FOREIGN KEY in one table points to a PRIMARY KEY in another table. The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.
For Example:
CREATE TABLE Orders
(
O_ID int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_ID int FOREIGN KEY REFERENCES Persons(P_ID)
);

SQL CHECK Constraint
The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column. If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.
For Example:
CREATE TABLE Persons
(
P_ID int NOT NULL CHECK (P_ID > 0),
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);

SQL DEFAULT Constraint
The DEFAULT constraint is used to insert a default value into a column.
The default value will be added to all new records, if no other value is specified.
For Example:
The following SQL creates a DEFAULT constraint on the "City" column when the "Persons" table is created:
CREATE TABLE Persons
(
P_ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255) DEFAULT 'Bharatpur'

);

The DROP TABLE statement:
The DROP TABLE statement is used to delete a table.
Syntax: DROP TABLE table_name

The DROP DATABASE Statement
The DROP DATABASE statement is used to delete a database.
Syntax: DROP DATABASE database_name

The ALTER TABLE Statement:
The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
Syntax:
To add a column in a table, use the following syntax:
ALTER TABLE table_name
ADD column_name datatype

To delete a column in a table, use the following syntax:
ALTER TABLE table_name
DROP COLUMN column_name

For Example:
If we want to add a column named "DateOfBirth" in the "Persons" table, we use the following SQL statement:
ALTER TABLE Persons
ADD DateOfBirth date

Change Data Type Example:
If we want to change the data type of the column named "DateOfBirth" in the "Persons" table.
We use the following SQL statement:
ALTER TABLE Persons
ALTER COLUMN DateOfBirth year

DROP COLUMN Example:
If we want to delete the column named "DateOfBirth" in the "Persons" table, we use the following SQL statement:
ALTER TABLE Persons
DROP COLUMN DateOfBirth

CREATE DATABASE Statement:
The CREATE DATABASE statement is used to create a database.
Syntax:
CREATE DATABASE dbname;
For Example:
The following SQL statement creates a database called "my_db":

CREATE DATABASE my_db;

SQL Date Functions:

The following table lists the most important built-in date functions in SQL server:

Function	Description
GETDATE()	Returns the current date and time
DATEPART()	Returns a single part of a date/time
DATEADD()	Adds or subtracts a specified time interval from a date
DATEDIFF()	Returns the time between two dates
CONVERT()	Displays date/time data in different formats

SQL NULL Values:

NULL values represent missing unknown data. By default, a table column can hold NULL values. If a column in a table is optional, we can insert a new record or update an existing record without adding a value to this column. This means that the field will be saved with a NULL value.

For Example:

SELECT LastName, FirstName, Address FROM Persons
WHERE Address IS NULL

SQL IS NOT NULL:

We will have to use the IS NOT NULL Operator as:

SELECT LastName, FirstName, Address FROM Persons
WHERE Address IS NOT NULL

SQL Functions:

SQL has many built-in functions for performing calculations on data.

SQL Aggregate Functions:

SQL aggregate functions return a single value, calculated from values in a column.

Some useful aggregate functions are:

- AVG() – Returns the average value
- COUNT() – Returns the number of rows
- FIRST() – Returns the first value
- LAST() – Returns the last value
- MAX() – Returns the largest value
- MIN() –Returns the smallest value
- SUM() – Returns the sum

SQL Scalar functions:

SQL scalar functions return a single value, based on the input value.

Useful scalar functions are:

- UCASE() – converts a field to upper case.
- LCASE() – converts a field to lower case.
- MID() – Extract characters from a text field.
- LEN() – Returns the length of a text field.
- ROUND() – Rounds a numeric field to the number of decimals specified
- NOW() – Returns the current system date and time

- FORMAT() – Formats how a field is to be displayed.

AVG()Function:
The AVG() function returns the average value of a numeric column.
Syntax: SELECT AVG(column_name) FROM table_name
For Example: SELECT AVG(price) AS PriceAverage FROM Proudcts;
The above SQL statement gets the average value of the "Price" column from the "Products" table.

COUNT() Function:
The COUNT() function returns the number of rows that matches a specified criteria.
Syntax: SELECT COUNT(column_name) FROM table_name;

FIRST() Function:
The FIRST() function returns the first value of the selected column.
Syntax: SELECT FIRST(column_name) FROM table_name;

LAST() Function:
The LAST() function returns the last value of the selected column.
Syntax: SELECT LAST(column_name) FROM table_name;

MAX() Function:
The MAX() function returns the largest value of the selected column.
Syntax: SELECT MAX(column_name) FROM table_name;
For Example:
SELECT MAX(Price) AS HighestPrice FROM Products;

MIN() Function:
The MIN() function returns the smallest value of the selected column.
Syntax: SELECT MIN(column_name) FROM table_name;
For Example:
SELECT MIN(Price) AS SmallestOrderPrice FROM Products;

SUM() Function:
The SUM() function returns the total sum of a numeric column.
Syntax: SELECT SUM(column_name) FROM table_name;
For Example:
SELECT SUM(Quantity) AS TotalItemOrdered FROM OrderDetails;

GROUP BY Statement:
The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.
Syntax:
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;

For Example:
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS
NumberOfOrders FROM Orders

LEFT JOIN Shippers
ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;

HAVING Clause:
The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.
Syntax:
SELECT column_name, aggregate_function (column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;

Example:
SELECT Employees.LastName, COUNT(Orders.OrderID) AS
NumberOfOrders FROM (Orders
INNER JOIN Employees
ON Orders.EmployeeID = Employee.EmployeeID)
GROUP BY LastName
HAVING COUNT (Orders.OrderID) > 10;

UCASE( ) Function:
The UCASE() function converts the value of a field to uppercase.
Syntax:
SELECT UCASE(column_name) FROM table_name;

Example:
SELECT UCASE(CustomerName) AS Customer, City
FROM Customers;

LCASE() Function:
The LCASE() function converts the value of a field to lowercase.
Syntax:
SELECT LCASE (column_name) FROM table_name;

For Example:
SELECT LCASE(CustomerName) AS Customer, City
FROM Customers;

MID() Function:
The MID() function is used to extract characters from a text field.
Syntax: SELECT MID(column_name, start, length) AS some_name FROM table_name;

For Example:
SELECT MID(City, 1, 4) AS ShortCity
FROM Customers;

LEN() Function:
The LEN() function returns the length of the value in a text field.
Syntax: SELECT LEN(column_name) FROM table_name;

For Example:
SELECT CustomerName, LEN(Address) as LengthOfAddress
FROM Customers;

ROUND() Function:
The ROUND() function is used to round a numeric field to the number of decimals specified.
Syntax:
SELECT ROUND(column_name, decimals) FROM table_name;
For Example:
SELECT ProductName, ROUND(Price, 0) AS RoundedPrice
FROM Products;

NOW() Function:
The NOW() function returns the current system date and time.
Syntax:
SELECT NOW() FROM table_name;
For example:
SELECT ProductName, Price, NOW() AS PerDate
FROM Products;

4.4 Joined relations, sub queries

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.
The most common type of join is: SQL INNER JOIN (simple join). AN SQL INNER JOIN returns all rows from multiple tables where the join condition is met.

For Example:
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID = Customers.CustomerID;

Different SQL JOINs:

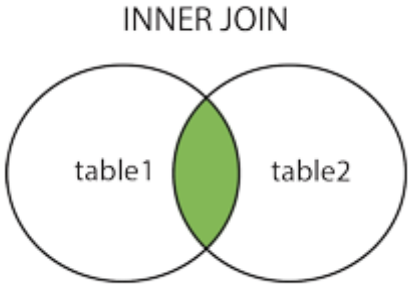
- INNER JOIN: Returns all rows when there is at least one match in BOTH tables.
- LEFT JOIN: Return all rows from the left table, and the matched rows from the right table.
- RIGHT JOIN: Return all rows from the right table, and the matched rows from the left table.
- FULL JOIN: Return all rows when there is a match in ONE of the tables.

INNER JOIN:

The INNER JOIN keyword selects all row from both tables as long as there is a match between the columns in both tables.

Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
                OR
SELECT column_name(s)
FROM table1
JOIN table2
ON table1.column_name = table2.column_name;
```



For Example:

The following SQL statement will return all customers with orders:

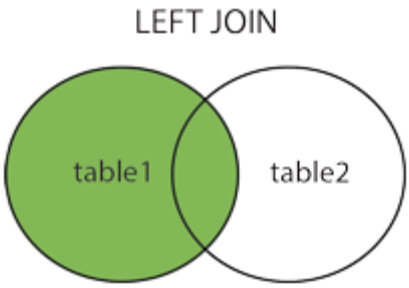
```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID =Orders.CustomerID
ORDER BY Customers.CustomerName;
```

SQL LEFT JOIN keyword:

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result in NULL in the right side when there is no match.

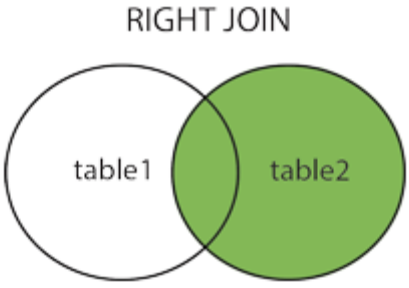
Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
                OR
SELECT column_name(s)
FROM table1
LEFT OUTER JOIN table2
ON table1.column_name = table2.column_name;
```



For Example:
The following SQL statement will return all customers, and any orders they might have:
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;

SQL RIGHT JOIN Keyword:
The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.
Syntax:
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
OR
SELECT column_name(s)
FROM table1
RIGHT OUTER JOIN table2
ON table1.column_name = table2.column_name;



For Example:
SELECT Orders.OrderID, Employees.FirstName
FROM Orders
RIGHT JOIN Employees
ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;

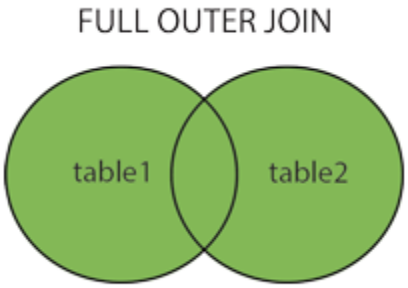
FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all rows from the lest table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

Syntax:

SELECT column_name(s)
FROM table1
FULL OUTER JOINtable2
ON table.column_name = table2.column_name;



For Example:

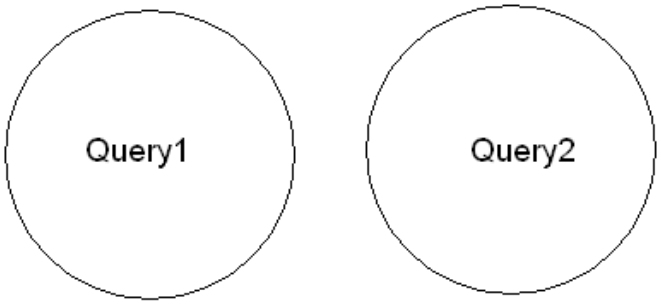
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;

4.5 Set operations

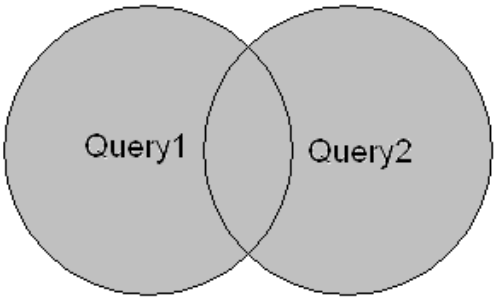
UNION [DISTINCT] and UNION ALL:

This usually are most widely used set operators. Quite many times one cannot get all the result from one Select statement. Then one of the UNIONS can help.

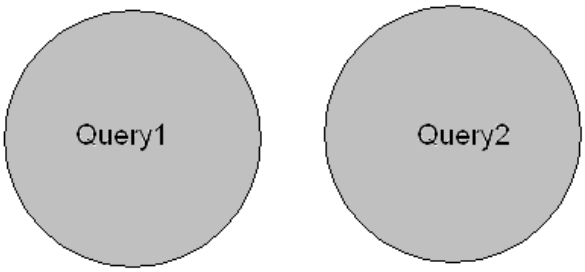
Graphically UNION can be visualised using Venn diagram. Assume we have two row sets.



Then Query1 UNION Query2 would be as follows:



Query1 UNION ALL Query2 would be as follows:

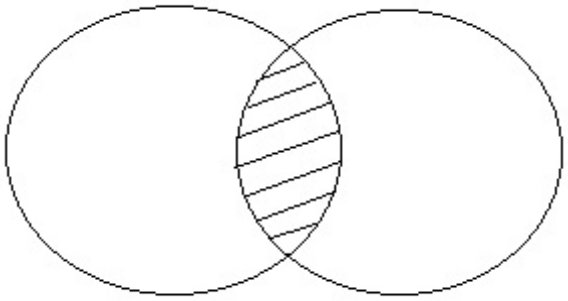


Example1:
SELECT city FROM Table1
UNION
SELECT city FROM table2;
Above example unions cities from table1 and table2.

Example2:
SELECT city FROM table2
UNION
SELECT city FROM table1;
Both the result of Example1 and Example2 are same.

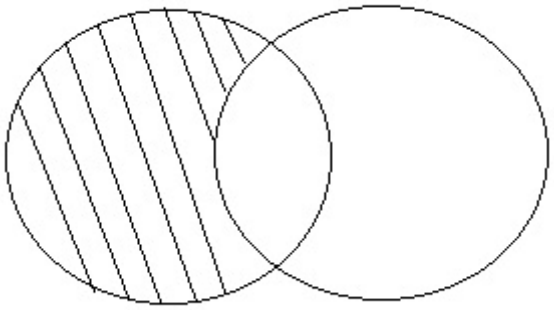
Example3:
SELECT city FROM table1
UNION ALL
SELECT city FROM table2;

Intersection:
Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statement. In case of Intersect the number of columns and datatype must be same.



For Example:
SELECT * FROM First
INTERSECT
SELECT * FROM Second

MINUS:
MINUS operation combines result of two select statements and return only those result which belongs to first set of result.



For Example:
SELECT * FROM First
MINUS
SELECT * FROM Second

4.6 Views

A view is a virtual table. A view consists of rows and columns just like a table. The difference between view and a table is that views are definitions built on top of other tables (or views), and do not hold data themselves. If data is changing in the underlying table, the same change is reflected in the view. A view can be built on top of a single table or multiple tables. It can also be built on top of another view.
Views offer the following advantages:

1. **Ease of use:**A view hides the complexity of the database tables from end users. Essentially we can think of views as a layer of abstraction on top of the database tables.
2. **Space savings:** Views takes very little space to store, since they do not store actual data.
3. **Additional data security:** Views can include only certain columns in the table so that only the non-sensitive columns are included and exposed to the end user. In addition, some databases allow views to have different security settings, thus hiding sensitive data from prying eyes.

Creating view statement:
Views can be considered as virtual tables. Generally speaking, a table has a set of definition, and it physically stores the data. A view also has a set of definition, which is build on top of table(s) or other view(s), and it does not physically store the data.
Syntax:
CREATE VIEW view_name AS

SELECT column_name(s)
FROM table_name
WHERE condition

For Example:
CREATE VIEW [Current Product List] AS
SELECT ProductID, ProductName
FROM Products
WHERE Discountinued=No

Example2:
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, UnitPrice
FROM Products
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Products)

Dropping VIEW:
We can delete a view with the DROP VIEW command.
Syntax: DROP VIEW view_name

Integrity Constraints

Integrity constraints are those constrains in database system which guard against invalid database operations or accidental damage to the database, by ensuring that authorized changes to the database. It does not allow to loss of data consistency in database, it ensures database consistency.

In fact, integrity constraints provide a way of ensuring the changes made to the database by authorized users do not result in a loss of data consistency. Example of integrity constraints in E-R model is : Key declaration (candidate key, primary key), Form of relationship (mapping cardinalities: one to one, one to many etc.).

In database management system we can enforce any arbitrary predicate as integrity constraints but it adds overhead to the database system so its cost should be evaluated, as far as possible integrity constraint should with minimal overhead.

5.1 Domain constraints

It is a set of all possible values for attribute known as its domain. Domain constraints enforce attribute should hold only particular types of attributes. A domain of possible values should be associated with every attribute. Domain constraints are the most elementary form of integrity constraint. It is tested by database system whenever a new data item is entered into database. System test values inserted in the database and test queries to ensure that the comparisons make sense.

Domain types in SQL:

SQL standard supports a variety of built in domain types including:

- **Char (n):** A fixed length character string with user specified length n.
- **Varchar(n):**A variable length string with user specified maximum length n.
- **Int:** An integer (Machine dependent).
- **Smallint:** A small integer.
- **Numeric (p,d):** A fixed point number with user specified precision. Where (.) is counted in p.
- **Real, double precision:** Floating point and double precision floating point numbers.
- **Float(n):** A floating point number with precision of at least n digits.
- **Date:** A calendar date containing a four digit year, month and day of the month.
- **Time:** The time of a day, in hours, minutes and seconds.
- **Timestamp:** A combination of date and time.

New domains can be created from existing data types. For example:

CREATE DOMAIN Dollars Numeric(12,2)

CREATE DOMAIN Pounds Numeric(12,2)

The Check clause in SQL allow domains to be restricted.
For Example:
CREATE DOMAIN salary-rate numeric(5)
constraintvalue-test check(value > = 5000)
The domain constraint ensures that the hourly0rate must greater than 5000.
The clause **constraint***value-test* is optional but useful to indicate which constraintan update violated.

Example -2
CREATE DOMAIN AccountType char(10)
constraint account-type-test
check (value in ('Checking', 'Saving'))

Example-3
CREATE DOMAIN account-number char(10)
Constraint account-number-null-test check (value not null)

5.2 Entity integrity

This rule is designed to assure that every relation has a primary key and that the data values for that primary key are all valid i.e. every primary key is not null. But in some case a particular attribute can not be assigned a data value. There are two situation where this one is likely occur: either there is no applicable data values or the applicable data values is not known when values are assigned. NULL is a value that may be assigned to an attribute when no other values applies or when the applicable value is unknown. States the following No primary key attribute may be NULL. A NULL is a value that may be assigned to an attribute when no other value applies or when the applicable value is unknown.

5.3 Referential integrity

Referential integrity is a condition which ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.

For Example:
If "B1" is a branch name appearing in one of the tuples in the account relation, then there exists a tuple in the branch relation where "B1" exist for branch name attribute.

Example:
Consider two relation department and employee as follows:
Department(deptno#, dname)
Employee (empno#, ename, deptno)

- Deletion of particular department from department table also need to delete records on employees they belongs to that particular department or delete need not be allow if there is any employee that is associated to that particular department that we are going to delete.
- Any update made in depno in department table deptno in employee must be updated automatically.
- Any update made in deptno in department table deptno in employee must be updated automatically.

- This implies primary key acts as a referential integrity constraint in a relation.

Formal Definition:

- Let $r_1(R_1)$ and $r_2(R_2)$ be relations with primary keys K_1 and K_2 respectively. The subsetα of R_2 is a foreign key referencing K_1 in relation r_1, if for every t_2 in r_2 there must be a tuple t_1 in r_1 such that $t_1[K_1] = t_2[\alpha]$.
- Referential integrity constraint also called subset dependency since its can be written as: $\Pi_{\alpha}(r_2) \subseteq K_1(r_1)$

Referential integrity in E-R Model:

- Consider relationship set *R* between entity sets E_1 and E_2. The relational schema for *R* includes the primary keys K_1 of E_1 and K_2 of E_2. Then K_1 and K_2 form foreign keys on the relational schemas for E_1 and E_2 respectively that leads referential integrity constraint.
- Weak entity sets are also a source of referential integrity constraints. A weak entity set must include the primary key attributes of the entity set on which it depends.

Referential integrity in SQL:

Using the SQL create table statement we can enforce:

- Primary key
- Unique
- Foreign key

For example:

```
CREATE TABLE Customer
(
customer-name char(20),
customer-street char(30),
customer-city char(30),
primary key (customer-name)
);

CREATE TABLE branch
(
branch-name char(15),
branch-city char(30),
assets integer,
primary key(branch-name)
);

CREATE TABLE account
(
account-number char(10),
branch-name char(15),
balance integer,
primary key(account-number),
foreign key(branch-name) references branch
)
```

```
CREATE TABLE depositor
(
customer-name char(20),
account-number char(10),
primary key (customer-name, account-number),
foreign key (account-number) references account,
foreign key(customer-name) references customer
)
```

Relational Database Design

6.1 Pitfalls of Relational Model: Redundancy and Anomalies

The main goal of relational database is to create/find good collection of relational schemas. Such that database should allow us to store data/information without unnecessary redundancy and should also allow to retrieve information easily. That is the goal of relational database design should concentrate:

- To avoid redundant data from database.
- To ensure that relationships among attributes are represented.
- To facilitate the checking of updates for the violation of database integrity constraints.

A bad relational database design may lead to:

- Repetition of information.
    - That is, it leads data redundancy in database, so obviously it requires much space.
- Inability to represent certain information.

For Example:
Consider the relational schema:
Branch_loan = (branch_name, branch_city, assets, customer_name, loan_no, amount)

branch_name	branch city	assets	customer name	loan no	amount
kathmandu	baneshwor	25000	rohan	L - 15	3000
Lalitpur	patan	19000	mohan	L - 17	5000
Kathmandu	baneshwor	25000	raju	L - 19	10000
Pokhara	Prithibi nagar	17000	manoj	L - 10	7000
Lalitpur	patan	19000	swikar	L - 30	9000

Redundancy:

- Data for branch_name, branch_city, assets are repeated for each loan that provides by bank.
- Storing information several times leads waste of storage space/time.
- Data redundancy leads problem in Insertion, deletion and update.

Insertion Problem:

- We cannot store information about a branch without loan information, we can use null values for loan information, but they are difficult to handle.

Deletion Problem:

- In this example, if we delete the information of Manoj, (i.e. DELETE FROM branch_loan WHERE customer_name = 'Manoj';), we cannot obtain the information of Pokhara branch (i.e. we don't know branch city of Pokhara, total asset of Pokhara branch etc.

Update problem:

- Since data are duplicated so multiple copies of same fact need to update while updating one. It increases the possibility of data inconsistency. When we made update in one copy there is possibility that only some of the multiple copies are update but not all, which lead data/database in inconsistent state.

6.2 Functional dependency

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 A_1, A_2, …, A_n, then those two tuples must have to have same values for attributes B_1, B_2, …. , B_n.

Functional dependency is represented by an arrow sign (→) that is, X → Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

Armstrong's Axioms:

If F is a set of functional dependencies then the closure of F, denoted as F*, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules that, when applied repeatedly, generates a closure of functional dependencies.

- **Reflexive rule:** If alpha is a set of attributes and beta is subset of alpha, then alpha holds beta.
- **Transitivity rule:** Same as transitive rule in algebra, if a→b holds and b→c holds, then a→c also holds. a→bis called as a functionally that determines b.

Trivial Functional Dependency

- **Trivial:** If a functional dependency (FD) X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.
- **Non-trivial:** If an FD X → Y holds, where Y is not a subset of X, then it is called anon-trivial FD.
- **Completely non-trivial:** If an FD X → Y holds, where X intersect Y = ϕ, it is said to be a completely non-trivial FD.

6.3 Normalization, its need and objectives

Database normalization is the process of organizing data into tables in such a way that the results of using the database are always unambiguous and as intended. Such normalization is intrinsic to relational database theory. It may have the effect of duplicating data within the database and often results in the creation of additional tables.

The concept of database normalization is generally traced back to E.F. Codd, an IBM researcher who, in 1970, published a paper describing the relational database model. What Codd describes as "a normal form for database relations" was an essential element of the

relational technique. While data normalization rules tend to increase the duplication of data, it does not introduce data redundancy, which is unnecessary duplication. Database normalization is typically a refinement process after the initial exercise of identifying the data objects that should be in the relational database, identifying their relationships and defining the tables required and the columns within each table.

Objectives of Normalization:

A basic objective of the first normal form defined by Codd in 1970 was to permit data to be queried and manipulated using a "universal data sub-language" grounded in first-order-logic. The objectives of normalization beyond 1NF (First Normal Form) were stated as follows by Codd:

1. To free the collection of relations from undesirable insertion, update and deletion dependencies;
2. To reduce the need for restructuring the collection of relations, as new types of data are introduced, and thus increase the life span of application programs;
3. To make the relational model more informative to users;
4. To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.

The following example give details of each of these objectives:

Free the database of modification anomalies:

Employees' Skills

Employee ID	Employee Address	Skill
426	Chhoprak-7, Gorkha	Typing
426	Chhoprak-7, Gorkha	Shorthand
519	Bharatpur-10, Chitwan	Public Speaking
519	Bharatpur-12, Chitwan	Carpentry

An **update anomaly.** Employee 519 is shown as having different addresses on different records.

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Mr. Sushil Bhattarai	10 – Feb – 2012	MGT – 111
407	Mr. Suman Poudel	19 – Apr – 2010	CMP – 101
407	Mr. Suman Poudel	19 – Apr – 2010	CMP – 201
424	Mr. Ujjwal Marahatta	29 – Mar – 2016	?

An **insertion anomaly.** Until the new faculty member, Mr. Ujjwal Marahatta, is assigned to teach at least course, his details cannot be recorded.

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Mr. Sushil Bhattarai	10 – Feb – 2012	MGT – 111
407	Mr. Suman Poudel	19 – Apr – 2010	CMP – 101
407	Mr. Suman Poudel	19 – Apr – 2010	CMP – 201

A **deletion anomaly.** All information about Mr. Sushil Bhattarai is lost if he temporarily ceases to be assigned to any courses.

6.4 1NF, 2NF, 3NF, BCNF and 4NF

First Normal Form (1NF):

As per First Normal Form, no two rows of data must contain repeating group of information i.e. each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row. Each table should be organized into rows, and each row should have a primary key that distinguishes it as unique.

The **Primary Key** is usually a single column, but sometimes more than one column can be combined to create a single primary key. For example consider a table which is not in First normal form:

Student	Age	Subject
Adam	15	Biology, Maths
Alex	14	Maths
Stuart	17	Maths

In First Normal Form, any row must not have a column in which more than one value is saved, like separated with commas. Rather than that, we must separate such data into multiple rows.

Student table following 1NF will be:

Student	Age	Subject
Adam	15	Biology
Adam	15	Maths
Alex	14	Maths
Stuart	17	Maths

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Second Normal Form (2NF):

As per Second Normal Form there must not be any partial dependency of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails **Second normal form.**

In example of First Normal Form there are two rows for Adam, to include multiple subjects that he has opted for. While this is searchable, and follows First Normal Form, it is an inefficient use of space. Also in the above table in First Normal Form, while the candidate key is {**Student, Subject}, Age** of Student only depends on Student column, which is incorrect as per Second Normal Form. To achieve second normal form, it would be helpful to split out the subjects into an independent table, and match them up using the student names as foreign keys.

New Student Table following 2NF will be:

Student	Age
Adam	15
Alex	14
Stuart	17

In Student table the candidate key will be **Student** column, because all other column i.e. **Age** is dependent on it.

New subject table introduced for 2NF will be:

Student	Subject
Adam	Biology
Adam	Maths
Alex	Maths
Stuart	Maths

In Subject Table the candidate key will be {**Student, Subject}** column. Now, both the above tables qualifies for Second Normal Form and will never suffer from Update Anomalies. Although there are a few complex cases in which table in Second Normal Form suffers Update Anomalies, and to handle those scenarios Third Normal Form is there.

Third Normal Form (3NF):

Third Normal Form applies that every non-prime attribute of table must be dependent on primary key, or we can say that, there should not be the case that a non-prime attribute is determined by another non-prime attribute. So this *transitive functional dependency* should be removed from the table and also the table must be in **Second Normal Form.** For example, consider a table with following fields:

Student_Detail Table:

Student_ID	Student_name	DOB	Street	City	State	Zip

In this table Student_ID is primary key, but street, city and state depends upon Zip. The dependency between zip and other fields is called **transitive dependency**. Hence, to apply **3NF**, we need to move the street, city and state to new table, with **zip** as primary key.

New Student_Detail Table:

Student_ID	Student_name	DOB	Zip

Address Table:

Zip	Street	City	State

The advantage of removing transitive dependency is,

- Amount of data duplication is reduced.
- Data integrity achieved.

Boyce and Codd Normal Form (BCNF):

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- and for each functional dependency (X → Y), X should be a super key.

Consider the following relationship : **R (A,B,C,D)**

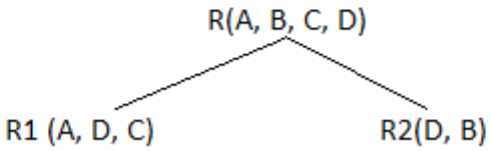
and following dependencies :

A -> BCD
BC -> AD
D -> B

Above relationship is already in 3rd NF. Keys are **A** and **BC**.

Hence, in the functional dependency, **A -> BCD**, A is the super key.
in second relation, **BC -> AD**, BC is also a key.
but in, **D -> B**, D is not a key.

Hence we can break our relationship R into two relationships **R1** and **R2**.



Breaking, table into two tables, one with A, D and C while the other with D and B.

Question: Normalize the following table up to 3NF.

RN	Name	Address	Phone	DOB	Tid	Dept	Tname	Tphone
1	Ram	Chitwan	056540123	2051/11/12	12	Management	Shankar	9845111111 056540111
2	Hari	Kathmandu	01444444	2051/3/13	11	Science	Manish	984111111 01411111
3	Sita	Bhaktapur	01611111	2052/3/5	10	Humanities	Smita	984111811
4	Alisha	Lalitput	01511111	2051/4/4	10	Humanities	Smita	984111811

Solution:

First Normal Form (1NF):

In the above table, the field Tphone holds multiple values in one cell. In this case, the primary key is RN. With the design like this table, we can have insert, update, delete and select anomalies. This table has more than one data in one cell so, it should be corrected as:

RN	Name	Address	Phone	DOB	Tid	Dept	Tname	Tphone
1	Ram	Chitwan	056540123	2051/11/12	12	Management	Shankar	9845111111
1	Ram	Chitwan	056540123	2051/11/12	12	Management	Shankar	056540111
2	Hari	Kathmandu	01444444	2051/3/13	11	Science	Manish	984111111
2	Hari	Kathmandu	01444444	2051/3/13	11	Science	Manish	01411111
3	Sita	Bhaktapur	01611111	2052/3/5	10	Humanities	Smita	984111811
4	Alisha	Lalitput	01511111	2051/4/4	10	Humanities	Smita	984111811

This table is in 1NF because every field of the table is atomic, i.e. small piece of data consists in the table field. Further, there are no duplication of rows or columns.

Second Normal Form (2NF):

In the above table, the RN and Tid with Tphone field can be used as primary keys. Similarly, the corresponding key field depends on its primary key like Name, Address, Phone

and DOB depends on RN field and Dept, Tname and Tphone depends on Tid. To change the table in 2NF, we need to decompose the table in multiple tables as:

Student's table:

RN	Name	Address	Phone	DOB
1	Ram	Chitwan	056540123	2051/11/12
2	Hari	Kathmandu	01444444	2051/3/13
3	Sita	Bhaktapur	01611111	2052/3/5
4	Alisha	Lalitput	01511111	2051/4/4

Teacher's table

Tid	Tname	Tphone
12	Shankar	9845111111
12	Shankar	056540111
11	Manish	984111111
11	Manish	01411111
10	Smita	984111811

here, Tid and Tphone jointly work as primary key.

Relation between teacher's and student's table:

RN	Tid
1	12
2	11
3	10
4	10

Third Normal Form (3NF):

The above tables are in 2NF but we still have Insert and Delete anomalies. To reduce these anomalies, these tables should be changed into 3NF. In the student table (RN, Name, Address, Phone, DOB), Name, Address and Phone are fully depend on it's primary key 'RN'. But the date of birth DOB is depends on student name 'Name' which is not a primary key. As applying the rules, the student tables will be decomposed into the following tables but other tables remain same because other supports the rule of 3NF.

Student1 table:

RN	Name	Address	Phone
1	Ram	Chitwan	056540123
2	Hari	Kathmandu	01444444
3	Sita	Bhaktapur	01611111
4	Alisha	Lalitput	01511111

Student2 table:

RN	DOB
1	2051/11/12
2	2051/3/13
3	2052/3/5
4	2051/4/4

Now, this scheme is free from insert, update, delete and select anomalies. Finally the given table is normalized.

Database Security

Database security refers to protection from malicious access. Absolute protection of the database from malicious abuse is not possible, but the cost to the perpetrator can be made high enough to deter most if not all attempts to access the database without proper authority. The data stored in the database need protection from unauthorized access and malicious destruction or alteration, in addition to the protection against accidental introduction of inconsistency that integrity constraints provide.

Needs of security:
To protect the database, we must take security measure at several levels:

Database system; some database system user may be authorized to access only a limited portion of the database. Other user may be allowed to issue queries, but mat be forbidden to modify the data. It is the responsibility of the database system to ensure that these authorization restrictions are not violated.

Operating system; No matter how secure the database system is, weakness in operating system security may as a means of unauthorized access to the database.

Network; since almost all database system allows remote access through terminals or network, software level security within the network software is as important as physical security, both on the internet and in private networks**.**

Physical; Sites with computer system must be physically secured against armed or surreptitious entry by intrudes.

Human; Users must be authorized carefully to reduce the chance of any user giving access to an intruder in exchange for a bribe or other favors. Security is need in database for the following reasons.

1. ***Protection from improper access***- only authorized users should be granted access to objects of DBMS. This control should be applied on smaller objects (record, attribute, value).
2. ***Protection from inference*** - inference of confidential information from available data should be avoided. This regards mainly statistical DBMSs.
3. ***Database integrity*** - partially is ensured with system controls of DBMS (atomic transactions) and various back-up and recovery procedures and partially with security procedures.
4. ***Operational data integrity*** - logical consistence of data during concurrent transactions (concurrency manager), serializability and isolation of transactions (locking techniques).

5. ***Semantic data integrity*** - ensuring that attribute values are in allowed ranges. This is ensured with integrity constraints.
6. ***Accountability and auditing*** - there should be possibility to log all data accesses.
7. ***User authentication*** - there should be unambiguous identification of each DBMS user. This is basis for all authorization mechanisms.
8. ***Management and protection of sensitive data*** - access should be granted only to narrow round of users.
9. ***Multilevel security*** - data may be classified according to their sensitivity. Access granting should then depend on that classification.
10. ***Confinement (subject isolation)*** - there is necessity to isolate subjects to avoid uncontrolled data flow between programs (memory channels, covert channels).

7.1 Access Control: Discretionary and Mandatory

Access control is responsible for control of rules determined by security policies for all direct accesses to the system. Traditional control systems work with notions *subject*, *object* and *operation*.
For better image look at the figure of secure DBMS.

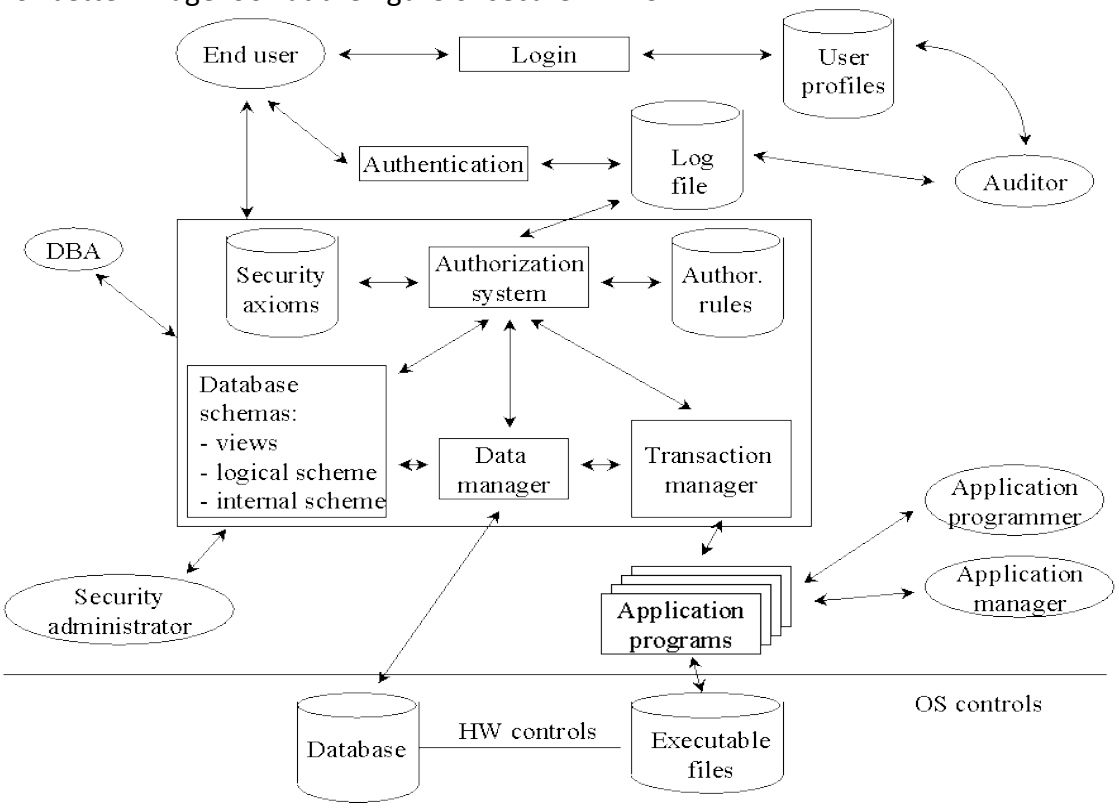


Fig: Schema of secure database management system

The term access control is something of an ambiguous term. To some it could be interpreted as controlling the access to a system from an external source (for example controlling the login process via which users gain access to a server or desktop system). In fact, such access control is actually referred to as Authentication or Identity Verification and is not what is mean by Access Control in this context.

The term Access Control actually refers to the control over access to system resources after a user's account credentials and identity have been authenticated and access to the system resources after a user's account credentials and identity have been authenticated and

access to the system granted. For example, a particular user, or group of users, might only be permitted access to certain files after logging into a system, while simultaneously being denied access to all other resources.

In addition to handling such concern any organization need to address five areas regard to achieving high data quality.

1. **Security policy and disaster recovery:** The organization must establish security policies and make detailed disaster recovery plans. It should determine how the organization is going to maintain a secure system. Disaster recovery will determine how the organization will continue to function should an emergency situation such as flood or fire.
2. **Personnel control:** Monitoring to ensure that the personnel are following established practices, taking regular, vacation, working with other employees, and so forth should be followed. Employee should be trained in those aspects of security and quality that are relevant to their jobs and be encouraged to be aware of and follow standard security and data quality measures.
3. **Physical access control:** Limited access to particular areas within a building is usually a part of controlling physical areas. Sensitive equipment, including hardware and peripherals such as printers can be controlled by placement in the secured areas. Other equipment may be locked to a desk or cabinet, or may have an alarm attached. Backup data tapes must be kept in fireproof data safes or at a safe location.
4. **Maintenance control:** An area of control that helps to maintain data quality but that is often overlooked is maintenance control. Organization should review external maintenance agreements for all hardware and software they are using to ensure that appropriate responses are agreed to for maintaining data quality.
5. **Data protection and privacy:** It refers about the right of individuals to not have personal information collected and misestimated casually have intensified as more of the population has become familiar with computers and as communications among computers have proliferated.

Mandatory Access Control (MAC):

Mandatory Access Control (MAC) is the strictest of all levels of control. The design of MAC was defined, and is primarily used by the government.

MAC takes a hierarchical approach to controlling access to resources. Under a MAC enforced environment access to all resource objects (such as data files) is controlled by settings defined by the system administrator. As such, all access to resource objects is strictly controlled by the operating system based on system administrator configured settings. It is not possible under MAC enforcement for users to change the access control of a resource.

Mandatory Access Control begins with security labels assigned to all resource objects on the system. These security labels contain two pieces of information - a classification (top secret, confidential etc) and a category (which is essentially an indication of the management level, department or project to which the object is available).

Similarly, each user account on the system also has classification and category properties from the same set of properties applied to the resource objects. When a user attempts to access a resource under Mandatory Access Control the operating system checks the user's classification and categories and compares them to the properties of the object's security label. If the user's credentials match the MAC security label properties of the object access is allowed. It is important to note that both the classification and categories must match. A user with top secret classification, for example, cannot access a resource if they are not also a member of one of the required categories for that object.

Mandatory Access Control is by far the most secure access control environment but does not come without a price. Firstly, MAC requires a considerable amount of planning before it can be effectively implemented. Once implemented it also imposes a high system management overhead due to the need to constantly update object and account labels to accommodate new data, new users and changes in the categorization and classification of existing users.

Discretionary Access Control:
Unlike Mandatory Access Control (MAC) where access to system resources is controlled by the operating system (under the control of a system administrator), Discretionary Access Control (DAC) allows each user to control access to their own data. DAC is typically the default access control mechanism for most desktop operating systems.

Instead of a security label in the case of MAC, each resource object on a DAC based system has an Access Control List (ACL) associated with it. An ACL contains a list of users and groups to which the user has permitted access together with the level of access for each user or group. For example, User A may provide read-only access on one of her files to User B, read and write access on the same file to User C and full control to any user belonging toGroup 1.
It is important to note that under DAC a user can only set access permissions for resources which they already own. A hypothetical User Acannot, therefore, change the access control for a file that is owned by User B. User A can, however, set access permissions on a file that she owns. Under some operating systems it is also possible for the system or network administrator to dictate which permissions users are allowed to set in the ACLs of their resources.

Discretionary Access Control provides a much more flexible environment than Mandatory Access Control but also increases the risk that data will be made accessible to users that should not necessarily be given access.

Role Based Access Control:
Role Based Access Control (RBAC), also known as Non discretionary Access Control, takes more of a real world approach to structuring access control. Access under RBAC is based on a user's job function within the organization to which the computer system belongs.

Essentially, RBAC assigns permissions to particular roles in an organization. Users are then assigned to that particular role. For example, an accountant in a company will be assigned to the Accountant role, gaining access to all the resources permitted for all accountants on the system. Similarly, a software engineer might be assigned to the developer role.

Roles differ from groups in that while users may belong to multiple groups, a user under RBAC may only be assigned a single role in an organization. Additionally, there is no way to provide individual users additional permissions over and above those available for their role. The accountant described above gets the same permissions as all other accountants, nothing more and nothing less.

Rule Based Access Control:
Rule Based Access Control (RBAC) introduces acronym ambiguity by using the same four letter abbreviation (RBAC) as Role Based Access Control.

Under Rules Based Access Control, access is allowed or denied to resource objects based on a set of rules defined by a system administrator. As with Discretionary Access Control, access properties are stored in Access Control Lists (ACL) associated with each resource object. When a particular account or group attempts to access a resource, the operating system checks the rules contained in the ACL for that object.

Examples of Rules Based Access Control include situations such as permitting access for an account or group to a network connection at certain hours of the day or days of the week.
As with MAC, access control cannot be changed by users. All access permissions are controlled solely by the system administrator.

7.2 Authorization and Authentication

Authentication:
Authentication means verifying the identity of someone (a user, device, or other entity) who wants to use data, resources, or applications. Validating that identity establishes a trust relationship for further interactions. Authentication also enables accountability by making it possible to link access and actions to specific identities. After authentication, authorization processes can allow or limit the levels of access and action permitted to that entity.

For simplicity, the same authentication method is generally used for all database users. To validate the identity of database users and prevent unauthorized use of a database user name, we can authenticate using any combination of the methods as follows:

- **Authentication by Operating System:** Once authenticated by the operating system, users can connect to database more conveniently, without specifying a user name or password. With control over user authentication centralized in the operating system, it need not store or manage user passwords, though it still maintains user names in the database.
- **Authentication by the Network:**
    - Third Party based authentication technologies: If network authentication services are available to us then database can accept authentication from the network service.
    - Public key infrastructure based authentication: Authentication system based on public key cryptography issue digital certificates to user clients, which use them to authenticate directly to servers in the enterprise without directly involving an authentication server.

  - Remote authentication: It supports remote authentication of users through remote dial in user service (RADIUS), a standard lightweight protocol used for user authentication, authorization, and accounting.
- **Authentication by the Oracle Database:** Oracle can authenticate users attempting to connect to a database by using information stored in that database. Database authentication includes the following facilities:
  - Password Encryption: To protect password confidentiality, Oracle always encrypts passwords before sending them over the network. It encrypts the passwords using a modified AES (Advanced Encryption Standard) algorithm.
  - Account locking: Oracle can lock a user's account after a specified number of consecutive failed log-in attempts. You can configure the account to unlock automatically after a specified time interval or to require database administrator intervention to be unlocked. The database administrator can also lock accounts manually, so that they must be unlocked explicitly by the database administrator.
  - Password lifetime and expiration: The database administrator can specify a lifetime for passwords, after which they expire and must be changed before account login is again permitted. A grace period can be established, during which each attempt to login to the database account receives a warning message to change the password. If it is not changed by the end of that period, then the account is locked.
- **Authentication by the secure socket layer protocol:** The secure socket layer (SSL) protocol is an application layer protocol. Users identified either externally or globally (external or global users) can authenticate to a database through SSL.
- **Authentication of Database Administrators:** Database administrators perform special operations (such as shutting down or starting up a database) that should not be performed by normal database users.

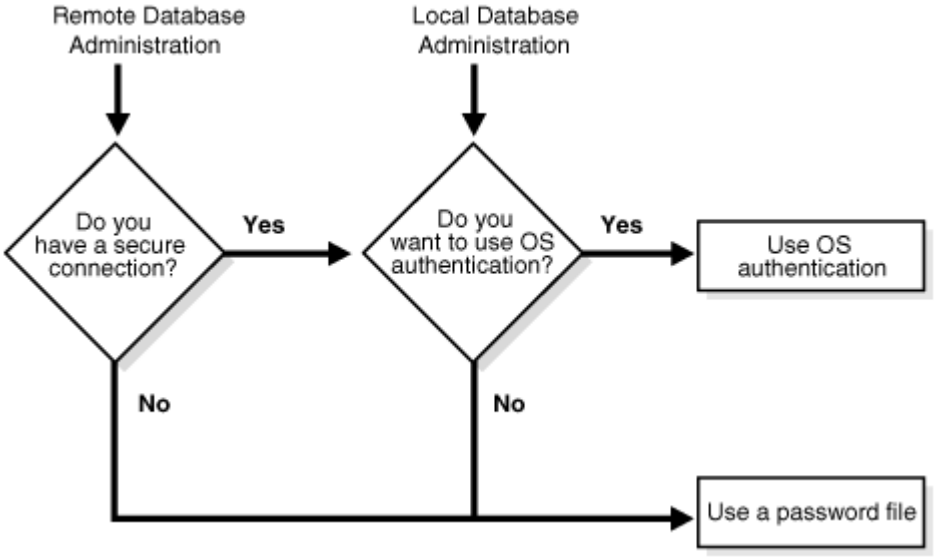


Fig: Database Administrator Authentication Methods

Authorization:

We may assign a user several forms of authorization on parts of the database. For example:

- **Read authorization:** allows reading, but not modification of data.
- **Insert authorization:** allows insertion of new data but not modification of existing data.

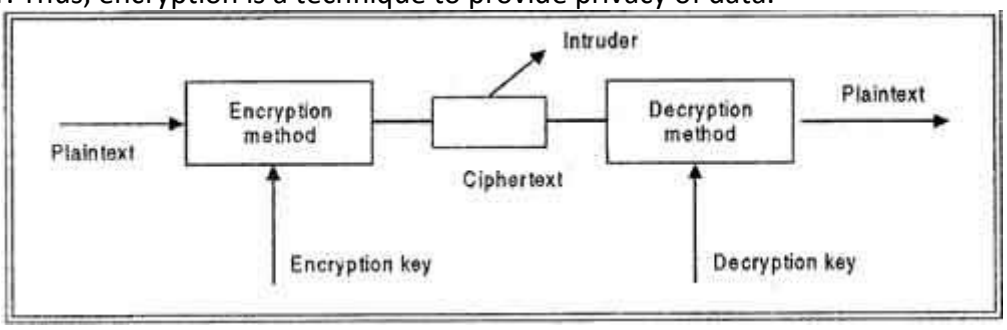
- **Update authorization:** allows modification but not deletion of data.
- **Delete authorization:**allows deletion of data. We may assign the user all, none, or a combination of these type of authorization. In addition to these forms of authorization for access to data. We may grant a user authorization to modify the database schema.
- **Index authorization:** allows the creation and deletion of indices.
- **Resources authorization:** allows the creation of new relations.
- **Alteration authorization:** allows the addition or deletion of attributes in a relation.
- **Drop authorization:** allows the deletion of relations.

The drop and delete authorization differs in that delete authorization allows deletion of tuples only. If a user deletes all tuples of relation, the relation still exists, but it is empty. If the relation is dropped, it no longer exists.
It regulates the ability to create a new relation through resources authorization. A user with resources authorization who creates a new relation is given all privileges on that relation automatically.

Index authorization may appear unnecessary, since the creation or deletion of an index does not alter data in relation. Rather, indices are a structure for performance enhancements. However, indices also consume space, and all database modification are required to updates indices if index authorization were granted to all customer, those who performed updates would be tempted to delete indices, where those who issued queries would be tempted to creates numerous indices, to allows the database administrator to regulates the use of system resources, it is necessary to treat index creation as a privilege.

7.3 Data encryption and decryption

A DBMS can use encryption to protect information in certain situations where the normal security mechanisms of the DBMS are not adequate. For example, an intruder may steal tapes containing some data or tap a communication line. By storing the transmitting data in an encrypted form, the DBMS ensures that such stolen data is not intelligible to the intruder. Thus, encryption is a technique to provide privacy of data.



In encryption, the message to be encrypted is known as plaintext. The plaintext is transformed by a function that is parameterized by a key. The output of the encryption process is known as the cipher text. Ciphertext is then transmitted over the network. The process of converting the plaintext to ciphertext is called as Encryption and process of converting the ciphertext to plaintext is called as Decryption. Encryption is performed at the transmitting end and decryption is performed at the receiving end. For encryption process we need the encryption key and for decryption process we need decryption key as shown in

figure. Without the knowledge of decryption key intruder cannot break the ciphertext to plaintext. This process is also called as Cryptography.

The basic idea behind encryption is to apply an encryption algorithm, which may' be accessible to the intruder, to the original data and a user-specified or DBA-specified encryption key, 'which is kept secret. The output of the algorithm is the encrypted version of the data. There is also a decryption algorithm, which takes the encrypted data and the decryption key as input and then returns the original data. Without the correct decryption key, the decryption algorithm produces gibberish. Encryption and decryption keys may be same or· different but there must be relation between the both which must me secret.

Techniques used for Encryption:
There are following techniques used for encryption process:

- Substitution ciphers
- Transposition ciphers

Substitution Ciphers: In a substitution cipher each letter or group of letters is replaced by another letter or group of letters to mask them For example: a is replaced with D, b with E, c with F and z with C. In this way *attack* becomes *DWWDFN.* The substitution ciphers are not much secure because intruder can easily guess the substitution characters.

Transposition Ciphers: Substitution ciphers preserve the order of the plaintext symbols but mask them-;-The transposition cipher in contrast reorders the letters but do not mask them. For this process a key is used. For example: *iliveinqadian* may be coded as*divienaniqnli.* The transposition ciphers are more secure as compared to substitution ciphers.

Algorithms for Encryption Process
There are commonly used algorithms for encryption process. These are:

• Data Encryption Standard (DES)
• Public Key Encryption

Data Encryption Standard (DES)

It uses both a substitution of characters and a rearrangement of their order on the basis of an encryption key. The main weakness of this approach is that authorized users must be told the encryption key, and the mechanism for communicating this information is vulnerable to clever intruders.

Public Key Encryption

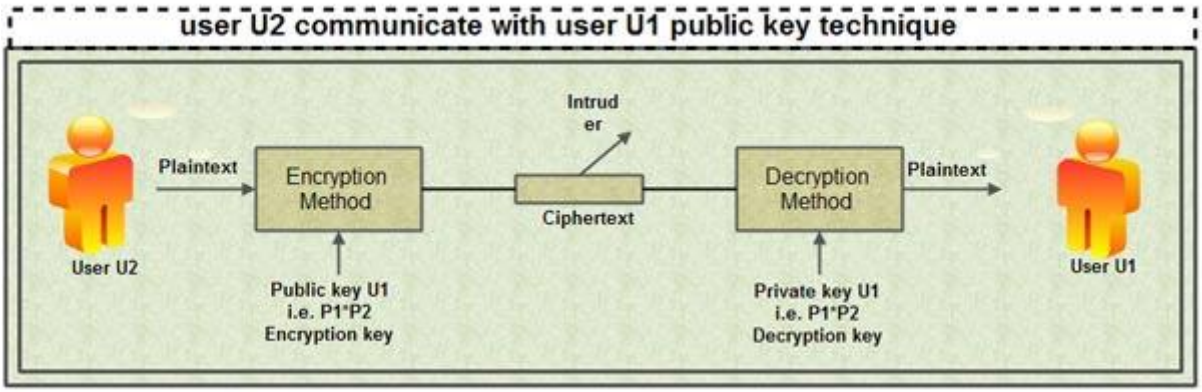
Another approach to encryption, called public-key encryption, has become increasingly popular in recent years. The encryption scheme proposed by Rivest, Shamir, and Adheman, called RSA, is a well-knownexample of public-key encryption. Each authorized user has a public encryption key, known to everyone and a private decryption key (used by the decryption algorithm), chosen by the user and known only to him or her. The encryption and decryption algorithms themselves are assumed to be publicly known.

Consider user called Suneet. Anyone can send Suneet a secret message by encrypting the message using Sunset's publicly known encryption key. Only Suneet can decrypt this secret message because the decryption algorithm required Suneet's decryption key, known only to Suneet. Since users choose their own decryption keys, the weakness 0f DES is avoided.

The main issue for public-key encryption is how encryption and decryption keys are chosen. Technically, public-key encryption algorithms rely on the existence of one-way functions, which are functions whose inverse is computationally very hard to determine.

The RSA algorithm, for example is based on the observation that although checking whether a given number of prime is easy, determining the prime factors of a nonprime number is extremely hard. (Determining the prime factors of a number with over 100 digits can take years of CPU-time on the fastest available computers today.)

We now sketch the intuition behind the RSA algorithm, assuming that the data to be encrypted is an integer 1. To choose an encryption key and a decryption key, our friend Suneet-- create a public key by computing the product of two large prime numbers: PI and P2. The private key consists of the pair (PI, P2) and decryption algorithms cannot be used if the product of PI and P2 is known. So we publish the product PI *P2, but an unauthorized user would need to be able to factor PIP2 to steal data. By choosing PI and P2 to be sufficiently large (over 100 digits), we can make it very difficult (or nearly impossible) for an intruder to factorize it.



Although this technique is secure, but it is also computationally expensive. A hybrid scheme used for secure communication is to use DES keys exchanged via a public-key encryption scheme and DES encryption is used on the data transmitted subsequently.

Disadvantages of encryption

There are following problems of Encryption:

- Key management (i.e. keeping keys secret) is a problem. Even in public-key encryption the decryption key must be kept secret.
- Even in a system that supports encryption, data must often be processed in plaintext form. Thus sensitive data may still be accessible to transaction programs.
- Encrypting data gives rise to serious technical problems at the level of physical storage organization. For example indexing over data, which is stored in encrypted form, can be very difficult.

Decryption: Decryption is the process of taking encoded or encrypted text or other data and converting it back into text that you or the computer can read and understand. This term could be used to describe a method of un-encrypting the data manually or with un-encrypting the data using the proper codes or keys.

Data may be encrypted to make it difficult for someone to steal the information. Some companies also encrypt data for general protection of company data and trade secrets. If this data needs to be viewable, it may require decryption. If a decryption passcode or key is not available, special software may be needed to decrypt the data using algorithms to crack the decryption and make the data readable.

Transaction Management, Recovery and Query Processing

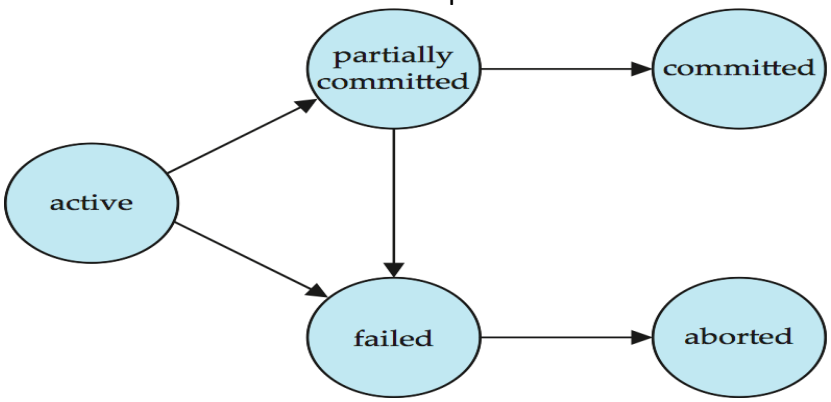
8.1 Introduction to transaction, ACID properties

A **transaction**is a *unit* of program execution that accesses and possibly updates various data items. E.g. transaction to transfer $50 from account A to account B:

1. **read**(*A*)
2. *A* := *A* – 50
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 50
6. **write**(*B*)

Transaction State

- **Active** –the initial state; the transaction stays in this state while it is executing
- **Partially committed** –after the final statement has been executed.
- **Failed** -- after the discovery that normal execution can no longer proceed.
- **Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - Restart the transaction
    - can be done only if no internal logical error
  - Kill the transaction
- **Committed** – after successful completion.



Two main issues to deal with:

1. Failures of various kinds, such as hardware failures and system crash
2. Concurrent execution of multiple transactions

Atomicity requirement

1. if the transaction fails after step 3 and before step 6, money will be "lost" leading to an inconsistent database state
   Failure could be due to software or hardware
2. the system should ensure that updates of a partially executed transaction are not reflected in the database

Durability requirement — once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures.

Isolation requirement — if between steps 3 and 6, another transaction T2 is allowed to access the partially updated database, it will see an inconsistent database (the sum A *+ B* will be less than it should be).

T1		**T2**
1.	**read** (*A*)	
2.	*A* := *A* – 50	
3.	**write**(*A*)	
		read(A), read(B), print(A+B)
4.	**read**(*B*)	
5.	*B* := *B* + 50	
6.	**write**(*B)*	

Isolation can be ensured trivially by running transactions **serially**. i.e. that is, one after the other.

ACID properties of transaction

A transaction is a unit of program execution that accesses and possibly updates various data items. To preserve the integrity of data the database system must ensure:

1. **Atomicity.** Either all operations of the transaction are properly reflected in the database or none are.
2. **Consistency.** Execution of a transaction in isolation preserves the consistency of the database.
3. **Isolation.** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.

   That is, for every pair of transactions $T_i$and T_j, it appears to $T_i$that either T_j, finished execution before T_i started, or T_j started execution after T_i finished.
4. **Durability.** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures

8.2 Introduction to concurrency control

The process of managing simultaneous operations against a database so that data integrity is maintained and the operations do not interfere with each other in a multiuser environment.

Authentication and authorization:
Databases are a method of storing and organizing large sets of information, but not everyone who has access to a database should be allowed to view or change all of the records inside of it. To manage this situation, database administrators use a security system to control database privileges. When a database receives a request for a specific resource, it goes through a process of authentication and authorization to determine whether or not to release the requested data.

Authentications:
Authentication verifies who you are. Its task is to identify a person/software connecting to a database. It is the process or act of confirming that a user who is attempting to log into a database is authorized to do so. Authentication acquires one more dimension because it may happen at different levels. It may be performed by the database itself, or the setup may be changed to allow either the operating system, or some other external method, to authenticate users.

8.3 Reasons of transaction failure, system recovery and media recovery

Transaction failure:
A transaction has to abort when it fails to execute or when it reaches a point from where it can't go any further. This is called transaction failure where only a few transactions or processes are hurt.

Reasons for a transaction failure could be:

- **Logical errors:** Where a transaction cannot complete because it has some code error or any internal error condition.
- **System errors:** Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

System Crash:
There are problems – external to the system – that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure.
Examples may include operating system errors.

Disk Failure:
In early days of technology evolution, it was a common problem when hard-disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or

any other failure, which destroys all or a part of disk storage.

Recovery and Atomicity:

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following −

- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Log-based Recovery

Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows −

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.

$<T_n$, Start>

- When the transaction modifies an item X, it write logs as follows −

$<T_n, X, V_1, V_2>$

It reads T_n has changed the value of X, from V_1 to V_2.

- When the transaction finishes, it logs −

$<T_n$, commit>

The database can be modified using two approaches −

- **Deferred database modification** − All logs are written on to the stable storage and the database is updated when a transaction commits.
- **Immediate database modification** − Each log follows an actual database modification. That is, the database is modified immediately after every operation.

Recovery with Concurrent Transactions

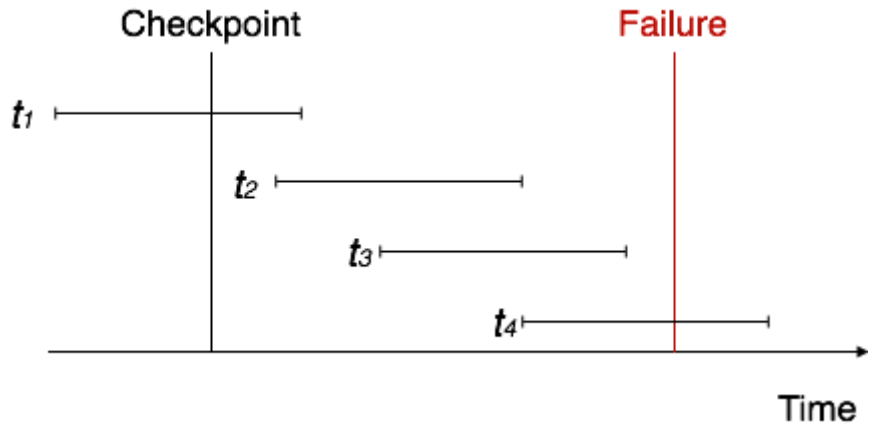
When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

Checkpoint

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. 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.

Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner –



- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with <T_n, Start> and <T_n, Commit> or just <T_n, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with <T_n, Start> but no commit or abort log found, it puts the transaction in undo-list.

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

8.4 Introduction to query processing, steps in query processing

Query Processing refers to the range of activities involved in extracting data from a database. The activities include translation of query in high-level database languages into expression that can be used at the physical level of the files system, a variety of query-optimizing transformations, and actual evaluation of queries.

Steps of query processing:

- Parsing and translation
- Optimization
- Evaluation

Before query processing can begin, the system must be translated, the query into a usable form. A language such as SQL is suitable for human use, but is still suited to be the system's internal representation of a query. A more useful internal representation is one based on the extended relational algebra.

Thus the first action the system must take in query processing is to translate a given query into its internal form. This translated process is similar to the work perform by the parser checks the syntax of the user's query, verifies that the relation names appearing in the query are names of the relations in the database, and so on. The system constructs a parse-tree representation of the query, which it then translates into a relational-algebra expression. If the query was expressed in term of a view, the translation phase also replaces all uses of the view by the relational- algebra expression that defines the view. Most compiler texts cover parsing in details.

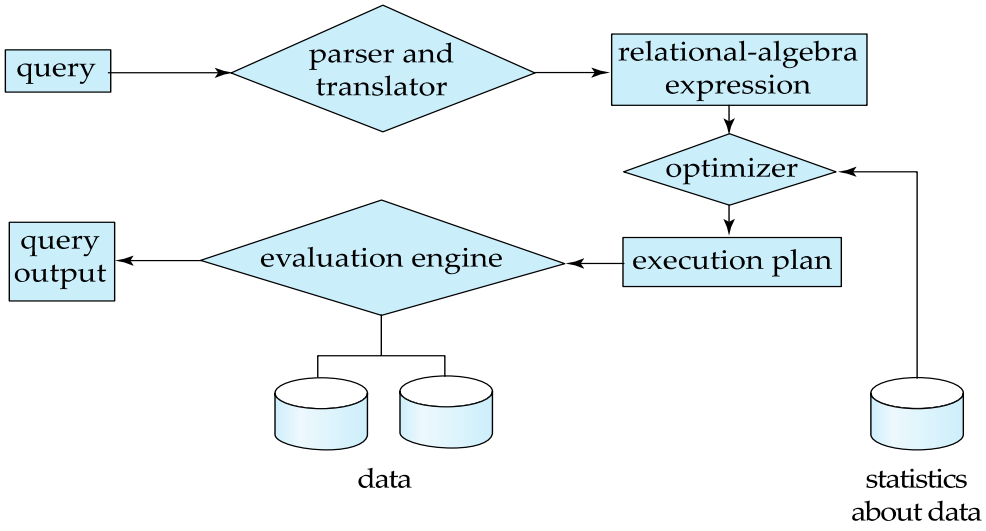


Fig: Steps in query processing

Parsing and translation:

- Translate the query into its internal form. This is then translated into relational algebra.
- Parser checks syntax, verifies relations

Query Optimization:

- Amongst all equivalent evaluation plans choose the one with lowest cost.
- Cost is estimated using statistical information from the database catalogue. For e.g. number of tuples in each relation, size of tuples, etc.

Evaluation

The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

The End