System Analysis And DataBase

اسم المادة : تحليل نظم وقواعد بيانات

المرحلة : الثانية – مسائي

مدرس المادة : جنان رضا مطر

اللقب العلمي : مدرس مساعد

System Analysis And DataBase

Chapter one
System analysis

- **Introduction**
- **Characteristics of a System**
- **Elements of a System**
- **Types of systems**
- **What Is an Information System?**
- **System Development Life Cycle SDLC**

System analysis

1.1 Introduction:

The term system is derived from the Greek word *systema*, which means an organized relationship among functioning units or components. A system exists because it is designed to achieve one or more objectives. We come into daily contact with the transportation system, the telephone system, the accounting system, the production system, and, for over two decades, the computer system. Similarly, we talk of the business system and of the organization as a system consisting of interrelated departments (subsystems) such as production, sales, personnel, and an information system.

The study of systems concepts, then, has three basic implications:

1. A system must be designed to achieve a predetermined objective.
2. Interrelationships and interdependence must exist among the components.
3. The objectives of the organization as a whole have a higher priority than the objectives of its subsystems.

1.2 Characteristics of a System

Our definition of a system suggests some characteristics that are present in all systems: organization (order), interaction, interdependence, integration and a central objective.

1.2.1 Organization

Organization implies structure and order. It is the arrangement of components that helps to achieve objectives.

1.2.2 Interaction

Interaction refers to the manner in which each component functions with other components of the system.

1.2.3 Interdependence

Interdependence means that parts of the organization or computer system depend on one another. They are coordinated and linked together according to a plan. In summary, no subsystem can function in isolation because it is dependent on the data (inputs) it receives from other subsystems to perform its required tasks.

1.2.4 Integration

Integration refers to the holism of systems. Synthesis follows analysis to achieve the central objective of the organization. Integration is concerned with how a system is tied together. It is more than sharing a physical part or location. It means that parts of the system work together within the system even though each part performs a unique function. Successful integration will typically produce a synergistic effect and greater total impact than if each component works separately.

1.2.5 Central objective

The last characteristic of a system is its central objective. Objectives may be real or stated. Although a stated objective may be the real objective, it is not uncommon for an organization to state one objective and operate to achieve another. The important point is that users must know the central objective of a computer application early in the analysis for a successful design and conversion. Political as well as organizational considerations often cloud the real objective. This means that the analyst must work around such obstacles to identify the real objective of the proposed change.

1.3 Elements of a System

In most cases, systems analysts operate in a dynamic environment where change is a way of life. The environment may be a business firm, a business application, or a computer system. To reconstruct a system, the following key elements must be considered:

1. Outputs and inputs.
2. Processor(s).
3. Control.
4. Feedback.
5. Environment.
6. Boundaries and interface.

1.3.1 Outputs and Inputs

A major objective of a system is to produce an output that has value to its user. Whatever the nature of the output (goods, services, or information), it must be in line with the expectations of the intended user. Inputs are the elements (material, human resources, and information) that enter the system for processing. Output is the outcome of processing. A system feeds on input to produce output in much the same way that a business brings in human, financial, and material resources to produce goods and services. It is important to point out here that determining the output is a first step in
specifying the nature, amount, and regularity of the input needed to operate a system.

1.3.2 Processor(s)

The processor is the element of a system that involves the actual transformation of input into output. It is the operational component of a system. Processors may modify the input totally or partially, depending on the specifications of the output. This means that as the output specifications change so does the processing. In some cases, input is also modified to enable the processor to handle the transformation.

1.3.3 Control

The control element guides the system. It is the decision – making subsystem that controls the pattern of activities governing input, processing, and output. In an organizational context, management as a decision – making body controls the inflow, handling and outflow of activities that affect the welfare of the business. In a computer system,

the operating system and accompanying software influence the behavior of the system. Output specifications determine what and how much input is needed to keep the system in balance. In systems analysis, knowing the attitudes of the individual who controls the area for which a computer is being considered can make a difference between the success and failure of the installation. Management support is required for securing control and supporting the objective of the proposed change.

1.3.4 Feedback

Control in a dynamic system is achieved by feedback. Feedback measures output against a standard in some form of cybernetic procedure that includes communication and control. Output information is fed back to the input and / or to management (Controller) for deliberation. After the output is compared against performance standards, changes can result in the input or processing and consequently, the output. Feedback may be positive or negative, routing or informational. Positive feedback reinforces the performance of the system. It is routine in nature. Negative feedback generally provides the controller with information for action. In systems analysis, feedback is important in different ways. During analysis, the user may be told that the problems in a given application verify the initial concerns and justify the need for change. Another form of feedback comes after the system is implemented. The user informs the analyst about the performance of the new installation. This feedback often results in enhancements to meet the user's requirements.

1.3.5 Environment

The environment is the "supra system" within which an organization operates. It is the source of external elements that impinge on the system. In fact, it often determines how a system must function. For example, the organization's environment, consisting of vendors, competitors, and others, may provide constraints and, consequently, influence the actual performance of the business.

1.3.6 Boundaries and interface

A system should be defined by its boundaries – the limits that identify its components, processes and interrelationship when it interfaces with another system. For example, a teller system in a commercial bank is restricted to the deposits, withdrawals and related activities of customers checking and savings accounts. It may exclude mortgage foreclosures, trust activities, and the like.

Each system has boundaries that determine its sphere of influence and control.

The following Figure 1.1 is a graphical representation of Functions of an information system.

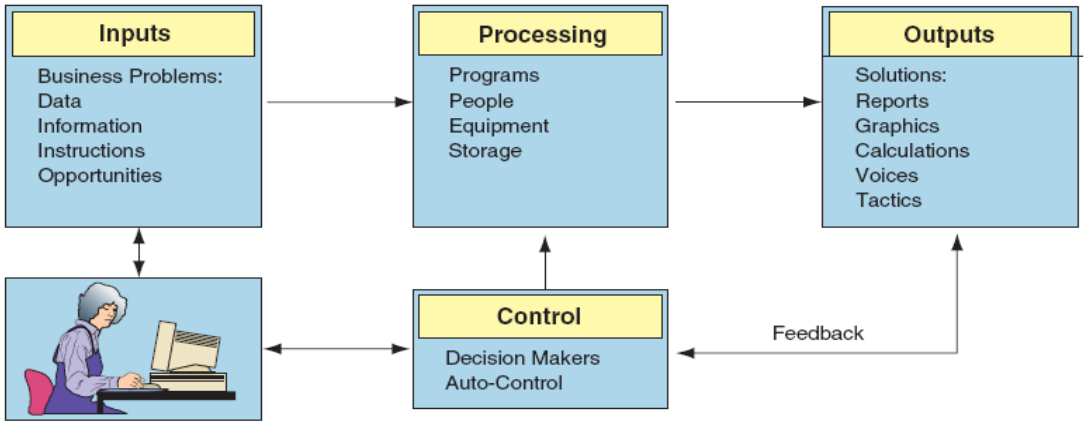


Figure 1.1: Functions of an information system

1.4 Types of systems

The frame of reference within which one views a system is related to the use of the systems approach for analysis. Systems have been classified in different ways. Common classifications are:

(1) physical or abstract

(2) open or closed

(3) “man – made” information systems.

1.4.1 Physical or abstract systems

Physical systems are tangible entities that may be static or dynamic in operation. For example, the physical parts of the computer center are the officers, desks, and chairs that facilitate operation of the computer. They can be seen and counted; they are static. In contrast, a programmed computer is a dynamic system. Data, programs, output, and applications change as the user's demands or the priority of the information requested changes. Abstract systems are conceptual or non-physical entities. They may be as straightforward as formulas of relationships among sets of variables or models – the abstract conceptualization of physical situations. A model is a representation of a real or a planned system. The use of models makes it easier for the analyst to visualize relationships in the system under study. The objective is to point out the significant elements and the key interrelationships of a complex system.

1.4.2 Open or Closed Systems

Another classification of systems is based on their degree of independence. An open system has many interfaces with its environment. It permits interaction across its boundary; it receives inputs from and delivers outputs to the outside. An information system falls into this category, since it must adapt to the changing demands of the user. In contrast, a closed system is isolated from environmental influences. In reality

1.4.3 Man – Made Information Systems

Ideally, information reduces uncertainty about a state or event. For example, information that the wind is calm reduces the uncertainty that the boat trip will be pleasant. An information system is the basis for interaction between the user and the analyst. It provides instruction, commands and feedback. It determines the nature of the relationships among decision-makers. In fact, it may be viewed as a decision center for personnel at all levels. From this basis, an information system may be defined as a set of devices, procedures and operating systems designed

around user based criteria to produce information and communicate it to the user for planning, control and performance. In systems analysis, it is important to keep in mind that considering an alternative system means improving one or more of these criteria. Many practitioners fail to recognize that a business has several information systems; each is designed for a purpose and works to accommodate data flow, communications, decision making, control and effectiveness.

1.5 What Is an Information System?

An information system can be defined ***technically*** as a set of interrelated components that collect (or retrieve), process, store, and distribute information to support decision making and control in an organization. In addition to supporting decision making, coordination, and control, information systems may also help managers and workers analyze problems, visualize complex subjects, and create new products. Three activities in an information system produce the information that organizations need to make decisions, control operations, analyze problems, and create new products or services. These activities are input, processing, Input captures or collects raw data from within the organization or from its external environment. Processing converts this raw input into a more meaningful form. Output transfers the processed information to the people who will use it or to the activities for which it will be used. Information systems also require feedback, which is output that is returned to appropriate members of the organization to help them evaluate or correct the input stage.

The following Figure 1.2 is a graphical representation of Components of Information System

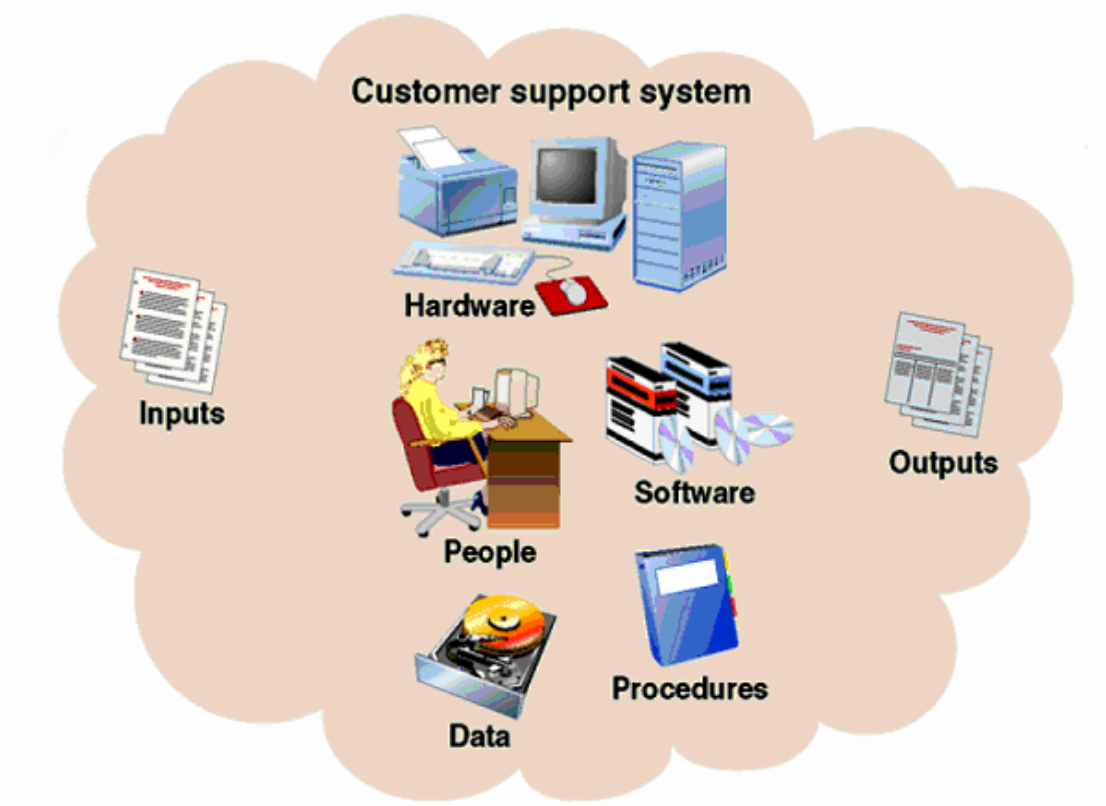


Figure 1. 2 Components of Information System

1.6 System Development Life Cycle SDLC

Information system development involves various activities performed together. The figure 1.3 suggest that the SDLC phases proceed in a logical path from start to finish.

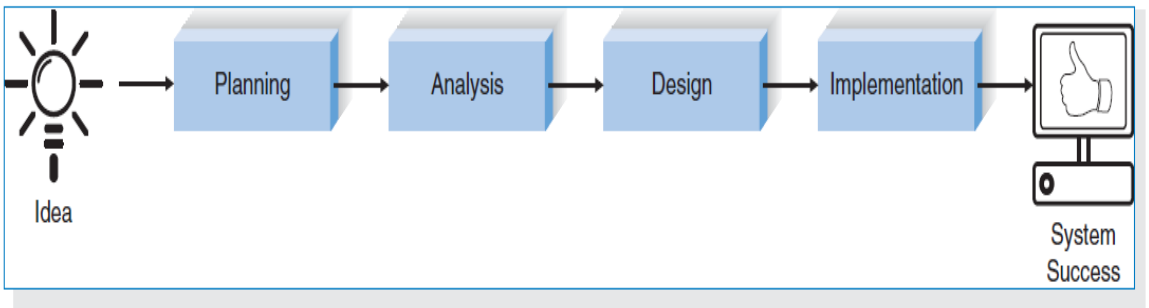


Figure 1.3 The Systems Development Life Cycle

The stages involved during System Life Cycle are ::

1.6.1Planning

The *planning phase* is the fundamental process of understanding *why* an information system should be built and determining how the project team will go about building it. It has two steps:

1.During *project initiation*, the system's business value to the organization is identified—how will it lower costs or increase revenues? Most ideas for new systems come from outside the IS area (from the marketing department, accounting department, etc.) in the form of a system request. A *system request* presents a brief summary of a business need, and it explains how a system that supports the need will create business value. The IS department works together with the person or department generating the request (called the *project sponsor*) to conduct a feasibility analysis.

The *feasibility analysis* examines key aspects of the proposed project:

- The technical feasibility (Can we build it?)
- The economic feasibility (Will it provide business value?)
- The organizational feasibility (If we build it, will it be used?)

The system request and feasibility analysis are presented to an information systems *approval committee* (sometimes called a *steering committee*), which decides whether the project should be undertaken.

2. Once the project is approved, it enters *project management.* During project management, the *project manager* creates a *work plan*, staffs the project, and puts techniques in place to help the project team control and direct the project through the entire SDLC. The deliverable for project management is a *project plan* that describes how the project team will go about developing the system.

1.6.2Analysis

The *analysis phase* answers the questions of *who* will use the system, *what* the system will do, and *where* and *when* it will be used. During this phase, the project team investigates any current system(s), identifies

improvement. opportunities, and develops a concept for the new system. This phase has three steps:

1. An *analysis strategy* is developed to guide the project team's efforts. Such a strategy usually includes a study of the current system (called the *as-is system*) and its problems, and envisioning ways to design a new system (called the *to-be system*).

2. The next step is *requirements gathering* (e.g., through interviews, group workshops, or questionnaires). The analysis of this information—in conjunction with input from the project sponsor and many other people—leads to the development of a concept for a new system. The system concept is then used as a basis to develop a set of business *analysis models* that describes how the business will operate if the new system were developed. The set typically includes models that represent the data and processes necessary to support the underlying business process.

3. The analyses, system concept, and models are combined into a document called the *system proposal*, which is presented to the project sponsor and other key decision makers (e.g., members of the approval committee) who will decide whether the project should continue to move forward. The system proposal is the initial deliverable that describes what business requirements the new system should meet. Because it is really the first step in the design of the new system, some experts argue that it is inappropriate to use the term *analysis* as the name for this phase; some argue a better name would be *analysis and initial design.* Because most organizations continue to use the name *analysis* for this phase, we will use it in this book as well. It is important to remember, however, that the deliverable from the analysis phase is both an analysis and a high-level initial design for the new system

1.6.3 Design

The *design phase* decides *how* the system will operate in terms of the hardware, software, and network infrastructure that will be in place; the

user interface, forms, and reports that will be used; and the specific programs, databases, and files that will be needed. Although most of the strategic decisions about the system are made in the development of the system concept during the analysis phase, the steps in the design phase determine exactly how the system will operate. The design phase has four steps:

1. The *design strategy* must be determined. This clarifies whether the system will be developed by the company's own programmers, whether its development will be outsourced to another firm (usually a consulting firm), or whether the company will buy an existing software package.

2. This leads to the development of the basic *architecture design* for the system that describes the hardware, software, and network infrastructure that will be used. In most cases, the system will add to or change the infrastructure that already exists in the organization. The *interface design* specifies how the users will move through the system (e.g., by navigation methods such as menus and on-screen buttons) and the forms and reports that the system will use.

3. The *database and file specifications* are developed. These define exactly what data will be stored and where they will be stored.

4. The analyst team develops the *program design*, which defines the programs that need to be written and exactly what each program will do.

This collection of deliverables (architecture design, interface design, database and file specifications, and program design) is the *system specification* that is used by the programming team for implementation. At the end of the design phase, the feasibility analysis and project plan are reexamined and revised, and another decision is made by the project sponsor and approval committee about whether to terminate the project or continue.

1.6.4 Implementation

The final phase in the SDLC is the *implementation phase*, during which the system is actually built (or purchased, in the case of a packaged

software design and installed). This is the phase that usually gets the most attention, because for most systems it is the longest and most expensive single part of the development process. This phase has three steps:

1. System *construction* is the first step. The system is built and tested to ensure that it performs as designed. Since the cost of fixing bugs can be immense, testing is one of the most critical steps in implementation. Most organizations spend more time and attention on testing than on writing the programs in the first place.

2. The system is installed. *Installation* is the process by which the old system is turned off and the new one is turned on. There are several approaches that may be used to convert from the old to the new system. One of the most important aspects of conversion is the *training plan*, used to teach users how to use the new system and help manage the changes caused by the new system.

3. The analyst team establishes a *support plan* for the system. This plan usually includes a formal or informal post-implementation review, as well as a systematic way for identifying major and minor changes needed for the system.

There is an implied phase is usually a subset of all the stages as in the modern SDLC models it is test phase, the testing activities are mostly involved in all the stages of SDLC. However this stage refers to the testing only stage of the product where products defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined.

The following Figure 1.4 is a graphical representation of the various stages of a typical SDLC.

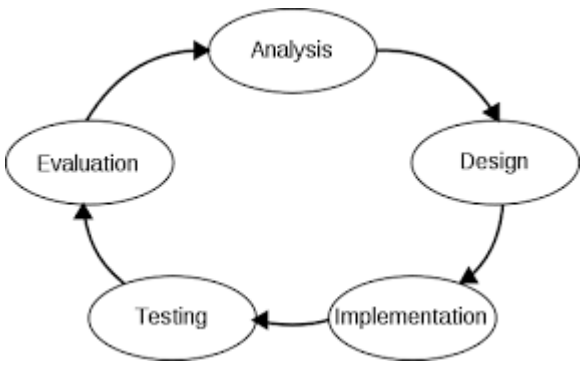


Figure 1.4 Various stages of SDLC

Chapter 2

Database and Database Management System (DBMS)

1. Introduction
2. **Database management system (DBMS)**
3. **Advantages of DBMS**
4. **Architecture of DBMS**
5. **Importance of DBMS**
6. **Components of DBS environment**
7. **Data independence**
8. **Instance , schema and mapping**
9. **ACID test**

2.1 Introduction

A **database** is a collection of information that is organized so that it can easily be accessed, managed, and updated. In one view, databases can be classified according to types of content: bibliographic, full-text, numeric, and images.

In computing, databases are sometimes classified according to their organizational approach. The most prevalent approach is the relational database, a tabular database in which data is defined so that it can be reorganized and accessed in a number of different ways. A distributed database is one that can be dispersed or replicated among different points in a network. An object-oriented programming database is one that is congruent with the data defined in object classes and subclasses.

Computer databases typically contain aggregations of data records or files, such as sales transactions, product catalogs and inventories, and customer profiles. Typically, a database manager provides users the capabilities of controlling read/write access, specifying report generation, and analyzing usage. Databases and database managers are prevalent in large mainframe systems, but are also present in smaller distributed workstation and mid-range systems such as the AS/400 and on personal computers. SQL(Structured Query Language) is a standard language for making interactive queries from and updating a database such as IBM's DB2, Microsoft's SQL Server, and database products from Oracle, Sybase, and Computer Associates.

2.1.1 Database System Applications

Databases are widely used. Here are some representative applications:

- **Banking:** For customer information, accounts, and loans, and banking transactions.
- **Airlines:** For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner—terminals situated around the world accessed the central database system through phone lines and other data networks.
- Universities: For student information, course registrations, and grades.

• **Credit card transactions:** For purchases on credit cards and generation of monthly statements.
• **Telecommunication:** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
• **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
• **Sales:** For customer, product, and purchase information.
• **Manufacturing:** For management of supply chain and for tracking production of items in factories, inventories of items in warehouses/stores, and orders for items.
• **Human resources:** For information about employees, salaries, payroll taxes and benefits, and for generation of paychecks.

2.1.2 Database Administrator

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator (DBA). The functions of a DBA include:
• **Schema definition.** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
• **Storage structure and access-method definition.**
• **Schema and physical-organization modification.** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.
• **Granting of authorization for data access.** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.
• **Routine maintenance**. Examples of the database administrator's routine maintenance activities are:

- ❖ Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.

- ❖ Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
- ❖ Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

2.2 Database management system (DBMS)

A **database-management system (DBMS)** is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.
Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

2.2.1 Advantages of DBMS

1. Database Development: It allows organizations to place control of database development in the hands of database administrators (DBAs) and other specialists.
2. Data independence: Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.
3. Efficient data access: A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. It allows different user application programs to easily access the same database. Instead of having to write computer programs to extract information, user can ask simple questions in a query language
4. Data integrity and security: If data is always accessed through the DBMS, the DBMS can enforce :

- Integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded.
- Also, the DBMS can enforce access controls that govern what data is visible to different classes of users.

5. Crash recovery: the DBMS protects users from the effects of system failures.
6. Data administration and Concurrent access: When several users share the data(more than one user access the database at the same time), DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time.

2.2.2 The Three-Schema Architecture

The goal of the three-schema architecture, is to separate the user applications from 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.

Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.

3.The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group.

The following Figure 2.1 is a graphical representation of three level of architecture.

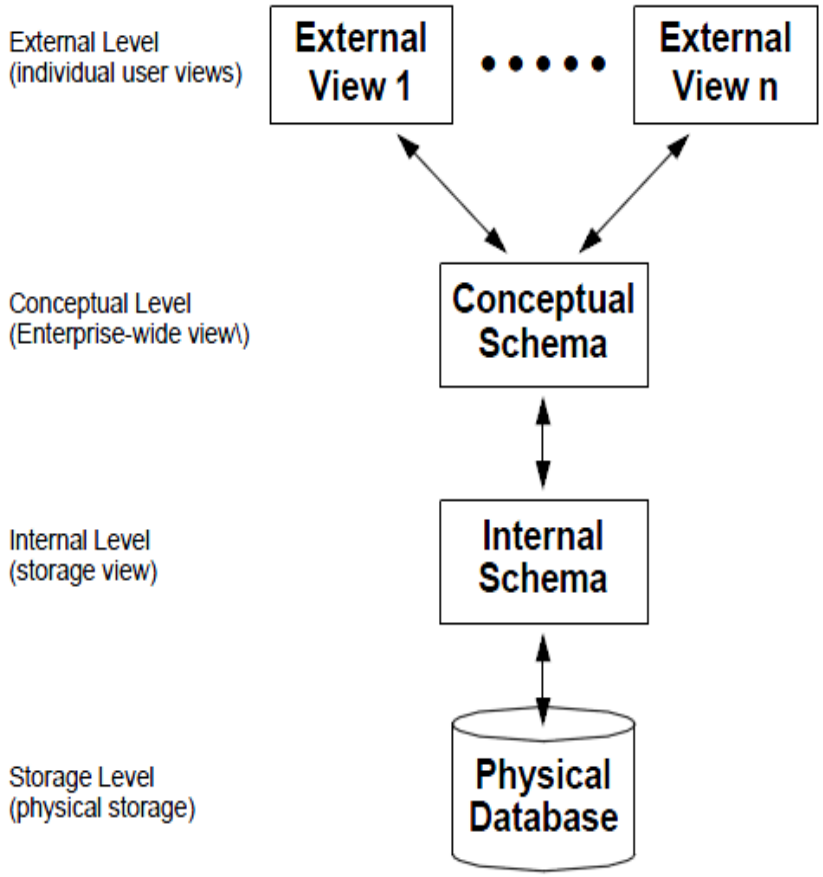


Figure 2.1 The three level of architecture

The processes of transforming requests and results between levels are called **mappings.**

2.3 Data Independence

The three-schema architecture can be used to explain the concept of data independence, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. Logical data independence 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 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.

2.4 Components of Database System Environment

The term **database system** refers to an organization of components that define and regulate the collection, storage, management, and use of data within a database environment. From a general management point of view, the database system is composed of the five major parts shown in the following Figure 2.2: hardware, software, people, procedures, and data.

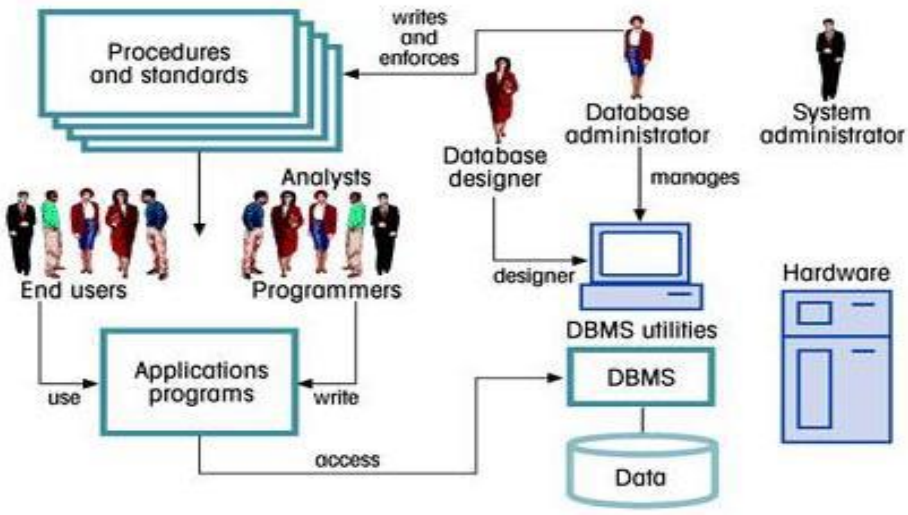


Figure 2.2 Components of database system environment

1. Hardware.

Hardware refers to all of the system's physical devices; for example, computers (PCs, workstations, servers, and supercomputers), storage devices, printers, network devices (hubs, switches, routers, fiber optics),and other devices (automated teller machines, ID readers, and so on).

2. Software.

Although the most readily identified software is the DBMS itself, to make the database system function fully, three types of software are needed: operating system software, DBMS software, and application programs and utilities.

manages all hardware components and makes it possible for all other software to run on the computers. Examples of operating system software include Microsoft Windows, Linux, MacOS, UNIX, and MVS.

b. DBMS software:

manages the database within the database system. Some examples of DBMS software include Microsoft's SQL Server, Oracle Corporation's Oracle, Sun's MySQL, and IBM's DB2.

c. Application programs and utility software:

3. People.

This component includes all users of the database system. On the basis of primary job functions, five types of users can be identified in a database system: system administrators, database administrators, database designers, system analysts and programmers, and end users. Each user type, described below, performs both unique and complementary functions.

1. **System administrators** oversee the database system's general operations.

2. **Database administrators**, also known as DBAs, manage the DBMS and ensure that the database is functioning properly.

3. **Database designers** design the database structure. They are, in effect, the database architects. If the database design is poor, even the best application programmers and the most dedicated DBAs cannot produce a useful database environment. Because organizations strive to optimize their data resources, the database

designer's job description has expanded to cover new dimensions and growing responsibilities.

4. **System analysts and programmers** design and implement the application programs. They design and create the data entry screens, reports, and procedures through which end users access and manipulate the database's data.
5. **End users** are the people who use the application programs to run the organization's daily operations. For example, salesclerks, supervisors, managers, and directors are all classified as end users. High-level end users employ the information obtained from the database to make tactical and strategic business decisions.

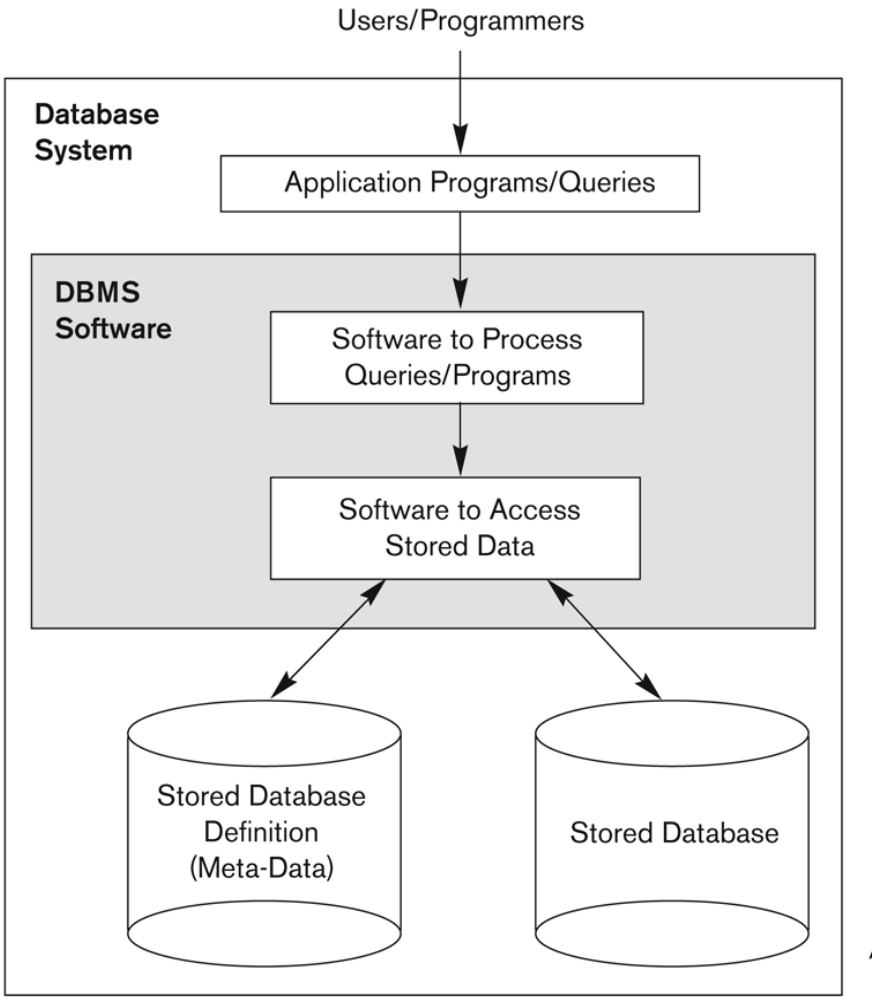
4. Procedures.

Procedures are the instructions and rules that govern the design and use of the database system. Procedures are a critical, although occasionally forgotten, component of the system. Procedures play an important role in a company because they enforce the standards by which business is conducted within the organization and with customers. Procedures are also used to ensure that there is an organized way to monitor and audit both the data that enter the database and the information that is generated through the use of those data.

5. Data.

The word data covers the collection of facts stored in the database. Because data are the raw material from which information is generated, the determination of what data are to be entered into the database and how those data are to be organized is a vital part of the database designer's job.

The following Figure 2.3 is a graphical representation of Database system.



(Database (DB) + DBMS → Database System (DBS))

Figure 2.3 Database system

2.5 Instance , schema and mapping

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database. The overall design of the database is called the database **schema**. Schemas are changed infrequently, if at all.

The concept of database schemas and instances can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a

given instant. The values of the variables in a program at a point in time correspond to an *instance* of a database schema.
Database systems have several schemas, partitioned according to the levelsof abstraction:

- Physical schema describes the database design at the physical level.
- Logical schema describes the database design at the logical level.
- A database may also have several schemas at the view level, sometimes called subschemas, which describe different views of the database.

Employee-schema=(emp-id, emp-name, salary,sex,department).

Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting application programs. Application programs are said to exhibit **physical data independence** if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.

The processes of transforming requests and results between levels are called **mappings.**

2.6 ACID Model

ACID ***(Atomicity, Consistency, Isolation, Durability)*** is a set of properties that guarantee database transactions are processed reliably. A transaction a single logical operation on the data is called a transaction. For example, a transfer of funds from one bank account to another, even though that might involve multiple changes (such as debiting one account and crediting another), is a single transaction.

The ACID model is one of the oldest and most important concepts of database theory. There are four goals that every

database management system should be achieved: atomicity, consistency, isolation and durability. No database can be considered **reliable** if the database fails to meet any of these four goals.Jim Gray defined these properties of a **reliable transaction system** in the late 1970s and developed technologies to automatically achieve them.

2.6.1 Characteristics of ACID Test

1- Atomicity

Atomicity requires that database modifications must follow an "all or nothing" rule. Each transaction is said to be atomic. If one part of the transaction fails, the entire transaction fails and the database state is unchanged. It is important that the database management system maintain the atomic nature of transactions in spite of any DBMS, operating system or hardware failure.

2- **Consistency**

Consistency states that only valid data will be written to the database. If a transaction is executed that violates the database's consistency rules, the entire transaction will be rolled back and the database will be restored to the state consistent with those rules. if a transaction successfully executes, it will take the database from one state that is consistent with the rules to another state that is also consistent with the rules.

3- Isolation

Isolation requires that multiple transactions occurring at the same time not impact each other's execution. For example, if salaam issues a transaction against a database at the same time that layla issues a different transaction; both transactions should operate on the database in an isolated manner. The database should either perform salaam's entire transaction before executing layla's or vice- versa. That leads to prevents salaam's transaction from reading intermediate data produced as a side effect of part of

layla's transaction that will not eventually be committed to the database. The isolation property does not ensure which transaction will execute first, but it will not interfere with each other. Isolation is helped to decrease the speed of this type of concurrency management. To respect the isolation property is better to use a **serial model** where no two transactions can occur on the same data at the same time and where the result is predictable.

2.7 Database Design

Database design is the process of producing a detailed data model of a database , this data model which can then be used to create a database.
The term database design can be used to describe many different parts of the design , it can be thought of as the logical design of the base data structures used to store the data. In the relational model these are the tables and view

The Design Process

The design process consists of the following steps:

1. Determine the purpose of your database - This helps prepare you for the remaining steps.

2. Find and organize the information required - Gather all of the types of information you might want to record in the database, such as product name and order number.

3. Divide the information into tables - Divide your information items into major entities or subjects, such as Products or Orders. Each subject then becomes a table.

4. Turn information items into columns - Decide what information you want to store in each table. Each item becomes a field, and is displayed as a column in the table. For example, an Employees table might include fields such as Last Name and Hire Date.

5. Specify primary keys - Choose each table's primary key. The primary key is a column that is used to uniquely identify each row. An example might be Product ID or Order ID.

6. Set up the table relationships - Look at each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.
7. Refine your design - Analyze your design for errors. Create the tables and add a few records of sample data. See if you can get the results you want from your tables. Make adjustments to the design, as needed.
8. Apply the normalization rules - Apply the data normalization rules to see if your tables are structured correctly. Make adjustments to the tables, as needed.

2.8 Database Model

A **database model** refers to the logical structure, representation or layout of a database and how the data will be stored, managed and processed within it. It helps in designing a database and serves as blueprint for application developers and database administrators in creating a database.

A database model is primarily a type of data model. Depending on the model in use, a database model can include entities, their relationships, data flow, tables and more. For example, within a hierarchal database mode, the data model organizes data in the form of a tree-like structure having parent and child segments.

Some of the popular database models include relational models, hierarchical models, flat file models, object oriented models, entity relationship models and network models.

In history of database design, three models have been in use.

- Hierarchical Model
- Network Model
- Relational Model

2.8.1 Hierarchical Model

In this model each entity has only one parent but can have several children . At the top of hierarchy there is only one entity which is called Root.

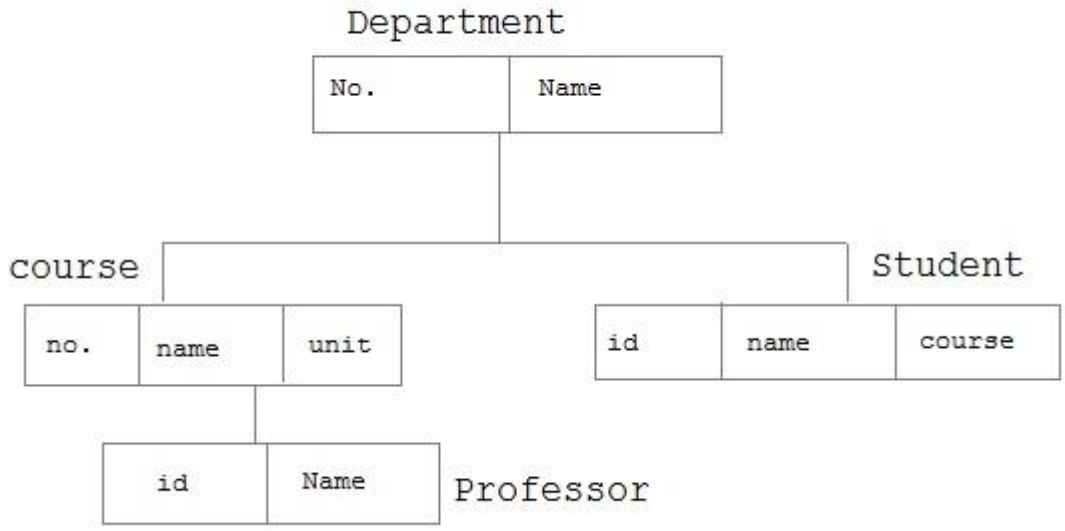


Figure 2.4 Hierarchical Model

2.8.2 Network Model

In the network model, entities are organized in a graph, in which some entities can be accessed through several path

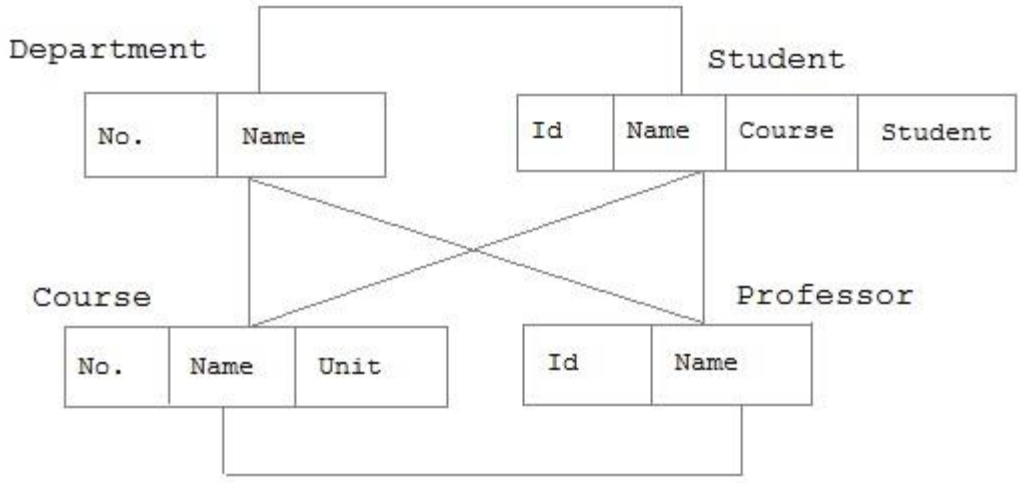


Figure 2.5 Network Model

2.8.3 Relational Model

In this model, data is organized in two-dimensional tables called relations. The tables or relation are related to each other.

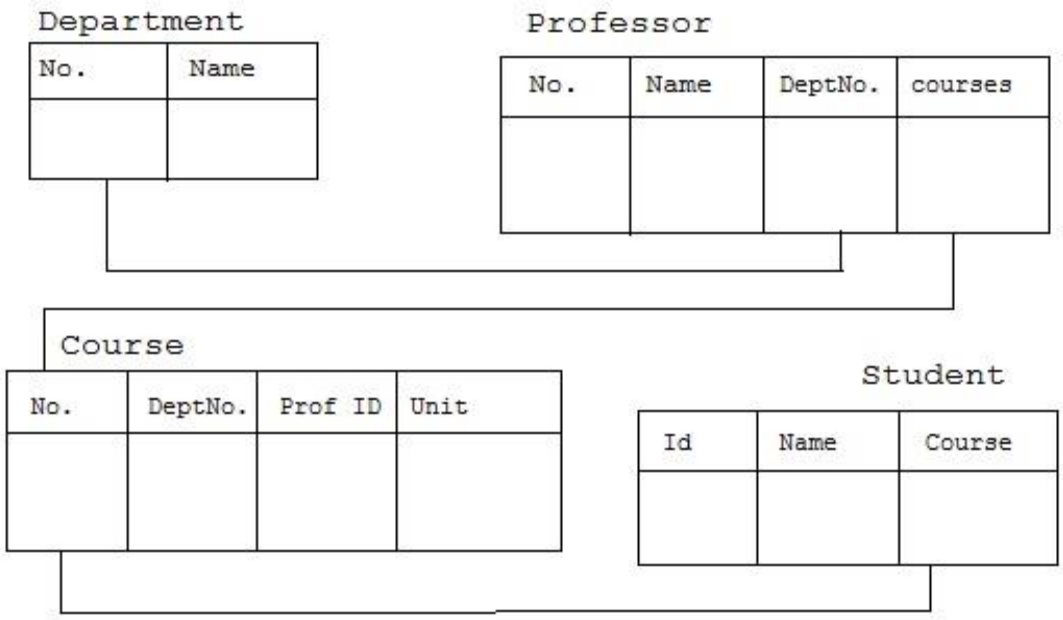


Figure 2.6 Relational Model

Chapter 3

RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)

1. Introduction
2. RDBMS Concepts
3. Database Keys

3.1 Introduction

A relational database management system (RDBMS) is a database engine/system based on the relational model specified by Edgar F. Codd--the father of modern relational database design--in 1970.

Most modern commercial and open-source database applications are relational in nature. The most important relational database features include an ability to use tables for data storage while maintaining and enforcing certain data relationships.

In 1970, Edgar F. Codd, a British computer scientist with IBM, published "A Relational Model of Data for Large Shared Data Banks." At the time, the renowned paper attracted little interest, and few understood how Codd's groundbreaking work would define the basic rules for relational data storage, which can be simplified as:

1. Data must be stored and presented as relations, i.e., tables that have relationships with each other, e.g., primary/foreign keys.
2. To manipulate the data stored in tables, a system should provide relational operators - code that enables the relationship to be tested between two entities.

Codd later published another paper that outlined the 12 rules that all databases must follow to qualify as relational. Many modern database systems do not follow all 12 rules, but these systems are considered relational because they conform to at least two of the 12rules.

Most modern commercial and open-source database systems are relational in nature and include well-known applications, e.g., Oracle DB (Oracle Corporation); SQL Server (Microsoft) and MySQL and Postgres (open source).

3.2 RDBMS Concepts

RDBMS is used to manage Relational database. **Relational database** is a collection of organized set of tables from which data can be accessed easily. Relational Database is most commonly used database. It consists of number of tables and each table has its own primary key.

3.2.1 What is Table ?

In Relational database, a **table** is a collection of data elements organized in terms of rows and columns. A table is also considered as convenient representation of **relations**. But a table can have duplicate tuples while a true **relation** cannot have duplicate tuples. Table is the most simplest form of data storage. Below is an example of Employee table.

ID	Name	Age	Salary
1	Adam	34	13000
2	Alex	28	15000
3	Stuart	20	18000
4	Ross	42	19020

3.2.2 What is a Record ?

A single entry in a table is called a **Record** or **Row**. A **Record** in a table represents set of related data. For example, the above **Employee** table has 4 records. Following is an example of single record.

1	Adam	34	13000

3.2.3 What is Field ?

A table consists of several records(row), each record can be broken into several smaller entities known as **Fields**. The above **Employee** table consist of four fields, **ID**, **Name**, **Age** and **Salary**.

3.2.4 What is a Column ?

In **Relational** table, a column is a set of value of a particular type. The term **Attribute** is also used to represent a column. For example, in Employee table, Name is a column that represent names of employee.

Name
Adam

Alex
Stuart
Ross

3.3 Database Keys

3.3.1 Introduction

For the purposes of clarity we will refer to keys in terms of RDBMS tables but the same definition, principle and naming applies equally to Entity Modeling and Normalization.

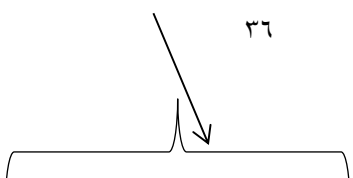
Keys are, as their name suggests, a key part of a relational database and a vital part of the structure of a table. They ensure each record within a table can be uniquely identified by one or a combination of fields within the table. They help enforce integrity and help identify the relationship between tables. There are three main types of keys, candidate keys, primary keys and foreign keys. There is also an alternative key or secondary key that can be used, as the name suggests, as a secondary or alternative key to the primary key

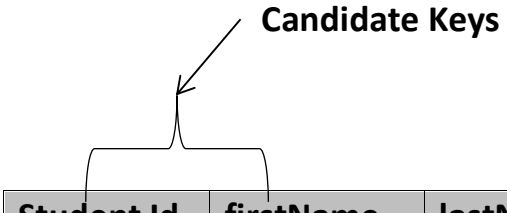
3.3.2 Super Key

A Super key is any combination of fields within a table that uniquely identifies each record within that table.

3.3.3 Candidate Key

A candidate is a subset of a super key. A candidate key is a single field or the least combination of fields that uniquely identifies each record in the table. The least combination of fields distinguishes a candidate key from a super key. Every table must have at least one candidate key but at the same time can have several.





Student Id	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042
L0023487	Peter	Murray	P301
L0018453	Anne	Norris	S042

As an example we might have a student_id that uniquely identifies the students in a student table. This would be a candidate key. But in the same table we might have the student's first name and last name that also, when combined, uniquely identify the student in a student table. These would both be candidate keys.

In order to be eligible for a candidate key it must pass certain criteria.

- It must contain unique values
- It must not contain null values
- It contains the minimum number of fields to ensure uniqueness
- It must uniquely identify each record in the table

Once your candidate keys have been identified you can now select one to be your primary key

3.3.4 Primary Key

A primary key is a candidate key that is most appropriate to be the main reference key for the table. As its name suggests, it is the primary key of reference for the table and is used throughout the database to help establish relationships with other tables. As with any candidate key the primary key must contain unique values, must never be null and uniquely identify each record in the table.

As an example, a student id might be a primary key in a student table, a department code in a table of all departments in an organisation. This module has the code DH3D 35 that is no doubt used in a database somewhere to identify RDBMS as a unit in a table of modules. In the table below we have selected the candidate key student_id to be our most appropriate primary key.

primary key

Student Id	**firstName**	**lastName**	**courseId**
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042
L0023487	Peter	Murray	P301
L0018453	Anne	Norris	S042

Primary keys are mandatory for every table each record must have a value for its primary key. When choosing a primary key from the pool of candidate keys always choose a single simple key over a composite key.

3.3.5 Foreign Key

A foreign key is generally a primary key from one table that appears as a field in another where the first table has a relationship to the second. In other words, if we had a table A with a primary key X that linked to a table B where X was a field in B, then X would be a foreign key in B.

An example might be a student table that contains the course_id the student is attending. Another table lists the courses on offer with course_id being the primary key. The 2 tables are linked through course_id and as such course_id would be a foreign key in the student table.

Student Id	**firstName**	**lastName**	**courseId**
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002

L0001198	Simon	McCloud	S042

foreign keys

Relationship

Primary keys

courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short course

3.3.6 Secondary Key or Alternative Key

A table may have one or more choices for the primary key. Collectively these are known as candidate keys as discuss earlier. One is selected as the primary key. Those not selected are known as secondary keys or alternative keys.

For example in the table showing candidate keys above we identified two candidate keys, studentId and firstName + lastName. The studentId would be the most appropriate for a primary key leaving the other candidate key as secondary or alternative key. It should be noted for the other key to be candidate keys, we are assuming you will never have a person with the same first and last name combination. As this is unlikely we might consider fistName+lastName to be a suspect candidate key as it would be restrictive of the data you might enter. It would seem a shame to not allow John Smith onto a course just because there was already another John Smith.

3.3.7 Simple Key

Any of the keys described before (ie primary, secondary or foreign) may comprise one or more fields, for example if firstName and lastName was our key this would be a key of two fields where as studentId is only one. A simple key consists of a single field to uniquely identify a record. In addition the field in itself cannot be broken down into other fields, for

example, studentId, which uniquely identifies a particular student, is a single field and therefore is a simple key. No two students would have the same student number.

3.3.8 Compound Key

A compound key consists of more than one field to uniquely identify a record. A compound key is distinguished from a composite key because each field, which makes up the primary key, is also a simple key in its own right. An example might be a table that represents the modules a student is attending. This table has a studentId and a moduleCode as its primary key. Each of the fields that make up the primary key are simple keys because each represents a unique reference when identifying a student in one instance and a module in the other.

3.3.9 Composite Key

A composite key consists of more than one field to uniquely identify a record. This differs from a compound key in that one or more of the attributes, which make up the key, are not simple keys in their own right. Taking the example from compound key, imagine we identified a student by their firstName + lastName. In our table representing students on modules our primary key would now be firstName + lastName + moduleCode. Because firstName + lastName represent a unique reference to a student, they are not each simple keys, they have to be combined in order to uniquely identify the student. Therefore the key for this table is a composite key.

3.4 Database Schema

When we talk about a database, we must differentiate between the **database schema**, which is the logical design of the database, and a **database instance**, which is a snapshot of the data in the database at a given instant in time.

The concept of a relation corresponds to the programming-language notion of a variable. The concept of a **relation schema** corresponds to the programming-language notion of type definition.

It is convenient to give a name to a relation schema, just as we give names to type definitions in programming languages. We adopt the

convention of using lowercase names for relations, and names beginning with an uppercase letter for relation schemas. Following this notation, we use *Account-schema* to denote the relation schema for relation *account*. Thus,

***Account-schema* = (*account-number*, *branch-name*, *balance*)**

We denote the fact that *account* is a relation on *Account-schema* by *account*(*Account-schema*)

In general, a relation schema consists of a list of attributes and their corresponding domains.

The concept of a **relation instance** corresponds to the programming language notion of a value of a variable. The value of a given variable may change with time; similarly the contents of a relation instance may change with time as the relation is updated. However, we often simply say "relation" when we actually mean "relation instance."

As an example of a relation instance, consider the *branch* relation of Figure 3.3. The schema for that relation is

***Branch-schema* = (*branch-name*, *branch-city*, *assets*)**

Note that the attribute *branch-name* appears in both *Branch-schema* and *Accountschema*.

This duplication is not a coincidence. Rather, using common attributes in relation schemas is one way of relating tuples of distinct relations. For example, suppose we wish to find the information about all of the accounts maintained in branches

Branch-name	Branch-city	Assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
Northtown	Rye	3700000
Perryridge	Horseneck	1700000
Powanl	Bennington	300000
Redwood	Paloalto	2100000
Round hill	Horseneck	8000000

Figure 3.3 The *branch* relation.

located in Brooklyn. We look first at the *branch* relation to find the names of all the branches located in Brooklyn. Then, for each such branch, we would look in the *account* relation to find the information about the accounts maintained at that branch.

This is not surprising—recall that the primary key attributes of a strong entity set appear in the table created to represent the entity set, as well

as in the tables created to represent relationships that the entity set participates in.

Let us continue our banking example. We need a relation to describe information about customers. The relation schema is

***Customer-schema* = (*customer-name*, *customer-street*, *customer-city*)**

Figure 3.4 shows a sample relation *customer* (*Customer-schema*). Note that we have omitted the *customer-id* attribute, which we used Chapter 2, because now we want to have smaller relation schemas in our running example of a bank database. We assume that the customer name uniquely identifies a customer—obviously this may not be true in the real world, but the assumption makes our examples much easier to read.

Customer-name	Customer-street	Customer-city
Adams	Spring	Pitsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sandhill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
johnson	Alma	Paloalto
Jones	Main	Harrison

Figure 3.4 The *customer* relation.

Chapter 4
Entity-Relationship Diagram (ERD)

4.1 Introduction

An entity-relationship diagram (ERD) is a data modeling technique that graphically illustrates an information system's entities and the relationships between those entities. An ERD is a conceptual and representational model of data used to represent the entity framework infrastructure.

Steps involved in creating an ERD include:

1. Identifying and defining the entities
2. Determining all interactions between the entities
3. Analyzing the nature of interactions/determining the cardinality of the relationships
4. Creating the ERD

An entity-relationship diagram (ERD) is crucial to creating a good database design. It is used as a high-level logical data model, which is useful in developing a conceptual design for databases.

An entity is a real-world item or concept that exists on its own. Entities are equivalent to database tables in a relational database, with each row of the table representing an instance of that entity.

An attribute of an entity is a particular property that describes the entity. A relationship is the association that describes the interaction between entities. Cardinality, in the context of ERD, is the number of instances of one entity that can, or must, be associated with each instance of another entity. In general, there may be one-to-one, one-to-many, or many-to-many relationships.

For example, let us consider two real-world entities, an employee and his department. An employee has attributes such as an employee number, name, department number, etc. Similarly, department number and name can be defined as attributes of a department. A department can interact with many employees, but an employee can belong to only one department, hence there can be a one-to-many relationship, defined between department and employee.

4.2 Components of E-R Diagram

Entity relational diagram (ER Diagram) is used to represent the requirement analysis at the conceptual design stage. the database is designed from the ER Diagram or we can say that ER Diagram is converted to the database.

Each entity in the ER Diagram corresponds to a table in the database.

The attributes of any an entity correspond to field of a table.

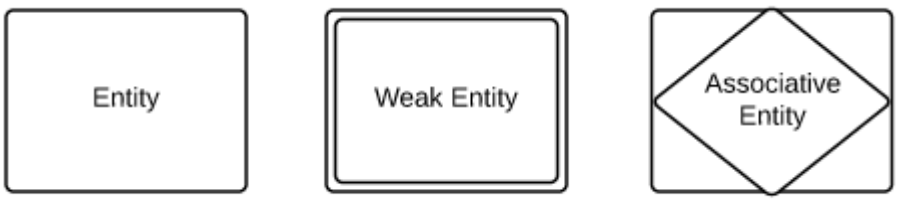
The ER Diagram is converted to the database.

The elements of an ERD are:

1. ENTITIES

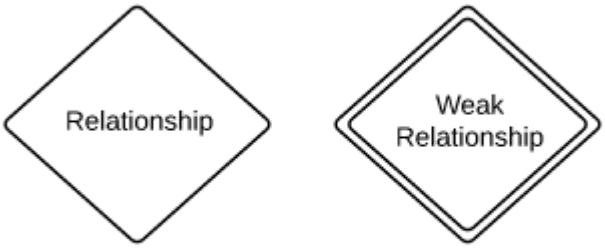
Entities are objects or concepts that represent important data. They are typically nouns, e.g. *customer*, *supervisor*, *location*, or *promotion*.

- **Strong entities** exist independently from other entity types. They always possess one or more attributes that uniquely distinguish each occurrence of the entity.
- **Weak entities** depend on some other entity type. They don't possess unique attributes (also known as a primary key) and have no meaning in the diagram without depending on another entity. This other entity is known as the owner.
- **Associative entities** are entities that associate the instances of one or more entity types. They also contain attributes that are unique to the relationship between those entity instances.



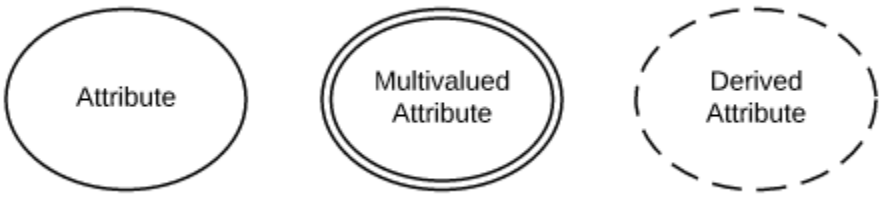
2. RELATIONSHIPS

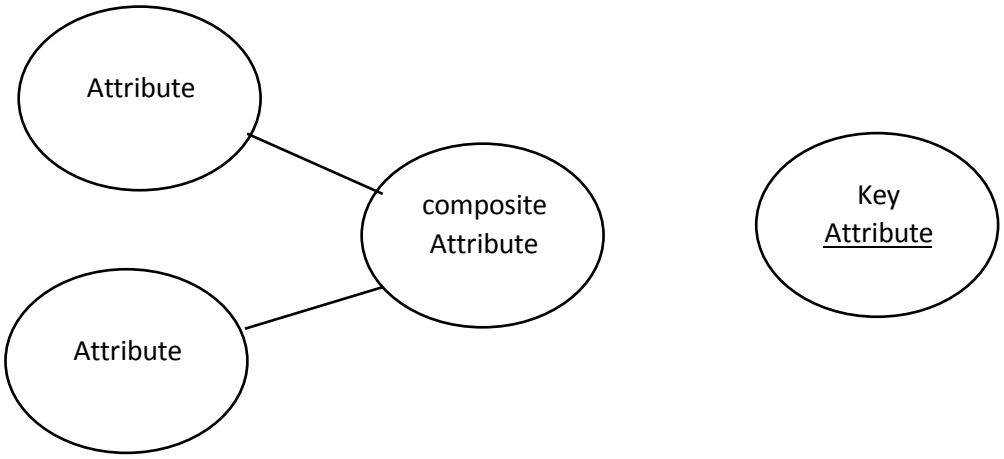
- **Relationships** are meaningful associations between or among entities. They are usually verbs, e.g. *assign*, *associate*, or *track*. A relationship provides useful information that could not be discerned with just the entity types.
- **Weak relationships**, or identifying relationships, are connections that exist between a weak entity type and its owner.
- **Ternary Relationship,** Relationship of degree three.



3. ATTRIBUTES

- **Attributes** are characteristics of either an entity, a many-to-many relationship, or a one-to-one relationship.
- **Multivalued attributes** are those that are capable of taking on more than one value.
- **Derived attributes** are attributes whose value can be calculated from related attribute values.
- **Composite** attributes are represented by ellipses that are connected with an ellipse. they are further divided in a tree like structure. Every node is then connected to its attribute
- **Key** attribute represents the main characteristic of an Entity. It is used to represent Primary key. Ellipse with underlying lines represent Key Attribute.



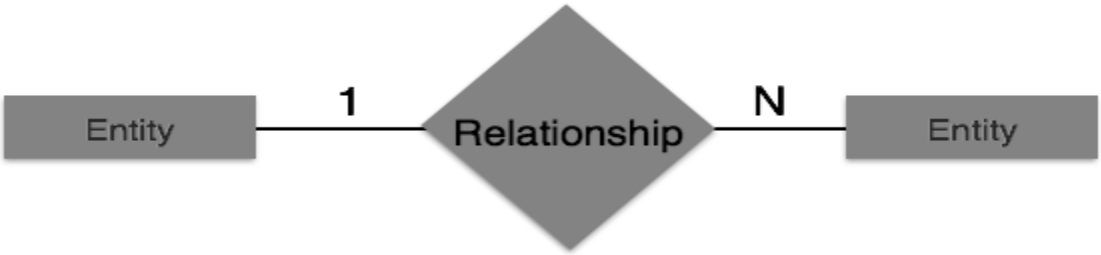


4.3 Binary Relationship and Cardinality

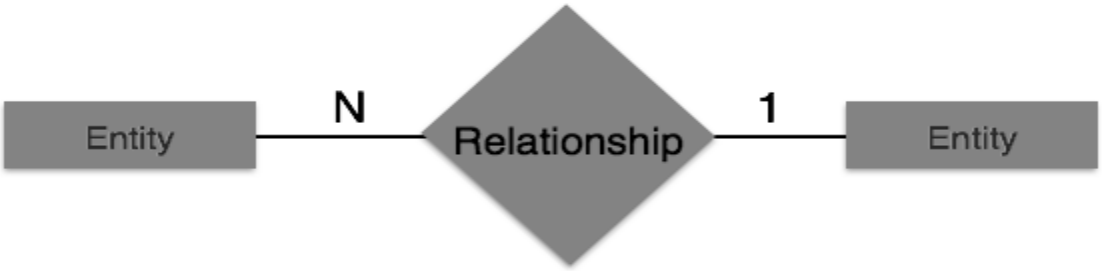
A relationship where two entities are participating is called a **binary relationship**. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

- **One-to-one** − When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.

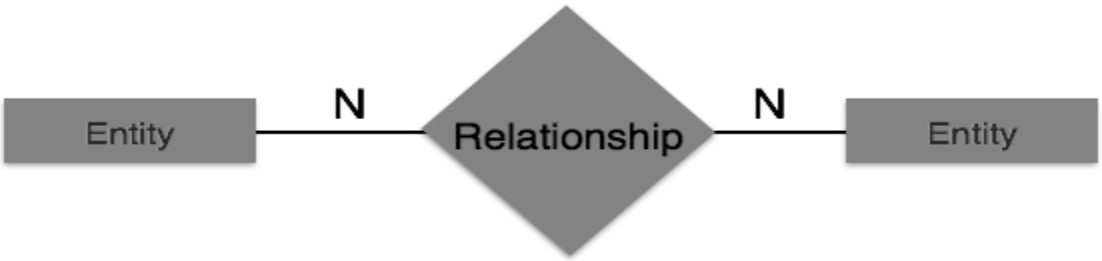
- **One-to-many** − When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.



- **Many-to-one** − When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.



- **Many-to-many** − The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.



Consider the entity-relationship diagram in Figure 4.1, 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 Figure
4.1, 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 either 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*.

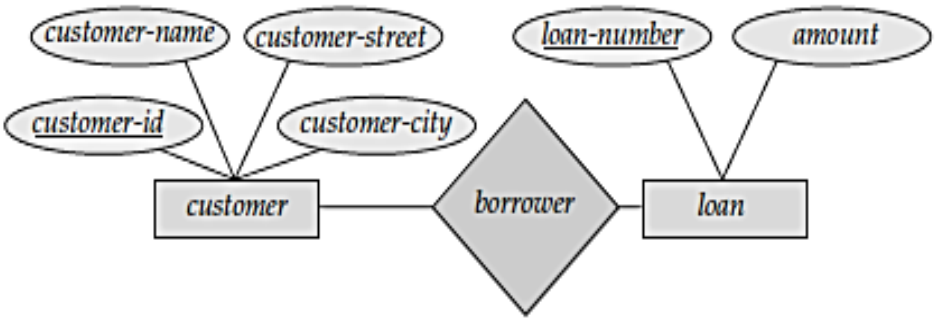


Figure 4.1 E-R diagram corresponding to customers and loans.

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*.
Returning to the E-R diagram of Figure 4.1, we see that the relationship set *borrower* is many-to-many. If the relationship set *borrower* were one-to-many, from *customer* to *loan*, then the line from *borrower* to *customer* would be directed, with an arrow pointing to the *customer* entity set (Figure 4.2a). Similarly, if the relationship set *borrower*

were many-to-one from *customer* to *loan*, then the line from *borrower* to *loan* would have an arrow pointing to the *loan* entity set (Figure 4.2b). Finally, if the relationship set *borrower* were one-to-one, then both lines from *borrower* would have arrows:

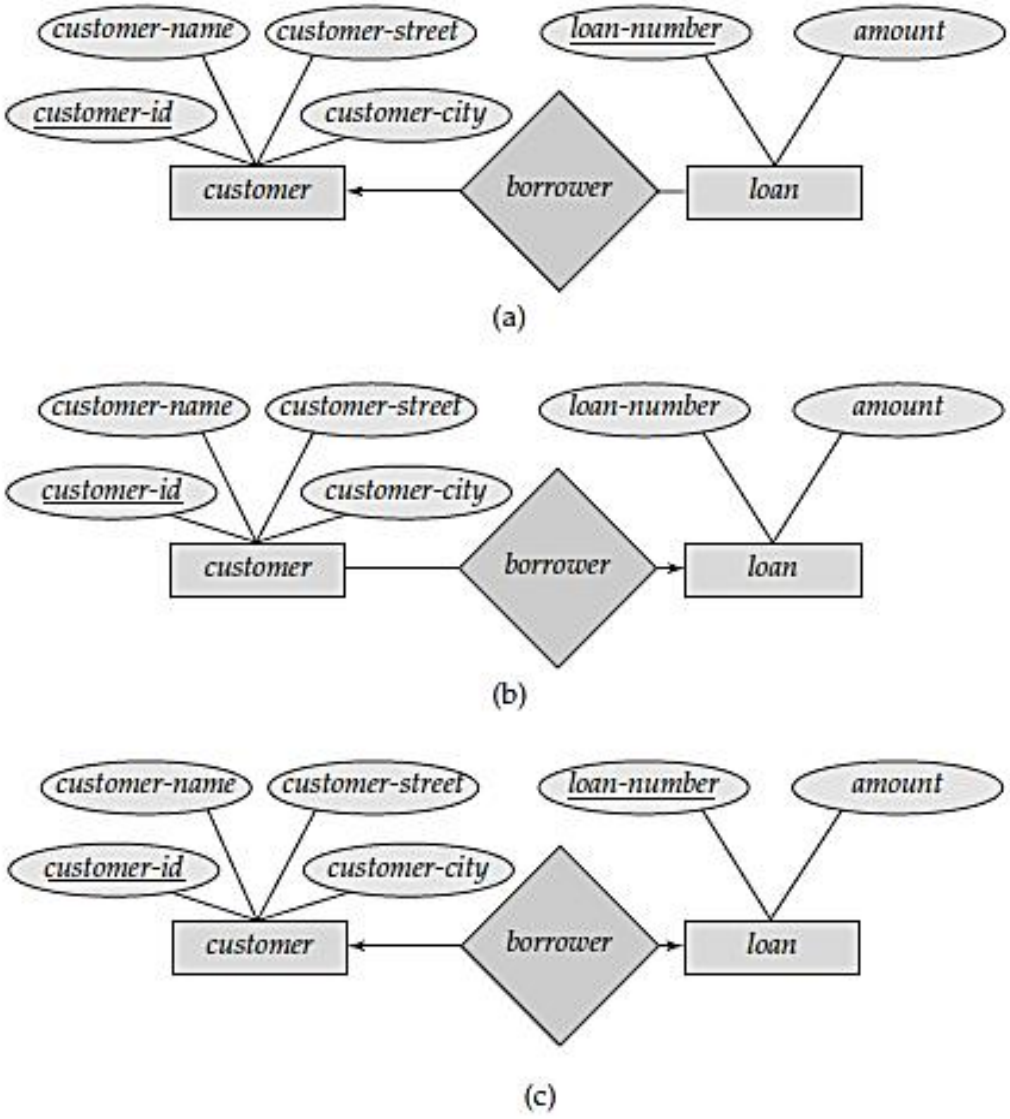


Figure 4.2 Relationships. (a) one to many. (b) many to one. (c) one-to-one.

Figure 4.3 also illustrates a multivalued attribute *phone-number*, depicted by a double ellipse, and a derived attribute *age*, depicted by a dashed ellipse.

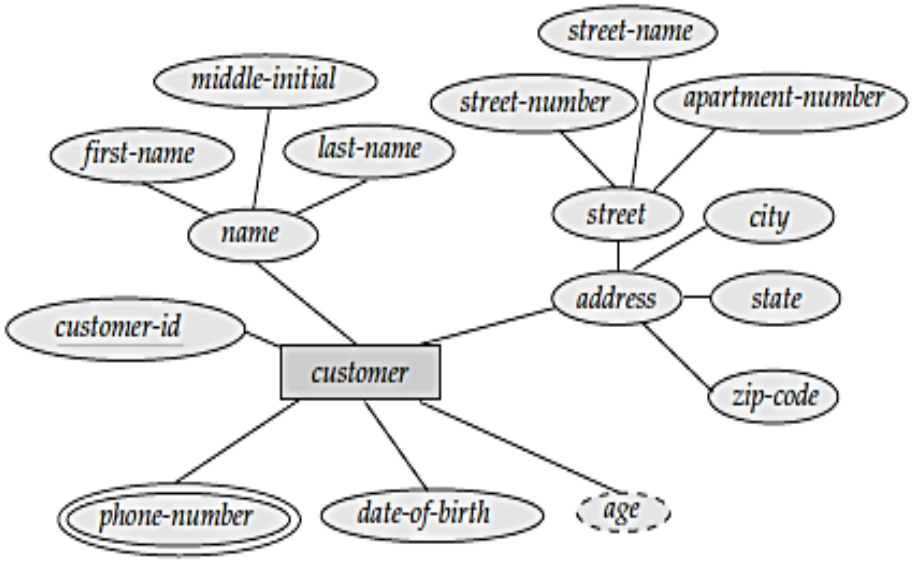


Figure 4.3 E-R diagram with composite, multivalued, and derived attributes.

Nonbinary relationship sets can be specified easily in an E-R diagram. Figure 4.4 consists of the three entity sets *employee*, *job*, and *branch*, related through the relationship set *works-on*.

We can specify some types of many-to-one relationships in the case of nonbinary relationship sets. Suppose an employee can have at most one job in each branch (for example, Jones cannot be a manager and an auditor at the same branch). This constraint can be specified by an arrow pointing to *job* on the edge from *works-on*.

We permit at most one arrow out of a relationship set, since an E-R diagram with two or more arrows out of a nonbinary relationship set can be interpreted in two ways.

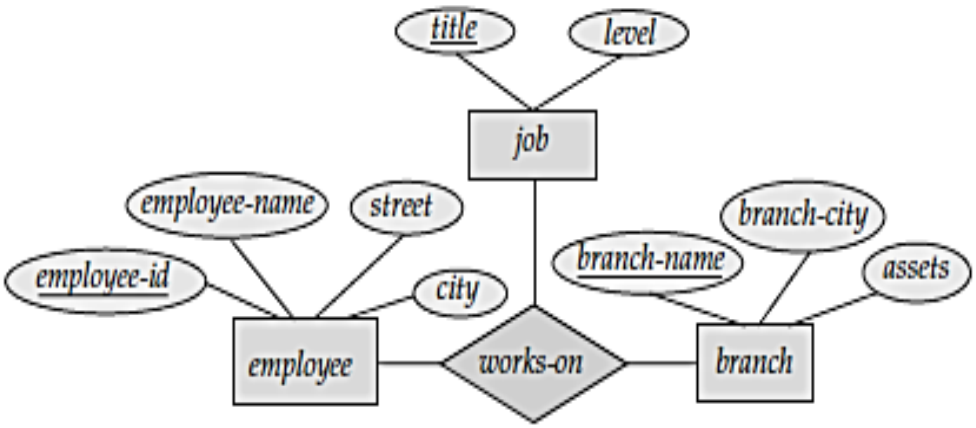


Figure 4.4 E-R diagram with a ternary relationship.

4.5 Reduction of an E-R Schema to Tables

We can represent a database that conforms to an E-R database schema by a collection of tables. For each entity set and for each relationship set in the database, there is a unique table to which we assign the name of the corresponding entity set or relationship set. Each table has multiple columns, each of which has a unique name.

Both the E-R model and the relational-database model are abstract, logical representations of real-world enterprises. Because the two models employ similar design principles, we can convert an E-R design into a relational design. Converting a database representation from an E-R diagram to a table format is the way we arrive at a relational-database design from an E-R diagram. Although important differences exist between a relation and a table, informally, a relation can be considered to be a table of values. The constraints specified in an E-R diagram, such as primary keys and cardinality constraints, are mapped to constraints on the tables generated from the E-R diagram.

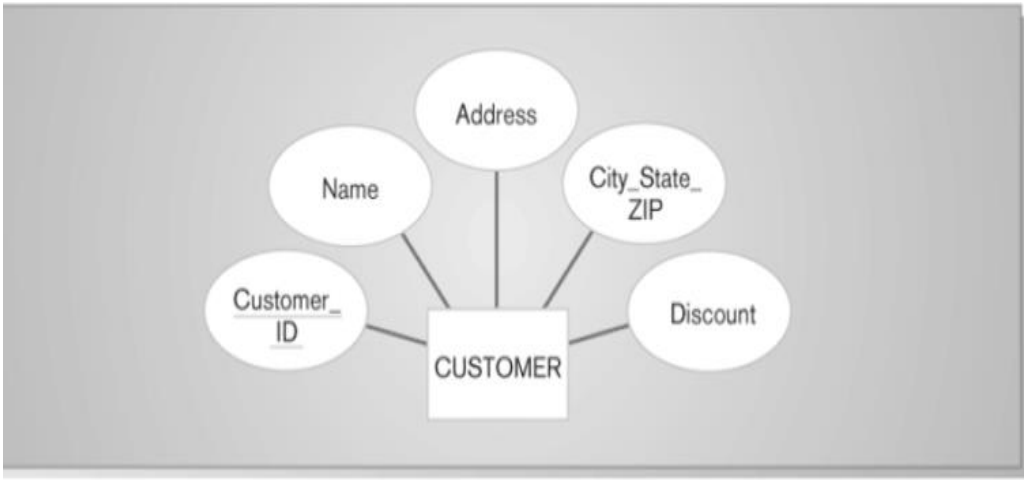
Example :

There is an entity:

customer-schema=(customer-id,name,address,city-state-ZIP,discount).

1.Transforming an entity to a relation – E/R Diagram.

2.Transforming an entity to a relation – relational .



CUSTOMER

Customer –ID	Name	Address	City –State-Zip	Discount
1273	Contemporary Designs	123 Oak St.	Austin,TX2888	5%
6390	Casual Comer	18 Hoosier Dr.	Bloomington ,IN5482	3%

Chapter 5
Normalization

1. Introduction
2. First normal form 1NF
3. Second normal form 2NF
4. Third normal form 3NF

5.1 What is Normalization?

Normalization is a *process* in which we systematically examine relations for *anomalies* and, when detected, remove those anomalies by splitting up the relation into two new, related, relations.

Normalization is an important part of the database development process: Often during normalization, the database designers get their first real look into how the data are going to interact in the database.

Finding problems with the database structure at this stage is strongly preferred to finding problems further along in the development process because at this point it is fairly easy to cycle back to the conceptual model (Entity Relationship model) and make changes.

Normalization can also be thought of as a trade-off between data redundancy and performance. Normalizing a relation reduces data redundancy but introduces the need for joins when all of the data is required by an application such as a report query.

Recall, the Relational Model consists of the elements: relations, which are made up of attributes.

A **relation** is a set of attributes with values for each attribute such that:

a. Each attribute (column) value must be a single value only.
b. All values for a given attribute (column) must be of the same data type.
c. Each attribute (column) name must be unique.
d. The order of attributes (columns) is insignificant
e. No two tuples (rows) in a relation can be identical.
f. The order of the tuples (rows) is insignificant.

Normalization Benefits:

1. Facilitates data integration.
2. Reduces data redundancy.
3. Provides a robust architecture for retrieving and maintaining data.
4. Compliments data modeling.
5. Reduces the chances of data anomalies occurring.

5.2 Problem Without Normalization

Without Normalization, it becomes difficult to handle and update the database, without facing data loss. **Insertion, Updating and Deletion Anomalies** are very frequent if Database is not Normalized. To understand these anomalies let us take an example of **Student** table.

S_id	S_Name	S_Address	Subject_opted
401	Adam	Noida	Bio
402	Alex	Panipat	Maths
403	Stuart	Jammu	Maths
404	Adam	Noida	Physics

- **Updating Anomaly :** To update address of a student who occurs twice or more than twice in a table, we will have to update **S_Address** column in all the rows, else data will become inconsistent.
- **Insertion Anomaly :** Suppose for a new admission, we have a Student id(S_id), name and address of a student but if student has not opted for any subjects yet then we have to insert **NULL** there, leading to Insertion Anamoly.
- **Deletion Anomaly :** If (S_id) 401 has only one subject and temporarily he drops it, when we delete that row, entire student record will be deleted along with it.

5.3 Functional Dependencies

The single most important concept in relational schema design theory is that of a functional dependency.

A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has *n* attributes *A*1, *A*2, ..., *An*.
If we think of the whole database as being described by a single **universal** relation schema *R* = {*A*1, *A*2, ... , *An*}.

A **functional dependency**, denoted by *X* → *Y*, between two sets of attributes *X* and *Y* that are subsets of *R, such that* any two tuples *t*1 and *t*2 in *r* that have *t*1[*X*] = *t*2[*X*], they must also have *t*1[*Y*] = *t*2[*Y*].
This means that the values of the *Y* component of a tuple in *r* depend on, or are *determined by,* the values of the *X* component;
We say that the values of the *X* component of a tuple uniquely (or **functionally**) *determine* the values of the *Y* component.
We say that there is a functional dependency from *X* to *Y*, or that *Y* is **functionally dependent** on *X*.

Functional dependency is represented as **FD** or **f.d.** The set of attributes *X* is called the **left-hand side** of the FD, and *Y* is called the **right-hand side**.
X functionally determines *Y* in a relation schema *R* if, and only if, whenever two tuples of *r*(*R*) agree on their *X*-value, they must necessarily agree on their *Y*-value.
If a constraint on *R* states that there cannot be more than one tuple with a given *X*-value in any relation instance *r*(*R*)—that is, *X* is a **candidate key** of *R*— this implies that *X* →*Y* for any subset of attributes *Y* of *R.*
If *X* is a candidate key of *R*, then *X* →*R.*

If *X*→*Y* in *R*, this does not imply that *Y*→*X* in *R*.
A functional dependency is a property of the **semantics** or **meaning of the attributes**.
Whenever the semantics of two sets of attributes in *R* indicate that a functional dependency should hold, we specify the **dependency as a constraint.**

Example :

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

The following FDs *may hold* because the four tuples in the current extension have no violation of these constraints:
$B \rightarrow C$; $C \rightarrow B$; $\{A, B\} \rightarrow C$; $\{A, B\} \rightarrow D$; and $\{C, D\} \rightarrow B$
However, the following *do not* hold because we already have violations of them in the given extension:
$A \rightarrow B$ (tuples 1 and 2 violate this constraint);
$B \rightarrow A$ (tuples 2 and 3 violate this constraint);
$D \rightarrow C$ (tuples 3 and 4 violate it).

5.3.1 Fully functional dependency (composite key)

If attribute B is functionally dependent on a composite key A but not on any subset of that composite key, the attribute B is fully functionally dependent on A.

5.3.2 Partial Dependency:

When there is a functional dependence in which the determinant is only part of the primary key, then there is a partial dependency.
For example if (A, B) → (C, D) and B→ C and (A, B) is the primary key, then the functional dependence B C is a partial dependency.

5.3.3 Transitive Dependency:

When there are the following functional dependencies such that X→Y, Y→ Z and X is the primary key, then X→Z is a transitive dependency because X determines the value of Z via Y.
Whenever a functional dependency is detected amongst nonprime, there is a transitive dependency.

The advantage of removing transitive dependency is,

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

5.4 Normalization of Relations

The normalization process, as first proposed by Codd (1972a),takes a relation schema through a series of tests to *certify* whether it satisfies a certain **normal form**.

The process, which proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary, can thus be considered as *relational design by analysis.*

Initially, Codd proposed three normal forms, which he called first, second, and third normal form.

5.4.1 How do you divide your tables?

The basic rule is that **each table should describe one type of things, each row in the table should contain about one such thing, and the data we stored for each thing should exist in only one row**.

This can often be sufficient to know. If one follows this basic rule, ones databases will get a good design and one avoids problems with redundancy, things that will not be possible to store, and tables that is hard to understand.

However, sometimes it is difficult to actually know what kind of “things” it is that one would like to store and which data that is related to them. Then we can take use of the theory of normalization.

It helps us to see exactly how different columns within a table are related and shows us how to divide the table to avoid our problems. Therefore we will start looking at the different **normal forms** that the theory of normalization describes. Normal forms are conditions that tables should fulfill. The simplest form is the first normal form and by adding more conditions one can define the second normal form, third normal form and further on.

5.4.2 The First Normal Form (1NF)

A database is in first normal form if it satisfies the following conditions:

1. All the key attributes are defined.
2. There are no repeating groups in the table.
3. The value of record must be atomic.
4. All attributes are dependent on the primary key.

Example1: Consider the following table **stud :**

stud

STU-ID	L-NAME	F-NAME
001	Smith	John
002	Smith	Susan
003	Beal	Fred
004	Thomoson	Marie
005	Tom	Jake
002	Smith	Susan
004	Thomoson	Marie
003	Beal	Fred

To bring this table to first normal form, we delete the duplication row, and now we have the following table stud1:

Stud1

STU-ID	L-NAME	F-NAME
001	Smith	John
002	Smith	Susan
003	Beal	Fred
004	Thomoson	Marie
005	Tom	Jake

Example2: Consider the following table student:

Student

STU-ID	CNAME	GRADE
001	English , Italian	A
002	German , English	B
003	Italian	C

To bring this table to first normal form, we convert data to **atomic value** , and now we have the following table student1:

Student1

STU-ID	CNAME	GRADE
001	English	A
001	Italian	A
002	German	B
002	English	B
003	Italian	C

5.4.3 Second Normal Form (2NF)

A database is in second normal form if it satisfies the following conditions:

- It is in first normal form
- All non-key attributes are fully functional dependent on the primary key

In a table, if attribute B is functionally dependent on A, but is not functionally dependent on a proper subset of A, then B is considered fully functional dependent on A. Hence, in a 2NF table, all non-key

attributes cannot be dependent on a subset of the primary key. Note that if the primary key is not a composite key, all non-key attributes are always fully functional dependent on the primary key.

A table that is in 1st normal form and contains only a single key as the primary key is automatically in 2nd normal form.

Example: Consider the following example:

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

This table has a composite primary key [Customer ID, Store ID]. The non-key attribute is [Purchase Location]. In this case, [Purchase Location] only depends on [Store ID], which is only part of the primary key. Therefore, this table does not satisfy second normal form.

To bring this table to second normal form, we break the table into two tables, and now we have the following:

TABLE_PURCHASE

Customer ID	Store ID
1	1
1	3
2	1
3	2
4	3

TABLE_STORE

Store ID	Purchase Location
1	Los Angeles
2	New York
3	San Francisco

What we have done is to remove the partial functional dependency that we initially had. Now, in the table [TABLE_STORE], the column [Purchase Location] is fully dependent on the primary key of that table, which is [Store ID].

5.4.4 Third Normal Form (3NF)

A database is in third normal form if it satisfies the following conditions:

- It is in second normal form
- There is no transitive functional dependency

By transitive functional dependency, we mean we have the following relationships in the table: A is functionally dependent on B, and B is functionally dependent on C. In this case, C is transitively dependent on A via B.

Consider the following example:

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

In the table able, [Book ID] determines [Genre ID], and [Genre ID] determines [Genre Type].

Therefore, [Book ID] determines [Genre Type] via [Genre ID] and we have transitive functional dependency, and this structure does not satisfy third normal form.

To bring this table to third normal form, we split the table into two as follows:

TABLE_BOOK

Book ID	Genre ID	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

TABLE_GENRE

Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

Now all non-key attributes are fully functional dependent only on the primary key. In [TABLE_BOOK], both [Genre ID] and [Price] are only dependent on [Book ID]. In [TABLE_GENRE], [Genre Type] is only dependent on [Genre ID].

Exercises:

5.1 List all functional dependencies satisfied by the relation of the following Figure:

A	B	C
a1	b1	c1
a1	b1	c2
a2	b1	c1
a2	b1	c3

5.2 List all functional dependencies satisfied by the relation of the following Figure:

Id	Name	Gender	Age
1	Orlando	Male	35
2	John	Male	35
3	Jane	Female	31
4	Jane	Female	30

5.3 Using Normalization convert this table to 1NF,2NF,3NF.

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

5.4 Using Normalization convert this table to 1NF,2NF,3NF.

PRODUCT	SUPPLIER	PRICE	CITY	POPULATION
Cars	Volvo	100000	Torslanda	80000
Cars	SAAB	150000	Södertälje	50000
Trucks	SAAB	400000	Södertälje	50000
Aspirin	Astra	10	Södertälje	50000

5.5 Using Normalization convert this table to 1NF,2NF,3NF.

Street	Zipcode	City	Length
Rydsvagen	58248	Linkoping	19km
Mardtorpsgatan	58248	Linkoping	0.7km
Storgatan	58223	Linkoping	1.5km
Storgatan	64631	Gnesta	0.014km

5.6 Using Normalization convert this table to 1NF,2NF,3NF.

Student	Course-id	Grade	Address
Erik	CIS331	A	80Ericsson Av.
Sven	CIS331	B	12Olafson ST.
Inge	CIS331	C	192Odin Blvd
Hildur	CIS362	A	212 Reyjavik ST.

Chapter 6

Structure Query Language (SQL)

1. Introduction SQL
2. Data Definition Language (DDL)
3. Data Manipulation Language (DML)
4. Data Control Language (DCL)

Structured Query Language(SQL)

6.1 Introduction

Structured Query Language (SQL) is a standard computer language for relational database management and data manipulation. SQL is used to query, insert, update and modify data. Most relational databases support SQL, which is an added benefit for database administrators (DBAs), as they are often required to support databases across several different platforms.

First developed in the early 1970s at IBM by Raymond Boyce and Donald Chamberlin, SQL was commercially released by Relational Software Inc. (now known as Oracle Corporation) in 1979. The current standard SQL version is voluntary, vendor-compliant and monitored by the American National Standards Institute (ANSI). Most major vendors also have proprietary versions that are incorporated and built on ANSI SQL, e.g., SQL*Plus (Oracle), and Transact-SQL (T-SQL) (Microsoft).

One of the most fundamental DBA rites of passage is learning SQL, which begins with writing the first SELECT statement or SQL script without a graphical user interfaces (GUI). Increasingly, relational databases use GUIs for easier database management, and queries can now be simplified with graphical tools, e.g., drag-and-drop wizards. However, learning SQL is imperative because such tools are never as powerful as SQL.

6.2 What can SQL do?

1. SQL can execute queries against a database .
2. SQL can retrieve data from a database .
3. SQL can insert records in a database .
4. SQL can update records in a database.
5. SQL can delete records from a database.
6. SQL can create new databases .
7. SQL can create new tables in a database.
8. SQL can create stored procedures in a database.
9. SQL can create views in a database .
10. SQL can set permissions on tables, procedures, and views .

6.3 Database Languages

A database system provides a **data definition language** to specify the database schema and a **data manipulation language** to express database queries and updates and a **data control language** to configure security access to relational databases . In practice, the data definition and data manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

6.3.1 Data Definition Language (DDL)

The SQL DDL allows specification of not only a set of relations, but also information about each relation, including

- The schema for each relation
- The domain of values associated with each attribute
- The integrity constraints
- The set of indices to be maintained for each relation
- The security and authorization information for each relation
- The physical storage structure of each relation on disk

Data Definition Language (DDL): statements are used to define the database structure or schema. Some examples:

- CREATE - to create objects in the database.
- ALTER - alters the structure of the database.
- DROP - delete objects from the database.
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed.
- COMMENT - add comments to the data dictionary.
- RENAME - rename an object.

6.3.2 Data Manipulation Language(DML)

A **data-manipulation language (DML)** is a language that enables users to access or manipulate data as organized by the appropriate data model.
Data manipulation is
• The retrieval of information stored in the database
• The insertion of new information into the database
• The deletion of information from the database
• The modification of information stored in the database
There are basically two types:

• **Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.
• **Declarative DMLs** (also referred to as **nonprocedural** DMLs) require a user to specify *what* data are needed *without* specifying how to get those data.
Declarative DMLs are usually easier to learn and use than are procedural DMLs.
However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data. The DML component of the SQL language is nonprocedural.
A **query** is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a **query language**. Although technically incorrect, it is common practice to use the terms *query language* and *data manipulation language* synonymously.

Data Manipulation Language (DML) statements are used for managing data within schema objects. Some examples:

- SELECT - retrieve data from the a 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.
- MERGE - UPSERT operation (insert or update).
- CALL - call a PL/SQL or Java subprogram.
- EXPLAIN PLAN - explain access path to data.
- LOCK TABLE - control concurrency.

6.3.3 Data Control Language:

Data Control Language (DCL) :statement is a subset of the Structured Query Language (SQL) that allows database administrators to configure security access to relational databases. Some examples:

- GRANT - gives user's access privileges to database.
- REVOKE - withdraw access privileges given with the GRANT command.

6.4 SQL - Data Types

SQL data type is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL.

You would use these data types while creating your tables. You would choose a particular data type for a table column based on your requirement.

The SQL standard supports a variety of built-in domain types, including:
• **char**(*n*): A fixed-length character string with user-specified length *n*. The full form, **character**, can be used instead.
• **varchar**(*n*): A variable-length character string with user-specified maximum length *n*. The full form, **character varying**, is equivalent.
• **int**: An integer (a finite subset of the integers that is machine dependent). The full form, **integer**, is equivalent.
• **smallint**: A small integer (a machine-dependent subset of the integer domain type).
• **numeric**(*p, d*): A fixed-point number with user-specified precision. The number consists of *p* digits (plus a sign), and *d* of the *p* digits are to the right of the decimal point. Thus, **numeric**(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.
• **real, double precision**: Floating-point and double-precision floating-point numbers with machine-dependent precision.
• **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 day, in hours, minutes, and seconds. A variant, **time**(*p*), can be used to specify the number of fractional digits for

seconds (the default being 0). It is also possible to store time zone information along with the time.

• **timestamp**: A combination of **date** and **time**. A variant, **timestamp**(*p*), can be used to specify the number of fractional digits for seconds (the default here being 6).

Date and time values can be specified like this:

date '2001-04-25'

time '09:30:00'

timestamp '2001-04-25 10:29:01.45'

Dates must be specified in the format year followed by month followed by day, as shown. The seconds field of **time** or **timestamp** can have a fractional part, as in the timestamp above. We can use an expression of the form **cast** *e* **as** *t* to convert a character string (or string valued expression) *e* to the type *t*, where *t* is one of **date, time**,

or **timestamp**. The string must be in the appropriate format as illustrated at the beginning of this paragraph.

To extract individual fields of a **date** or **time** value *d*, we can use **extract** (*field* **from** *d*), where *field* can be one of **year, month, day, hour, minute**, or **second.**

6.4.1 SQL Data Type Quick Reference

However, different databases offer different choices for the data type definition.

The following table shows some of the common names of data types between the various database platforms:

Data type	Access	SQL Server	Oracle	MySQL	PostgreSQL
Boolean	Yes/No	Bit	Byte	N/A	Boolean
Integer	Number (integer)	Int	Number	Int Integer	Int Integer
Float	Number (single)	Float Real	Number	Float	Numeric
Currency	Currency	Money	N/A	N/A	Money
string (fixed)	N/A	Char	Char	Char	Char
string (variable)	Text (<256) Memo (65k+)	Varchar	Varchar Varchar2	Varchar	Varchar
binary object	OLE Object Memo	Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB)	Long Raw	Blob Text	Binary Varbinary

Note: Data types might have different names in different database. And even if the name is the same, the size and other details may be different! **Always check the documentation!**

6.5 Data Definition Language (DDL) command

6.5.1. Create Command

create is a DDL command used to create a table or a database.

6.5.1.1 Creating a Database

To create a database in RDBMS, *create* command is uses. Following is the Syntax,

```
create database database-name;
```

Example for Creating Database

```
create database Test;
```

The above command will create a database named **Test**.

6.5.1.2 Creating a Table

create command is also used to create a table. We can specify names and data types of various columns along. Following is the Syntax,

```
create table table-name
{
 column-name1 datatype1,
 column-name2 datatype2,
 column-name3 datatype3,
 column-name4 datatype4
};
```

create table command will tell the database system to create a new table with given table name and column information.

Example for creating Table

```
create table Student(id int, name varchar, age int);
```

The above command will create a new table **Student** in database system with 3 columns, namely id, name and age.

6.5.2 Alter command

alter command is used for alteration of table structures. There are various uses of *alter* command, such as,

- to add a column to existing table

- to rename any existing column
- to change data type of any column or to modify its size.
- *alter* is also used to drop a column.

6.5.2.1 To Add Column to existing Table

Using alter command we can add a column to an existing table. Following is the Syntax,

```
alter table table-name add(column-name datatype);
```

Here is an Example for this,

```
alter table Student add(address char);
```

The above command will add a new column *address* to the **Student** table

6.5.2.2 To Add Multiple Column to existing Table

Using alter command we can even add multiple columns to an existing table. Following is the Syntax,

```
alter table table-name add(column-name1 datatype1, column-name2 datatype2, column-name3 datatype3);
```

Here is an Example for this,

```
alter table Student add(father-name varchar(60), mother-name varchar(60), dob date);
```

The above command will add three new columns to the **Student** table

6.5.2.3 To Add column with Default Value

alter command can add a new column to an existing table with default values. Following is the Syntax,

```
alter table table-name add(column-name1 datatype1 default data);
```

Here is an Example for this,

```
alter table Student add(dob date default '1-Jan-99');
```

The above command will add a new column with default value to the **Student** table

6.5.2.4 To Modify an existing Column

alter command is used to modify data type of an existing column . Following is the Syntax,

```
alter table table-name modify(column-name datatype);
```

Here is an Example for this,

```
alter table Student modify(address varchar(30));
```

The above command will modify *address* column of the **Student table**

6.5.2.5 To Rename a column

Using alter command you can rename an existing column. Following is the Syntax,

```
alter table table-name rename old-column-name to column-name;
```

Here is an Example for this,

```
alter table Student rename address to Location;
```

The above command will rename *address* column to *Location*.

6.5.2.6 To Drop a Column

alter command is also used to drop columns also. Following is the Syntax,

```
alter table table-name drop(column-name);
```

Here is an Example for this,

```
alter table Student drop(address);
```

The above command will drop *address* column from the **Student table**

6.5.3 Truncate Command

truncate command removes all records from a table. But this command will not destroy the table's structure. When we apply truncate command on a table its Primary key is initialized. Following is its Syntax,

```
truncate table table-name
```

Here is an Example explaining it.

```
truncate table Student;
```

The above query will delete all the records of **Student** table.

truncate command is different from **delete** command. delete command will delete all the rows from a table whereas truncate command re-initializes a table(like a newly created table).

For eg. If you have a table with 10 rows and an auto_increment primary key, if you use *delete* command to delete all the rows, it will delete all the rows, but will not initialize the primary key, hence if you will insert any row after using delete command, the auto_increment primary key will start from 11. But in case of *truncate*command, primary key is re-initialized.

6.5.4 Drop command

drop query completely removes a table from database. This command will also destroy the table structure. Following is its Syntax,

```
drop table table-name
```

Here is an Example explaining it.

```
drop table Student;
```

The above query will delete the **Student** table completely. It can also be used on Databases. For Example, to drop a database,

```
drop database Test;
```

The above query will drop a database named **Test** from the system.

6.5.5 Rename query

rename command is used to rename a table. Following is its Syntax,

```
rename table old-table-name to new-table-name
```

Here is an Example explaining it.

```
rename table Student to Student-record;
```

The above query will rename **Student** table to **Student-record**.

6.6 Data Manipulation Language(DML) command

Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

6.6.1) INSERT command

Insert command is used to insert data into a table. Following is its general syntax,

```
INSERT into table-name values(data1,data2,..)
```

Lets see an example,

Consider a table **Student** with following fields.

S_id	S_Name	age

```
INSERT into Student values(101,'Adam',15);
```

The above command will insert a record into **Student** table.

S_id	S_Name	age
101	Adam	15

Example to Insert NULL value to a column

Both the statements below will insert NULL value into **age** column of the Student table.

```
INSERT into Student(id,name) values(102,'Alex');
```

Or,

```
INSERT into Student values(102,'Alex',null);
```

The above command will insert only two column value other column is set to null.

S_id	S_Name	age
101	Adam	15
102	Alex	

Example to Insert Default value to a column

```
INSERT into Student values(103,'Chris',default)
```

S_id	S_Name	age
101	Adam	15
102	Alex	
103	Chris	14

Suppose the **age** column of student table has default value of 14.

Also, if you run the below query, it will insert default value into the age column, whatever the default value may be.

```
INSERT into Student values(103,'Chris')
```

6.6.2) UPDATE command

Update command is used to update a row of a table. Following is its general syntax,

```
UPDATE table-name set column-name = value where condition;
```

Let's see an example,

```
update Student set age=18 where s_id=102;
```

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Chris	14

Example to Update multiple columns

```
UPDATE Student set s_name='Abhi',age=17 where s_id=103;
```

The above command will update two columns of a record.

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

6.6.3) Delete command

Delete command is used to delete data from a table. Delete command can also be used with condition to delete a particular row. Following is its general syntax,

```
DELETE from table-name;
```

Example to Delete all Records from a Table

```
DELETE from Student;
```

The above command will delete all the records from **Student** table.

Example to Delete a particular Record from a Table

Consider the following **Student** table

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

```
DELETE from Student where s_id=103;
```

The above command will delete the record where s_id is 103 from **Student** table.

S_id	S_Name	age
101	Adam	15
102	Alex	18

6.7 WHERE clause

Where clause is used to specify condition while retrieving data from table. *Where* clause is used mostly with *Select*, *Update* and *Delete* query. If condition specified by *where* clause is true then only the result from table is returned.

Syntax for WHERE clause

```
SELECT column-name1,
 column-name2,
 column-name3,
 column-nameN
from table-name WHERE [condition];
```

Example using WHERE clause

Consider a **Student** table,

s_id	s_Name	Age	address
101	Adam	15	Noida
102	Alex	18	Delhi
103	Abhi	17	Rohtak
104	Ankit	22	Panipat

Now we will use a SELECT statement to display data of the table, based on a condition, which we will add to the SELECT query using WHERE clause.

```
SELECT s_id,
 s_name,
 age,
 address
from Student WHERE s_id=101;
```

s_id	s_Name	Age	address
101	Adam	15	Noida

6.8 SELECT Query

Select query is used to retrieve data from a tables. It is the most used SQL query. We can retrieve complete tables, or partial by mentioning conditions using WHERE clause.

Syntax of SELECT Query

```
SELECT column-name1, column-name2, column-name3, column-nameN
from table-name;
```

Example for SELECT Query

Conside the following **Student** table,

S_id	S_Name	Age	address
101	Adam	15	Noida
102	Alex	18	Delhi
103	Abhi	17	Rohtak
104	Ankit	22	Panipat

```
SELECT s_id, s_name, age from Student.
```

The above query will fetch information of s_id, s_name and age column from Student table

S_id	S_Name	Age
101	Adam	15
102	Alex	18
103	Abhi	17
104	Ankit	22

Example to Select all Records from Table

A special character **asterisk** `*` is used to address all the data(belonging to all columns) in a query. *SELECT* statement uses `*` character to retrieve all records from a table.

```
SELECT * from student;
```

The above query will show all the records of Student table, that means it will show complete Student table as result.

S_id	S_Name	Age	address
101	Adam	15	Noida
102	Alex	18	Delhi
103	Abhi	17	Rohtak
104	Ankit	22	Panipat

Example to Select particular Record based on Condition

```
SELECT * from Student WHERE s_name = 'Abhi';
```

103	Abhi	17	Rohtak

Example to Perform Simple Calculations using Select Query

Conside the following **Employee** table.

Eid	Name	Age	Salary
101	Adam	26	5000
102	Ricky	42	8000
103	Abhi	22	10000
104	Rohan	35	5000

```
SELECT eid, name, salary+3000  from Employee;
```

The above command will display a new column in the result, showing 3000 added into existing salaries of the employees.

Eid	Name	salary+3000
101	Adam	8000

102	Ricky	11000
103	Abhi	13000
104	Rohan	8000

6.9 Like clause

Like clause is used as condition in SQL query. **Like** clause compares data with an expression using wildcard operators. It is used to find similar data from the table.

Wildcard operators

There are two wildcard operators that are used in like clause.

- **Percent sign** `%` : represents zero, one or more than one character.
- **Underscore sign** `_` : represents only one character.

Example of LIKE clause

Consider the following **Student** table.

s_id	s_Name	Age
101	Adam	15
102	Alex	18
103	Abhi	17

```
SELECT * from Student where s_name like 'A%';
```

The above query will return all records where **s_name** starts with character 'A'.

s_id	s_Name	Age
101	Adam	15
102	Alex	18
103	Abhi	17

Example:

```
SELECT * from Student where s_name like '_d%';
```

The above query will return all records from **Student** table where **s_name** contain 'd' as second character.

s_id	s_Name	Age
101	Adam	15

Example:

```
SELECT * from Student where s_name like '%x';
```

The above query will return all records from **Student** table where **s_name** contain 'x' as last character.

s_id	s_Name	Age
102	Alex	18

6.9 Order By Clause

Order by clause is used with **Select** statement for arranging retrieved data in sorted order. The **Order by**clause by default sort data in ascending order. To sort data in descending order **DESC** keyword is used with**Order by** clause.

Syntax of Order By

```
SELECT column-list|* from table-name order by asc|desc;
```

Example using Order by

Consider the following **Emp** table,

Eid	Name	Age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

```
SELECT * from Emp order by salary;
```

The above query will return result in ascending order of the **salary**.

Eid	Name	Age	salary
403	Rohan	34	6000
402	Shane	29	8000
405	Tiger	35	8000

401	Anu	22	9000
404	Scott	44	10000

Example of Order by DESC

Consider the **Emp** table described above,

```
SELECT * from Emp order by salary DESC;
```

The above query will return result in descending order of the **salary**.

Eid	Name	age	Salary
404	Scott	44	10000
401	Anu	22	9000
405	Tiger	35	8000
402	Shane	29	8000
403	Rohan	34	6000

6.10 HAVING Clause

having clause is used with SQL Queries to give more precise condition for a statement. It is used to mention condition in Group based SQL functions, just like WHERE clause.

Syntax for having will be,

```
select column_name, function(column_name)
FROM table_name
```

```
WHERE column_name condition
GROUP BY column_name
HAVING function(column_name) condition
```

Example of HAVING Statement

Consider the following **Sale** table.

oid	order_name	previous_balance	customer
11	ord1	2000	Alex
12	ord2	1000	Adam
13	ord3	2000	Abhi
14	ord4	1000	Adam
15	ord5	2000	Alex

Suppose we want to find the customer whose previous_balance sum is more than 3000.

We will use the below SQL query,

```
SELECT *
from sale group customer
having sum(previous_balance) > 3000
```

Result will be,

oid	order_name	previous_balance	customer
11	ord1	2000	Alex

6.11 Distinct keyword

The **distinct** keyword is used with **Select** statement to retrieve unique values from the table. **Distinct** removes all the duplicate records while retrieving from database.

Syntax for DISTINCT Keyword

```
SELECT distinct column-name from table-name;
```

Example

Consider the following **Emp** table.

Eid	Name	Age	Salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	10000
404	Scott	44	10000
405	Tiger	35	8000

```
select distinct salary from Emp;
```

The above query will return only the unique salary from **Emp** table

salary
5000
8000
10000

6.12 AND & OR operator

AND and **OR** operators are used with **Where** clause to make more precise conditions for fetching data from database by combining more than one condition together.

6.12.1 AND operator

AND operator is used to set multiple conditions with *Where* clause.

Example of AND

Consider the following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

```
SELECT * from Emp WHERE salary < 10000 AND age > 25
```

The above query will return records where salary is less than 10000 and age greater than 25.

Eid	Name	Age	Salary
402	Shane	29	8000
405	Tiger	35	9000

6.12.2 OR operator

OR operator is also used to combine multiple conditions with *Where* clause. The only difference between AND and OR is their behavior. When we use AND to combine two or more than two conditions, records satisfying all the condition will be in the result. But in case of OR, at least one condition from the conditions specified must be satisfied by any record to be in the result.

Example of OR

Consider the following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

```
SELECT * from Emp WHERE salary > 10000 OR age > 25
```

The above query will return records where either salary is greater than 10000 or age greater than 25.

402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

6.13 SQL Constraints

SQl Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into following two types,

- **Column level constraints :** limits only column data
- **Table level constraints :** limits whole table data

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

6.13.1 NOT NULL Constraint

NOT NULL constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value. One important point to note about NOT NULL constraint is that it cannot be defined at table level.

Example using NOT NULL constraint

```
CREATE table Student(s_id int NOT NULL, Name varchar(60), Age int);
```

The above query will declare that the **s_id** field of **Student** table will not take NULL value.

6.13.2 UNIQUE Constraint

UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. UNIQUE constraint can be applied at column level or table level.

Example using UNIQUE constraint when creating a Table (Table Level)

```
CREATE table Student(s_id int NOT NULL UNIQUE, Name varchar(60), Ag
e int);
```

The above query will declare that the **s_id** field of **Student** table will only have unique values and won't take NULL value.

Example using UNIQUE constraint after Table is created (Column Level)

```
ALTER table Student add UNIQUE(s_id);
```

The above query specifies that **s_id** field of **Student** table will only have unique value.

6.13.3 Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

Example using PRIMARY KEY constraint at Table Level

```
CREATE table Student (s_id int PRIMARY KEY, Name varchar(60) NOT NU
LL, Age int);
```

The above command will creates a PRIMARY KEY on the `s_id`.

Example using PRIMARY KEY constraint at Column Level

```
ALTER table Student add PRIMARY KEY (s_id);
```

The above command will creates a PRIMARY KEY on the `s_id`.

6.13.4 Foreign Key Constraint

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see it using two table.

Customer_Detail Table :

c_id	Customer_Name	address
101	Adam	Noida
102	Alex	Delhi
103	Stuart	Rohtak

Order_Detail Table :

Order_id	Order_Name	c_id
10	Order1	101
11	Order2	103
12	Order3	102

In **Customer_Detail** table, c_id is the primary key which is set as foreign key in **Order_Detail** table. The value that is entered in c_id which is set as foreign key in **Order_Detail** table must be present

in **Customer_Detail**table where it is set as primary key. This prevents invalid data to be inserted into c_id column of **Order_Detail**table.

Example using FOREIGN KEY constraint at Table Level

```
CREATE table Order_Detail(order_id int PRIMARY KEY,
order_name varchar(60) NOT NULL,
c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id));
```

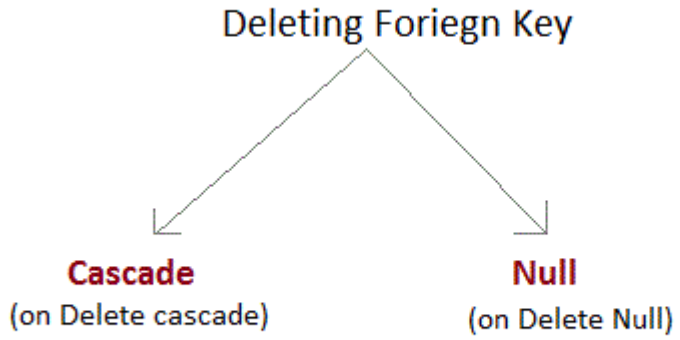
In this query, c_id in table Order_Detail is made as foriegn key, which is a reference of c_id column of Customer_Detail.

Example using FOREIGN KEY constraint at Column Level

```
ALTER table Order_Detail add FOREIGN KEY (c_id) REFERENCES Custome
r_Detail(c_id);
```

Behavior of Foreign Key Column on Delete

There are two ways to maintain the integrity of data in Child table, when a particular record is deleted in main table. When two tables are connected with Foreign key, and certain data in the main table is deleted, for which record exit in child table too, then we must have some mechanism to save the integrity of data in child table.



- **On Delete Cascade :** This will remove the record from child table, if that value of foreign key is deleted from the main table.
- **On Delete Null :** This will set all the values in that record of child table as NULL, for which the value of foreign key is selected from the main table.
- If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.

```
ERROR : Record in child table exist
```

6.13.4 CHECK Constraint

CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

Example using CHECK constraint at Table Level

```
create table Student(s_id int NOT NULL CHECK(s_id > 0),
Name varchar(60) NOT NULL,
Age int);
```

The above query will restrict the s_id value to be greater than zero.

Example using CHECK constraint at Column Level

```
ALTER table Student add CHECK(s_id > 0);
```

6.14 SQL Functions

SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations,

string concatenations, sub-strings etc. SQL functions are divided into two categories,

- Aggregate Functions
- Scalar Functions

6.14.1 Aggregate Functions

These functions return a single value after calculating from a group of values. Following are some frequently used Aggregate functions.

1) AVG()

Average returns average value after calculating from values in a numeric column.

Its general Syntax is,

```
SELECT AVG(column_name) from table_name
```

Example using AVG()

Consider following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find average of salary will be,

```
SELECT avg(salary) from Emp;
```

Result of the above query will be,

avg(salary)
8200

2) COUNT()

Count returns the number of rows present in the table either based on some condition or without condition.

Its general Syntax is,

```
SELECT COUNT(column_name) from table-name
```

Example using COUNT()

Consider following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to count employees, satisfying specified condition is,

```
SELECT COUNT(name) from Emp where salary = 8000;
```

Result of the above query will be,

count(name)
2

Example of COUNT(distinct)

Consider following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query is,

```
SELECT COUNT(distinct salary) from emp;
```

Result of the above query will be,

count(distinct salary)
4

3) FIRST()

First function returns first value of a selected column

Syntax for FIRST function is,

```
SELECT FIRST(column_name) from table-name
```

Example of FIRST()

Consider following **Emp** table

Eid	Name	Age	Salary

401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query

```
SELECT FIRST(salary) from Emp;
```

Result will be,

first(salary)
9000

4) LAST()

LAST return the return last value from selected column

Syntax of LAST function is,

```
SELECT LAST(column_name) from table-name
```

Example of LAST()

Consider following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000

404	Scott	44	10000
405	Tiger	35	8000

SQL query will be,

```
SELECT LAST(salary) from emp;
```

Result of the above query will be,

last(salary)
8000

5) MAX()

MAX function returns maximum value from selected column of the table.

Syntax of MAX function is,

```
SELECT MAX(column_name) from table-name
```

Example of MAX()

Consider following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find Maximum salary is,

```
SELECT MAX(salary) from emp;
```

Result of the above query will be,

MAX(salary)
10000

6) MIN()

MIN function returns minimum value from a selected column of the table.

Syntax for MIN function is,

```
SELECT MIN(column_name) from table-name
```

Example of MIN()

Consider following **Emp** table,

Eid	Name	Age	Salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find minimum salary is,

```
SELECT MIN(salary) from emp;
```

Result will be,

MIN(salary)
8000

7) SUM()

SUM function returns total sum of a selected columns numeric values.

Syntax for SUM is,

```
SELECT SUM(column_name) from table-name
```

Example of SUM()

Consider following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find sum of salaries will be,

```
SELECT SUM(salary) from emp;
```

Result of above query is,

SUM(salary)
41000

6.14.2 Scalar Functions

Scalar functions return a single value from an input value. Following are some frequently used Scalar Functions.

1) UCASE()

UCASE function is used to convert value of string column to Uppercase character.

Syntax of UCASE,

```
SELECT UCASE(column_name) from table-name
```

Example of UCASE()

Consider following **Emp** table

Eid	Name	Age	Salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query for using UCASE is,

```
SELECT UCASE(name) from emp;
```

Result is,

UCASE(name)
ANU
SHANE
ROHAN
SCOTT

TIGER

2) LCASE()

LCASE function is used to convert value of string column to Lowecase character.

Syntax for LCASE is,

```
SELECT LCASE(column_name) from table-name
```

Example of LCASE()

Consider following **Emp** table

Eid	Name	Age	Salary
401	anu	22	9000
402	shane	29	8000
403	rohan	34	6000
404	scott	44	10000
405	Tiger	35	8000

SQL query for converting string value to Lower case is,

```
SELECT LCASE(name) from emp;
```

Result will be,

LCASE(name)
anu
shane
rohan

scott
tiger

3) MID()

MID function is used to extract substrings from column values of string type in a table.

Syntax for MID function is,

```
SELECT MID(column_name, start, length) from table-name
```

Example of MID()

Eid	Name	Age	Salary
401	anu	22	9000
402	shane	29	8000
403	rohan	34	6000
404	scott	44	10000
405	Tiger	35	8000

Consider following **Emp** table

SQL query will be,

```
select MID(name,2,2) from emp;
```

Result will come out to be,

MID(name,2,2)
Nu

ha
oh
co
ig

4) ROUND()

ROUND function is used to round a numeric field to number of nearest integer. It is used on Decimal point values. Syntax of Round function is,

```
SELECT ROUND(column_name, decimals) from table-name
```

Example of ROUND()

Consider following **Emp** table

Eid	Name	Age	Salary
401	anu	22	9000.67
402	shane	29	8000.98
403	rohan	34	6000.45
404	scott	44	10000
405	Tiger	35	8000.01

SQL query is,

```
SELECT ROUND(salary) from emp;
```

Result will be,

ROUND(salary)
9001
8001
6000
10000
8000

6.15 Join in SQL

SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. SQL Join is used for combining column from two or more tables by using values common to both tables. **Join** Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table, is**(n-1)** where **n**, is number of tables. A table can also join to itself known as, **Self Join**.

Types of Join

The following are the types of JOIN that we can use in SQL.

- Inner
- Outer
- Left
- Right

6.15.1 Cross JOIN or Cartesian Product

This type of JOIN returns the Cartesian product of rows of from the tables in Join. It will return a table which consists of records which

combines each row from the first table with each row of the second table.

Cross JOIN Syntax is,

```
SELECT column-name-list
from table-name1
CROSS JOIN
table-name2;
```

Example of Cross JOIN

The **class** table,

ID	NAME
1	abhi
2	adam
4	alex

The **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Cross JOIN query will be,

```
SELECT *
 from class,
 cross JOIN class_info;
```

The result table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	1	DELHI
4	alex	1	DELHI
1	abhi	2	MUMBAI
2	adam	2	MUMBAI
4	alex	2	MUMBAI
1	abhi	3	CHENNAI
2	adam	3	CHENNAI
4	alex	3	CHENNAI

6.15.2 INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the query.

Inner Join Syntax is,

```
SELECT column-name-list
from table-name1
INNER JOIN
table-name2
WHERE table-name1.column-name = table-name2.column-name;
```

Example of Inner JOIN

The **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu

The **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Inner JOIN query will be,

```
SELECT * from class, class_info where class.id = class_info.id;
```

The result table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI

6.15.3 Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and same data type present in both the tables to be joined.

Natural Join Syntax is,

```
SELECT *
from table-name1
```

```
NATURAL JOIN
table-name2;
```

Example of Natural JOIN

The **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu

The **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Natural join query will be,

```
SELECT * from class NATURAL JOIN class_info;
```

The result table will look like,

ID	NAME	Address
1	abhi	DELHI
2	adam	MUMBAI
3	alex	CHENNAI

In the above example, both the tables being joined have ID column(same name and same data type), hence the records for which value of ID

matches in both the tables will be the result of Natural Join of these two tables.

6.15.5 Outer JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

- Left Outer Join
- Right Outer Join
- Full Outer Join

6.15.5.1 Left Outer Join

The left outer join returns a result table with the **matched data** of two tables then remaining rows of the **left**table and null for the **right** table's column.

Left Outer Join syntax is,

```
SELECT column-name-list
from table-name1
LEFT OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

Left outer Join Syntax for **Oracle** is,

```
select column-name-list
from table-name1,
table-name2
on table-name1.column-name = table-name2.column-name(+);
```

Example of Left Outer Join

The **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

The **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Left Outer Join query will be,

```
SELECT * FROM class LEFT OUTER JOIN class_info ON (class.id=class_info
.id);
```

The result table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI

4	anu	null	null
5	ashish	null	null

6.15.5.2 Right Outer Join

The right outer join returns a result table with the **matched data** of two tables then remaining rows of the **right table** and null for the **left** table's columns.

Right Outer Join Syntax is,

```
select column-name-list
from table-name1
RIGHT OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

Right outer Join Syntax for **Oracle** is,

```
select column-name-list
from table-name1,
table-name2
on table-name1.column-name(+) = table-name2.column-name;
```

Example of Right Outer Join

The **class** table,

ID	NAME
1	abhi
2	adam
3	alex

4	anu
5	ashish

The **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Right Outer Join query will be,

```
SELECT * FROM class RIGHT OUTER JOIN class_info on (class.id=class_inf
o.id);
```

The result table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
null	null	7	NOIDA
null	null	8	PANIPAT

6.15.3 Full Outer Join

The full outer join returns a result table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table.

Full Outer Join Syntax is,

```
select column-name-list
from table-name1
FULL OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

Example of Full outer join is,

The **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

The **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Full Outer Join query will be like,

```
SELECT * FROM class FULL OUTER JOIN class_info on (class.id=class_info.
id);
```

The result table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	null
5	ashish	null	null
Null	null	7	NOIDA
Null	null	8	PANIPAT

6.16 SQL Alias

Alias is used to give an alias name to a table or a column. This is quite useful in case of large or complex queries. Alias is mainly used for giving a short alias name for a column or a table with complex names.

Syntax of Alias for table names,

```
SELECT column-name
from table-name
as alias-name
```

Following is an Example using Alias,

```
SELECT * from Employee_detail as ed;
```

Alias syntax for columns will be like,

```
SELECT
column-name as alias-name
fromtable-name
```

Example using alias for columns,

```
SELECT customer_id as cid from Emp;
```

Example of Alias in SQL Query

Consider the following two tables,

The **class** table,The **class_info** table,

ID	Name
1	abhi
2	adam
3	alex
4	anu
5	ashish

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Below is the Query to fetch data from both the tables using SQL Alias,

```
SELECT C.id, C.Name, Ci.Address from Class as C, Class_info as Ci where C
.id=Ci.id;
```

Result table look like,

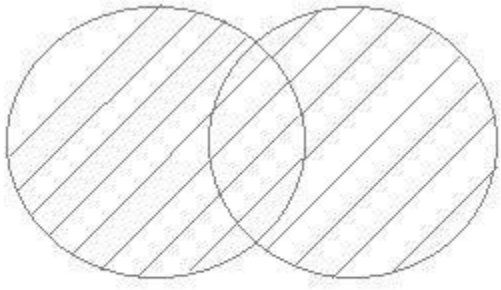
ID	Name	Address
1	abhi	DELHI
2	adam	MUMBAI
3	alex	CHENNAI

6.17 Set Operation in SQL

SQL supports few Set operations to be performed on table data. These are used to get meaningful results from data, under different special conditions.

6.17.1 Union

UNION is used to combine the results of two or more Select statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and data type must be same in both the tables.



Example of UNION

The **First** table,

ID	Name
1	abhi
2	adam

The **Second** table,

ID	Name
2	adam
3	Chester

Union SQL query will be,

```
select * from First
UNION
```

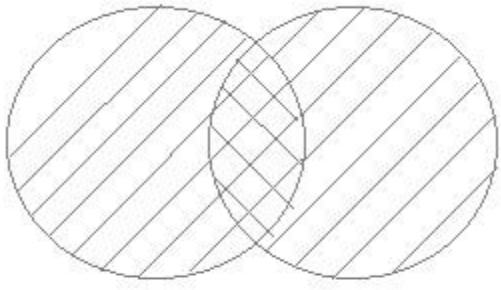
```
select * from second
```

The result table will look like,

ID	NAME
1	abhi
2	adam
3	Chester

Union All

This operation is similar to Union. But it also shows the duplicate rows.



Example of Union All

The **First** table,

ID	NAME
1	abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester

Union All query will be like,

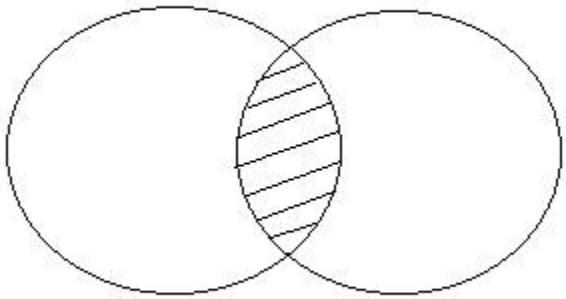
```
select * from First
UNION ALL
select * from second
```

The result table will look like,

ID	NAME
1	abhi
2	adam
2	adam
3	Chester

6.17. Intersect

Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and data type must be same. MySQL does not support INTERSECT operator.



Example of Intersect

The **First** table,

ID	NAME
1	Abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester

Intersect query will be,

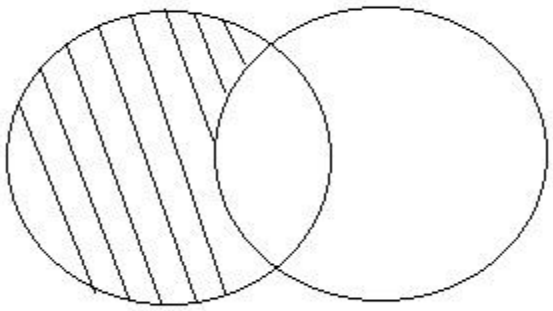
```
select * from First
INTERSECT
select * from second
```

The result table will look like

ID	NAME
2	adam

6.17.4 Minus

Minus operation combines result of two Select statements and return only those result which belongs to first set of result. MySQL does not support INTERSECT operator.



Example of Minus

The **First** table,

ID	NAME
1	Abhi
2	Adam

The **Second** table,

ID	NAME
2	adam
3	Chester

Minus query will be,

```
select * from First
MINUS
select * from second
```

The result table will look like,

ID	NAME
1	Abhi

6.18 SQL Sequence

Sequence is a feature supported by some database systems to produce unique values on demand. Some DBMS like **MySQL** supports AUTO_INCREMENT in place of Sequence. AUTO_INCREMENT is applied on columns, it automatically increments the column value by 1 each time a new record is entered into the table. Sequence is also somewhat similar to AUTO_INCREMENT but its has some extra features.

Creating Sequence

Syntax to create sequences is,

```
CREATE Sequence sequence-name
start with initial-value
increment by increment-value
maxvalue maximum-value
cycle|nocycle
```

initial-value specifies the starting value of the Sequence, **increment-value** is the value by which sequence will be incremented and **maxvalue** specifies the maximum value until which sequence will increment itself.**cycle** specifies that if the maximum value exceeds the set limit, sequence will restart its cycle from the begining. **No cycle** specifies that if sequence exceeds **maxvalue** an error will be thrown.

Example to create Sequence

The sequence query is following

```
CREATE Sequence seq_1
start with 1
increment by 1
```

```
maxvalue 999
cycle ;
```

Example to use Sequence

The **class** table,

ID	NAME
1	abhi
2	adam
4	alex

The sql query will be,

```
INSERT into class value(seq_1.nextval,'anu');
```

Result table will look like,

ID	NAME
1	abhi
2	adam
4	Alex
1	Anu

Once you use `nextval` the sequence will increment even if you don't Insert any record into the table.

6.19 SQL View

A view in SQL is a logical subset of data from one or more tables. View is used to restrict data access.

Syntax for creating a View,

```
CREATE or REPLACE view view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Example of Creating a View

Consider following **Sale** table,

Oid	order_name	previous_balance	Customer
11	ord1	2000	Alex
12	ord2	1000	Adam
13	ord3	2000	Abhi
14	ord4	1000	Adam
15	ord5	2000	Alex

SQL Query to Create View

```
CREATE or REPLACE view sale_view as select * from Sale where custome
r = 'Alex';
```

The data fetched from select statement will be stored in another object called **sale_view**. We can use create seperately and replace too but using both together works better.

Example of Displaying a View

Syntax of displaying a view is similar to fetching data from table using Select statement.

```
SELECT * from sale_view;
```

6.19.1 Force View Creation

force keyword is used while creating a view. This keyword force to create View even if the table does not exist. After creating a force View if we create the base table and enter values in it, the view will be automatically updated.

Syntax for forced View is,

```
CREATE or REPLACE force view view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

6.19.2 Update a View

Update command for view is same as for tables.

Syntax to Update a View is,

```
UPDATE view-name
set value
WHERE condition;
```

If we update a view it also updates base table data automatically.

6.19.3 Read-Only View

We can create a view with read-only option to restrict access to the view.

Syntax to create a view with Read-Only Access

```
CREATE or REPLACE force view view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition with read-only
```

The above syntax will create view for read-only purpose, we cannot Update or Insert data into read-only view. It will throw an error.

Types of View

There are two types of view,

- Simple View
- Complex View

Simple View	Complex View
Created from one table	Created from one or more table
Does not contain functions	Contain functions
Does not contain groups of data	Contains groups of data