

UNIT 1: Introduction to Database Management System

Introduction, Characteristics of the Database Approach, Advantages of Using DBMS Approach. Transaction management and Structure of a DBMS, Database System Concepts and Architecture: Data Models, Schemas, and Instances, The 3-level architecture of DBMS – Hierarchical, Network, and Relational Model and Data Independence, Database Languages and Interfaces, The Database System Environment, Centralized and Client-Server Architectures, Classification of Database Management Systems, Queries in DBMS.

INTRODUCTION TO DBMS

Databases and database systems are very important part of real life of modern world. Most of the people are interact with database for their day to day activities. For example people are interacting with bank database whenever they are using ATM for money transactions. People are interacting with railway reservation database whenever they are booking and cancelling railway ticket. We interact with database system whenever we purchase something on online, airline reservation, hotel reservation, college admission etc.

Database system plays a great role on the growing use of computers. It play a major role in almost all the fields in which computers are used like business, e-commerce, medicine, engineering, law, education, library science, agriculture, aeronautics, arts etc. Let us discuss some of the common terms used in database.

BASICS OF DATABASE

Q. What is Data, Database and Database Management system?

Data: Data is known facts or any raw facts, figures, statistics that can be recorded. Data may be in the form of numbers, characters, multimedia pictures, sound messages etc. (e.g. 1, ABC, 19 etc).

Information: The processed data is called information. : if we organize the above data (e.g. 1, ABC, 19 etc). in the following way, then they collectively represent meaningful.

Example

Roll	Name	Age
1	ABC	19



Database: A database is a **collection of logically related data** that can be recorded and have an implicit meaning.

- It is a collection of interrelated data.
- These can be stored in the form of tables.
- A database can be of any size and varying complexity.
- A database may be generated and manipulated manually or it may be computerized.

Ex: Student database, employee database etc.

1. employee database : (only one table)
 - a. EMP (EmpNo: int, name: string, dob: date, PhNo: int)
2. Student database: (Contains 4 Tables)
 - a. Student (snum: integer, sname: string, major: string, level: string, age: integer)
 - b. Class (name: string, meets at: string, room: string, fid: integer)
 - c. Enrolled (snum: integer, cname: string)
 - d. Faculty (fid: integer, fname: string, deptid: integer)

Data Base Management System (DBMS): It is a **collection of programs that enables user to create and maintain a database**. In other words it is general-purpose software that provides the users with the processes of **defining, constructing, maintaining and manipulating the database for various applications**. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

EX: UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment.

Database Applications:

- Banking: all transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Online retailers: order tracking, customized recommendations
- Manufacturing: production, inventory, orders, supply chain

- Human resources: employee records, salaries, tax deductions

Q. Explain the applications of Database

- *Sales*: For customer, product, and purchase information.
- *Accounting*: For payments, receipts, account balances, assets and other accounting information.
- *Human resources*: For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
- *Manufacturing*: For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.
- *Online retailers*: For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations.
- *Banking*: For customer information, accounts, loans, and banking transactions.
- *Credit card transactions*: For purchases on credit cards and generation of monthly statements.
- *Finance*: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.
- *Universities*: For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).
- *Airlines*: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.

Q. Discuss the main characteristics of the database approach and how it is better from traditional file system?

Characteristics of Database Approach:

Self-describing nature of a database system

- A DBMS catalog stores the description of a particular database (e.g. data structures, types, and constraints)
- The description is called meta-data.
- This allows the DBMS software to work with different database applications.

Insulation between programs and data / Program-data independence:

- Program-data independence.
- Allows changing data structures and storage organization without having to change the DBMS access programs.

Data Abstraction:

- A data model is used to hide storage and implementation details and present the users with a conceptual view of the database.
- Programs refer to the data model constructs rather than data storage details

Support of multiple views of the data:

- Each user may see a different view of the database, which describes only the data of interest to that user.

Sharing of data and multi-user transaction processing:

- Allowing a set of concurrent users to retrieve from and to update the database.
- Concurrency control within the DBMS guarantees that each transaction is correctly executed or aborted
- Recovery subsystem ensures each completed transaction has its effect permanently recorded in the database
- OLTP (Online Transaction Processing) is a major part of database applications. This allows hundreds of concurrent transactions to execute per second correctly and efficiently.

Security of data:

- In a centralized database system, the privilege of modifying the database is not given to everyone. This is given only to database administrator.
- The DBA has full control over the database and he can implement security by placing restrictions on the data. Based on the permissions granted to them, the user can add, modify or query data.

For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about the customer account.

Q. What are the advantages of using DBMS briefly explain all the advantages?

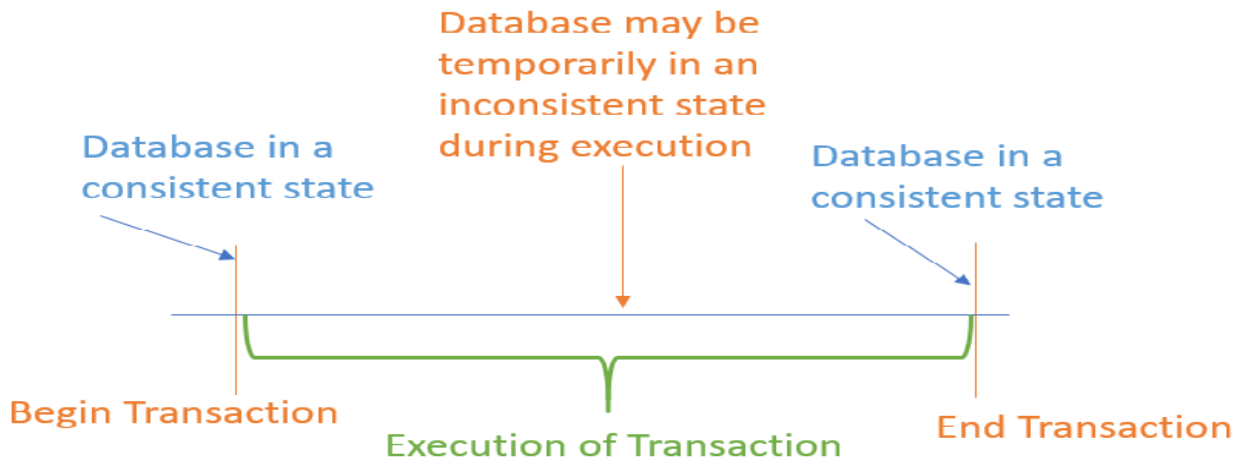
Advantages or capabilities of using DBMS:

- **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*.
- **Efficient data access:** A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently.
- **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.
- **Restricting unauthorized access :** When several users share the data, centralizing the administration of data can offer significant improvements.
 - Experienced professionals who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and for fine-tuning the storage of the data to make retrieval efficient.
- **Concurrent access:** A 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.

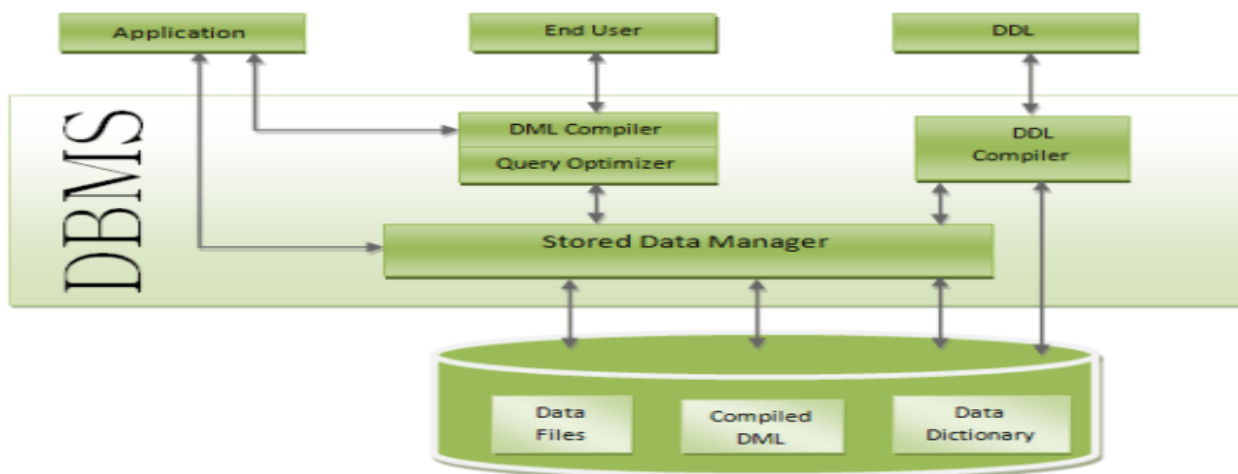
- **Providing backup and recovery:** A DBMS must provide facilities for recovering from hardware and software failures. The backup recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update program, the recovery subsystem is responsible, for making sure that the database is restored to the state it was in before the program started executing. The recovery subsystem could ensure that the program is resumed from the point at which it was interrupted so that its full effect is recorded in the database.
- **Reduced application development time:** Clearly, the DBMS supports many important functions that are common to many applications accessing data stored.
- **Controlling redundancy:** Redundancy is storing the same data multiple times leads to many problems. In the database approach, entire data is stored in a single location in a single database. Data redundancy can be avoided by using different constraints on the data. This doesn't permit any inconsistency and it saves storage space and manpower.
- **Permitting inference and actions using rules:** Database systems provide capabilities for defining deduction rules for inference new information from the stored database facts. Such systems are called deductive database system.
- **Permitting Multiple user interface:** Many types of users with varying levels of technical knowledge use a database. Hence a database should provide a variety of user interfaces. These include
 - query language for casual user
 - programming language interfaces for application programmers
 - forms and command for parametric users
 - menu driven interfaces and natural language interfaces for standalone users

Transaction management:

Transaction management is **a logical unit of processing** in a DBMS which entails one or more database access operation. It is a transaction is a program unit whose execution may or may not change the contents of a database.



DBMS STRUCTURE:



- **Applications:** – It can be considered as a user-friendly web page where the user enters the requests. Here he simply enters the details that he needs and presses buttons to get the data.
- **End User:** – They are the real users of the database. They can be developers, designers, administrators, or the actual users of the database.
- **DDL:** – Data Definition Language (DDL) is a query fired to create database, schema, tables, mappings, etc in the database. These are the commands used to create objects

like tables, indexes in the database for the first time. In other words, they create the structure of the database.

- **DDL Compiler:** – This part of the database is responsible for processing the DDL commands. That means this compiler actually breaks down the command into machine-understandable codes. It is also responsible for storing the metadata information like table name, space used by it, number of columns in it, mapping information, etc.
- **DML Compiler:** – When the user inserts, deletes, updates or retrieves the record from the database, he will be sending requests which he understands by pressing some buttons. But for the database to work/understand the request, it should be broken down to object code. This is done by this compiler.
- **Query Optimizer:** – When a user fires some requests, he is least bothered how it will be fired on the database. He is not all aware of the database or its way of performance. But whatever be the request, it should be efficient enough to fetch, insert, update, or delete the data from the database. The query optimizer decides the best way to execute the user request which is received from the DML compiler.
- **Stored Data Manager:** – This is also known as Database Control System. It is one of the main central systems of the database. It is responsible for various tasks
 - It converts the requests received from query optimizer to machine-understandable form. It makes actual requests inside the database. It is like fetching the exact part of the brain to answer.
 - It helps to maintain consistency and integrity by applying the constraints. That means it does not allow inserting/updating / deleting any data if it has child entry. Similarly, it does not allow entering any duplicate value into database tables.
 - It controls concurrent access. If there are multiple users accessing the database at the same time, it makes sure, all of them see correct data. It guarantees that there is no data loss or data mismatch happens between the transactions of multiple users.
 - It helps to back up the database and recovers data whenever required. Since it is a huge database and when there is any unexpected exploit of the transaction, and reverting the changes is not easy. It maintains the backup of all data so that it can be recovered.

- **Data Files:** – It has the real data stored in it. It can be stored as magnetic tapes, magnetic disks, or optical disks.
- **Compiled DML:** – Some of the processed DML statements (insert, update, delete) are stored in it so that if there are similar requests, it will be re-used.
- **Data Dictionary:** – It contains all the information about the database. As the name suggests, it is the dictionary of all the data items. It contains a description of all the tables, view, materialized views, constraints, indexes, triggers, etc.

DBMS Concepts and System:

Database system: Database and DBMS software together called as database system. If not only contain database but also a complete definition or description of the database structure and constraints. This definition stored in what is called DBMS catalog.

DBMS catalog: The catalog contains information such as the structure of each file, the type and storage format of each data item and various constraints on the data.

Metadata: The information such as structure of each file, the type and storage format of each data items and constraints on the data stored in DBMS catalos is called metadata. It describes the structure of the primary database. It contains **data about data**.

NAME	AGE	GENDER	HEIGHT (CM)
A	20	MALE	172
B	21	MALE	168
C	19	FEMALE	160
D	20	MALE	163

- Data dictionary: A **Data Dictionary** is a collection of names, definitions, and attributes about **data** elements that are being used or captured in a **database**,
 - A **Data Dictionary** also provides metadata about **data** elements.

Data Dictionary

Data Dictionary outlining a Database on Driver Details in NSW

Field Name	Data Type	Data Format	Field Size	Description	Example
License ID	Integer	NNNNNN	6	Unique number ID for all drivers	12345
Surname	Text		20	Surname for Driver	Jones
First Name	Text		20	First Name for Driver	Arnold
Address	Text		50	First Name for Driver	11 Rocky st Como 2233
Phone No.	Text		10	License holders contact number	0400111222
D.O.B	Date / Time	DD/MM/YYYY	10	Drivers Date of Birth	08/05/1956

A field is one item of information in a database record e.g. in an the below table, it is ROLL, NAME & AGE.

Roll	Name	Age
1	ABC	19
2	DEF	22
3	XYZ	28

A record consists of all the fields about an individual entry in a database e.g. all the details about one student in a school management database.

A file is the complete collection of all the records e.g. the complete collection of all the student records.

Table or **Relation**: Collection of related records. The columns of this relation are called **Fields**, **Attributes** or **Domains**. The rows are called **Tuples** or **Records**.

Roll	Name	Age
1	ABC	19
2	DEF	22
3	XYZ	28

Q. Database system concepts and architecture

Database Instances: Database 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.

Database Schema: The overall design of the database is called the database schema. A schema is a collection of named objects. Schemas provide a logical classification of objects in the database.

Data model: It is a set of concepts that can be used to describe the structure of a database that is to say data types, relationships and constraints.

DETAIL Database system concepts:

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

Schemas and Instances:

The description of the database is called the database schema, which is specified during the database design and is not expected change frequently. Most data models have certain conventions for displaying the schemas as diagrams. A displayed schema is called a schema diagram.

Consider an example for schema diagram:

Student

Regno	Sname	Class	Course	Address
-------	-------	-------	--------	---------

Department

Dept_no	Dept_name	Dept_location
---------	-----------	---------------

Course

Course_no	Course_name	Course_strength
-----------	-------------	-----------------

Employee

Emp_name	Emp_no	Salary	Address	Dob
----------	--------	--------	---------	-----

Instances: The collection of data stored in the database at a particular moment in time called a database instance or database state or snapshot. The actual data in a database may change frequently. For example, if we add a new student record or if we delete a record, then the database changes from one state to another state. When a new database is created its state is empty.

In a

Regno	Sname	Class	Course	Address
1001	Smith	II SEM	BCA	M.G.Road,Bangalore

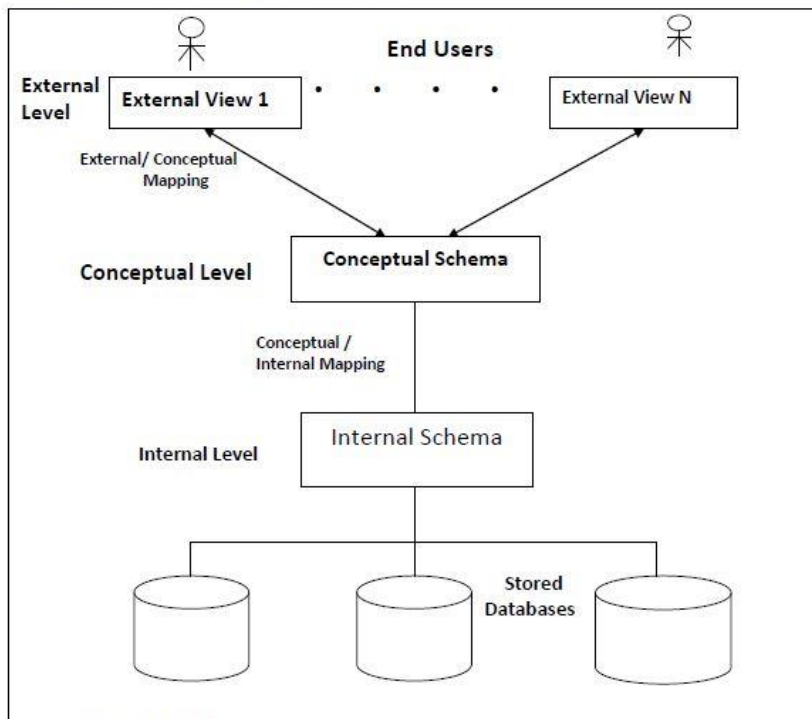
 figure, you see the relation student containing one student record. If a record is added to student table, then the state of the table changed from one to two records. This implies that the instance of the database may change time to time and it is also called as database state.

When a new database is created its state is empty. This means that there is no data in the database. However new records are inserted it's no longer empty.

Q. Explain the three schema architecture is also called ANSI/SPARC architecture or three-level architecture.

- A commonly used view of data approach is the three-level architecture suggested by the ANSI/SPARC (American National Standards Institute/Standards Planning and Requirements Committee). ANSI/SPARC proposed an architectural framework for databases.

The three schema architecture of database:



1. The **External Schema** is the view that the individual user of the database has. This view is often a restricted view of the database and the same database may provide a number of different views for different classes of users.

A **view** is a relation that collects certain information from a set of base relations and separate view relation.

2. The **Conceptual schema** (sometimes called the logical schema) describes the stored data in terms of the data model of the DBMS. In a relational DBMS, the conceptual schema describes all relations that are stored in the database. It hides physical storage details, concentrating upon describing entities, data types, relationships, user operations, and constraints.
3. The **physical schema** specifies additional storage details. Essentially, the physical schema summarizes how the relations described in the conceptual schema are actually stored on secondary storage devices such as disks and tapes. It tells us what data is stored in the database and how.

The DBMS must transform a request specified by the external schema into a request against a conceptual schema and then into a request on the internal schema for processing over the stored database.

The process of transforming requests and result between the three levels of schema is known as mapping.

- **1. External / conceptual mapping:** The mapping between external and conceptual level gives the correspondence between records and relationships of external and conceptual levels.
- **2. Conceptual/ Internal mapping:** The mapping between conceptual and internal levels gives the correspondence between the structure [metadata] and size and position of the data in the stored database.

Data Independence:

There are two types of data independences:

1. Logical data independence
2. Physical data independence

Logical data independence: It is 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 or data item) should not affect the external schema.

Example: Adding a phone number column to employee table. Only the view definition and the mappings are required to be changed in logical data independence. Application programs that reference this external schema construct must work as before, after changing the conceptual schema.

Physical data independence: It is capacity to change the internal schema without having to change conceptual or external schema. Some times to improve the performance of retrieval of data. We may need to change the internal storage structure or access path (Say some files need to be re-organized for efficient retrieval), than external or conceptual schema need not be changed.

Database languages and Interfaces

The DBMS provide appropriate language for each category of users. Some of the DBMS languages are,

Data definition language (DDL): This is used by the database Administrators [DBA's] and by the database designers to define schema objects. The DBMS will have a DDL compiler whose functions is to process DDL statements and generates schema objects [tables, indexes etc]. Which are stored in special file called data dictionary [system catalog]. The DDL is used to specify the conceptual schema only.

E.g. CREATE, DROP, TRUNCATE and ALTER Statements

- **Stored definition language (SDL):** It is used to specify internal schema.

- **View definition language (VDL):** It is used to specify views and their mappings to conceptual schema.

Data manipulation languages (DML): Once the database schemas are compiled, database can be manipulated using the language called DML.

Manipulation of database may include operations such as,

- Retrieval of data
- Insertion of data
- Deleting of data
- Modification of data

E.g. SELECT, DELETE, INSERT and UPDATE statements

Data Control Language (DCL): It is used to manage different transactions occurring within a database.

E.g. GRANT and REVOKE statements

Transaction Control Language (TCL): It is used to manage different transactions occurring within a database.

E.g. COMMIT ROLLBACK and SAV EPOINT statements.

DBMS Interfaces

Interfaces are the programs which enable the user to interact with the DBMS.

The different user-friendly interfaces provided by a DBMS may include the followings:

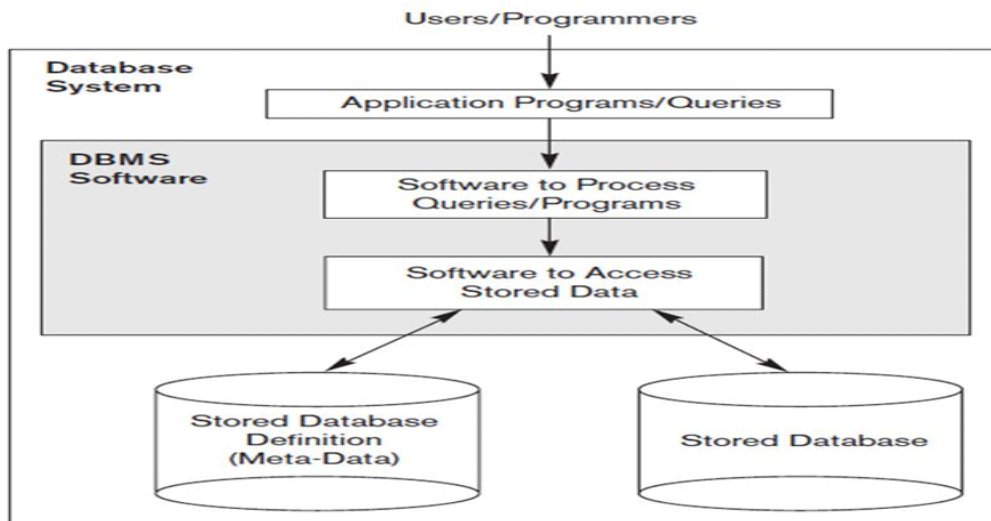
- **Menu based interfaces:** These interfaces presents the user with list of options called menus. Pull down menus are becoming very popular technique in windows based user interfaces; it provides the user to develop their own customized design.
- **Graphical user interfaces:** A GUI typically displays a schema to the user in diagrammatic form. The most GUI uses pointing device, such as mouse to pick certain parts of the displayed schema diagram.
- **Natural language interfaces:** These interfaces accept request written in English or some other language and attempt to “understand” them. This interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing.
- **Interfaces for parametric users:** Parametric users, such as bank tellers and reservation clerks, who are using only limited operation repeatedly, have usually some function keys to repeat the requests. System analysts design a special interface using function keys for specific class of users.

- **Interface for the Database administrator (DBA):** Most database systems contain privileged commands that can be used only by the staff of DBA. These include commands for creating schema objects, for creating new users, granting accounts authorization. **EX:** Grant, revoke etc.

Q. Explain the concept of Database system environment with a neat diagram?

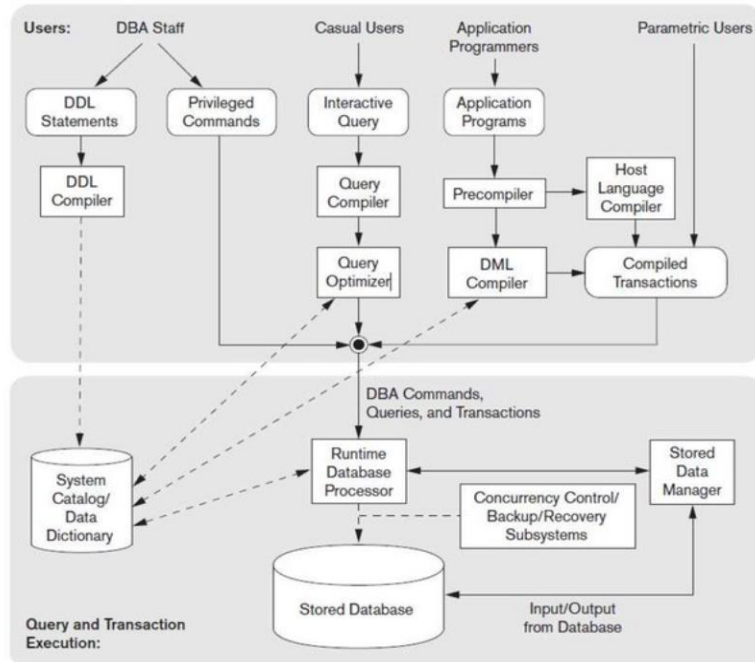
- We can divide the figure into Upper and Lower
 - Upper part – for Users
 - Lower part – For storage

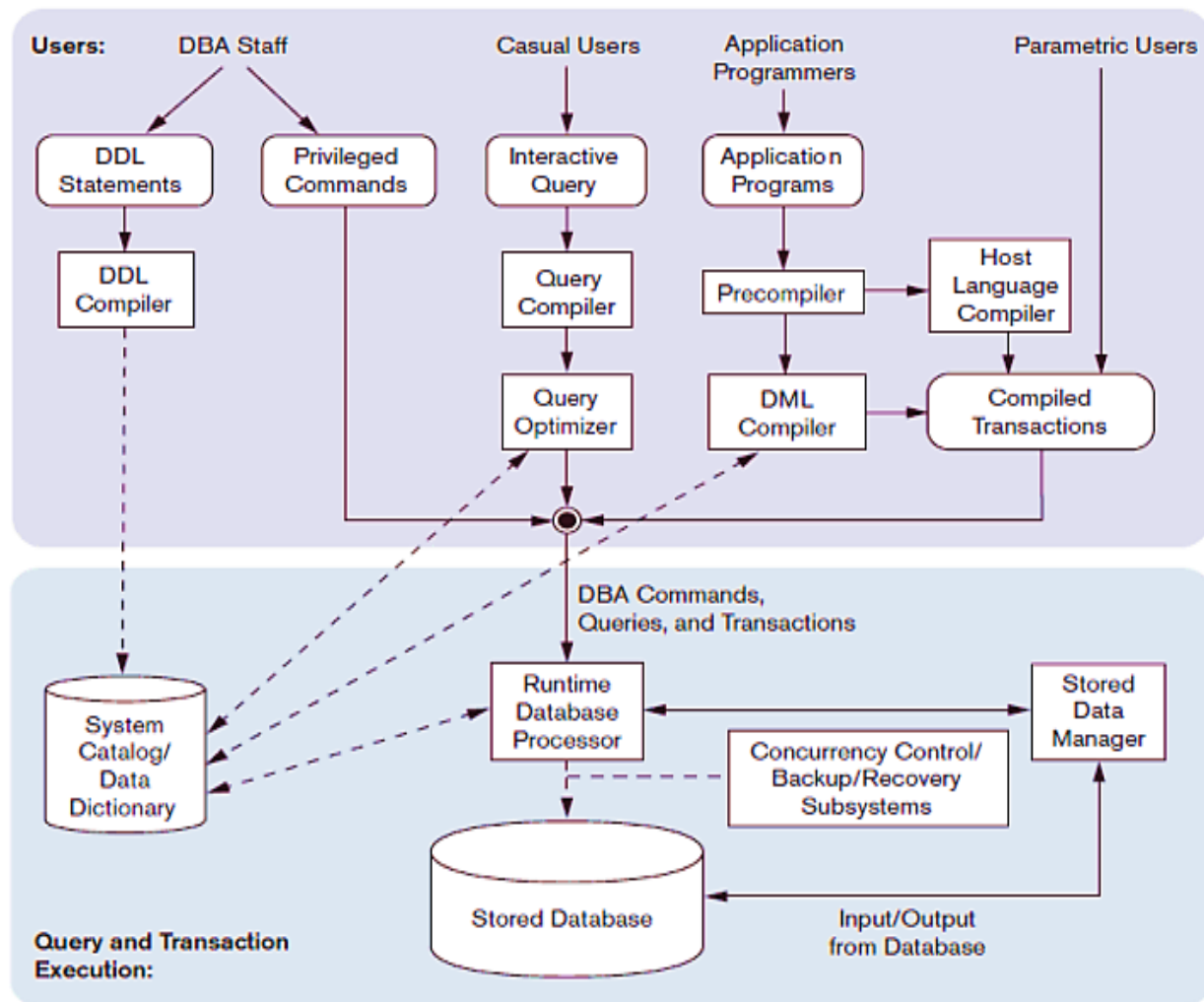
Simplified Database System Environment



The Database System Environment (typical)

- Upper part:
 - 4 kinds of usage modes
 - Database languages (SQL) and programming languages
- Lower part:
 - Core DBMS functionality
 - Runtime database processor has binary API which deals with actual data





- **Database system environment:**

- Upper : The top part of Figure,

- **DBA STAFF:**

It shows interfaces for **the DBA staff**, The DBA staff works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands.

- **DDL COMPILER** → The DDL compiler processes schema definitions, specified in the DDL and stores descriptions of the schemas (meta-data) in the DBMS catalog.
- The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints.

- **casual users** who work with interactive interfaces to formulate queries,
 - **interactive query interface**: Casual users and persons with occasional need for information from the database interact using some form of interface
 - **QUERY COMPILER**: These queries are parsed and validated for correctness of the query syntax, the names of files and a query compiler that compiles them into an internal form.
 -
- **application programmers** who create programs using some host programming languages
 - The **Precompiler** extracts DML commands from an application program written in a host programming language. These commands are sent to the **DML compiler**
 - **DML compiler** for compilation into object code for database access. The rest of the program is sent to the host language compiler.
 - The object codes for the DML commands and the rest of the program are linked, forming a **canned transaction**
- **parametric users** who do data entry work by supplying parameters to predefined transactions.
 - The object codes for the DML commands and the rest of the program are linked, forming a **canned transaction** whose executable code includes calls to the runtime database processor.

Lower part: **Database System Utilities and components**

- **RUNTIME DATABASE PROCESSOR**: handles database access at runtime
- **Stored data manager**: data transfer between disk and main memory
- In addition to possessing the software modules just described, most DBMSs have database utilities that help the DBA manage the database system.
- Common utilities have the following types of functions:
 - **Loading**. A loading utility is used to load existing data files—such as text files or sequential files—into the database. Usually, the current (source) format of the data file and the desired (target) database file structure are specified to the utility, which then automatically reformats the data and stores it in the database.

- A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium. The backup copy can be used to restore the database in case of disk failure.
- Database storage reorganization. This utility can be used to reorganize a set of database files into different file organizations, and create new access paths to improve performance.
- Performance monitoring. Such a utility monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files or whether to add or drop indexes to improve performance.
- Other utilities may be available for sorting files, handling data compression, monitoring access by users, interfacing with the network, and performing other functions.

Classification of Database Management Systems

The main classification - data model

- Relational data model (SQL systems)
- Object data model
- Document-based (JSON), graph-based, column-based, and key-value data models
- Legacy applications still work databases based on the hierarchical and network (COBOL) data models
- Experimental DBMSs are based on a tree-structured data model (XML)

Number of users:

- Single-user at a time
- Multiuser

Number of sites:

- Centralized
- Distributed (big data): homogeneous vs. heterogeneous (the same/different DBMS)

Cost:

- Free (MySQL, PostgreSQL)
- Commercial

Purpose – types of access path:

- General purpose
- Special purpose (online transaction processing (OLTP) systems)

20

Q. Define Data model and discuss the categories of data models?

Data model: It is a set of concepts that can be used to describe the structure of a database that is to say data types, relationships and constraints. The most popular high level model is the **entity relationship model**. It is easy to understand by the professionals and non technical users. It easily distinguishes the data, relationships between data and constraints. It also explains the way the organization uses and manages the information.

Categories of data model: Data model are categorized based on the types of concepts that they provide to describe the database structure.

1. **High level or conceptual data model:** This model provides the concepts that are close to user view. The concepts such as entity attribute and relationships.

An **entity** represents a real world object or concept such as employee or a project that is stored in a database.

An **attribute** represents of interest that further describes an entity, such as employee's name, age, salary and etc.

A **relationship** among two or more entities represents an association among two or more entities, for example **works on** relationship between an employee and project.

2. **Physical or low level data model:** This model provides the concept that describes the details of how data is stored in the computer. Concepts provided by low level data models are generally meant for computer specialist, not for typical end users.

3. **Representational or implementation data model:** this model is in between these two extremes of the physical model and the conceptual model. This does not hide all the storage details from the user and it can be implemented on a computer system directly. It represents data by using record structures and hence is sometimes called record-based data models.

Hierarchical, Network, and Relational Model

These include three most popular data models:

1. **Relational model: Relational Model:** The Relational Model uses a collection of tables both data and the relationship among those data. Each table has multiple columns and each column has a unique name.

Relational database comprising of two tables.

Student_Table

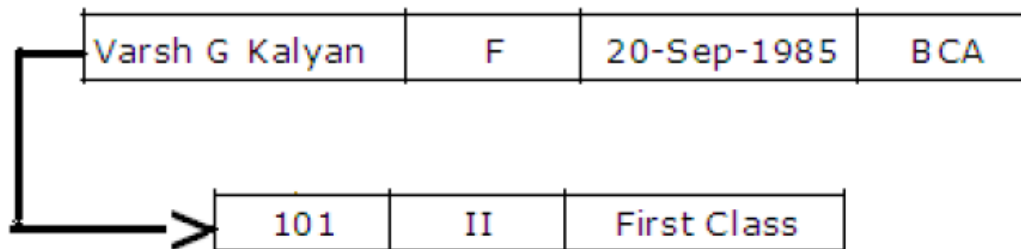
Regno	Name	Gender	DOB	Course
101	Varsh G Kalyan	F	20-Sep-1985	BCA
102	Mohith G Kalyan	M	20-Aug-1980	BBM
103	Nisarga	F	15-Jul-1983	BCom
104	Eenchara	F	04-Dec-1985	BCA

Result_Table

Regno	Sem	Result
101	II	First Class
102	II	First Class
103	II	Passes
104	II	Second Class

2. Network model :

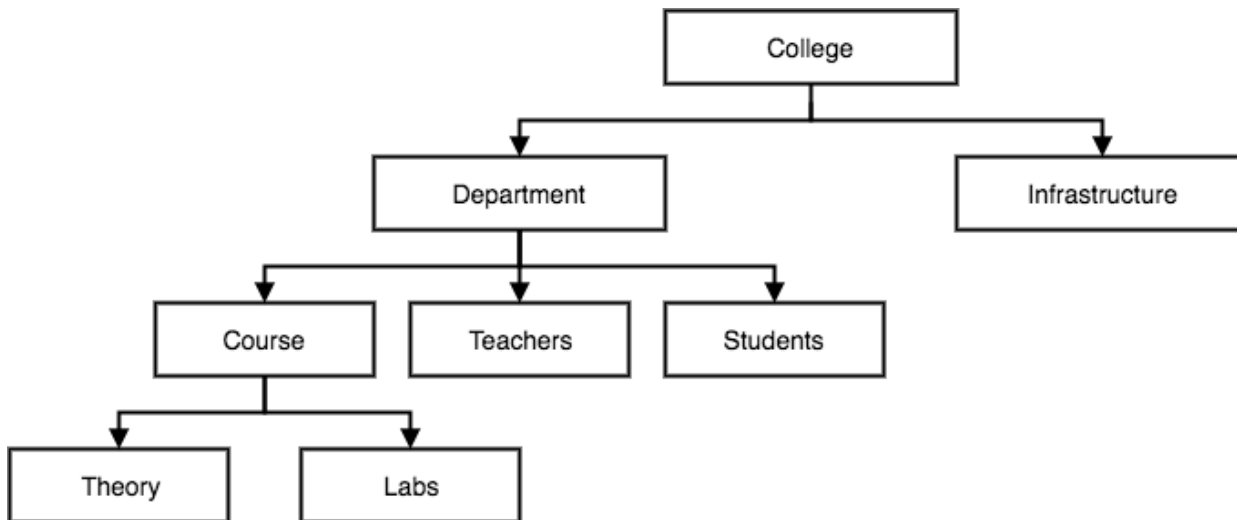
The data in the network model are represented by collection of records and relationships among data are represented by links, which can be viewed as pointers



3. Hierarchical model.

A hierarchical data model is a data model which the data is organized into a tree like structure.

The structure allows repeating information using parent/child relationships: each parent can have many children but each child has one parent. All attributes of a specific record are listed under an entity type.



Q. What is Schema?

Schemas: The description of the database is called the database schema, which is specified during the database design and is not expected change frequently. Most data models have certain conventions for displaying the schemas as diagrams. A displayed schema is called a schema diagram.

Centralized Architecture:

The client/server architecture was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers

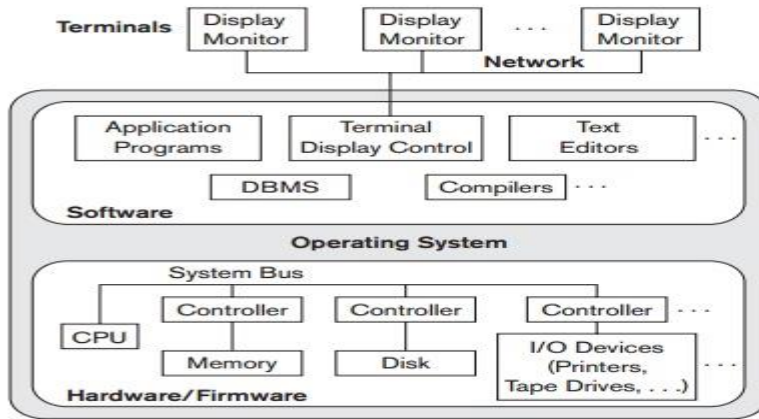


Figure 2.4
A physical centralized architecture.

Web servers, e-mail servers, and other software and equipment are connected via a network. The idea is to define specialized servers with specific functionalities. For example, it is possible to connect a number of PCs or small workstations as clients to a file server that maintains the files of the client machines. Another machine can be designated as a printer server by being connected to various printers; all print requests by the clients are forwarded to this machine.

Client/Server Architecture:

In client-server architecture many clients connected with one server. The server is centerlines.it provides services to all clients. All clients request to the server for different Service. The server displays the results according to the client's request.

Client/server architecture is a computing model in which the server hosts (computer), send and manages most of the resources and works to be required by the client. In this type of architecture has one or more client computers attached to a central server over a network. This system shares different resources.

Client/server architecture is also called as a networking computing model and client-server network because all the requests and demands are sent over a network.

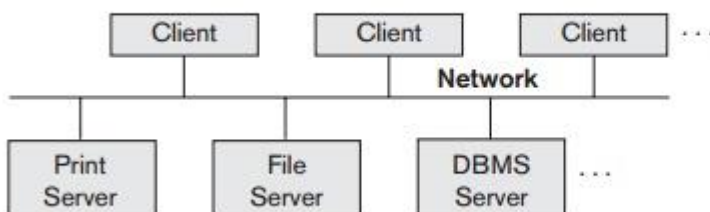


Figure 2.5
Logical two-tier client/server architecture.

Q. What is Database 3-TIER Architecture in DBMS?

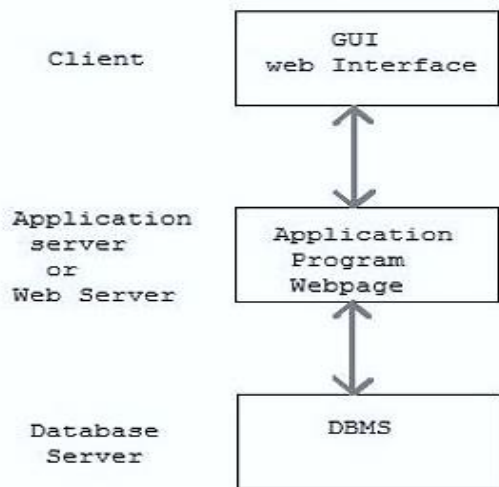
A **Database Architecture** is a representation of DBMS design. It helps to design, develop, implement, and maintain the database management system. A DBMS architecture allows dividing the database system into individual components that can be independently modified, changed, replaced, and altered. It also helps to understand the components of a database.

A Database stores critical information and helps access data quickly and securely. Therefore, selecting the correct Architecture of DBMS helps in easy and efficient data management.

There are mainly three types of DBMS architecture:

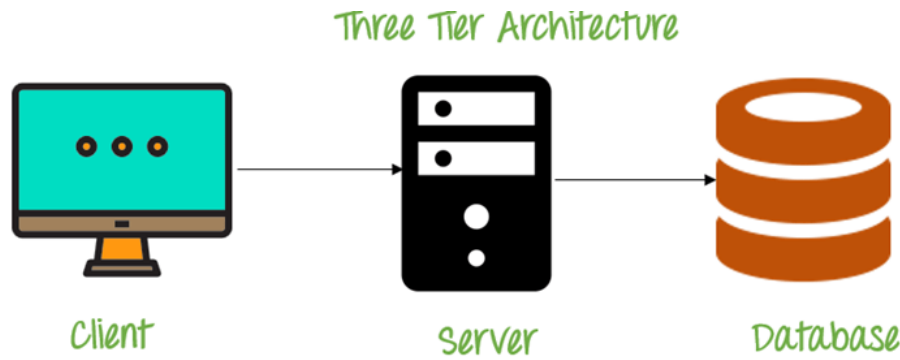
- One Tier Architecture (Single Tier Architecture)
- Two Tier Architecture
- Three Tier Architecture

Tier Architecture in DBMS is the most popular client server architecture in DBMS in which the development and maintenance of functional processes, logic, data access, data storage, and user interface is done independently as separate modules.



Three Tier architecture contains a presentation layer, an application layer, and a database server. 3-Tier database Architecture design is an extension of the 2-tier client-server architecture. A 3-tier architecture has the following layers:

1. Presentation layer (your PC, Tablet, Mobile, etc.)
2. Application layer (server)
3. Database Server



3 Tier Architecture Diagram

The Application layer resides between the user and the DBMS, which is responsible for communicating the user's request to the DBMS system and send the response from the DBMS to the user. The application layer(business logic layer) also processes functional logic, constraint, and rules before passing data to the user or down to the DBMS.

The goal of Three Tier client-server architecture is:

- To separate the user applications and physical database
- To support DBMS characteristics
- Program-data independence
- Supporting multiple views of the data

Example: book your ticket by using IRCTC WEB PAGE.

Summary of Architecture

- An Architecture of DBMS helps in design, development, implementation, and maintenance of a database

<ul style="list-style-type: none"> • CENTRALISED 	<ul style="list-style-type: none"> • The simplest database system architecture is 1 tier where the Client, Server, and Database all reside on the same machine • ALL in One machine
<ul style="list-style-type: none"> • 2- TIER CLIENT/ SERVER 	<ul style="list-style-type: none"> • A two-tier architecture is a database architecture in DBMS where presentation layer(interface)

	runs on a client and data is stored on a server
<ul style="list-style-type: none"> 3-TIER CLIENT/ SERVER 	<ul style="list-style-type: none"> Three-tier client-server architecture consists of the Presentation layer (PC, Tablet, Mobile, etc.), Application layer (server) and Database Server

Differentiate between centralized and distributed data base

Centralized	Distributed
Database is maintained at one site	Database is maintained at a number of different sites
If centralized system fails, entire system is halted.	If one system fails, system continues work with other sites
Less reliable	More reliable

Data modelling using entity relational model

ER-Model is used to represent objects in the real world and of relationship among these objects, which represents the overall logical structure of a database. We have also seen that the data model that is independent of both the DBMS software and the hardware is the conceptual model. ER-model is a high level conceptual data model developed by Chen in 1976 to facilitate database design.

***Explain High level Conceptual data models for database design(10 M)

The database design begins with the software requirement specification of the given problem. The precise requirement collection is very important to have a good database design. This is then to be analyzed.

The next step is to create a conceptual schema that describes the data type, relationships, and constraints. This is called the conceptual design. It deals with the high-level data descriptions of the problem and hence implementation details are hidden.

The implementation model consists of two phases:

- ❖ Logical database design
- ❖ Physical database design

The database design process consists of the following steps.

1. Requirement collection and analysis
2. Conceptual design
3. Logical design [Implementation of data model]
4. Physical design

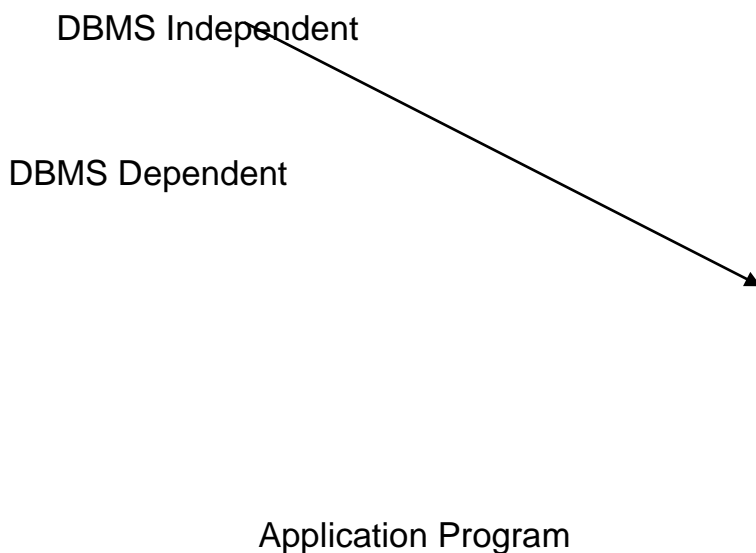
Requirement collection and analysis: This phase is also called feasibility phase. During this phase, the database designers interview database users. This helps them to understand and document their data requirements.

In this phase through the interviews and reviewing all related documents and policies in the organization, the following items are identified.

- a. Clear and concise definition of the problem.
- b. Local dependency lists.
- c. Local dependency diagrams.
- d. Local Schema.

Then a concise set of user's requirements is written. After that the detailed and complete form of users requirements is written. In addition to this, the known functional

requirements of the application are to be specified. The functional requirement consists of user-defined operations (transactions) that will be applied to the database, including both retrievals and updates. Data flow diagrams [DFD], sequence diagrams, scenarios are some techniques used for specifying functional requirements.



Conceptual design: Conceptual schema for the database is created using a high-level conceptual data model. The conceptual schema describes data requirements of the users, entity types, relationships and constraints. These concepts do not include implementation details.

Logical design [Implementation of data model]: This step involves implementation of data model. Since the conceptual schema is transformed from the high level model into the implementation data model, this step is also known as **Data model mapping**.

Logical database design is the process of designing a model of the information in an enterprise based on the chosen database model.

This results in a database schema in the implementation data model such as relational or object-relational database model.

Physical design: During this step, the internal storage structures, indexes, access paths and file organization for the database files are specified.

Physical database design is the process of describing the implementation of the database on the disk. It describes the internal storage structures, indexes, access paths, base relations, security issues and constraints.

Database design involves the following steps:

1. Identifying all the required files.
In database terminology, files are called record types.
2. Identifying the files of each of those record types.
Fields in database terminology are called attributes. Note that attributes in semantic object oriented database are called properties.
3. Identifying the primary key of each of these files.
Note that the primary key is field that uniquely identifies a specific record in the file.
4. Identifying the relationships between record types.

An Example – Company Database:

1. The company consists of several departments and every department has a manager. It is also required to record when he has become the manger.
2. Several employees' works for a department.
3. A department have several locations.
4. Each department controls several projects.

5. Employee details like Social Security Number (SSN), name, the department he works for etc are to be stored.
6. An employee works for only one department, but he/she can work on more than one project.
7. We need to know the employee dependent details for some specific purpose like PF, Insurance and etc.

Overview of Database Design, Entities, Attributes and Relationships

Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems

It helps produce database systems

1. That meet the requirements of the users
2. Have high performance.

The main objectives of database designing are to produce logical and physical designs models of the proposed database system.

- **The logical model** concentrates on the data requirements and the data to be stored independent of physical considerations. It does not concern itself with how the data will be stored or where it will be stored physically.
- **The physical data** design model involves translating the logical design of the database onto physical media using hardware resources and software systems such as database management systems (DBMS).

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

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

Attributes

Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes. There exists a domain or range of

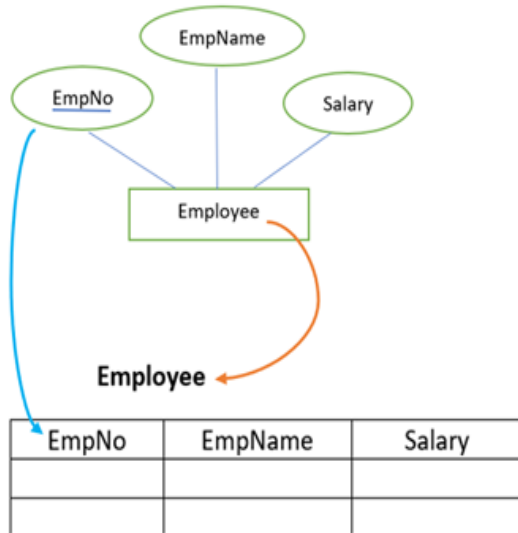
values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Q. What is an entity?

Entity: The E-R model describes data as entities, relationships and attributes:

An **entity** is a thing in the real world with an independent existence that is distinguishable from all other objects.

Example: Each employee in an organization is an entity. A company, a job, a book, etc all are entities. Each entity has particular properties called attributes that describes it.



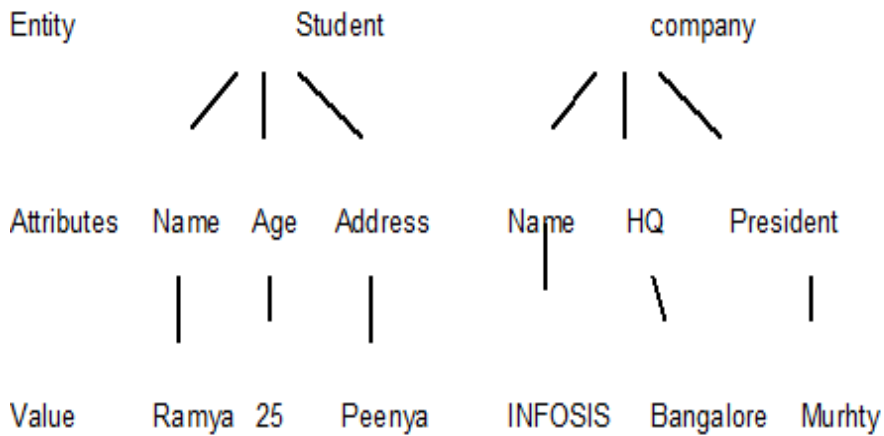
Q. What is an attribute?

An **attribute** is a property that describes an entity.

Example:

1. The attributes of a person's entity are his name, address, job, salary etc. for an entity each attributes will have a value.
2. **student** (**regno**, **name**, **age**, **address**)
3. **Employee** (**ID**, **name**, **dept**, **salary**)
4. **Company** (**name**, **HQ**, **president**)

Attributes:



Q. Explain the various types of attributes?

Types of Attributes

- Simple Attribute
- Composite
- Single valued Attribute
- Multivalued Attribute
- Stored attribute
- Derived attribute
- Complex
- Null value Attribute
- Key attribute

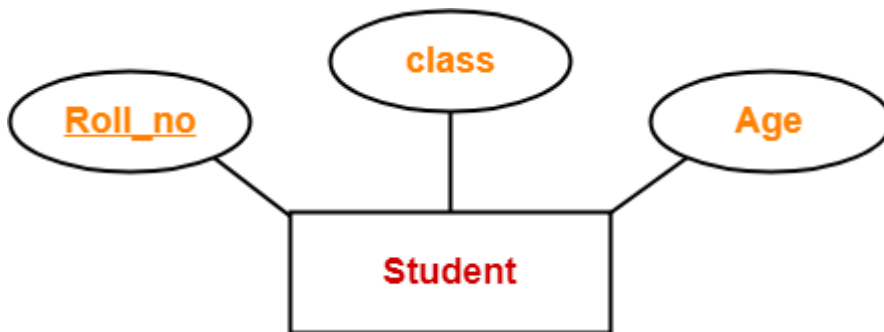
Each attribute is associated set of values called domain.

TYPES OF ATTRIBUTES:

Simple or atomic attributes: These attributes can't be subdivided into further. In other words, the atomic attributes hold **single value and not composed of any attributes**.

Example: Regno, ID, DOB, age, Deptno, sex.

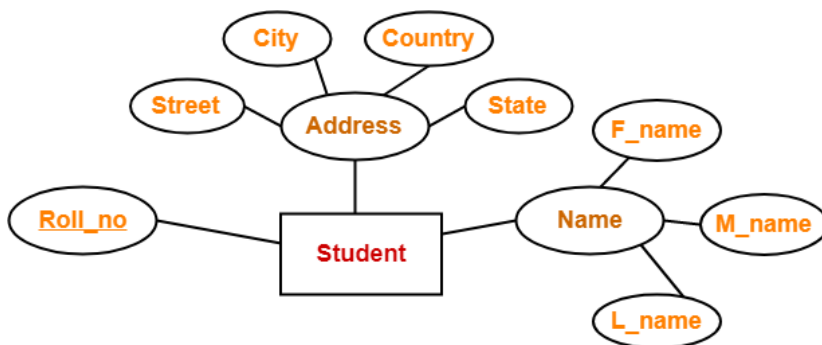
For example, a student's phone number is an atomic value of 10 digits.



Composite attributes: Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.

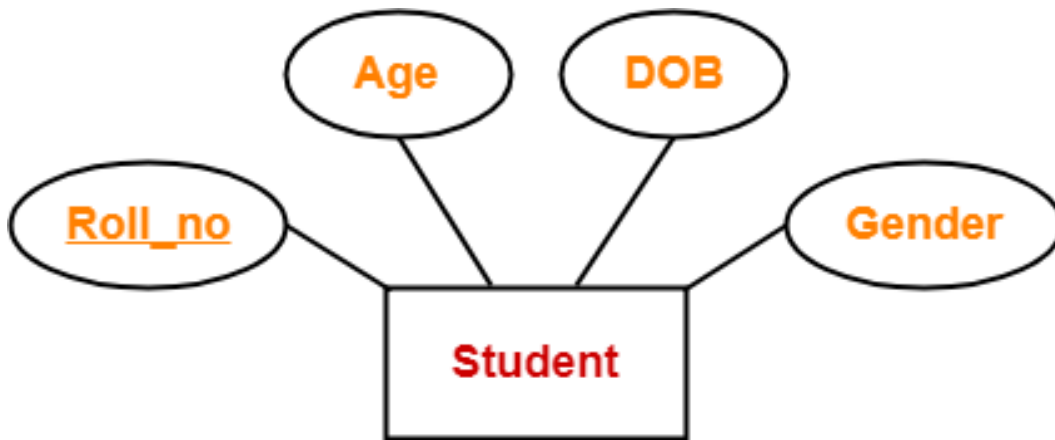
Example:

1. The address attribute can be subdivided into **house no, street_name, area, city and country.**
2. The name attributes, which can be composed of **first name, middle name, and last name.**

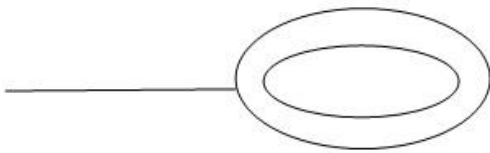


Single valued attributes: An attribute that can take only one value at a time is called single valued attributes. Usually, for a particular entity, each attributes will have single value.

Example: The age attribute will have a single value.

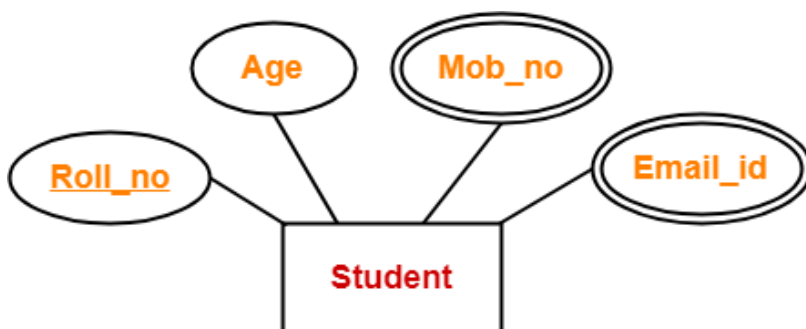


Multi valued attributes: Some attribute have more than one value for the same attribute and are called multi valued attributes.



Example: A college degree attribute can have multiple values.

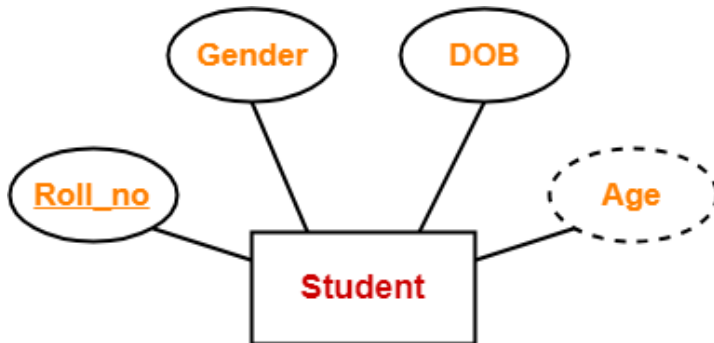
Degree [B,Sc, MCA, PhD]
 Admin [Bangalore, Mumbai]
 Carcolor [Red, Black].



Derived attributes: If the value of an attribute can be derived from some other attributes, then such attributes are called **derived attributes**. Since age can be determined by knowing birthday and current date. **The age attribute is said to be derived attribute.**



Another example, **gross pay of an employee** this can be derived by knowing **basic pay, allowances, and deduction for employee**.



Stored attributes: The value of certain attributes cannot be obtained or derived from some other attributes, that is, they are not derived from any other attributes, these types of attributes is called as stored attributes.

Example: Birth date, Book_ID, SSN, Part#.

The attribute SSN is direct attribute and is not dependent on any other attribute or derivable from one or more other attributes.

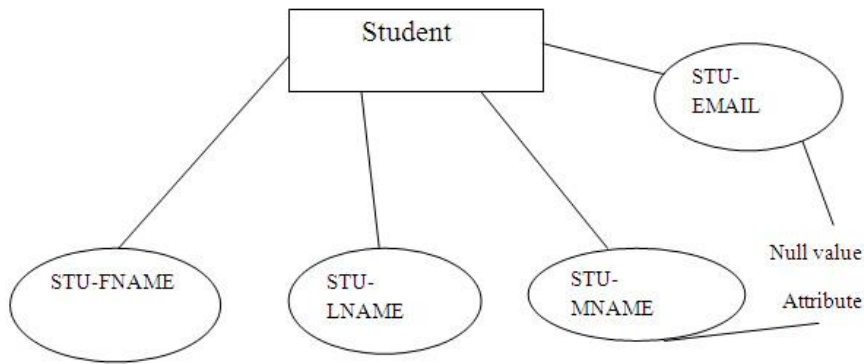
Complex attribute: A attribute which is both composite and multivalued. complex Attribute is a type of attribute in database.

For example, **A person can have many phone numbers, many e-mail addresses, home addresses etc.** So if there is a attribute in the name of 'Contact detail, It can be a complex attribute.

Example: A person can have more than one residence; each residence can have more than one phone.

Null attributes: A null attributes used when an attributes does not have any value. A null value does not mean that the value is equal to zero but indicate no value is stored for that attribute.

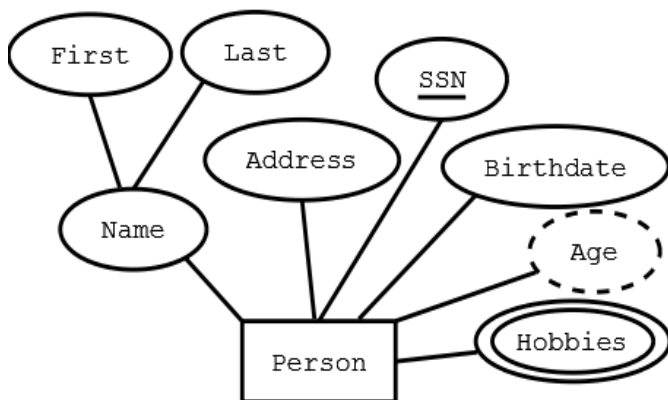
Example: Abasement number attribute of an address applies only to address that are in apartment buildings, and not to other types of residence such as single family home. Email: All employees in an employee database may not have e-mail address.



Key Attributes: This attribute represents the main characteristic of an entity i.e. primary key. Key attribute has clearly different value for each element in an entity set.

Example: The entity student ID is a key attribute because no other student will have the same ID.

Graphical Representation in E-R diagram




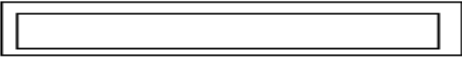

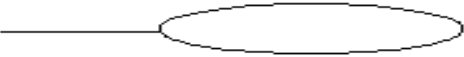
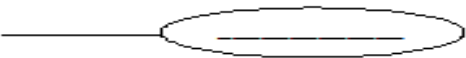
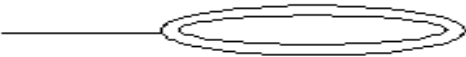

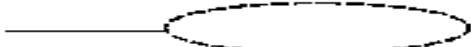
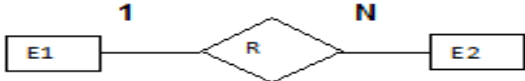
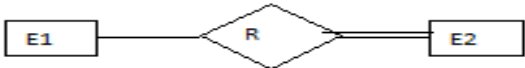
Rectangle -- Entity

Ellipses -- Attribute (underlined attributes are [part of] the primary key)

Double ellipses -- multi-valued attribute

Dashed ellipses-- derived attribute, e.g. age is derivable from birthdate and current date.

ER MODEL NOTATION

ER Symbols :	Meaning
	Entity
	Weak Entity
	Relation ships
	Attributes
	Key attribute
	Multi valued attribute
	Composite attributes
	Derived attribute
	Cardinality ratio 1:N E1, E2 in R
	Total participation E2 in R

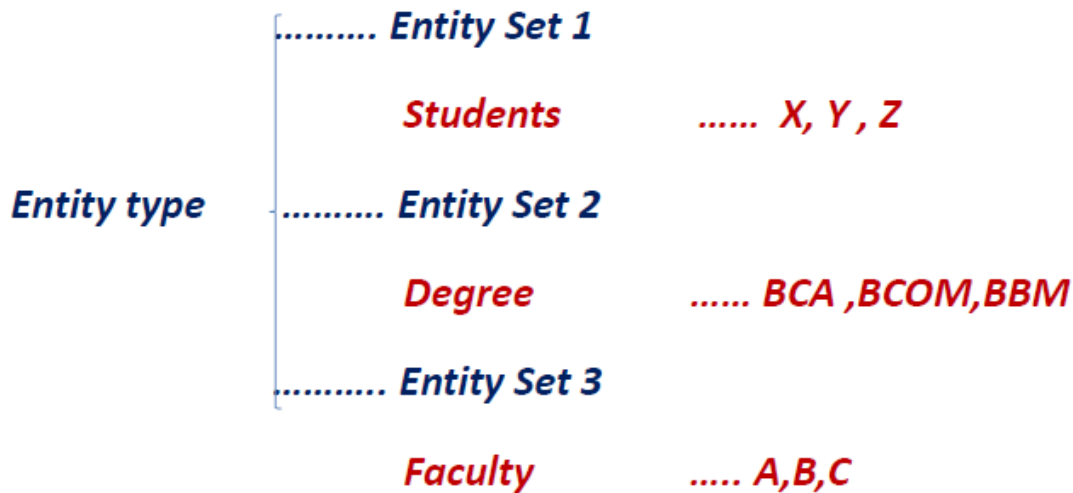
Q. Define the following?

Entity types, Entity sets, Key attributes, value sets.

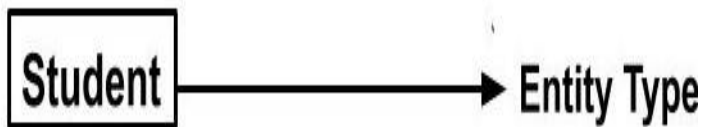
Entity set: It is a set of entities of the same type that share the same properties or attributes. The set of all employees working for the same department can be defined as entity set employee but each entity has own values for each attributes.

Entity Type: An entity type defines a set of entities that have the same attributes. Each entity type in the database is described by its name and a list of attributes.

Example :



An entity type describes the schema or intention for a set of entities that share the same structure. The individual entities of a particular entity type are grouped into a collection or entity set. Which is also called the extension of the entity type.

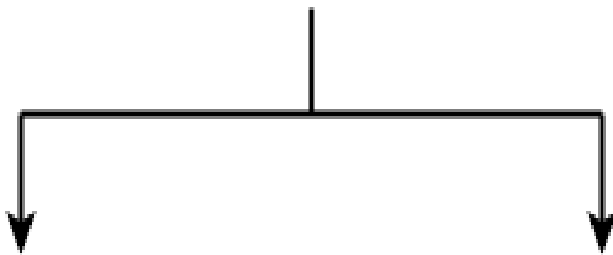


Roll_no	Student_name	Age	Mobile_no
1	Andrew	18	7089117222
2	Angel	19	8709054568
3	Priya	20	9864257315
4	Analisa	21	9847852156

Entity

ENTITY TYPES

Entity Set



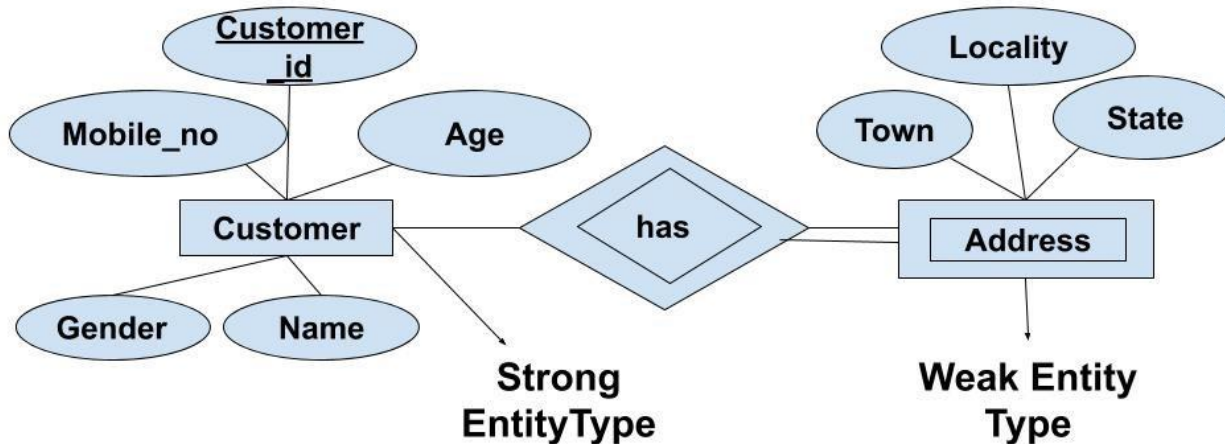
Strong Entity Set

Weak Entity Set

The **entity** can be classified into two basic categories, strong **entity set** and weak **entity set**.

An **entity set** that has a primary key is the strong **entity set**.

An **entity** that doesn't have a primary key is a weak **entity set**. A weak **entity set** is dependent on one strong **entity set** to distinguish its **entities**.



There are two types of weak entities: **associative entities** and **subtype entities**.

An **associative entity** is uniquely identified by concatenating the primary keys of the two entities it connects.

For example, the number of hours the Employee worked on a particular Project is an attribute of the relationship between Employee and Project, not of either Employee or Project.

Subtype: where one entity (the subtype) inherits the attributes of another entity (the supertype).

Relationships between an entity supertype and its subtypes are always described as “is a.” For example, Employee is a Permanent Employee, Employee is a Part-time Employee.

Key attributes of an entity type:

- A key attribute is a minimal set of attributes of an entity set, which uniquely identifies an entity in an entity set. A key may be a single attributes or may be more than one attribute.

An entity type usually an attribute whose values are distinct for each individual entity such an attribute is called a key attribute. Whose values can be used to identify each entity.

Example:

For the company entity, name can be the key attribute because no two companies can have the same name. For the student entity, regno can be the key attribute.

Value sets (Domain) of attributes:

A set of values that may be assigned to the attributes of each individual entity, in an entity set is called the value set or domain.

Keys play an important role in the relational database.

- It is used to **uniquely identify any record or row of data from the table**. It is also used to establish and identify relationships between tables.
- **For example:** In Student table, ID is used as a key because it is unique for each student.
- In PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.

STUDENT	PERSON
ID	Name
Name	DOB
Address	Passport_Number
Course	License_Number
	SSN

Q. Key attributes (Primary key) of an entity type:

A key attribute is a minimal set of attributes of an entity set, which uniquely identifies an entity in an entity set. A key may be a single attributes or may be more than one attribute.

An entity type usually an attribute whose values are distinct for each individual entity such an attribute is called a key attribute. Whose values can be used to identify each entity?

Example:

1. For the company entity, name can be the key attribute because no two companies can have the same name.
2. For the student entity, regno can be the key attribute.

Q. What is weak entity?

Weak Entity

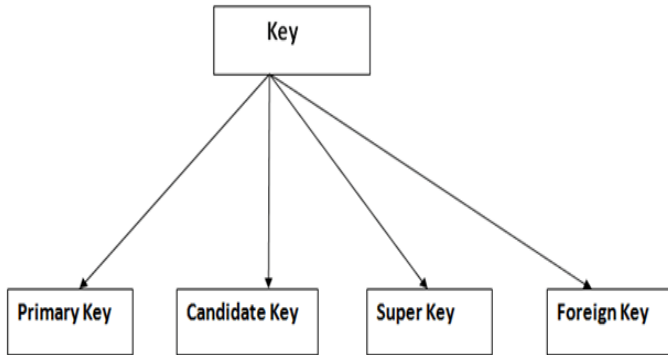
Entities which do not have key attribute (primary key) is known as weak entity weak entity is also known as Child entity or subordinate entity.

Q. What is strong Entity?

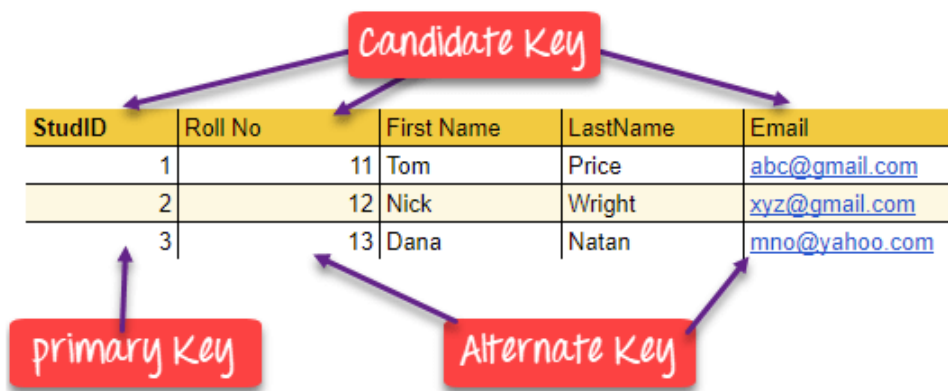
Strong entity

Entities which have key attribute is known as Strong entity

Types of key:



- Super Key: A set of attributes (one or more) that collectively identifies an entity in an entity set.
- Candidate Key: A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- Primary Key: A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.



1. **Super key:** It is a set of one or more attributes that taken collectively, allows us to identify uniquely an entity in the entity set.

- The combination of customer_name and social_security_numbers is a super key for the entity set customer.
- The customer_name attribute is not a super key because several people might have the same name.
- The social_security_number attribute could also be super key since it is sufficient to distinguish one customer entity from another.

2. **Candidate key:** A super key may contain extraneous attributes. A super key for which no proper subset is a super key. i.e., minimal super key is called candidate key.

- The combination (**customer_name, SSN**) forms super key, the attribute **SSN alone is a candidate key**.

3. **Primary key:** It is a candidate key chosen by the database designers as the principal means of identifying entities within an entity set.

4. **Prime attribute:** An attribute of relation schema **R** is called a prime attribute of **R** if it is a member of some candidate key of **R**.

5. **Non prime attribute:** An attribute is called non prime if it is not a prime attribute i.e., it is not a member of candidate key.

6. **Weak entity set:** An entity set not having sufficient attributes to form a primary key is called weak entity set.

7. **Strong entity set:** An entity set that has a primary key is termed a strong entity set.

8. **Foreign key:** A set of attributes in a relation is a foreign key if it satisfies the following conditions.

- It should have the same domain as the primary key attributes of another relation schema and is said to refer to this relation.
- A value of foreign key in tuple t1 either occurs as a value of primary key for some tuple t2 in another relation.

9. **Composite key:** If a key has more than one attribute, it is called composite key.

- **Example:** Consider the relation schema BRANCH with attributes **branch_name, branch_city**, identify super key, candidate key and primary key.
- {**Branch_name, Branch_city**} → **Super key**.
- {**Branch_name**} → **Candidate key**.

- {Branch_name} à Primary key.

SUMMARY OF KEYS:

- Super Key - A super key is a group of single or multiple keys which identifies rows in a table.
- Primary Key - is a column or group of columns in a table that uniquely identify every row in that table.
- Candidate Key - is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes.
- Alternate Key - is a column or group of columns in a table that uniquely identify every row in that table.
- Foreign Key - is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity.
- Compound Key - has two or more attributes that allow you to uniquely recognize a specific record. It is possible that each column may not be unique by itself within the database.
- Composite Key - An artificial key which aims to uniquely identify each record is called a surrogate key. These kind of key are unique because they are created when you don't have any natural primary key.
- Surrogate Key - An artificial key which aims to uniquely identify each record is called a surrogate key. These kind of key are unique because they are created when you don't have any natural primary key.

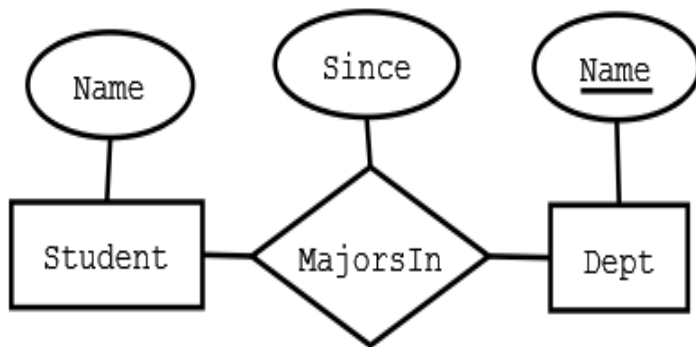
Relationships

Relationship: connects two or more entities into an association/relationship

- "John" majors in "Computer Science"

Relationship Type: set of similar relationships

- *Student* (entity type) is related to *Department* (entity type) by *MajorsIn* (relationship type).



Relationships types, sets and instances:

RELATIONSHIP Definition: A relationship is an association among two or more entities. A relationship captures how two or more entities are related to one another.

Example:

An ‘owns’ relationships between a company and computer

A ‘Supervises’ relationships between an employee and department.

A ‘Performs’ relationships between an artist and song.

A ‘Proved’ relationship between a mathematical and a theorem.

Relationships are represented as diamonds symbol, connected by lines to each of the entities in the relationship in an ER diagram.

- **Relationship:**
 - association between multiple entities

Example-

'Enrolled in' is a relationship that exists between entities **Student** and **Course**.



Relationship types: A relationship type R among 'n' entity type E_1, E_2, \dots . Defines a set of associations among entities from these types.

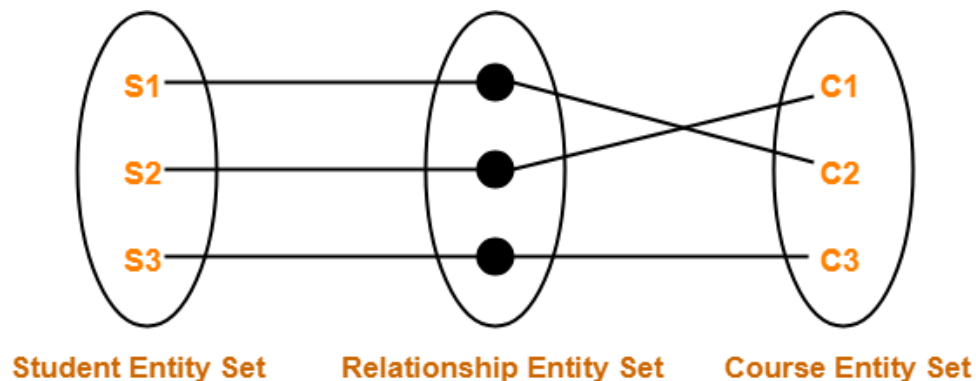
Example: Relationship type **works_for** between two entities of company database, namely employee and department.

Relationship Instance: Each relationship instance ri is an association of entities where the association includes exactly one entity from each participating entity type.

Example: Each relationship instance in **works_for** relationship type associates an employee entity and one department entity.

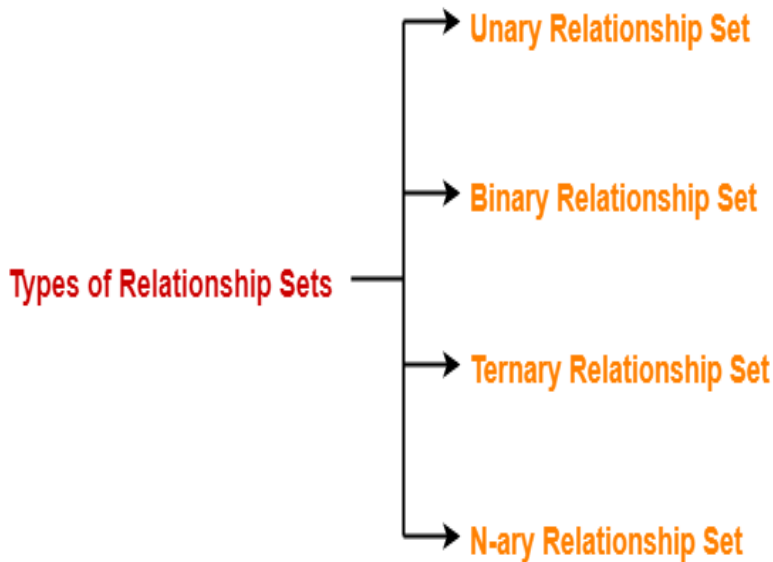
Relationship Set: A collection of relationships of same type is called the relationship set. OR

Relationship Set: A relationship set is a set of relationships of same type



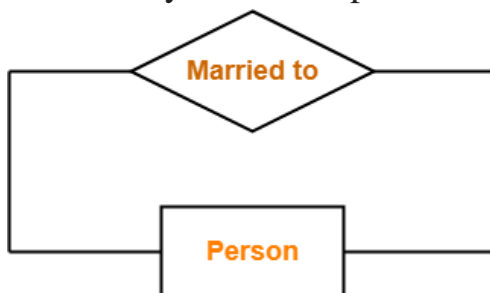
Set Representation of ER Diagram

Types of Relationship Sets-



Degree of a relationship: Degree of a relationship type is the number of entities participating in the relation.

Unary relationship set is a relationship set where only one entity set participates in a relationship set. A unary relationship is **when both participants in the relationship are the**



same entity. **Unary Relationship Set**

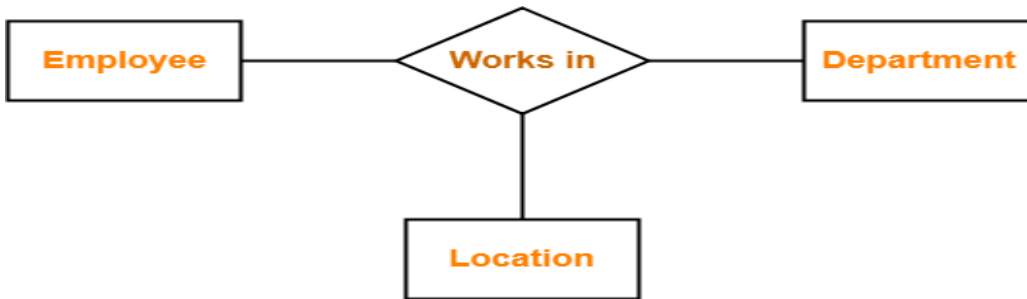
- A **Binary Relationship** is the **relationship between two different Entities** i.e. it is a relationship of role group of one entity with the role group of another entity. Binary relationship set is a relationship set where two entity sets participate in a relationship set.



Binary Relationship Set

- **Ternary relationship** set is a relationship set where three entity sets participate in a relationship set.

Ternary relationship exists when there are three types of entity and we call them a degree of relationship is 3. Since the number of entities increases due to this, it becomes very complex to turn E-R into a relational table.



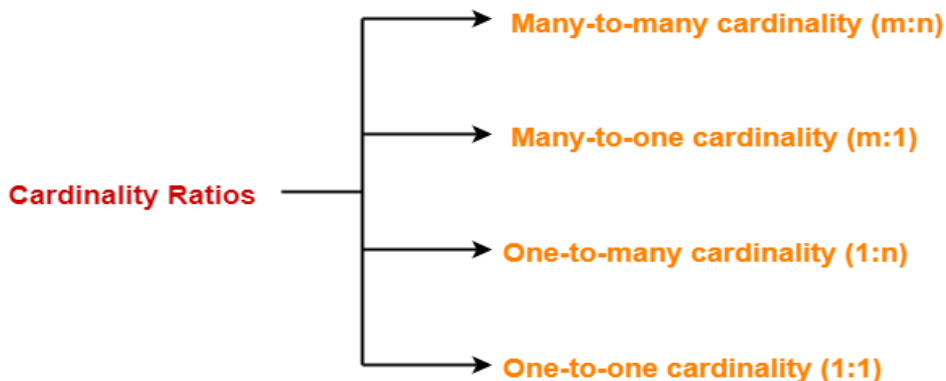
Ternary Relationship Set

4. N-ary Relationship Set-

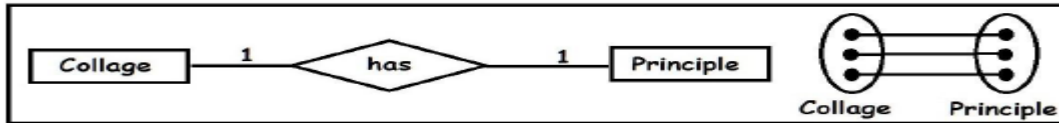
Nary relationship set is a relationship set where 'n' entity sets participate in a relationship set.

Cardinality: Cardinality constraint defines the maximum number of relationship instances in which an entity can participate. **Cardinality** is the number of instances of one entity that can or must be associated with each instance of another entity

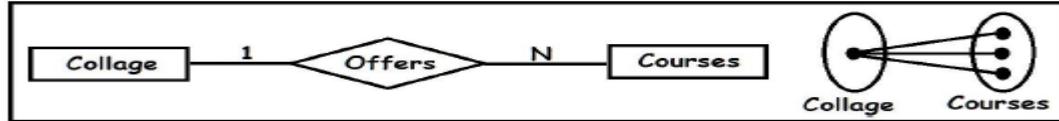
TYPES OF CARDINALITY/ MAPPING



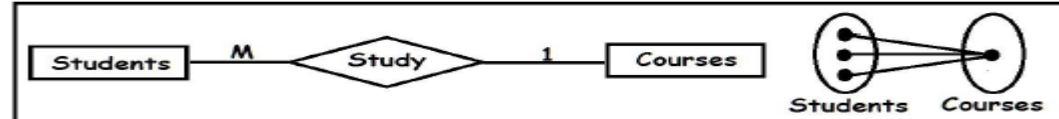
1. **One-to-One Relationship (1:1):** For ex: (1:1) college can have only one principle



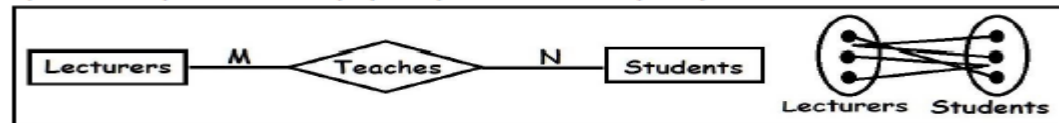
2. **One-to-Many Relationship (1:N):** For ex: (1:N) Collage offers many courses



3. **Many-to-One Relationship (M:1):** For ex: (M-1) STUDENTS STUDY COURSE



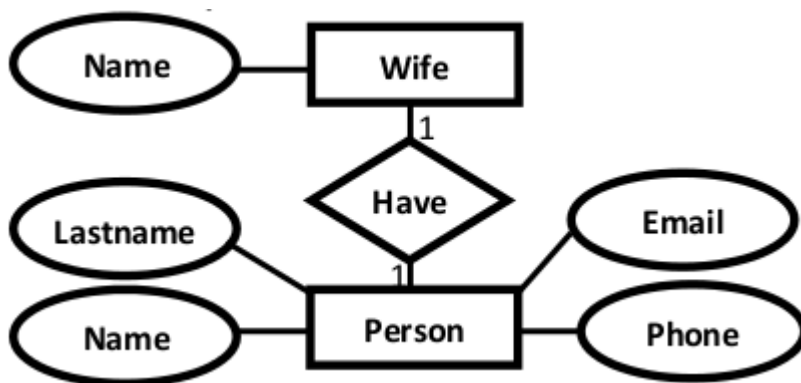
4. **Many-to-Many Relationship (M:N):** For ex: (M:N) students enrolled in a classes



Cardinality types/ Ratios:

1:1 Relationships :

One-to-one: one entity from entity set A can be associated with at most one entity of entity set B and vice versa.



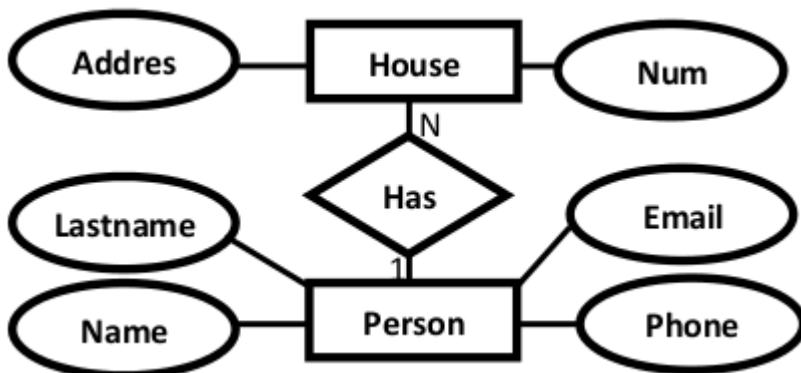
Persons(personid , name, lastname, email , *wifeid*)

Wife (wifeid , name)

1:N Relationships

One-to-many: One entity from entity set A can be associated with more than one entities of entity set B but from entity set B one entity can be associated with at most one entity.

For instance, the Person can have a **House** from zero to many , but a **House** can have only one **Person**. To represent such relationship the **personid** as the Parent node must be placed within the Child table as a foreign key but **not the other way around as shown next:**



It should convert to :

Persons(personid , name, lastname, email)

House (houseid , num , address, *personid*)

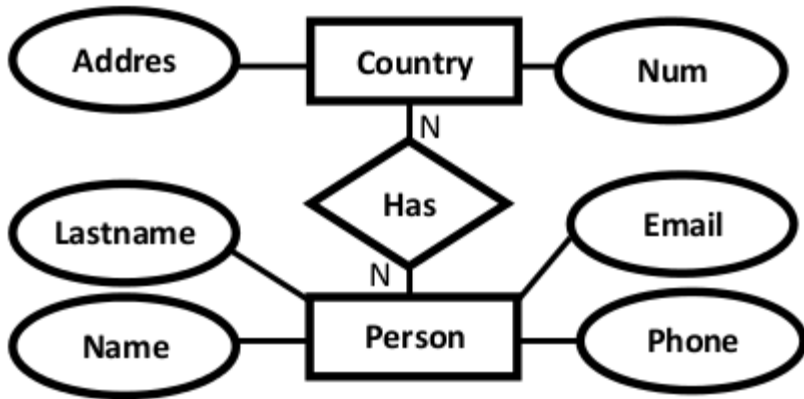
Many-to-one(M:1): More than one entities from entity set A can be associated with at most one entity of entity set B but one entity from entity set B can be associated with more than one entity from entity set A.



Beginnerbook.com

N: N Relationships (M:M)

Many-to-many: one entity from A can be associated with more than one entity from B and vice versa.



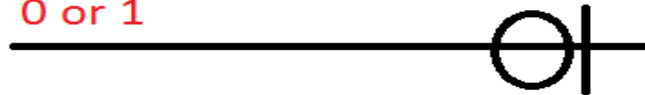
Persons(personid , name, lastname, email)

Countries (countryid , name, code)

HasRelat (hasrelatid , **personid** , **countryid**)

One card
holder can
have:

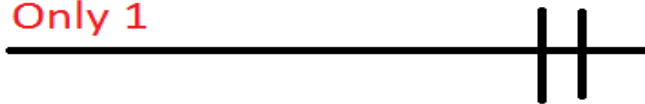
0 or 1



Credit card

One card
holder can
have:

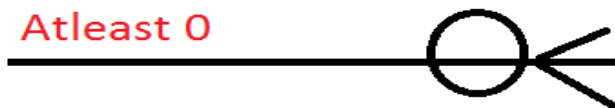
Only 1



Credit card

One card
holder can
have:

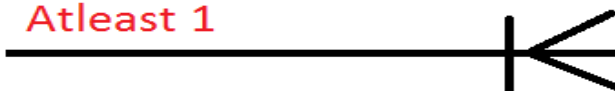
Atleast 0



Credit card

One card
holder can
have:

Atleast 1



Credit card


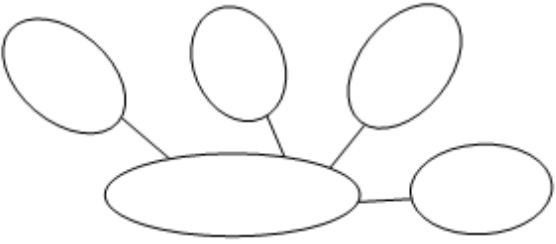

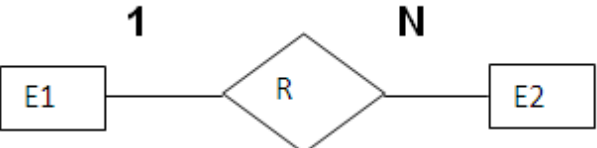
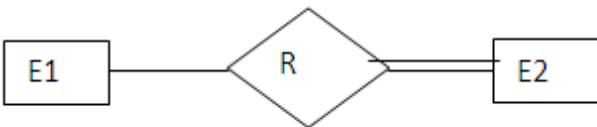
Proper naming of Schema constructs

- When designing a data base schema, the choice of names for entity types, attributes, relationship types & role is not always straight forward

- One should choose names that convey as much as possible, the meaning attached to the different constructs in the schema
- It is preferable to use Singular names for entity types, rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type
- Each object in a schema is known as schema construct

ER DIAGRAM.

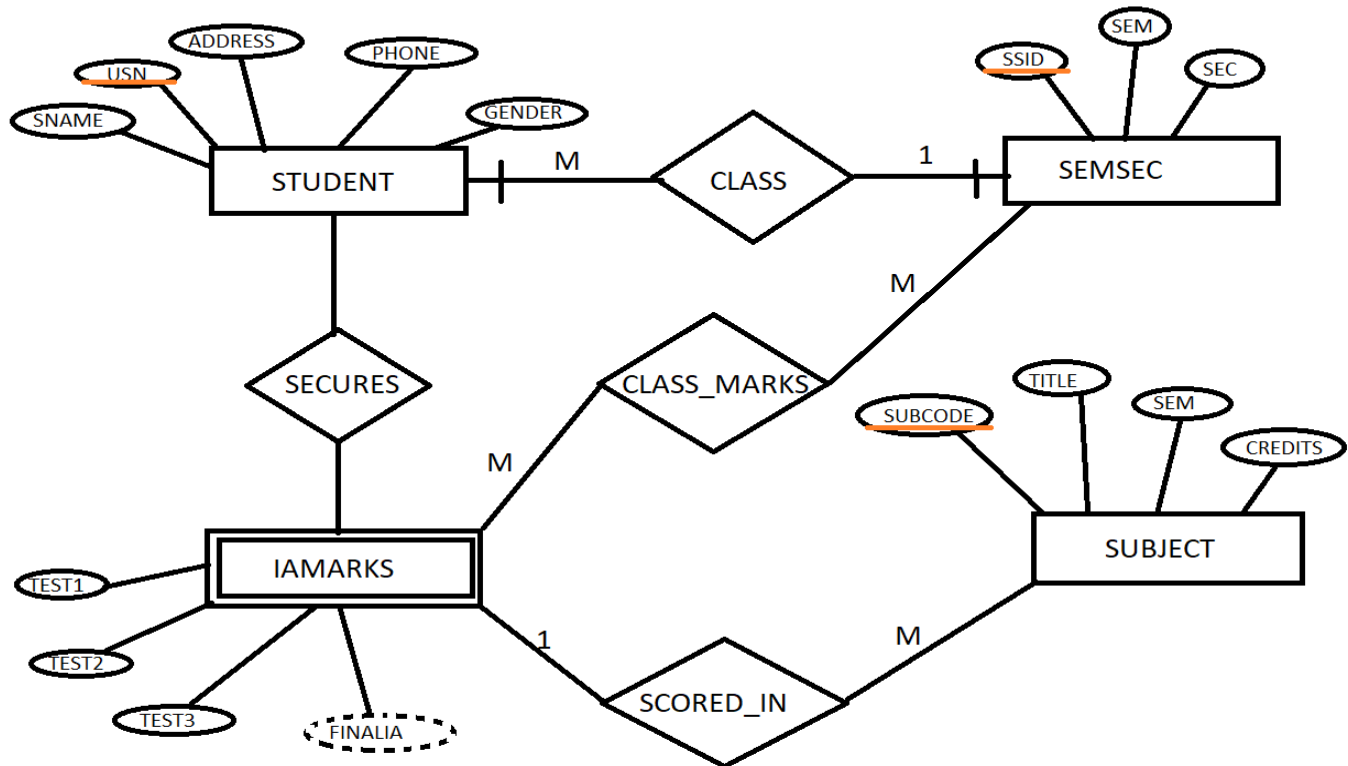
An Entity Relationship [ER] diagram is graphical depiction of organizational system elements & the association among the elements where elements are entities & the association is relationship. Thus, the diagrammatic notation associated with ER model is known as ER diagram.

	Multi valued attribute
	Composite attributes
	Derived attribute
	Cardinality ratio 1:N E1, E2 in R
	Total participation E2 in R

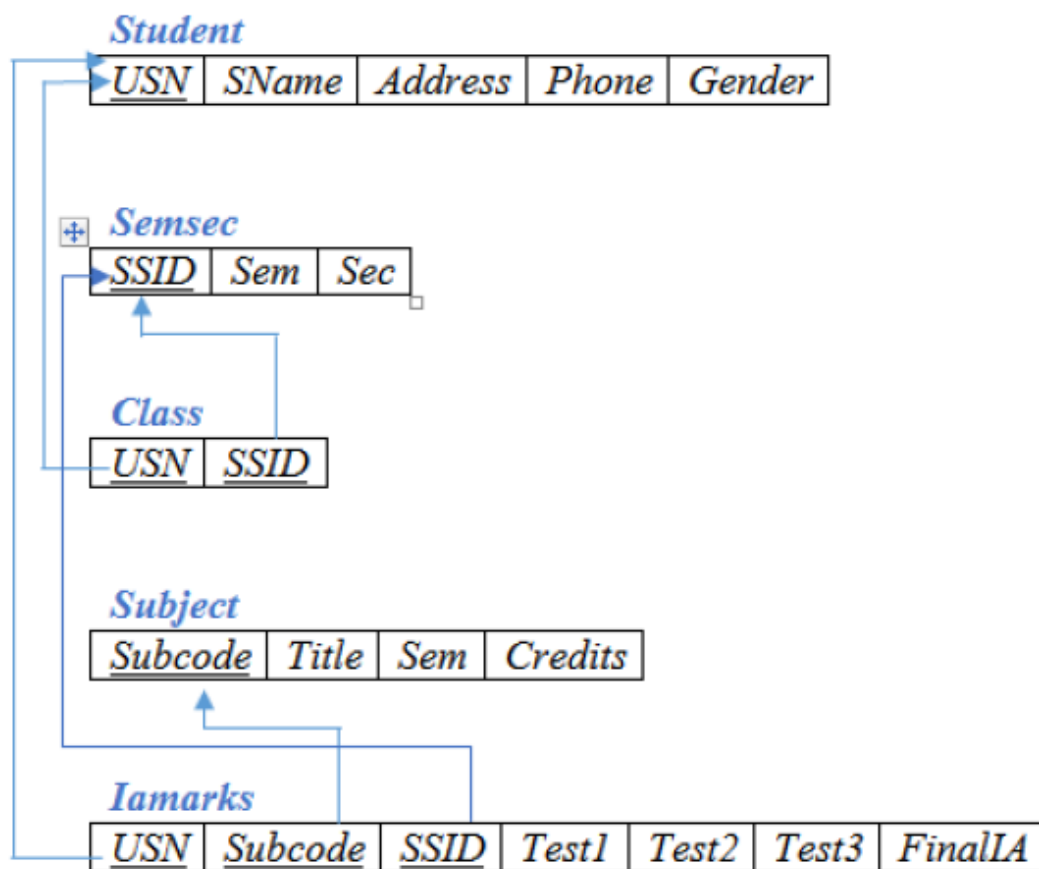
1. Draw E-R diagram and convert entities and relationships to relation table for a given scenario. (eg. College)



PRESIDENCY COLLEGE
(AUTONOMOUS)
AFFILIATED TO BENGALURU CITY UNIVERSITY, APPROVED BY AICTE, DELHI & RECOGNISED BY THE GOVT. OF KARNATAKA
RE-ACCREDITED BY NAAC WITH 'A+' GRADE



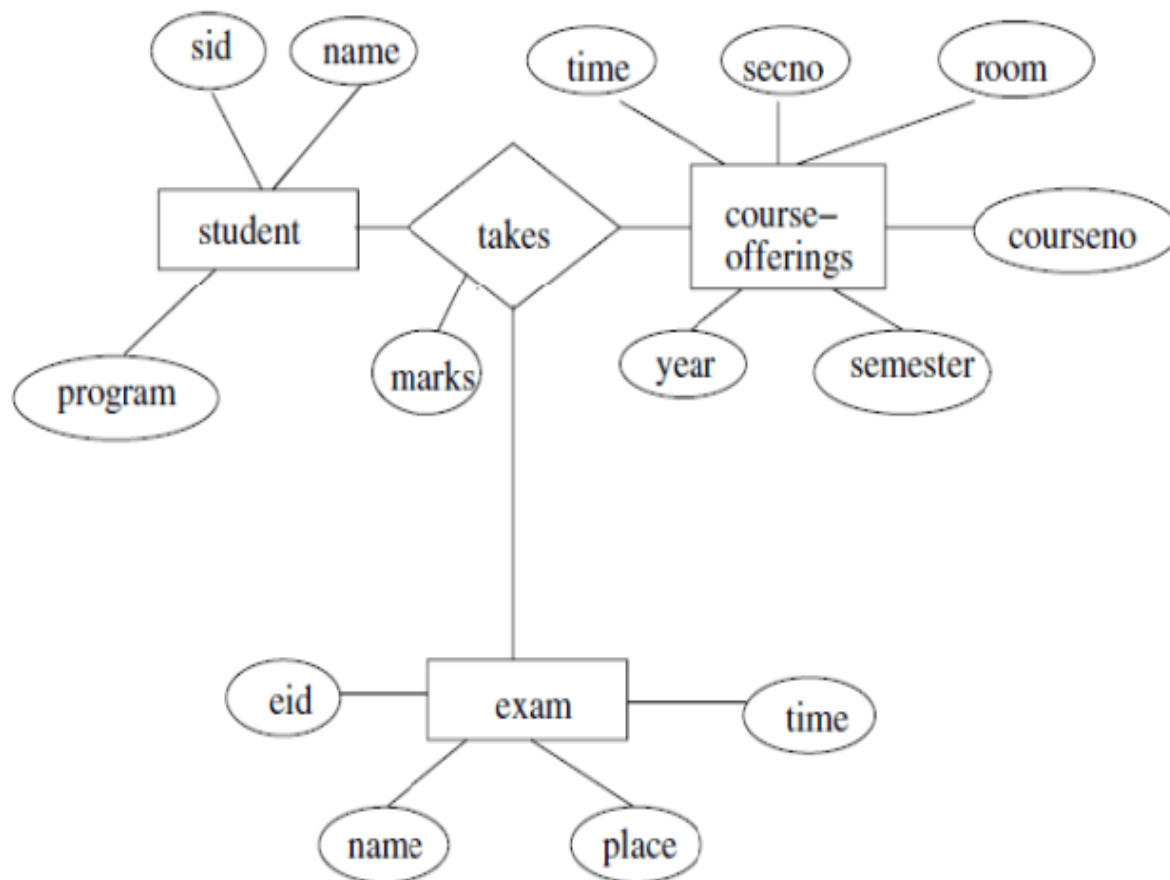
MAPPING ENTITY RELATIONSHIP MODEL TO RELATIONAL TABLE



Consider a database used to record the marks that students get in different exams of different course offerings.

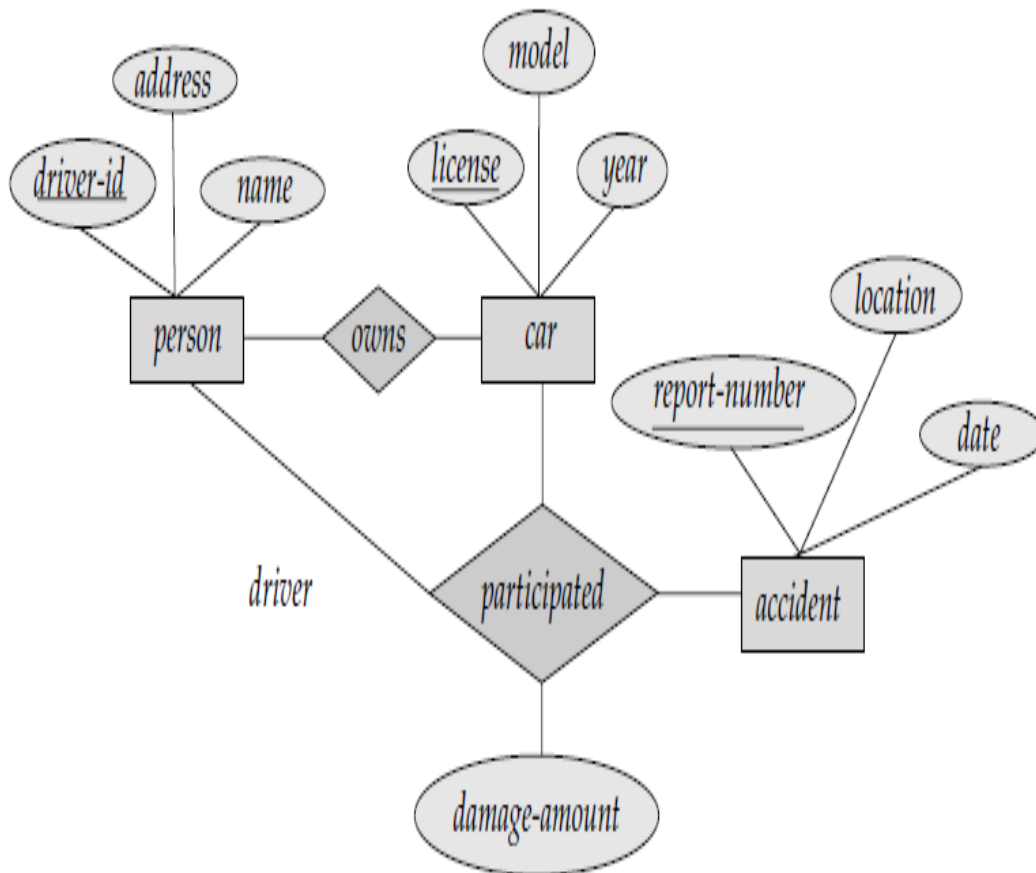
a) Construct an E-R diagram that models exams as entities, and uses a ternary relationship, for the above database.

Answer a :



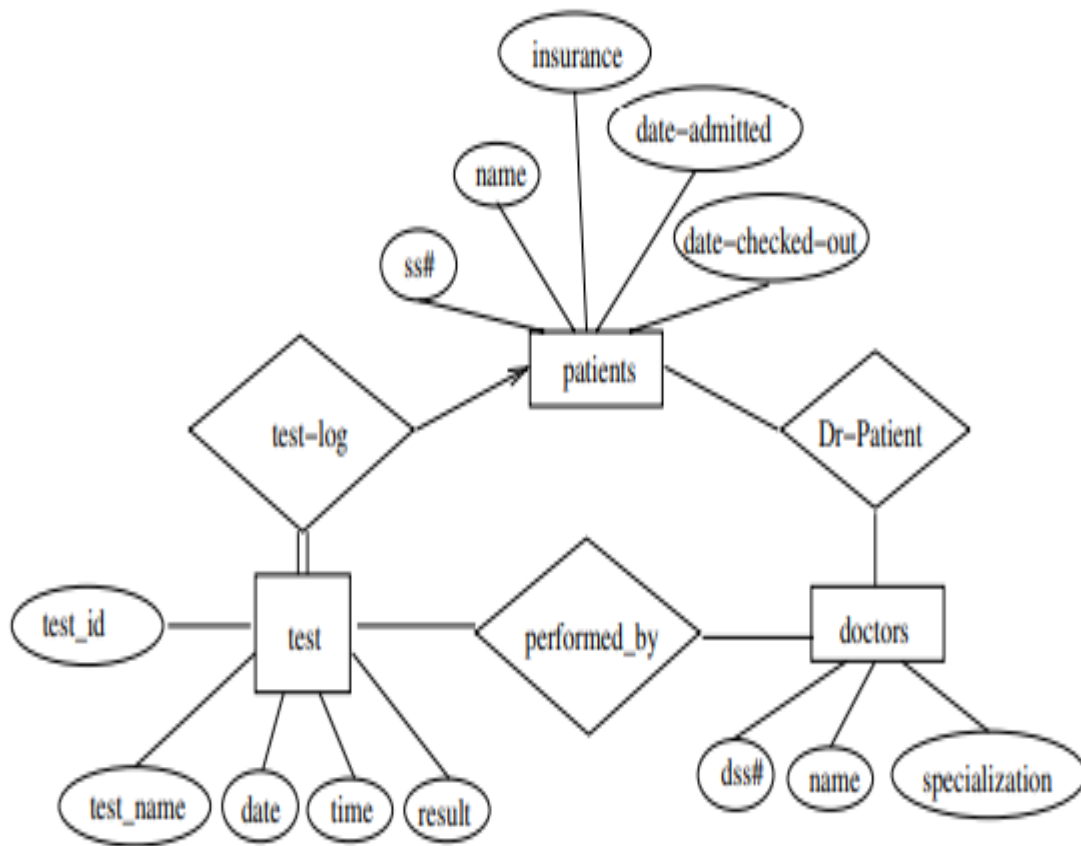
Construct an E-R diagram for a car-insurance company

- whose customers own one or more cars each.
- Each car has associated with it zero to any number of recorded **accidents**.

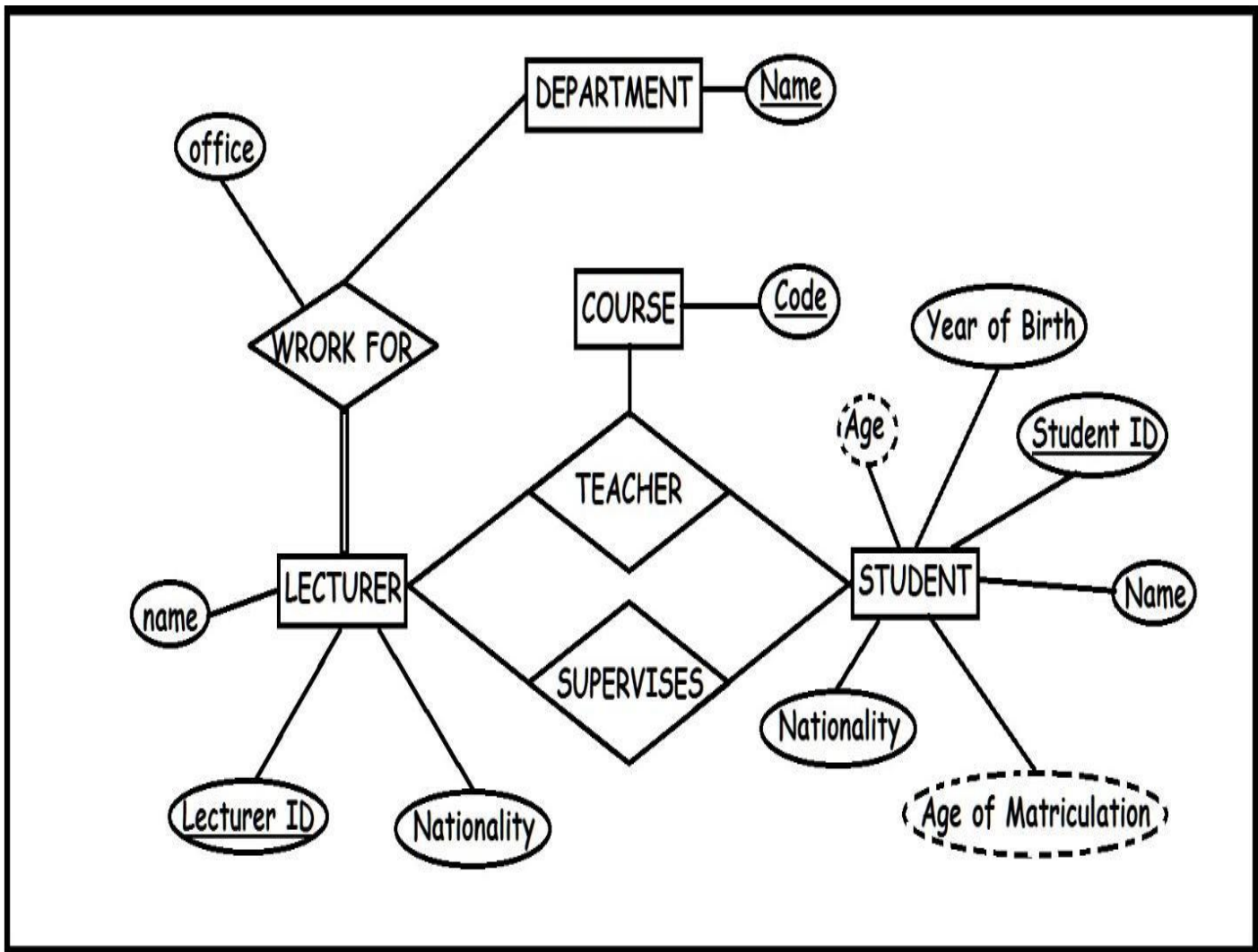


Construct an E-R diagram for a hospital

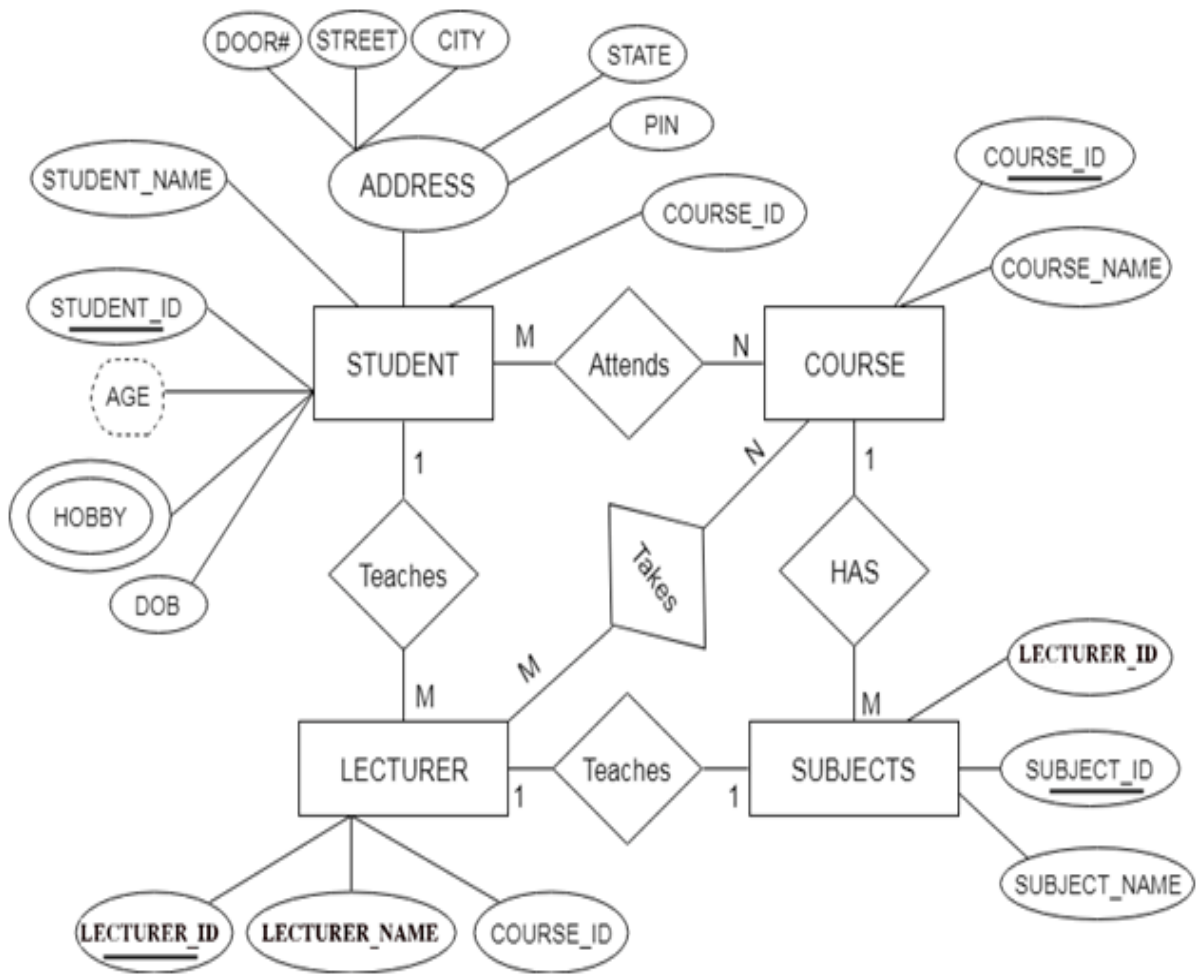
- with a set of patients and a set of medical doctors.
- Associate with each patient a log of the various tests and examinations conducted.



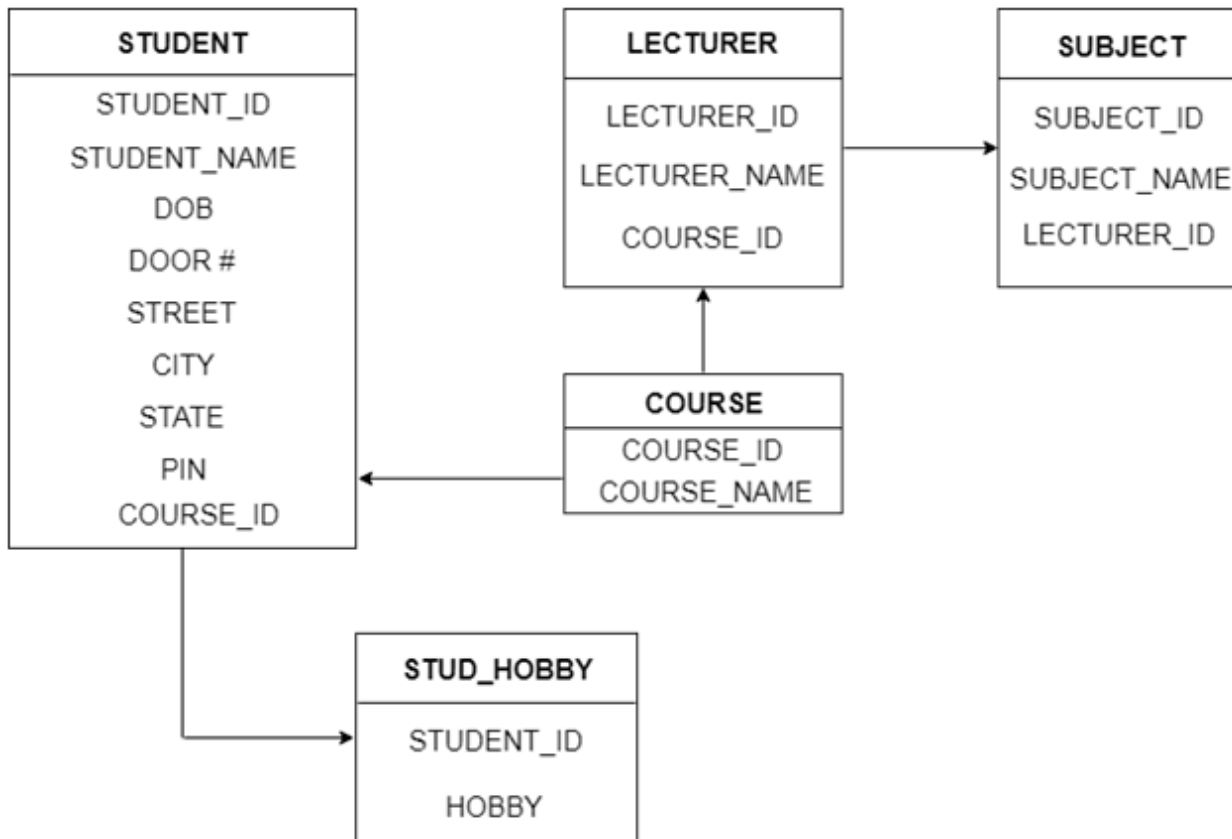
ER DIAGRAM FOR UNIVERSITY



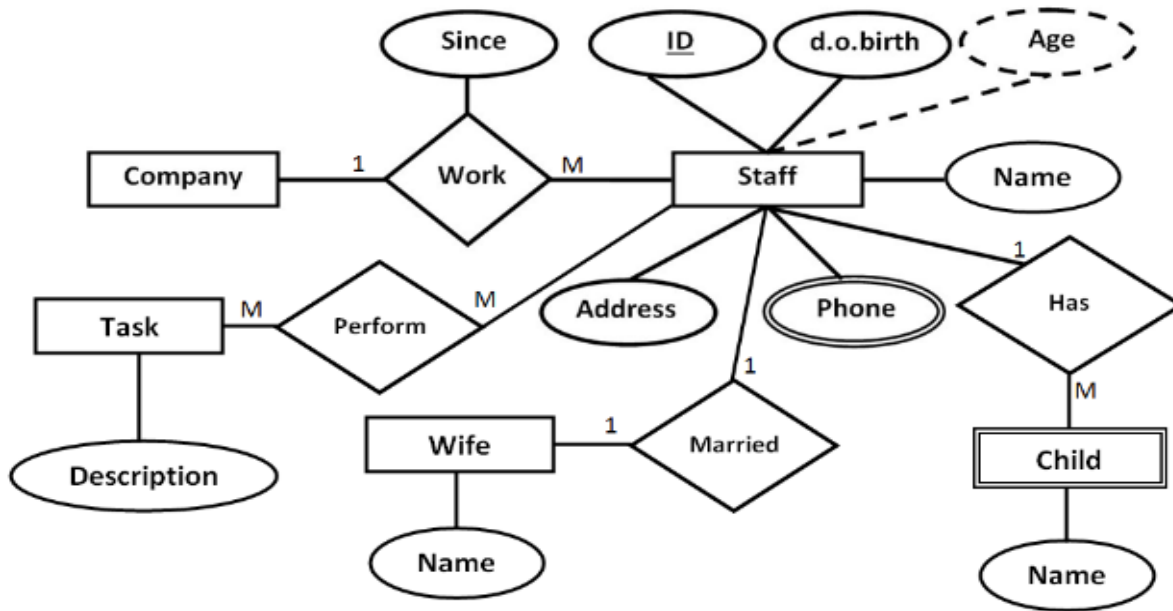
- Mapping Process (Algorithm)
 - Create table for each entity.
 - Entity's attributes should become fields of tables with their respective data types.
 - Declare primary key.



ER MAPPING TO RELATIONAL DATABASE



COMPANY ER DIAGRAM



The relational schema for the ER Diagram is given below as:

Company(CompanyID , name , address)

Staff(StaffID , dob , address , *WifeID*)

Child(ChildID , name , *StaffID*)

Wife (WifeID , name)

Phone(PhoneID , phoneNumber , *StaffID*)

Task (TaskID , description)

Work(WorkID , *CompanyID* , *StaffID* , since)

Perform(PerformID , *StaffID* , *TaskID*)

Problem

Draw of ER model for company considering the following constraints –

- In a company, an employee works on many projects which are controlled by one department.
- One employee supervises many employees.
- An employee has one or more dependents.

- One employee manages one department.

Solution

Follow the steps given below to draw an ER model for the company –

Step 1 – Identify the entity sets

The entity set has multiple instances in a given business scenario.

As per the given constraints the entity sets are as follows:

- Employee
- Department
- Project
- Dependent

Step 2 – Identify the attributes for the given entities

- Employee – The relevant attributes are name, ssn, sex, address, salary.
- Department – The relevant attributes are Name, number of employees, location.
- Project – The relevant attributes are number, name, location.
- Dependent – The relevant attributes are name, sex, birth date, relationship.

Step 3 – Identify the Key attributes

- SSN is the key attribute for Employee.
- Number is the key attribute for the Department.
- Number is the key attribute for Project.
- Name is the key attribute for a dependent entity.

Step 4 – Identify the relationship between entity sets

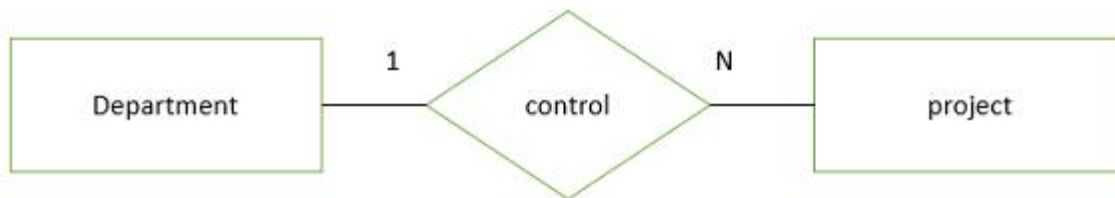
- Multiple employees work for a single department and one department has multiple employees. Hence, the relationship between employee and department is many to one.



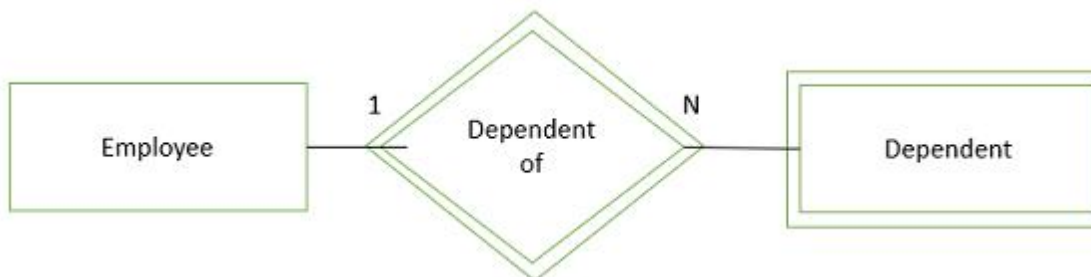
- Single employee manages the entire department and one department is handled by one manager.



- Each department is controlled by the number of projects and the number of projects handled by a single department. Hence, the relationship between department and project is one-to-many.



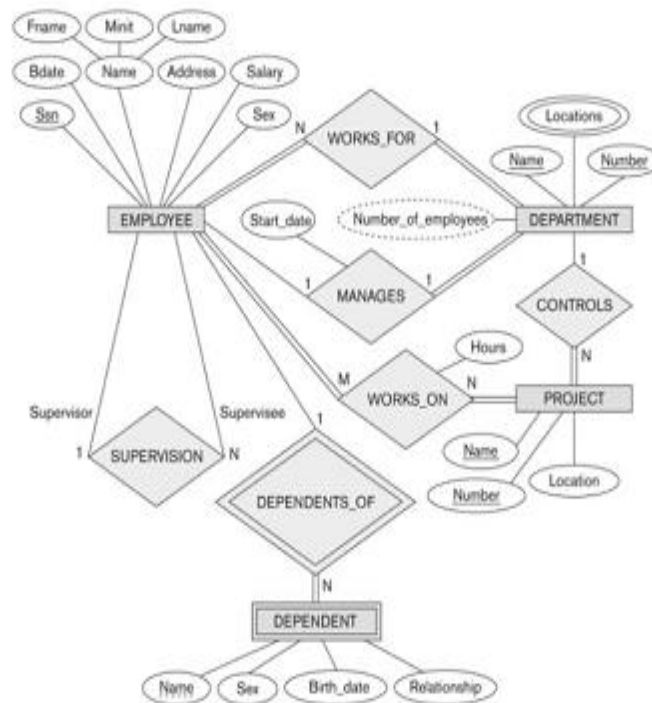
- One employee dependent is of multiple dependent and multiple dependents are of single employee. Hence, the relationship between employee and dependent is one- to-many.



Dependent is a weak entity which is denoted by a double rectangle, the entity which does not contain a primary key is known as weak entity.

Step 5 – Complete ER diagram

The complete ER diagram is as follows –



File organisation

- A database consists of a huge amount of data. The data is grouped within a table in RDBMS, and each table has related records.
- A user can see that the data is stored in form of tables, but in actual this huge amount of data is stored in physical memory in form of files.
- **File** – A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.

Types of File Organizations –

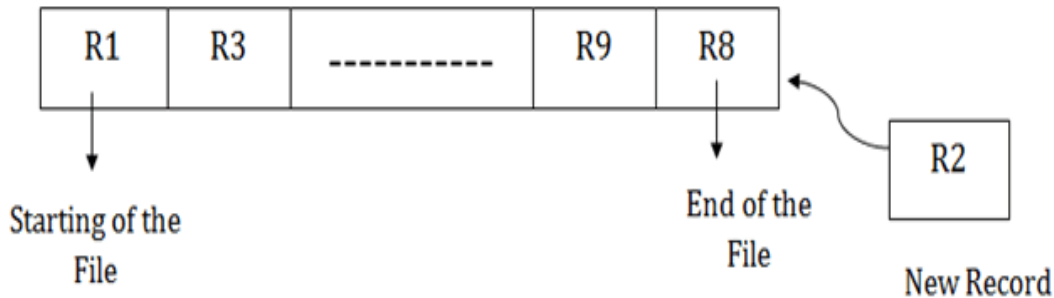
- Various methods have been introduced to Organize files. These particular methods have advantages and disadvantages on the basis of access or selection .
- Sequential File Organization(SORTED)
 - Pile File Method
 - Sorted File Method
- Heap File Organization (UNSORTED)
- Hash File Organization
 - External
 - Internal
- **Sequential File Organization:** (SORTED) This method is the easiest method for file organization. In this method, files are stored sequentially.
- This method can be implemented in two ways:

Pile File Method:

- It is a quite simple method. In this method, we store the record in a sequence, i.e., **one after another. Here, the record will be inserted in the order in which they are inserted into tables.**
- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.

Insertion of the new record:

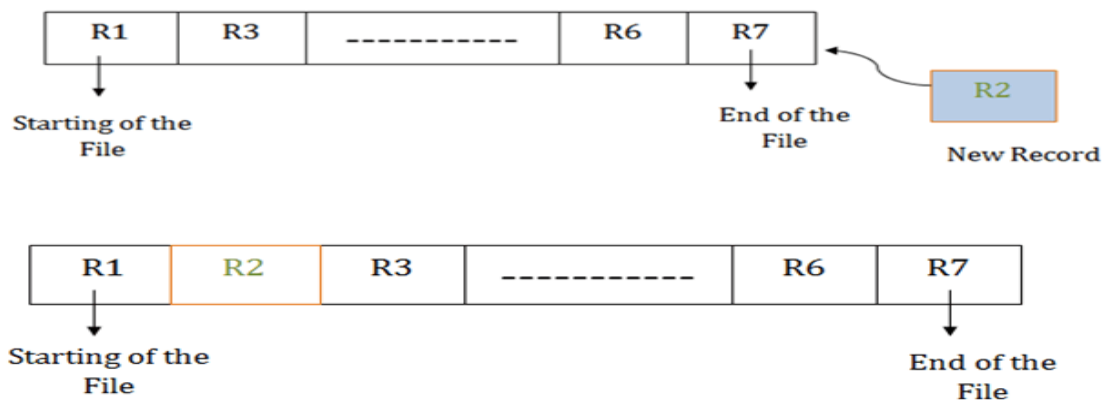
Suppose we have four records R1, R3 and so on upto R9 and R8 in a sequence. Hence, records are nothing but a row in the table. Suppose we want to insert a new record R2 in the sequence, then it will be placed at the end of the file. Here, records are nothing but a row in any table.



• Sorted File Method:

- In this method, the new record is always inserted at the file's end, **and then it will sort the sequence in ascending or descending order**. Sorting of records is based on any primary key or any other key.
- In the case of modification of any record, it will update the record and then sort the file, and lastly, the updated record is placed in the right place.

Suppose there is a preexisting sorted sequence of four records R1, R3 and so on upto R6 and R7. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file, and then it will sort the sequence.



• Pros of sequential file organization

- It contains a fast and efficient method for the huge amount of data.

- In this method, files can be easily stored in cheaper storage mechanism like magnetic tapes.
- It is simple in design. It requires no much effort to store the data.
- This method is used when most of the records have to be accessed like grade calculation of a student, generating the salary slip, etc.
- This method is used for report generation or statistical calculations.
- **Cons of sequential file organization**
- It will waste time as we cannot jump on a particular record that is required but we have to move sequentially which takes our time.
- Sorted file method takes more time and space for sorting the records.

Hashing technique:

- Hashing is a method, which provides very fast access to records on certain search conditions.
- The search condition must be an equality condition on a single field called the hash field of the file.
- The hash field is also a key field of the file, in which case it is called hash key.
- **The basic terms associated with the hashing technique are:**
- **1. Hash table:** It is simply an array that is having address of records.
- **2. Hash function:** It is the transformation of a key into the corresponding location or address in the hash table. (it can be defined as function that takes key as input and transforms it into a hash table index)
- **3. Hash key:** Let R be a record and its key hashes into a key called hash key.

Assume a table with 8 slots :

Hash key = key % table size

$$4 = 36 \% 8$$

$$2 = 18 \% 8$$

$$0 = 72 \% 8$$

$$3 = 43 \% 8$$

$$6 = 6 \% 8$$

[0]	72
[1]	
[2]	18
[3]	43
[4]	36
[5]	
[6]	6
[7]	

Types of Hashing

1. Internal Hashing: The hashing technique employed for internal files is known as Internal Hashing. Internal files are the files stored in primary memory
2. External Hashing: The hashing technique employed for disk files is known as External hashing

Internal hashing: For internal files, hash table is an array of records, having array index ranges from 0 to M-1, let us consider a hash function $H(K)$ such that $H(K) = \text{key MOD } M$ which produce a remainder between 0 and M-1 depending on the value of key, this value then used for the record By

Hashing is typically implemented as a **hash table** through the use of an array of records. Let the array index range is from 0 to M-1, as shown in Figure(a); then we have **M slots** whose addresses correspond to the array indexes. We choose a hash function that transforms the hash field value into an integer between 0 and M-1. One common hash function is the $h(K) = K \bmod M$ function, which returns the remainder of an integer hash field value K after division by M ; this value is then used for the record address.

- 🍌 Array of M positions for use in internal hashing.
- (f) Collision resolution by chaining records.



There are numerous methods for collision resolution, including the following:

Chaining. For this method, various overflow locations are kept, usually by extending the array with a number of overflow positions. Additionally, a pointer field is added to each record location. A collision is resolved by placing the new record in an unused overflow location and setting the pointer of the occupied hash address location to the address of that overflow location. A linked list of overflow records for each hash address is thus maintained, as shown in Figure (b).

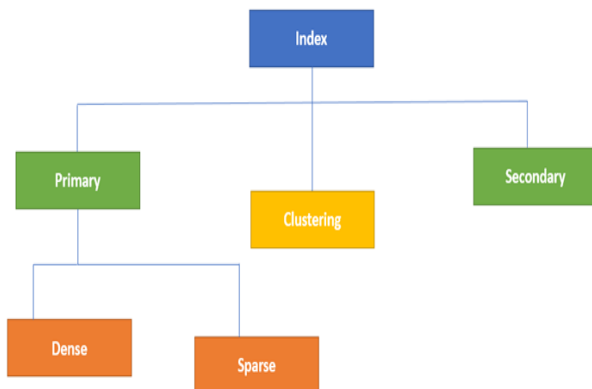


Multiple hashing. The program applies a second hash function if the first results in a collision. If another collision results, the program uses open addressing or applies a third hash function and then uses open addressing if necessary.

- **Indexing** is a data structure technique which allows you to quickly retrieve records from a database file.
- An Index is a small table having only two columns.
- The first column comprises a copy of the primary or candidate key of a table.
- Its second column contains a set of pointers for holding the address of the disk block where that specific key value stored.

• **Types of Single-level Ordered Indexes**

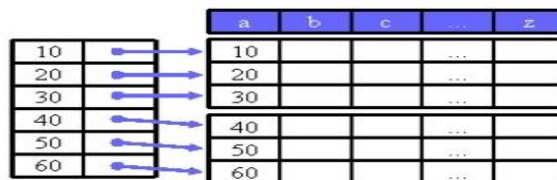
- Primary Indexes
- Clustering Indexes
- Secondary Indexes
- Multilevel Indexes
- Indexes on Multiple Keys



- **Primary Index** – Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Secondary Index** – Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.

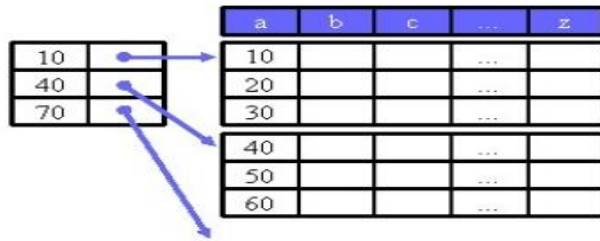
- **Clustering Index** – Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.
- **Primary Index**
- Primary Index is an ordered file which is fixed length size with two fields.
 - The first field is the same as a primary key and second,
 - field is pointed to that specific data block.
 - In the primary Index, there is always one to one relationship between the entries in the index table.
- The primary Indexing in DBMS is also further divided into two types.
 - Dense Index
 - Sparse Index
- In a dense index, a record is created for every search key valued in the database. This helps you to search faster but needs more space to store index records.
- In this Indexing, method records contain search key value and points to the real record on the disk.

✓ A **dense** index has one index entry for every search key value



- every data record is represented in the index
 - an existence search of a record may be done via index only
 - the record is directly found in a block using the pointer, i.e., no search within the block
- It is an index record that appears for only some of the values in the file. Sparse Index helps you to resolve the issues of dense Indexing in DBMS.
 - In this method of indexing technique, a range of index columns stores the same data block address, and when data needs to be retrieved, the block address will be fetched.

A **sparse** index has one index entry for every data block, i.e., the key of the first record



- only one index field per block, i.e., less index data
- cannot find out if a record exists only using the index
- **Multilevel Indexing in Database** is created when a primary index does not fit in memory.
- In this type of indexing method, you can reduce the number of disk accesses to short any record and kept on a disk as a sequential file and create a sparse base on that file.

Multi-key indexes

Multi-key indexes are **used to index all the elements of an array, or all the elements and/or all the keys of a map**. As a result, for each table row, the index contains as many entries as the number of elements/entries in the array/map that is being indexed (modulo duplicate elimination).

Unit-03

Data modelling using entity relational model

ER-Model is used to represent objects in the real world and of relationship among these objects, which represents the overall logical structure of a database. We have also seen that the data model that is independent of both the DBMS software and the hardware is the conceptual model. ER-model is a high level conceptual data model developed by Chen in 1976 to facilitate database design. The ER model is extremely useful in mapping and interaction of real world enterprises onto a conceptual schema. The main usage is in the design of the database.

*****Explain High level Conceptual data models for database design(10 M)**

The database design begins with the software requirement specification of the given problem. The precise requirement collection is very important to have a good database design. This is then to be analyzed.

The next step is to create a conceptual schema that describes the data type, relationships, and constraints. This is called the conceptual design. It deals with the high-level data descriptions of the problem and hence implementation details are hidden.

The implementation model consists of two phases:

- ❖ Logical database design
- ❖ Physical database design

The database design process consists of the following steps.

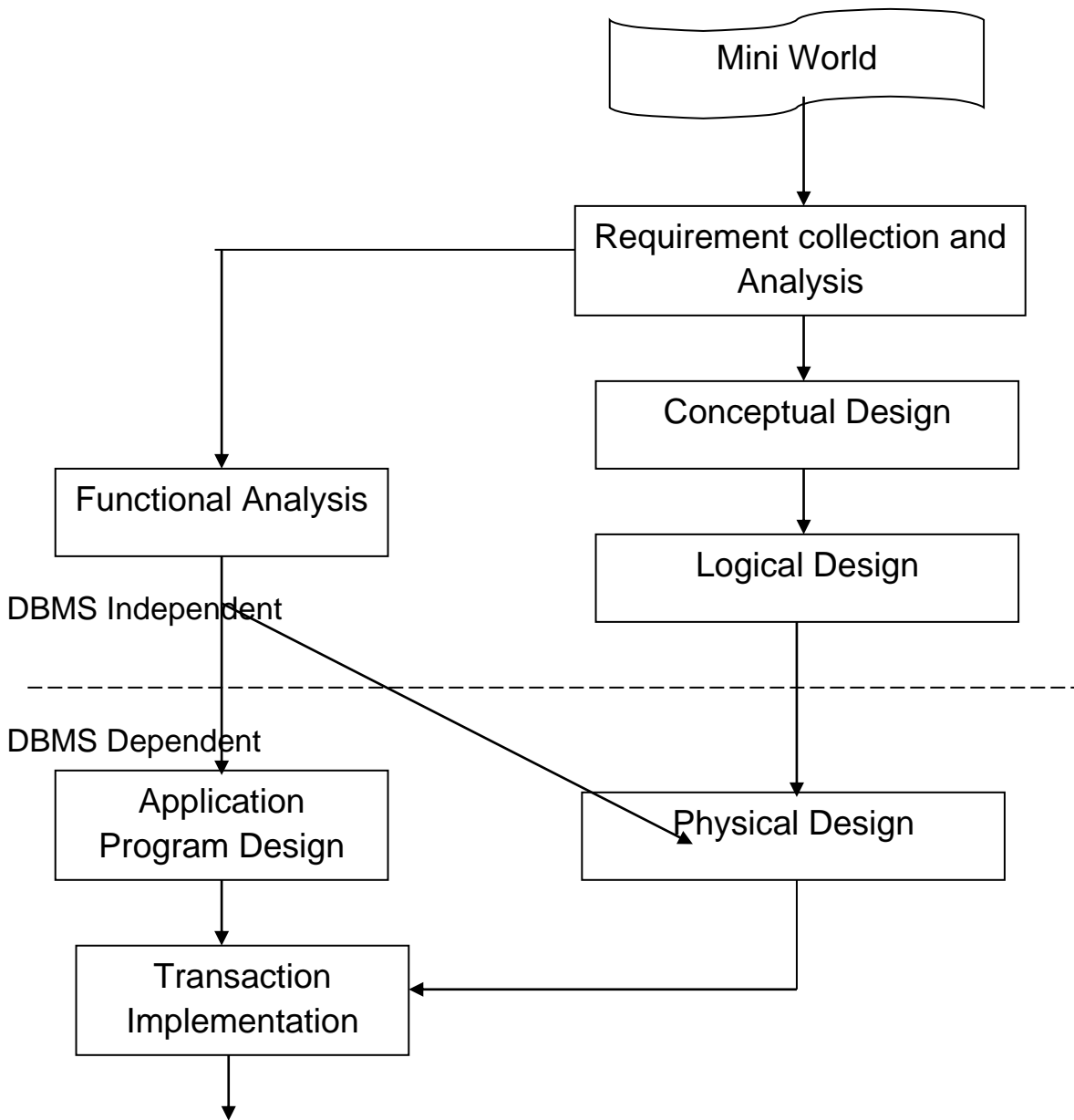
5. Requirement collection and analysis
6. Conceptual design
7. Logical design [Implementation of data model]
8. Physical design

Requirement collection and analysis: This phase is also called feasibility phase. During this phase, the database designers interview database users. This helps them to understand and document their data requirements.

In this phase through the interviews and reviewing all related documents and policies in the organization, the following items are identified.

- e. Clear and concise definition of the problem.

- f. Local dependency lists.
- g. Local dependency diagrams.
- h. Local Schema.



Application Program

Conceptual design: Conceptual schema for the database is created using a high-level conceptual data model. The conceptual schema describes data requirements of the users, entity types, relationships and constraints. These concepts do not include implementation details.

Logical design [Implementation of data model]: This step involves implementation of data model. Since the conceptual schema is transformed from the high level model into the implementation data model, this step is also known as **Data model mapping**.

Logical database design is the process of designing a model of the information in an enterprise based on the chosen database model.

This results in a database schema in the implementation data model such as relational or object-relational database model.

Physical design: During this step, the internal storage structures, indexes, access paths and file organization for the database files are specified.

Physical database design is the process of describing the implementation of the database on the disk. It describes the internal storage structures, indexes, access paths, base relations, security issues and constraints.

Database design involves the following steps:

5. Identifying all the required files.

In database terminology, files are called record types.

6. Identifying the files of each of those record types.

Fields in database terminology are called attributes. Note that attributes in semantic object oriented database are called properties.

7. Identifying the primary key of each of these files.

Note that the primary key is field that uniquely identifies a specific record in the file.

8. Identifying the relationships between record types.

An Example – Company Database:

8. The company consists of several departments and every department has a manager. It is also required to record when he has become the manger.
9. Several employees' works for a department.
10. A department have several locations.
11. Each department controls several projects.
12. Employee details like Social Security Number (SSN), name, the department he works for etc are to be stored.
13. An employee works for only one department, but he/she can work on more than one project.
14. We need to know the employee dependent details for some specific purpose like PF, Insurance and etc.

Entity types, entity sets, attributes and keys:

Entity: The E-R model describes data as entities, relationships and attributes:

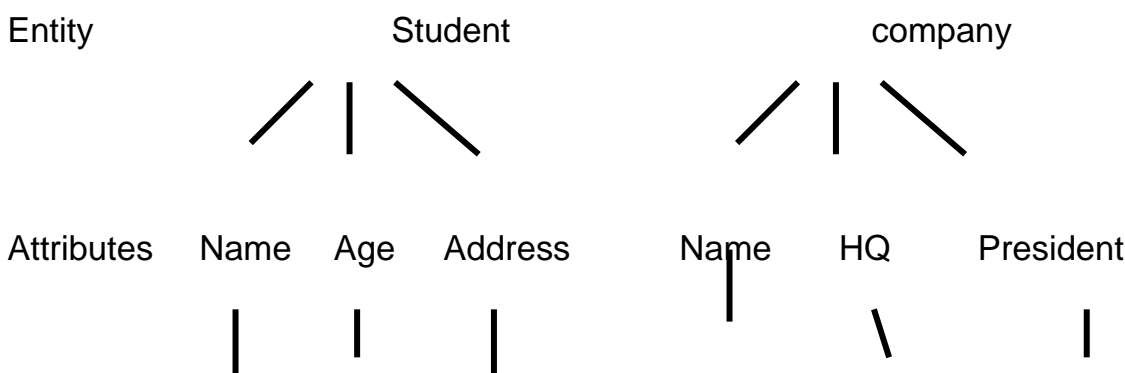
An **entity** is a thing in the real world with an independent existence that is distinguishable from all other objects.

Example: Each employee in an organization is an entity. A company, a job, a book, etc all are entities. Each entity has particular properties called attributes that describes it.

An **attribute** is a property that describes an entity.

Example:

1. The attributes of a person's entity are his name, address, job, salary etc. for an entity each attributes will have a value.
2. **student (regno, name, age, address)**
3. **Employee (ID, name, dept, salary)**
4. **Company (name, HQ, president)**



Value Ramya 25 Peenya INFOSIS Bangalore Murthy

Types of attributes: Each attribute is associated set of values called domain.

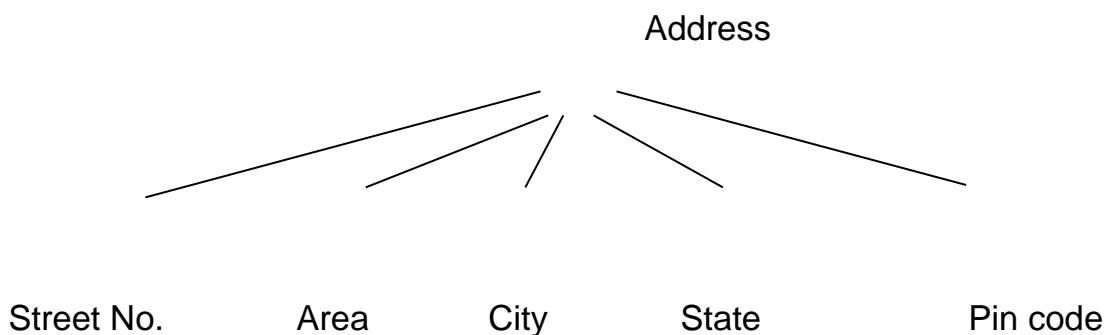
1. Simple or atomic attributes: These attributes can't be subdivided into further. In other words, the atomic attributes hold single value and not composed of any attributes.

Example: Regno, ID, DOB, age, Deptno, sex.

2. Composite attributes: Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.

Example: 1. the address attribute can be subdivided into house no, street_name, area, city and country.

2. The name attributes, which can be composed of first name, middle name, and last name.



3. Single valued attributes: An attribute that can take only one value at a time is called single valued attributes. Usually, for a particular entity, each attributes will have single value.

Example: The age attribute will have a single value.

4. Multi valued attributes: Some attribute have more than one value for the same attribute and are called multi valued attributes.

Example: A **college degree** attribute can have multiple values.

Degree [B,Sc, MCA, PhD]

Admin [Bangalore, Mumbai]

Carcolor [Red, Black].

5. Derived attributes: If the value of an attribute can be derived from some other attributes, then such attributes are called derived attributes. Since age can be determined by knowing birthday and current date. The age attribute is said to be derived attribute.

Another example, gross pay of an employee this can be derived by knowing basic pay, allowances, and deduction for employee.

6. Stored attributes: The value of certain attributes cannot be obtained or derived from some other attributes, that is, they are not derived from any other attributes, these types of attributes is called as stored attributes.

Example: Birth date, Book_ID, SSN, Part#.

The attribute SSN is direct attribute and is not dependent on any other attribute or derivable from one or more other attributes.

7. Null attributes: A null attributes used when an attributes does not have any value. A null value does not mean that the value is equal to zero but indicate no value is stored for that attribute.

Example: Abatement number attribute of an address applies only to address that are in apartment buildings, and not to other types of residence such as single family home.

2. Email: All employees in an employee database may not have e-mail address.

8. Key attributes: An entity type usually has an individual whose values are distinct for each individual entity. Such an attribute is called a key attribute. These attributes which uniquely identifies every instance of the entity is termed as the primary key.

Entity types, Entity sets, Key attributes, value sets.

A database usually contain group of entities that are similar. For example, a company with number of employees may have to store similar information for each of employees. These employee entities share the same attributes, but each entity has its own values for each attributes.

Entity set: It is a set of entities of the same type that share the same properties or attributes. The set of all employees working for the same department can be defined as entity set employee but each entity has own values for each attributes.

Entity Type: AN entity type defines a set of entities that have the same attributes. Each entity type in the database is described by its name and a list of attributes.

Entity type name	Employee			Company	
Attributes	Name,	Age,	Salary	Name,	HQ
Entity set [Extension]	Mahesh	E1 35	20,000	BPL	C1 Chennai
	Nagaraj	E2 32	15,000	Infosys	C2 Bangalore
	Manju	E3 30	25,000	Wipro	C3 Delhi

An entity type describes the schema or intention for a set of entities that share the same structure. The individual entities of a particular entity type are grouped into a collection or entity set. Which is also called the extension of the entity type.

Key attributes of an entity type:

A key attribute is a minimal set of attributes of an entity set, which uniquely identifies an entity in an entity set. A key may be a single attributes or may be more than one attribute.

An entity type usually an attribute whose values are distinct for each individual entity such an attribute is called a key attribute. Whose values can be used to identify each entity?

Example:

1. For the company entity, name can be the key attribute because no two companies can have the same name.
2. For the student entity, regno can be the key attribute.

Value sets (Domain) of attributes:

A set of values that may be assigned to the attributes of each individual entity, in an entity set is called the value set or domain.

Example:

1. For an example entity, if age limit is 20 to 58, then the value set (DOMAIN) attributes age consists of integer from 20 to 58.
Age: Domain is [20-58]
2. For salary attribute, the value set may be again a range from minimum of 5,000 to maximum 50,000.
Salary: Domain is [5000- 50000].

Mathematically it can be represented as follows:

$$A: E \rightarrow P(V)$$

Where

A is an attribute.

E is an entity set.

V is value set.

P (V) is a power set, which includes all the subsets of V.

The value of attribute A of entity 'e' is referred by A (e). It includes single attribute or composite attribute or even null attributes. If the attribute is single, the value set contains only one value. If it is multi valued attribute, there will be several values in value set. For a composite attribute the value set V is obtained by the Cartesian product of the value sets of simple component attributes of A.

Relationship:

Definition: A relationship is an association among two or more entities. A relationship captures how two or more entities are related to one another.

Example:

1. An **'owns'** relationships between a company and computer
2. A **'Supervises'** relationships between an employee and department.
3. A **'Performs'** relationships between an artist and song.
4. A **'Proved'** relationship between a mathematical and a theorem.

Relationships are represented as diamonds symbol, connected by lines to each of the entities in the relationship in an ER diagram.

Relationships types, sets and instances:

Relationship types: A relationships type R among 'n' entity' type E1, E2..... Defines a set of associations among entities from these types.

Example: Relationship type **works_for** between two entities of company database, namely employee and department.

Relationship Instance: Each relationship instance **ri** is an association of entities where the association includes exactly one entity from each participating entity type.

Example: Each relationship instance in **works_for** relationship type associates an employee entity and one department entity.

Relationship Set: A collection of relationships of same type is called the relationship set.

Employee

Works_for

Department

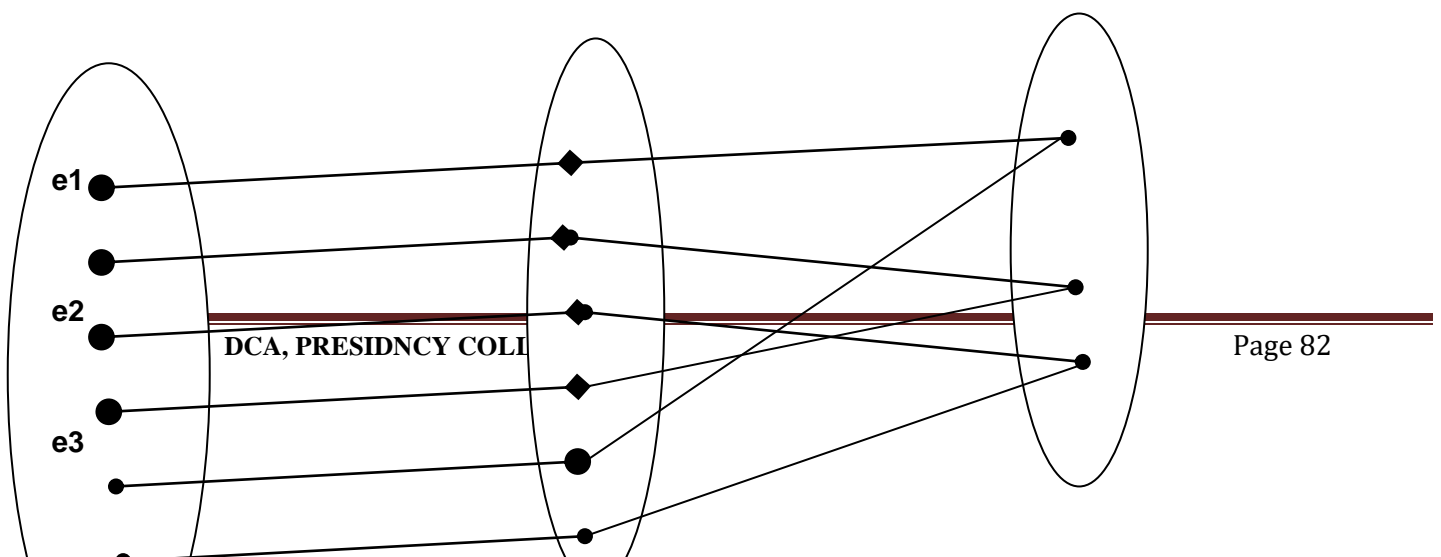


Fig: Some instances of the works_for relation ship.

Explain cardinality in dbms or Explain different types of relationship in dbms.

Relationship between entity sets: relationship between entity sets are of four types:

1. Many to Many (M : N)
2. One to Many (1 : M)
3. Many to One (M:1)
4. One to One (1:1)

One to One relationship (1:1) relationships: An entity in A is associated with at most one entity in B and vice versa.



The relationship between a department and a manger is usually one to one; there is only one manger per department and manger mange's only one department.

One to Many relationship (1:M) relationships: An entity in A is associated with any number in B, an entity in B however can be associated with at most one entity in A.



A 1: N relationship exists from the entity department to the entity employee because there are several employees working for the department.

Many to One relationship (M:1) relationships: An entity in A is associated with at most one entity in B. An entity in B however it can be associated with any number of entities in A. many Depositors deposit to single account.



Many to Many relationship (M:N) relationships: An entity in A is associated with any number of entities in B and, an entity in B is associated with any number of entities in A.



Employee can work several and that several employees can work on a project.

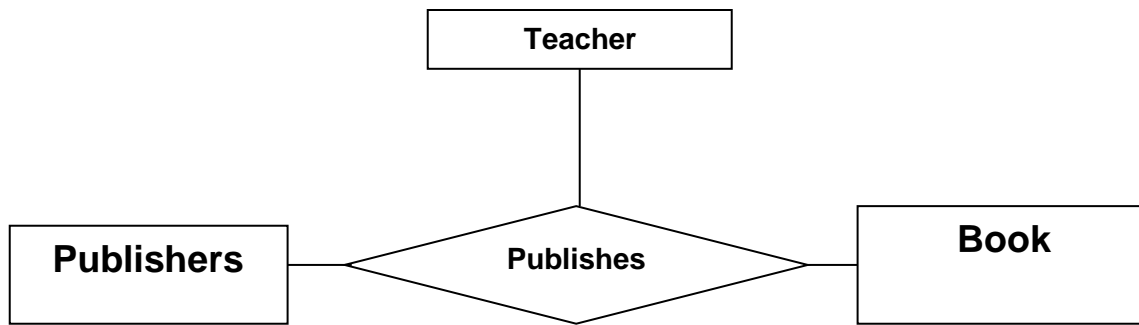
Degree of a relationship: Degree of a relationship type is the number of entities participating in the relation.

Binary relationship: A relationship of degree 2 is called binary relationship. Example, the figure shows an example of binary relationships.



The publishes relationships is of degree two because they are two entities publisher and book which are related.

Ternary relationship: Relations of degree three are called ternary relationship.



Allocating file blocks on disks: There are various methods for allocating the blocks of a file on a disk.

- **Contiguous allocation:** Here, file blocks are allocated to consecutive disks blocks. This makes reading the whole file very fast using double buffering, but it makes expanding the file difficult.
- **Linked allocation:** In linked allocation each file block contains a pointer to next file block. A combination of the two allocates clusters of consecutive disk blocks and the cluster are linked together clusters are sometime called segments or extents.
- **Indexed allocations:** Here one or more index blocks contain pointers to the actual file blocks.

Files of ordered records [sorted files]: The records of file can be physically ordered based on the values of one of their fields called the ordering fields. If the ordering field is also a key field of the file, a field guaranteed to have a unique value in each record then the field is also called a ordering key for the file.

Advantages:

1. Reading the records in order of the ordering field values becomes extremely efficient since no sorting is required.

2. Finding the next record in an ordering field usually requires no additional block access, because the next record is in the same block as the current one.
3. Using a search condition based on the value of an ordering key field results in faster access when the binary search technique is used.

Inserting and deleting records are expensive operations for an ordered file because the records must remain physically ordered. To insert a new record, we must find the correct position in the file, based on its ordering field value, and then make space in the file to insert the record in that position. For a large file this can be very time consuming. For record deletion the problem is less severe, if we use deletion markers and recognize the file periodically.

Hash file → a direct file

Explain Hashing technique:

Hashing is a method, which provides very fast access to records on certain search conditions. The search condition must be an equality condition on a single field called the hash field of the file. The hash field is also a key field of the file, in which case it is called hash key.

Key value → Hash function → Address.

The basic terms associated with the hashing technique are:

1. **Hash table:** It is simply an array that is having address of records.
2. **Hash function:** It is the transformation of a key into the corresponding location or address in the hash table. (it can be defined as function that takes key as input and transforms it into a hash table index)
3. **Hash key:** Let R be a record and its key hashes into a key called hash key.

Internal hashing: For internal files, hash table is an array of records, having array index ranges from 0 to M-1, let us consider a hash function $H(K)$ such that $H(K) = \text{key MOD } M$ which produce a remainder between 0 and M-1 depending on the value of key, this value then used for the record address. The problem with most hashing function is that they do not guarantee that distinct value will hash to distinct address, situation that occurs when two non-identical keys are hashed into the same location.

For example, let us assume that there are two identical keys $K_1 = 352$ and we have some mechanism to convert key values to address. Then the simple hashing function is

$$H(K) = K \text{ Mod } 10$$

$H(K)$ = Produces bucket address.

To insert a record with key value K , we must hash its key first. For example, consider $H(K1) = K1 \% 10$ will get 2 has the hash value. The record with key value 342 is placed on the location 2, another record with 352 as its key value produces same hash address, i.e., $H(K1) = H(K2)$, we try to place the record at the location where the record with key $K1$ is already stored their occurs a collision.

The process of finding another position is called **collision resolution**.

The methods that can be employed for collision resolution are,

- **Open addressing:** With open addressing we resolve the hash clash by inserting the record in the next available free or empty location in the table.
- **Chaining:** Various overflow locations are kept, a pointer field is added to each record and setting the pointer to address of that overflow location.

Multiple hashing: If the first hash function results in a collision, then the program applies a second hash function, if another collision results the program uses open addressing or applies a third hash function and then uses open addressing if necessary.

Functional dependencies and normalization for relational databases

A relational model consists of three basic components:

1. A set of domains and a set of relations
2. Operations on relation
3. Integrity rules

Introduction to normalization: normalization is the process of building database structures to store data, because any application ultimately depends on its data structures. If the data structures are poorly designed, the application will start form poor foundation. Normalization is the formal process for deciding which attributes should be grouped together in a relation. Normalization serves as a tool for validating and improving the logical design, so that logical design avoids unnecessary duplication of data, i.e., it eliminates redundancy and promotes integrity. In normalization process we analyze and decompose the complex relations into smaller, simpler and well structured relations.

Definition of important terms:

1. **Super key:** It is a set of one or more attributes that taken collectively, allows us to identify uniquely an entity in the entity set.
 - The combination of customer_name and social_security_numbers is a super key for the entity set customer.
 - The customer_name attribute is not a super key because several people might have the same name.
 - The social_security_number attribute could also be super key since it is sufficient to distinguish one customer entity from another.
2. **Candidate key:** A super key may contain extraneous attributes. A super key for which no proper subset is a super key. i.e., minimal super key is called candidate key. The combination (customer_name, SSN) forms super key, the attribute SSN alone is a candidate key.
3. **Primary key:** It is a candidate key chosen by the database designers as the principal means of identifying entities within an entity set.
4. **Prime attribute:** An attribute of relation schema **R** is called a prime attribute of **R** if it is a member of some candidate key of **R**.
5. **Non prime attribute:** An attribute is called non prime if it is not a prime attribute i.e., it is not a member of candidate key.
6. **Weak entity set:** An entity set not having sufficient attributes to form a primary key is called weak entity set.
7. **Strong entity set:** An entity set that has a primary key is termed a strong entity set.
8. **Foreign key:** A set of attributes in a relation is a foreign key if it satisfies the following conditions.
 - It should have the some domain as the primary key attributes of another relation schema and is said to refer to this relation.
 - A value of foreign key in tuple t1 either occurs as a value of primary key for some tuple t2 in another relation.
9. **Composite key:** If a key has more than one attribute, it is called composite key.

Example: Consider the relation schema BRANCH with attributes **branch_no**, **branch_city**, identify super key, candidate key and primary key.

{Branch_name, Branch_city} → Super key.

{Branch_name} → Candidate key.

{Branch_name} → Primary key.

Insertion of a new data values to a relation should be possible without being forced to leave blank field for some attributes.

Deletion of a tuple should be possible without losing vital information unknowingly.

Updating a value of an attribute in tuple should be possible without exhaustively searching all the tuple in the relation.

10. Relation key: Given a relation, if the value of an attribute X uniquely determines the value of all other attribute in a row, then X is said to be the key of that relation. Sometimes more than one attribute is needed to uniquely determine other attribute in relation row. In that case, such a set of attributes is the key.

******Normalization:**

Database designers proposed designed guidelines in the form of normal forms to relations starting from first normal form to fifth normal form. The process of converting relations from one normal form to another normal form is known as normalization.

Functional dependencies play very important role in conversion process. The step involved in normalization is as follows, start from 1st normal form [1NF], then go to second normal form [2NF], then to third normal form [3NF] and Boyce coad normal form [BCNF], all of these are based on FD's and key concepts.

Another restriction that is imposed is that if a relation is in 2NF, it should be in 1NF and if a relation is in 3NF, it should be in 2NF, and finally a relation is in BCNF, it should be in 3NF.

First normal form [1NF]: A relation schema R is the 1NF if every attribute of R takes only a single value [atomic value], when a table contains multi-valued attributes we say that is in un-normalized form.

Example: Consider the employee table **Employee [SSN, Name, Age, Dependent]** shown in figure (a). This relation is not in 1NF, because of the repeating group or multi-valued attributes dependents.

Employee

SSN	Name	Age	Dependent
1234	Pooja	20	{Deepa, Divya}
1235	Praveen	32`	Pradeep

Fig (a): Employee table with multi-valued attribute.

Employee

SSN	Name	Age	Dependent
1234	Pooja	20	Deepa
1234	Pooja	20	Divya
1235	Praveen	32`	Pradeep

Fig (b): Employee table which is in 1NF.

Second normal form [2NF]: A relation schema R is in 2NF if every non-prime attribute in R is fully functionally dependent on primary key [or every candidate key].

Example: Consider the EMP-PROJ relation shown in fig (c) with attributes {SSN, PNO, Hours, Ename, Pname, Ploc}, we shell show that this relation is not in 2NF as per the definition.

EMP_PROJ

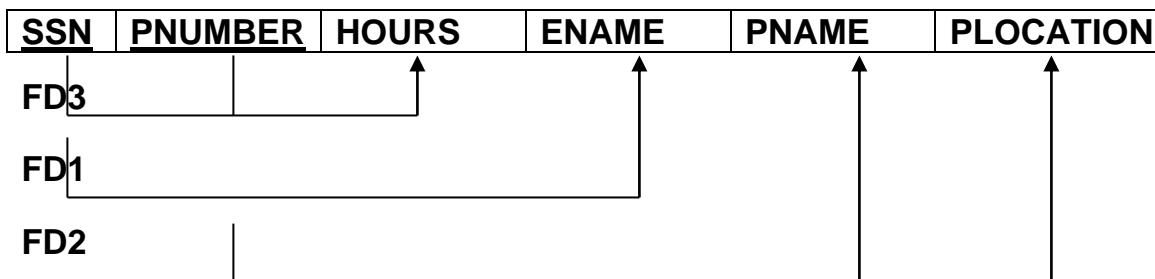


Fig (c): Emp_Proj table that is not in 2NF.

FD1: SSN → ENAME

FD2: PNUMBER → {PNAME, PLOCATION}.

FD3: {SSN, PNUMBER} → HOURS

The above relation is in first normal form but not in 2NF, because FD1 AND FD2. According to the definition every non-prime attribute should be fully functionally dependent on primary key. When we remove PNO from the key, the FD stills hold [FD1] and therefore this relation must be decomposed. It can easily be done based on the three FD's shown already and the resulting tables are shown in figure (d).

SE

PPP

SPH

<u>SSN</u>	EName	
<u>PNUMBER</u>	PNAME	PLOC
<u>SSN</u>	<u>PNUMBER</u>	HOURS

Fig (d): Relation that are in 2NF.

Three decomposed relation are SE {SSN, Ename}, PPP {Pnumber, Pname, Ploc}, SPH{SSN, PNUMBER, Hours} and the underlined attributes are the key of respective relations. Another way to prove that those relation are in 2NF, is that each relation has a single candidate key.

Third normal form [3NF]: A relation schema R is in 3NF, if it is in 2NF and non-prime attribute of R is transitively dependent on the primary key.

Example: Consider the relation EMP_DEPT {SSN, Name, DOB, Address, DNO, Dname, MGRSSN} shown in the figure (e) that contains information about employee and department.

EMP_DEPT

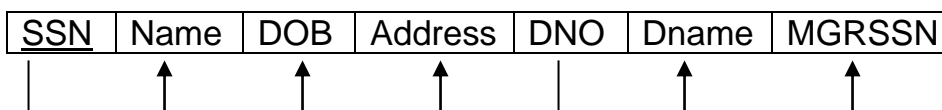


Fig (e): A relation EMP_DEPT [not in 3NF].

FD1: SSN → {Name, DOB, Address, DNO}

FD2: DNO → {Dname, MGRSSN}

It is obvious that relation EMP_DEPT is not in 3NF, because of attribute DNO which transitive that is FD1 we write SSN→DNO and FD2 we write DNO→MGRSSN and clearly DNO is transitive attribute. Hence DNO is undesirable, because it is not a key of EMP_DEPT.

Therefore EMP_DEPT should be decomposed into two relations employee and department corresponding to FD1 and FD2. These tables are shown in figure (f).

Employee:

<u>SSN</u>	Name	DOB	Address	DNO
------------	------	-----	---------	-----

Department:

DNO	Dname	MGRSSN
-----	-------	--------

Fig (e): EMP_DEPT after decomposing (both are in 3NF).

Both the relation employee and department are in 3NF as no transitive dependency exists.

Boyce Codd Normal form [BCNF]: A relation is in BCNF, if and only if every determinate in table is a candidate key.

Example: Let us demonstrate the concept of BCNF with a relation as shown in figure (g). The relation viva schedule has attributes Enroll#, viva-date, viva-time, Examiner#, Room#.

Viva schedule

<u>Enroll#</u>	Viva-date	Viva-time	Examiner#	Room#
----------------	-----------	-----------	-----------	-------

Fig (g): Viva schedule table not in BCNF.

FD1: Enroll#, Viva_date → Viva_time, Examiner#, Room#

FD2: Examiner#, Viva_date, Viva_time → Enroll#

FD3: Viva_date, Examiner# → Room#

FD1, FD2 and FD3 are FD's for the above relations.

As per definition of BCNF, every determinate must be a candidate key. The determinate attribute set FD1 Enroll#, Viva_date is a candidate key, but not FD3, therefore this relation is not in BCNF and requires decomposition.

Viva_date

<u>Enroll#</u>	<u>Viva-date</u>	Viva-time	Examiner#
----------------	------------------	-----------	-----------

Examiner_Rooms

<u>Examiner#</u>	<u>Viva- date</u>	Room#

Fig (h): Viva Schedule after decomposition BCNF.

The decomposition rule for BCNF is to create a new table based upon the violating FD's. Now we can verify the insertion anomalies is avoided after decomposition compared to the original relation.

Relational Data Model and Relational Algebra

Relational model has established it self as the primary data model for commercial data processing application.

The relational model represents the database as a collection of relations having a set of rows and columns. Each of which is assigned a unique name, relation consists of a relational schema [structure of table] and relational instance [data in a table at a particular time], and there is a close correspondence between the concepts of table and the mathematical concepts of relation.

In relational model we use certain conversions, for instance a row is called as tuple and a column is termed as an attribute and a domain of a relational schema is a pool of legal values.

Characteristics of a relation:

1. The tuples in the relation need not be ordered.
2. Each tuple in the relation is an empty.

Domain: A domain D is set of automatic values. For each attribute empname, the domain is set of all empname.

Let D1: Denotes the attribute of empnames.

D2→ Set of all empno

D3→ Set of all empaddress

D4→ Set of all Phone.

D5→ Set of all salary.

In general, a table of n columns must be subset of $D_1 \times D_2 \times D_3 \times \dots \times D_{n-1} \times D_n$. In relational model terminology the data type describing the type of values that can appear in each column is called a **Domain**.

Entity set: The number of tuples in a relation is called as entity set.

Schema based constraints or Relational Model constraints:

These include:

- Domain constraint
 - Key constraint
 - Constraints on NULL.
 - Entity integrity constraints.
 - Referential Integrity Constraints
-
- **Domain constraint:** It specifies that, the value of each attribute A must be an atomic value from the domain $\text{dom}(A)$ for that attribute. It also specifies that, integer type holds only integer but not float value.
-
- **Key constraint:**
 - A relation is defined as a set of tuples.
 - All tuples in a relation must also be distinct, this means that no 2 tuples can have same value [maintains uniqueness], $t_1[\text{sk}] \neq t_2[\text{sk}]$, here sk is a super key.
 - In any schema there will be a super key (PK) to distinguish tuples.
 - Every relation can have at least one super key.
 - In general, a relation schema may have more than one key, in this case each of these keys is called a candidate key.
-
- **Constraints on NULL:** Another constraint on attribute specifies whether null values are or are not permitted. The constraint is specified as NOT NULL.

Entity integrity constraints: It states that no PK value can be NULL, because PK value is used to identify individual tuples in a relation. Having NULL values for the PK implies that we cannot identify some tuples.

▪ **Referential Integrity Constraints:** Tables can be related by common columns, a referential integrity constraint requires that the value in the foreign key matches a value in the primary key [unique or primary key of the same or different table referenced key].

Foreign Key: The condition for a foreign key specifies referential integrity constraints between the two relation schemas R1 AND R2.

An attribute A is foreign key, it specifies the following rules.

- A rule defined on a column in one table that allows insert or update of a row only if the value for the column [dependent table] matches a value in a column of the referenced table. “ **That is the attribute in foreign key have the same domain (values) as the primary key attribute primary key of another relation [parent table]**”.

RELATIONAL ALGEBRA:

Relational algebra is a procedural query language. It consists of a set of operations that takes one or two relations [tables] as input and produce a new relation as their result. For example a select operation on a relation produces another relation [table].

The relational algebra operations are divided into two groups.

1. Set operations

- ✚ UNION
- ✚ INTERSECTION
- ✚ DIFFERENCE and
- ✚ CARTESIAN

2. Developed specifically for the relational database.

- ✚ SELECT
- ✚ PROJECT and
- ✚ JOIN

Set theoretic operation: These are used to merge the elements of two sets in various ways, including UNION, INTERSECTION and DIFFERENCE three of these operations require the table to be union compatible. The two relations are said to be UNION compatible if the following conditions are satisfied.

- The two relations /tables (say R and S) that have the same number of columns (have the same degree).
- Each column of the first relation / table (R) must be the same data type as the corresponding column of the second relation or table (S).

Relation R and S

R	
ENO	NAME
1	Jyothi
2	Ganga
3	Girija
4	Ankith

R	
ENO	NAME
3	Girija
4	Ankith
5	Madhu
6	Tarun

UNION [U]: The result of this operation denoted by $R \cup S$, is a relation that includes all tuples that either in R or in S or in both. Duplication tuples will not appear in the output.

$$Q = R \cup S \quad Q = \{t / t \in R \vee t \in S\}$$

Q = R ∪ S	
ENO	NAME
1	Jyothi
2	Ganga
3	Girija
4	Ankith
5	Madhu
6	Tarun

INTERSECTION [∩]: The intersection operation selects the common tuple from the two relations.

$$Q = R \cap S \quad Q = \{t / t \in R \wedge t \in S\}$$

Q = R ∩ S	
ENO	NAME
3	Girija
4	Ankith

DIFFERENCE (-): The result of difference operation consist of all tuples in R but not in S

$$Q = R - S \quad Q = \{t / t \in R \wedge t \notin S\}$$

Q =R-S	
ENO	NAME
1	Jyothi
2	Ganga

CARTESIAN PRODUCTS (X): The Cartesian product or cross product is binary operation that is used to combine two relations. Assuming R and S as relations of n and m attributes respectively, the Cartesian products $R \times S$ can be written as, $R(A_1, A_2, A_3, \dots, A_n) \times S(B_1, B_2, B_3, \dots, B_m)$. The result of the above set operation is,
 $Q = R \times S(A_1, A_2, A_3, \dots, A_n, B_1, B_2, B_3, \dots, B_m)$.

The total number of columns in Q degree (Θ) = $n + m$.

$\left. \begin{array}{l} \text{Total no. of tuples} \\ \text{in } Q: \text{Count}(Q) \end{array} \right\} = \left\{ \begin{array}{l} \text{Number of Tuples in } R \\ \times \\ \text{Number of tuples in } S \end{array} \right\}$

R	
DNO	NAME
1	COMPUTER
2	MANAGEMENT
3	SCIENCE

S	
PNO	NAME
10	NETWORKING
11	PAYROLL

Cartesian product of R and S can be written as

R S $\leftarrow R \times S$			
R		S	
DNO	DNAME	PNO	PNAME
1	COMPUTER	10	NETWORKING
1	COMPUTER	11	PAYROLL
2	MANAGEMENT	10	NETWORKING
2	MANAGEMENT	11	PAYROLL
3	SCIENCE	10	NETWORKING
3	SCIENCE	11	PAYROLL

The relation R has two columns and three tuples. The relation S has two columns and two tuples, so the Cartesian product has four columns ($2+2$) and 6 tuples (3×2).

The Cartesian product operation applied by itself is generally meaningless. It is useful only when followed by a selection and projection operations.

UNARY RELATIONAL OPERATIONS:

1. Select operation (σ) :
2. Projection Operation (π):
3. Renaming operation(ρ) :

1. Select operation (σ): It selects required rows from the table. This operation is used to select the subset of the tuples from a relation that satisfy a selection condition or search criteria. Mathematical symbol ' σ ' [sigma] is used to denote the select operator. The general syntax for selection operation is shown below.

Sigma <selection condition> {<relation name>}

The <selection condition> is a Boolean expression, consist of attribute names, comparison operator like =, !=, <, > <=, >= and Boolean operator like AND, OR and NOT.

EXAMPLE.

1. Select the employees who are working in department 10 and whose salary is greater than Rs 5000.

σ (DNO =10 AND Salary > 5000) EMPLOYEE

2. Select the employees who working in department 10 and earning more than 5000 or employees who working in department 20 and whose are earning more than 10000 Rupees.

σ (DNO =10 AND Salary > 5000) OR(DNO =20 And Salary > 10000) EMPLOYEE

2. Projection Operation (π): Projection operation is used to select only few columns from a relation. The mathematical symbol π (PI) is used to denote the project operation. The general syntax for projection operation is shown below.

π <attribute list> {<Relation>}

Here <attribute list> is a list of attributes from the relation R, hence the degree (number of columns) of the result is equal to the number of attributes specified in the attribute list.

Example:

1. Select the name and salary of all the employees.

π Name, Salary (Employee)

This query selects only name and salary attributes from relation EMPLOYEE.

2. Select name and address of all employees working for department 10.

π Name, Address (σ DNO =10) EMPLOYEE.

3. Renaming operation(ρ): This operation is used to rename the relations or attributes.

The symbol ρ is used to denote the rename operator. In some situations, it is better to break down complex query into two or more simple queries, we must rename the relations. It improves the readability and facilitates better understanding.

The syntax is as follows:

Rename <old table> to <new table>

ρ S New attributes names (R) -It renames both the relations and its attributes.

ρ S (R) -It renames the relation only.

ρ New attributes names (R) -It renames only the attributes

NATURAL JOIN (*): It is denoted by * symbol. The standard definition of natural join requires that the join attribute have the same name in both relations. In general, natural join is performed by equating all attribute pairs that have the same name in two relations.

General form is:

$Q \leftarrow R * \langle \text{list 1} \rangle \langle \text{list 2} \rangle S$

Here list 1 specifies list of attributes from R and list 2 specifies a list of attributes from S.

Department	
DNO	DNAME
1	Admin
2	Research
3	Accounts

Project		
PNO	PNAME	DNUM
10	Library MGT	2
20	ERP	1
30	Hospital MGT	3
40	Wireless n/w	2

PROJ_DEPT			
PNO	PNAME	DNUM	DNAME
10	Library MGT	2	Research
20	ERP	1	Admin
30	Hospital MGT	3	Accounts
40	Wireless n/w	2	Research

Here, the joining is done over the attribute DNO of department relation and DNUM of project relation. In fact, DNUM of project is a foreign key which references DNO of department. Generally, in a natural join, the joining attribute is implicatively considered. Suppose if the two relations have no attribute(s) in common, $R * S$ simply the cross product of these two relations, joining can be done between any set of attributes and need not be always with respect to the primary key and foreign key combinations.

The expected size of the join result divided by maximum size, i.e., $n_R * n_S$ leads to a ration called JOIN selectivity.

OUTER JOIN (+): It returns both matching and non-matching rows, it differs from the inner JOIN in that the rows in one table having no matching rows in one table will also appear in the results table with NULLS in the other attribute position, instead of being ignored as in the case with the inner JOIN. It output rows even if they do not satisfy the JOIN condition, the outer JOIN operator (+) is used with the table having no matching rows.

WORKER		
NAME	AGE	ADDRESS
Ankith	23	-
Bhrath	21	-
Chaya	20	-
Deepa	21	-
Ganga	24	-
Lokesh	25	-
Nataraju	23	-

Madhu	22	-
-------	----	---



WORKER SKILLS	
NAME	SKILL
Ankith	Work
Ganga	Smithy
Lokesh	Driver
Mahesh	Fitter
Pandu	Smithy
Madhu	Fitting



Example:



```
select A.NAME, AGE, SKILL
from WORKER A, WORK_SKILL B
where A.NAME = B.NAME(+);
```

RESULT		
NAME	AGE	SKILL
Ankith	23	Work
Bhrath	21	
Chaya	20	
Deepa	21	
Ganga	24	Smithy
Lokesh	25	Driver
Nataraju	23	
Madhu	22	Fitting

In the above example, even though there is no matching row with B.Name, all workers are listed along with age and skill. If there is no match simply got an empty skill column. The outer join can be used when we want to keep all the tuples in R of in S are those in both relations, whether or not they have matching tuples in the other relation.

LEFT OUTER JOIN (): It is denoted by  , the left outer join operation keeps every tuple in the first or left relations R in relation R, if no matching tuple is found in S in the JOIN result are failed with null values.

RIGHT OUTER JOIN (): It is denoted by  , keeps every tuple in the second or right relation S in the result of R, S.

FULL OUTER JOIN (): It is denoted by  , keeps all tuples in both the left and right relations when no matching tuples are found, filled with NULL values are needed.

Example:

BRANCH_LOAN		
BNAME	LOANNO	LOAN_AMT
X	L120	30000
Y	L220	50000
Z	L440	60000

CUSTOMER_LOAN	
CNAME	LOANNO
A	L120
B	L320
C	L440

LEFT OUTER JOIN:

BNAME	LOANNO	LOAN_AMT	CNAME	LOANNO
X	L120	30000	A	L120
Z	L440	60000	C	L440
Y	L220	50000	NULL	NULL

RIGHT OUTER JOIN:

BNAME	LOANNO	LOAN_AMT	CNAME	LOANNO
X	L120	30000	A	L120
Z	L440	60000	C	L440
NULL	NULL	NULL	B	L320

FULL OUTER JOIN:

BNAME	LOANNO	LOAN_AMT	CNAME	LOANNO
X	L120	30000	A	L120
Z	L440	60000	C	L440

Y	L220	50000	NULL	NULL
NULL	NULL	NULL	B	L320

DIVISION (\div): A division operation is useful for a special kind of query, occasionally it may be used to solve certain kind of problems.

Consider the relation P(P) and Q(Q) as shown figure, the result of dividing P by Q is the relation R and it has two tuples. For each tuple in R, its product with the tuples of Q must be in P. In our example (a1,b1) and (a2,b2) must both be tuples in P, the same is true for (a5,b1) and (a5,b2).

Example of division operation $R = P \div Q$

P(P)		Q(Q)	R(R) (result)
A	B		
a1	b1		
a1	b2		
a2	b1		
a3	b1		
a4	b2		
a5	b1		
a5	b2		

Relational Database Language

The name SQL is derived from structured query language. Originally, SQL was called SEQUEL (for Structured English Query Language) and was designed to implemented at IBM research as the interface from experimental relational database system called SYSTEM R. SQL is now the standard language for commercial relational DBMS'S.

SQL is a non-procedural language, means that SQL specifies what data to be retrieved from the database rather than how to retrieve the data.

SQL is a comprehensive database language; it has statements for data definition, query and update. It has facilitates for defining views on the database, for specifying security and authorization, for defining integrity constraints and for specifying transaction controls.

The SQL statements can be grouped into following categories:

1. Data Definition Language [DDL]
2. Data Manipulation Language [DML]
3. Data Control Language [DCL]
4. Transaction Control Language [TCL]
5. Data Query Language [DQL]

1. Data Definition Language [DDL]: Provides commands for defining schema, tables, indexes, sequences etc. Commands are:

CREATE: For creating schema, table, indexes, sequences.

ALTER: For altering table, indexes, sequences.

DROP: For dropping table, indexes, sequences

2. Data Manipulation Language [DML]: Provides commands for manipulating the data.

INSERT: - For inserting values or data into the table.

DELETE: - For deleting the data from the table.

UPDATE: - For updating the data in the table

3. Data Control Language [DCL]: These statements are used to give permissions to the user, take back the given permission from the user, lock certain permission from the user.

GRANT: - Giving permissions on object privileges to user.

REVOKE: - Take back the given permission from the user.

4. Transaction Control Language [TCL]: It is used to control transactions.

COMMIT: - To permanently save the changes made to transaction.

ROLLBACK: - To undo or cancel (discards) the changes up to the previous commit point.

5. Data Query Language [DQL]: It is used retrieve data from the database

SELECT:

SQL uses the terms tables, row and column for relation, tuple and attribute respectively.

An SQL schema is identified by a schema name, and includes an authorization identifier to indicate the user or account who owns the schema, as well as descriptors for each element

in the schema. Schema elements include the tables, constraints, views, domains and other constraints (such as authorization grants) that describe the schema. A schema is created via create schema statements.

For example, the following statement creates a schema called COMPANY owned by the user with authorization identifier JSMITH:

SQL DATA TYPES: The following table lists the various data types available for attribute in SQL

DATA TYPE	DESCRIPTION
CHAR(size)	Fixed length character Size- number of characters, Max size = 200 Characters
VARCHAR(size)	Varying length characters. Size- number of characters. Max. size = 400 characters. This is also specified as VARCHAR2(Size)
NUMBER(size)	Integer number without decimal point Max. Size = 40 digits
NUMBER(size, d)	Integer numbers with decimal point or floating points numbers. Size = total number of digits and d- number of digits after decimal point.
INTEGER OR INT	Integer number. Size can't be specified.
SMALL INT	Integer number
DECIMAL (i, j)	Integer numbers with decimal point or real numbers or floating point numbers. i- Precision is the total number of decimal digits. j- Scale, is the number of digits after decimal point.

DATE	For representing date, which has 10 positions and its components are year, month and day typically in the form YYYY-MM-DD.
TIME	For representing time, which has 8 positions and its components are year. Minutes and second. Typically in the form HH: MM:SS.

The DROP Table command: If a base relation within a schema is not needed any longer, the relation and its definition can be deleted by using the drop Table command.

For example: If we want to delete a employee table from company schema:

DROP TABLE EMPLOYEE;

DROP TABLE EMPLOYEE CASCADE;

This command drops employee table with all constraints and views that reference the table automatically from the schema along with the table itself.

DROP TABLE EMPLOYEE RESTRICT;

This command drops a table only it is not referenced in any constraints.

Aggregate Functions: Many database applications require the aggregate of summarization of data. The aggregation operations are counting, summing, average and finding the maximum and minimum values on tables.

SQL provides the following built in aggregate functions, which are listed in the following table:

COUNT ()	Counts the number of values in columns
COUNT(*)	Counts the number of rows of the query result
SUM()	Finds the sum of the values in a column

AVG()	Returns the average of the values in a column
MAX()	Returns the maximum value in a column
MIN()	Returns the minimum value in a column

1. COUNT ():

EX: Count the number of distinct salary values in the database

SEELCT COUNT (DISTINCT SALARY) from EMPLOYEE;

EX: Count the number of employees in the department 10.

SELECT COUNT (FName) from EMPLOYEE where DNO =10;

2. COUNT (*):

Ex: Retrieve the total number of employees in the company.

SELECT COUNT (*) from EMPLOYEE;

Ex: Retrieve the number of employees in the research department.

SELECT COUNT (*) from EMPLOYEE, DEPARTMENT

Where DNO = DNumber AND DNAME = 'Research';

3. SUM ():

Ex: Find the sum of the salaries of all employees

SELECT SUM (Salary) from EMPLOYEE;

4. AVG ():

Ex: Find the average salary of all employees:

SELECT AVG (Salary) from EMPLOYEE;

5. MAX():

EX: Find the maximum salary of employees.

SELECT MAX (Salary) from EMPLOYEE;

6. MIN ():

Ex: Find the minimum salary of employee.

SEELCT MIN (Salary) from EMPLOYEE;

Ex: Find the sum of the salaries of all employees of research department, as well as the maximum salary, minimum salary and the average salary in this department.

SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)

from EMPLOYEE, DEPARTMENT

Where DNO = DNumbe AND DName ='Resesarch';

VIEWS (Virtual Tables) in SQL: A View in SQL is a single table that is derived from other tables. These other tables could be base tables or previously defined Views. A view does not exist in physical form. It is called as **virtual table** in contrast to base tables whose tuples are actually stored in the databases. The advantages of using Views are:

1. It restricts data access from tables, i.e., it provides security.
2. Reduces joining of tables each and every time, if we create a view for frequently refreshing a join operation of two or more tables. Base tables are also called **Defining** tables. The VIEW can be created using **create View command**.

The general syntax for creating a VIEW is as follows:

CREATE VIEW <View Name> AS

SELECT <Table name1>, <Table name2>

WHERE [Condition];

Ex: Create a view WORKS_ON1 with attribute from EMPLOYEE, PROJECT and WORKS_ON Base tables.

CREATE VIEW WORKS_ON1 AS

```

SELECT FName LName, PName, Hours
    From EMPLOYEE, PROJECT, WORKS_ON
    Where SSN = ESSN and PNO = PNUMBER;

```

INDEXES: Index is a database object created on one or more columns of a table which provides faster access to data.

The General syntax for creating index is as follows:

```

CREATE INDEX <index name>
    ON <Table name> (column1, column2 ..... )

```

Ex: CREATE INDEX idxemp ON EMPLOYEE (SSN);

Embedded SQL: In this approach the SQL statements are embedded directly into the program's source code, along with other programming language statements. Special delimiters specify the beginning and end of the SQL statements in the program. For example, the following code segment shows how an SQL statement is included in a COBOL program.

COBOL Statements

```

-  -----
-  -----
-  -----
EXEC SQL

```

```

SELECT Name INTO: WS_Name from EMPLOYEE

```

```

    Wjere empno = 100

```

```

    END-EXEC

```

```

    -----

```

```

    -----

```

The embedded SQL statements are thus used in the application to perform the data manipulation and access tasks. A special SQL pre compiler accepts the combined source code and compiles it to convert it into the executable form.

PL/SQL

PL/SQL is a non procedural language. It means that we can state what we want without specifying the way (procedure) to do it. Hence it lacks the programming capacity PL/SQL helps us to write program to do specific operations as we write in other programming language like BASIC, C, C++ and etc.

The create table command of SQL is not allowed in PL/SQL code is compiled and it cannot refer to object that do not as yet exist at compile time.

It has several high level programming language features such as block structure, variables, constants and types, the assignment statement, conditional statements, loops, customized error handling and structure data.

Limitations of SQL:

1. SQL does not have any procedural capabilities. It does not provide the programming techniques of condition checking, looping and branching that is vital for data testing before its permanent storage.
2. SQL statements are passed to the oracle engine one at a time, each time on SQL statement is executed, a call is made to engines resources \, consider a multi user issuing SQL statements simultaneously, it causes more traffic on the network.
3. While processing on SQL sentence if an error occurs, the oracle engine displays its own error messages, SQL has no facilities for programmed handling of errors that arise during the manipulation of data
4. It is not a fully structured programming language.

Advantages of PL/SQL:

1. PL/SQL is a development tool that not only supports SQL data manipulation but also provides facilities of conditional checking, branching and looping.

2. PL/SQL sends on entire block of SQL statements to the oracle engine all together communication between the program block and the oracle engine reduces considerably reducing network traffic. The code is processed much faster.
3. PL/SQL allows declaration and use of variables in blocks of code. These variables can be used to store intermediate results of query for later processing or calculate values and insert them into an oracle table later.
4. Using PL/SQL all sorts of calculations can be done quickly and efficiently without the use of the oracle engine. This improves transaction performance.
5. Applications written in PL/SQL are portable to any computer hardware and operating system, where oracle is operational.

PL/SQL Block: A PL/SQL block has a definite structure, which can be divided into 4 sections.

- The declaration section
- The begin section
- The exception section
- The end section.

Memory variables, constants and other oracle objects can be declared in declaration section.

SQL executable statements and PL/SQL executable statements which describe processes that have to be applied to table data are specified in the begin section. Actual data manipulation, retrieval, looping and branching constructs are specified in this section.

The exception section deals with handling of errors that arise during the execution of data manipulation statements, which make up the PL/SQL code block. Errors can arise due to syntax, logic and/or validation rule violation.

The end section marks the end of the PL/SQL block.

PL/SQL Character set:

1. Upper case letters: A...Z
2. Lower case letters: a...z
3. Numerals: 0...9
4. Special symbols: (,), +, -, *, /, <, >, =, !, :, ;, ., ' , ", @, #, \$, %, &, _, \, {, }, [,], ?, |.
5. Literals: A sequence of valid PL/SQL characters is called literal.

a. Numeric Literal: [int / float]:

Example: 12, 14, -10, 12E10, 15.04E10

b. String literal: [a sequence of more than one character enclosed with a single code]

Example: 'Good Moring' , 'how are you', '1234.56'.

c. Character Literal: A string literal only one character is called character literal.

Example: 'y','0','5'

d. Boolean Literal: The value TRUE, FALSE, NULL are called Boolean literal.

PL/SQL Data types:

1. Number
2. Char
3. Date
4. Boolean
5. % type: This data type is used to declare a variable similar to the data type of a column in an existing table or that of a variable defined earlier in the block.

PL/SQL Variable: PL/SQL variables are declared in the declarative part of a PL/SQL block.

- ❖ A variable name must begin with an alphabet followed by alpha-numerals.
- ❖ The maximum length of a variable is 30 characters.
- ❖ Only underscore (_) special character allowed.

Syntax:

Variable name < data type> (size).

Example:

STUD_NAME varchar (30);

SALARY number (6, 2);

- ❖ Using the assignment operator.

Constant: If a particular item does not change its value in PL/SQL block, we can declare it as a constant in the block; its value must be assigned along with the declaration.

Syntax: <constant_name> constant < data type>:= <value>;

Example: discount CONSTANT number (5, 2):= 3.50;

Comments: This line is not executable, we can place comments anywhere inside a PL/SQL block. Comments are very useful to the programmers for later references and debugging.

1. **Single line comments (- -):** A comment having only one line is called a single line comment beginning with two dashes.
2. **Multiline comments (/* ..*/):** If a comment runs over more than one line, we need not begin a line with - -, but just mark the beginning of the comment with /* and mark the end of the comment with */.

PL/SQL Execution:

1. Invoke editor from edit menu of Oracle SQL * Plus environment, Notepad editor is invoked.
2. Enter the PL/SQL program.
3. Save and exist using file menu.
4. To execute PL/SQL type '/' at SQL prompt and press enter.
5. Error will be indicated, go back and correct them.
6. When execution is successful, the message 'PL/SQL Procedure Successfully Completed' is displayed.

SQL>

Wrote file afiedt.buf

```
1 declare
2 n1 number;
3 n2 number;
4 s number;
5 begin
6 n1:=10;
7 n2:=20;
8 s:=n1 + n2;
9 dbms_output.put_line('Sum of 10 and 20 is'||s);
10* end;
```

SQL> /

Sum of 10 and 20 is 30

PL/SQL procedure successfully completed.

Unit-04 Transaction Processing Concept

Introduction: Transaction management is the ability of a DBMS to manage the various transactions that occur within the system. Transaction is the set of program statements or collection of operations that form a single logical unit of work. A DBMS should ensure that the transactions are executed properly, either the entire transaction should execute or none of the operations should have been executed. This is also called atomic operation. The DBMS should execute this task or transaction in total to avoid inconsistency.

DEFINITION: A transaction is an atomic unit comprised of one or more SQL statements. A transaction begins with the first executable statements and ends when it is committed or rolled back.

Single user/ multi user systems: A DBMS is a single user if at most one user at a time can use the system. It is multiuser if many users can use the system and have access to the database concurrently.

Example: An airline reservation system is used by 100's of travel agencies and clerks concurrently.

Multiuser can access the database and use computer systems simultaneously. Because of the concept of multiprogramming, this system acquires some commands from one process then suspends that process and executes some command from the next process, therefore it is interleaved.

In a single user system can execute at most one process at a time.

Example of transaction processing system:

- Reservation systems.
- Credit card processing system
- Stock market processing system.
- Super market processing system
- Insurance processing system.

Transaction and system concepts: A transaction is a logical unit of database processing that includes one or more database access operation (insertion, delete etc) only retrieving of data is called read only transaction.

A transaction includes two basic database access operations: they are,

1. **Read_item(x):** It reads a database item named as 'x' into a program variable.
2. **Write_item(x):** Writes the value of the program variable x into the database.

Read_item(x) include the following steps:

1. Find the address of the disk block that contain item 'x'.
2. Copy the disk block into a buffer in main memory.
3. Copy item 'x' from the buffer to the program variable x.

Executing the write_item(x) includes the following steps:

1. Find the address of the disk block that contains item (x).
2. Copy that disk block into a buffer in main memory.
3. Copy item(x) from the program variable into its current location in the buffer.
4. Store the update block from the buffer to disk.

Why concurrency control is needed?

In a multiuser database, transaction submitted by the various user may execute concurrently and many update the same data concurrently executing transactions must be guarantied to produce the same effect as serial execution of transaction (one by one). **Several problems can occur when current transaction execute in uncontrolled manner**, there fore primary concern of multiuser database include how to control data concurrency and consistency.

Data concurrency: Access to data concurrently (simultaneously) used by many users must be co-ordinates.

Data consistency: A user always see a consistent (accurate) view of all data committed by other transactions as of that time and all change made by the user up to that time. Several problems can occur when concurrent transactions execute in an uncontrolled manner.

Example: Airline reservation database, in which a record is stored for each airline flight. Each record includes the number of reserved seats on the flight as a named data item.

A transaction is a particular execution of program on a specific data, flight and number of seats. a database program can be used to execute many transactions, each with different flight and number of seats to be booked.

T1	T2
read_item(x);	read_item(x)
x:=x-N;	x:=x+M;
write_item(x);	write_item(x);
read_item(y);	
y:=y+N;	
write_item(y);	
fig(a): Transaction [t1]	fig(b) Transaction[T2].

.fig (9.1): Simple transactions.

When a database access program is written, it has the flight numbers, their dates and number of seats to be booked as parameter. So same program can be used to execute many transactions, each with different flights and numbers of seat to be booked.

Transaction T1 and T2 are specific executions of the programs that refer to the specific flights whose number of seats are stored in data item x and y in the database.

The problems that occur when these two transactions run concurrently.

- The Lost update problem.
- Temporary update (or Dirty read) problem.

- The incorrect summary problem.

The Lost update problem: Suppose transaction T1 and T2 are submitted at the same time, when these two transactions are executed concurrently as shown in figure (9.1.a), then the final value of X is incorrect because T2 reads the value of x before T1 changes it in the database and hence the updated value resulting from T1 is lost.

Example: $x=80$ at the start (80 reservation at the beginning), $n=5$ (T1 transfer 5 seats reservations from flight x to y) and $m=4$ (T2 reserves 4 seats on x), the final result should be $x = 79$ but due to interleaving of operations $x = 84$ because updating T1 that removed the 5 seats from x was Lost.

T1	T2
read_item(x);	
$x:=x-N$;	
	read_item(x)
	$x:=x+M$;
write_item(x);	
read_item(y);	
	write_item(x);
$y:=y+N$;	-item x has an incorrect, because its update by T1 is lost.
write_item(y);	
fig(a): Transaction [t1]	fig(b) Transaction [T2].

Figure (9.2): The Lost update problem.

2. Dirty read Problem: This operation occurs when one transaction updates a database item and then the transaction fails for some reason, the update item is accessed by another transaction before it is changed back to its original value.

Example: T1 updates item x and then fails before completion, so the system must change x back to original value, before it can be do so, however, transaction T2 reads the temporary value of x, which will be no record permanently in the database because of the failure of T1. The value of item x that is read by T2 is called **Dirty data**. Because it has been created by a transaction that has not completed and committed yet, hence this problem is also known as the temporary update problem.

T1	T2
read_item(x);	
x:=x-N;	
write_item(x);	
	read_item(x)
	x:=x+M;
	write_item(x);
read_item(y);	
-transaction T1 fails and must change the value of x back to its old value.	-T2 has read the temporary incorrect value of x.
fig(a): Transaction [t1]	fig(b) Transaction[T2].

Figure(9.3): Temporary update problem.

3. Incorrect summary problem: If one transaction is calculating an aggregate summary function on a number of records while other transaction are updating some of the records, the aggregate functions may calculate some values before they are update and others after they are updated.

Example: Transaction T3 is calculating the total number of reservations on all the flights, mean while transaction T1 is executing. The T3 reads the value of x after n seats have

been subtracted from it but reads the value of y before those n seats have been added to it.

T1	T3	
	Sum:=0	
	read_item(a)	
	sum := sum+a;	
read_item(x);		
x:=x-N;		
write_item(x);		
	read_item(x);	
	sum:=sum+x; ←	T3 reads X after N is subtracted and reads y before N is added.
	read_item(y);	
	sum:= sum+y;	
read_item(y);		
y:=y+N;		
write_item(y);		
	-a wrong summary is the result.	
fig(a): Transaction [t1]	fig(b) Transaction[T2].	

Figure (9.4): Incorrect summary problem:

Why recovery is needed?

A major responsibility of database administrator is to prepare for the possibilities of hardware, software, network and system failure. It is usually desirable to recover the databases and return to normal operation as quickly as possible, recovery should proceed in such a manner to protect the database and users from unnecessary problem.

Whenever a transaction is submitted to a DBMS for execution the system is responsible for making sure that either,

- All the operations in the transaction are completed successfully and their effects are recorded permanently in the database or,
- The transaction has no effect on the database; this may happen if a transaction fails after executing some of its operations but before executing all of them.

Types of transaction failures:

- **A computer failure (system crash):** Hardware, software and network error occurs in the computer system during transaction execution.
- **Transaction or system error:** Some operation in the transaction may cause it to fail, such as integer overflow or division by 'zero' etc.
- **Local errors or exception conditions deleted by the transaction:** During transaction execution, certain conditions may occur that perform cancellation of the transaction. For example: data for the transaction may not be found.
- **Concurrency control enforcement:** The concurrency control method may decide to abort the transaction to be restarted later, because several transactions are in a state of dead lock.
- **Disk Failure:** Some disk block may lose their data because of a read or write malfunctions.
- **Physical problems and catastrophes:** This refers to list of problem that includes power or air conditioning failure, fire, theft, overwriting disks and etc.

Explain Transactions states and additional operations:

A transaction is an atomic unit of work that is entirely completed or not done at all. For recovery purpose the system needs to keep track of when the transaction starts,

terminates, commits or aborts. Hence the recovery managers keeps track of the following operations.

1. **Begin transactions:** This marks the beginning of the transaction execution.
2. **Read/write:** These specify read/write operation execution.
3. **End transaction:** This specifies the read and write transaction operations have ended and marks the end of the transaction execution. At this point it may be necessary to check whether the changes can be permanently applied to the database or aborted.
4. **Commit transaction:** This signals a successful end of the transaction so that they changes executed by the transaction can be committed to the database.
5. **Roll back:** This signals that the transactions has ended unsuccessfully, so that any changes that the transaction may have applied to database must be undone.

Transaction states: A transaction moves through various execution states. This can be explained with state transaction diagram shown in figure 9.5.

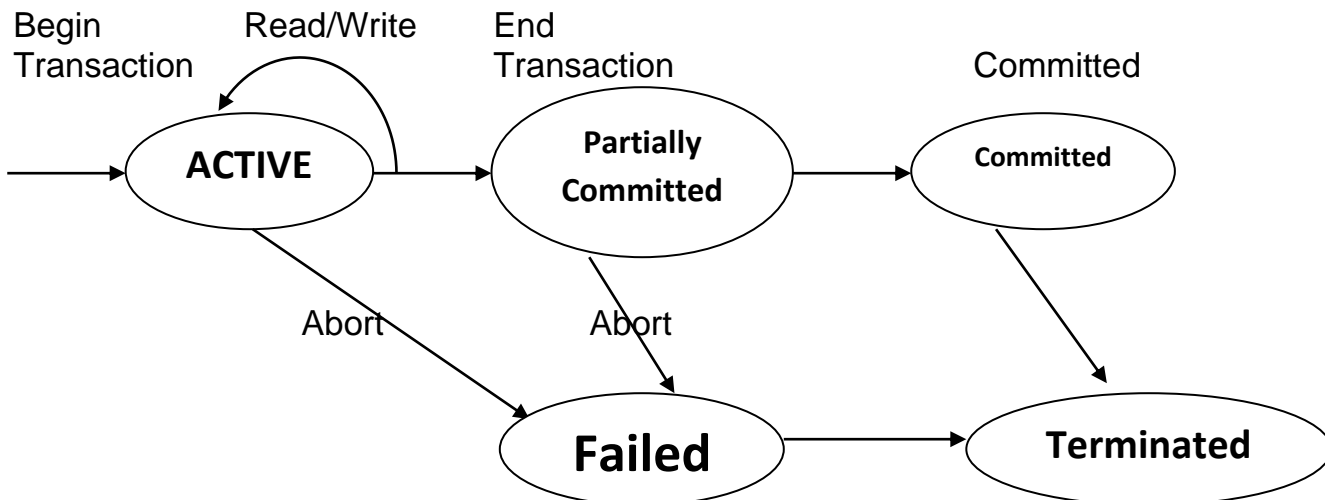


Figure (9.5): State transaction diagram.

Figure shows a state transaction that describes how a transaction moves through its execution states. A transaction goes into an active state immediately after it starts execution. Where it can issue Read and Write operation.

When the transaction ends, it moves to partially committed state. At this point some recovery protocols needed to ensure that there is no system failure. Once this check is successful, the transaction is said to have reached its commit point and enters the committed state.

However, a transaction can go to the failed state, if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its write operations on the database. The terminated state corresponds to the transaction leaving the system or end of the transaction.

Desirable properties of transaction; [ACID Properties]

To ensure data integrity, the DBMS should maintain the following transaction proprieties; these are often called the ACID properties.

1. **Atomicity:** A transaction is an atomic unit of processing it is either performed in it's entirely (completely) or not performed at all.
2. **Consistency preservation:** A transaction is said to be consistency preserving if it's complete execution takes the database from one consistent state to another.
3. **Isolation:** The execution of a transaction should not be interfered with by other transaction executing concurrently. It should appear as though it is being executed in isolation. This ensures that each transaction appears to execute in isolation from other transaction, even through hundreds of transactions may be executing concurrently.
4. **Durability:** The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

Concurrency control techniques:

Some of the main techniques used to control concurrent execution of transaction are based on the concept of locking data items. A lock is a restriction on access to the data in a multiuser environment. It prevents multiple users from changing the same data simultaneously. If locking is not used, data within the database may become logically incorrect and may produce unexpected results.

Two types of LOCKs can be used, they are,

1. Binary LOCK.
2. Shared/ Exclusive LOCK.

1. Binary LOCK: A binary lock can have two states of values.

- Locked (1),
- Unlocked (0).

A distinct lock is associated with each database item x . If the value of the lock on ' x ' is 1, item x cannot be accessed by a database operation that requires the item. If the value of the lock on x is 0, the item can be accessed when requests.

Two operations, lock_item and unlock_item are used with binary locking. A transaction requests access to an item x by first issuing a lock_item operation. If $LOCK[x] = 1$, the transaction is forced to wait, if $LOCK[x] = 0$, it is set to 1 and the transaction is allowed to access item x , when the transaction is through using the item, it issues an unlock_item[x] operation, which sets $LOCK[X]$ to 0 (unlocks the item) so that x may be accessed by another transaction, hence, a binary lock enforces mutual exclusion on the data item.

Lock_item(X):

B: if $LOCK(x) = 0$ ("item is unlocked")

Then $LOCK(x) \leftarrow$ ("lock the item")

Else begin

wait (until $lock(X) = 0$ and the lock manager wakes up the transaction);

goto B

end;

unlock_item(X):

$LOCK(X) \leftarrow$ ("unlock the item")

If any transaction are waiting

Then wake up one of the waiting transactions:

If the simple binary locking schema described here is used. Every transaction must obey the following rules.

- ❖ A transaction ' T ' must issue the operation lock_item(X) before any read_item(X) or write_item(X) operations are performed in T .

- ❖ A transaction 'T' must issue the operation unlock_item(X) after all read_item(X) and write_item(X) operations are completed in T.
- ❖ A transaction T will not issue a lock_item(X) operation if it already holds the lock on item X.
- ❖ A transaction 'T' will not issue an unlock_item(X) operation unless it already holds the lock on item x.

2. Shared LOCK: it is used for read only operations. i.e., Used for operations that does not change or update the data.

Example: select statement.

Shared lock allow current transaction to read (select) a data. No other transaction can modify the data while shared locks exist. Shared locks are released as soon as the data has been read.

3. Exclusive LOCK:

.Exclusive LOCK: Exclusive locks are used for data modification operations such as update, delete and insert.

Once a transaction puts the X lock on a particular resource, no other transaction can put any kind of lock on this resource.

This resource is exclusively reserved for the first transaction and no other transaction can use it for read or write operation.

Hence X lock allows least concurrency

- The effect of a lock is to lock other transaction out of the object.

–

- **Compatibility matrix**

A \ B	X	S	—
X	N	N	Y
S	N	Y	Y
—	Y	Y	Y

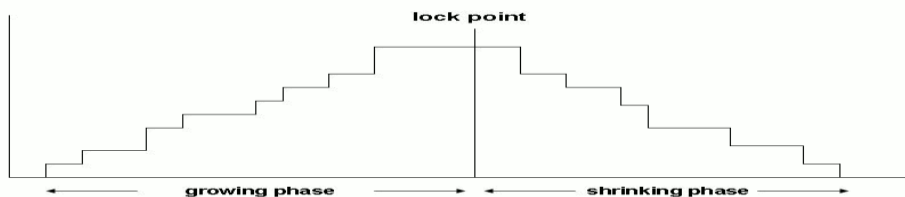
____:no lock

N: request not compatible

Y: request compatible

The two Phase locking(2PL) protocol

- A transaction is said to follow the Two-Phase Locking protocol if Locking and Unlocking can be done in two phases.
- **Growing Phase** In this phase we put read or write lock based on need on the data. In this phase we does not release any lock.
- **Growing Phase:** New locks on data items may be acquired but none can be released.
- **Shrinking Phase** This phase is just reverse of growing phase. In this phase we release read and write lock but doesn't put any lock on data.
- **Shrinking Phase:** Existing locks may be released but no new locks can be acquired.



	T ₁	T ₂
1	lock-S(A)	
2		lock-S(A)
3	lock-X(B)	
4
5	Unlock(A)	
6		Lock-X(C)
7	Unlock(B)	
8		Unlock(A)
9		Unlock(C)
10

Transaction T₁ :

The growing Phase is from steps 1-3.

The shrinking Phase is **Transaction T₁** :

The growing Phase is from steps 1-3.

The shrinking Phase is from steps 5-7.

Lock Point at 3

Transaction T₂ :

The growing Phase is from steps 2-6.

The shrinking Phase is from steps 8-9.

Lock Point at 6

What is LOCK POINT? The Point at which the growing phase ends, i.e., when a transaction takes the final lock it needs to carry on its work.

Timestamp:

Timestamp is a unique identifier created by the DBMS to identify a transaction. Timestamp values are assigned in the order in which the transactions are submitted to the system. A timestamp of transaction T can be referred by TS(T).

The idea for this scheme is to order the transactions based on their timestamps.

TIME	10:00	10:02	10:05	10:20
TRANSACTION	T1	T2	T3	T4
TIMESTAMP	100	200	300	400
	OLDER			YOUNGER

RTS= 30 WTS= 20

10	20	30
T1	T2	T3
R(A)		
	R(A)	

		R(A)
--	--	------

10	20	30
T1	T2	T3
		W(A)
W(A)		
	W(A)	

TIME STAMP ORDERING ALGORITHM

1) Transaction T_i issues a Read (A) operation

- If $WTS(A) > TS(T_i)$, rollback T_i
- Otherwise execute R(A) operation set $RTS(A) = \text{MAX}\{RTS(A), TS(T_i)\}$

2) Transaction T_i issues write operation

- If $RTS(A) > TS(T_i)$ then rollback T_i
- If $WTS(A) > TS(T_i)$ then rollback T_i
- Other wise execute Write(A) operation Set $WTS(A) = TS(T_i)$