MEDARAMETLA ANJAMMA MASTAN RAO PG COLLEGE

I MCA I SEMESTER

MCA 102-DBMS

DATABASES & DATABASE USERS

Introduction : Databases and database systems are an essential components of everyday life in modern society. Due to increasing use of databases as a corporate resource and growth of information technology, all the organizations have come to realize the importance of speedy access and reliable data for correct decision making. The data is primarily the known facts that can be stored and have intrinsic meaning. For example a dictionary is a store of huge amount of information which is in organized form.

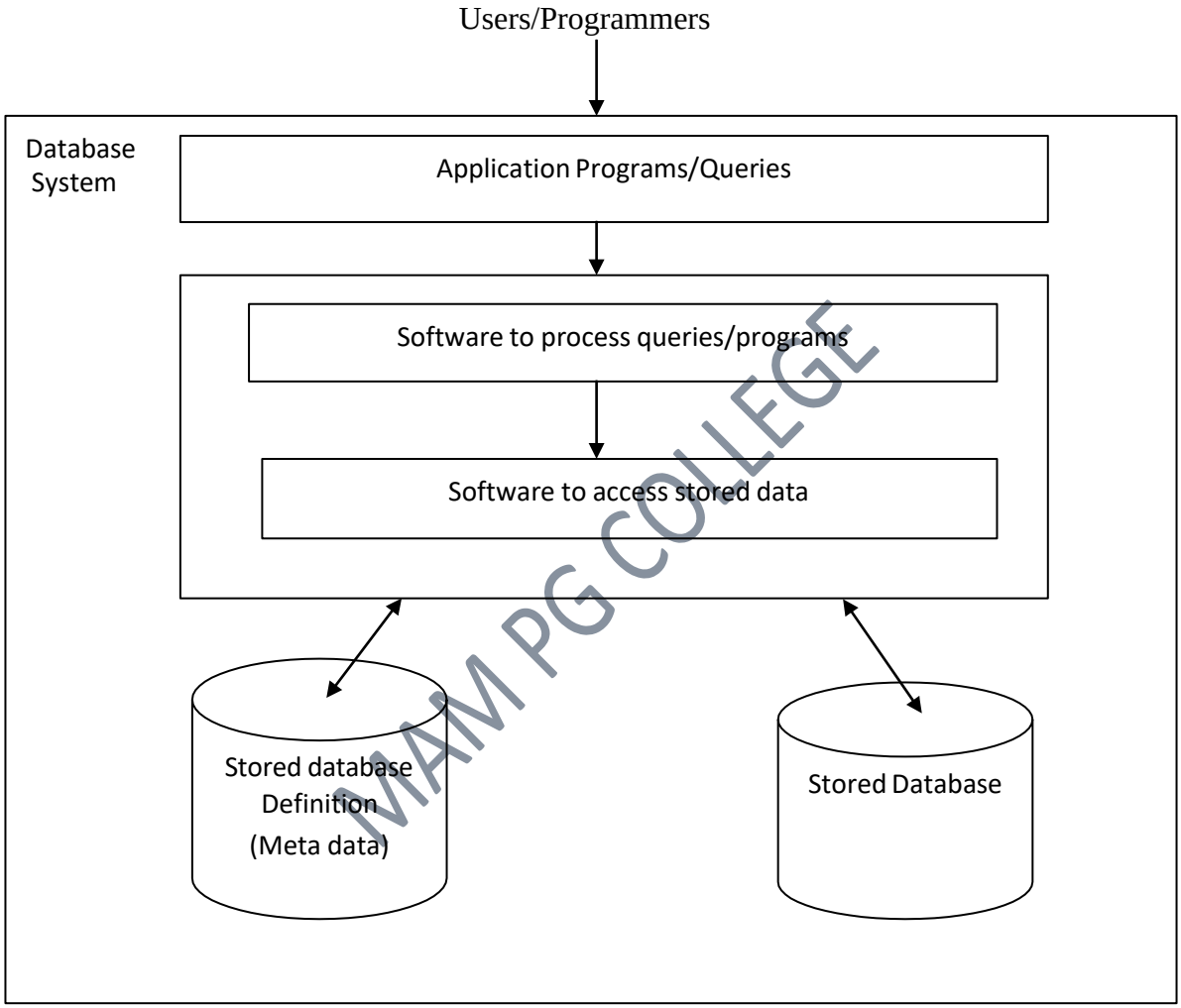
DATABASE : It is a collection of interrelated data. The collection of related data with an implicit meaning is called Database. Nowadays databases and database technology have a major impact on growing use of computers. Databases play a critical role in almost all major areas where computers are used, including business, e-commerce, engineering, medicine, law, education and library science etc.

The database has the following implicit properties :

1. A database represents some aspects of real world, sometimes called mini world oruniverse of discourse (UOD). Changes to the mini world are reflected in the database.
2. A database is a logically coherent collection of data with some inherent meaning. A random assortment of data can't correctly be referred to as a database.
3. A database is designed, build and populated with database for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

DBMS(Database Management Systems) : A DBMS is a collection of program that enables the users to create and maintain a database. It is a general purpose software system that facilities the process of defining, constructing, manipulating and sharing databases among various users and applications. Defining database involves specifying data types, structures and constraints of data to be stored in the database. This definition is stored in the form of database catalog or dictionary. It is sometimes called as Meta Data. Constructing the database is the process of storing the data on some storage medium that is controlled by DBMS. Manipulating adatabase includes functions such as querying database, retrieve specific data, updating database

and generating reports from the database. Sharing a database allows multiple users and programs to access database simultaneously. Other important functions provided by DBMS include protecting and maintaining it over a long period of time. Protection includes system protection against hardware or software malfunction and security protection against unauthorizedaccess.



Characteristics of Database Approach : A Traditional File System (TFS) existed before DBMS came into existence. Because of drawback in TFS, the database systems were introduced. The main characteristics of database approach versus the file processing approach are described below.

1. **Self describing nature of database system** : A fundamental characteristic of the database approach is that database system not only contains database itself but also a complete definition or description of database structure and constraints. This is stored as DBMS catalog. The information stored in catalog is also called Meta Data. This catalog is used by DBMS and also by users who need information about structure of database. However in TFS data definition is typically a part of application programs. So if file was used in more than one application, then we had to create a separate copy for each application. This multiple copies of same data file leads to redundancy problems.

2. **Insulation between programs and data abstraction** : In TFS, the structure of data files is embedded in application programs, so any changes to structure of file may require changing all application programs that access that file. However DBMS don't require such changes because structure of data files stored separately in DBMS catalog from the access programs. This nature is referred as Program-Data independence. In some cases user can define operations as a part of database definition. This operation has two parts like interface and implementation. User application programs can operate on data by invoking these operations through their names and arguments regardless of how the operations are implemented. This is termed as Program-operation independence. The characteristic that allow program-data and program-operation independences is called Data Abstraction.

3. **Support of Multiple View of data** : A database has many users, each of whom may require a different perspective or view of the database. This view may be a subset of database. It may contain virtual data that is derived from database files but not explicitly stored.

4. **Sharing of data and multi user transaction processing** : A multiuser DBMS allows multiple users to access the database at the same time. This is essential when data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control to ensure that several users trying to update same data, so that the result of updates is correct. The concept of transaction has become central to many database applications. A transaction is executing program or process that includes

one or more database access. Each transaction is supposed to execute without interference from other transactions. The DBMS enforce several transactions properties.

Advantages of DBMS : The main advantages of using DBMS are

1. **Controls Redundancy :** The redundancy is storing same data at multiple places which leads to several problems. Centralized control of data by DBA avoids unnecessary duplication of data. So it can effectively reduce the total amount of storage required. DMBS should have the capability to control this redundancy.

2. **Restricts Unauthorized access :** Confidential data must not be accessed by unauthorized users. DBMS should provide a security and authorization system that DBA uses to create user accounts. DBMS can ensure that proper access procedures should follow accessing sensitive data. Different levels of security could be implemented for various types of data and operations.

3. **Provides persistent storage for program objects :** Databases can be sued to provide persistent storage for program objects and data structures. An object is said to be persistent when it is retrieved after the termination of the program. So the persistent storage of program objects and data structures is an important function of databasesystem.

4. **Provides storage structures for efficient query processing :** Because database is typically stored on disk, DBMS must provide specialized data structures to speed up disk search for desired records. Auxiliary files called indexes are used for this purpose. DBMS must provide capabilities for efficient query processing and updates. The query processing and optimization module of DBMS is responsible for choosing efficient queryexecution plan for each query based on existing storage structure.

5. **Provides Backup & Recovery :** A DBMS must provide the facilities for recovery from hardware and software failures. For example when computer fails in the middle of transaction, the recovery subsystem is responsible for making that database is restored to the state it was before the transaction stated executing.

6. **Provides Multiple User Interfaces :** DBMS should provide different user interfaces because different users with different levels of knowledge use the database. For example query language for casual users and programming language interface for application programs etc.

7. **Represents complex relationships among data :** A database may have varieties of data that are interrelated in many ways. A DBMS must have the capability to represent a variety of complex relationships among data to define new relationships and to retrieve and update data easily and efficiently.

8. **Enforce Integrity Constraints :** Database systems must hold certain integrity constraints to make the data valid. So DBMS must provide the capabilities for defining and enforcing these constraints. Some constraints can be specified to DBMS and automatically enforced. Other constraints may have to be checked by update programs or at the time of data entry.

Disadvantages of DBMS : The DBMS has its own disadvantages. The main disadvantage of DBMS is its cost. The overhead costs of using DBMS are due to following reasons.

1. High initial investment in hardware, software and training.
2. The generality that DBMS provides for defining and processing data.
3. Overhead for providing security, concurrency control, recovery and integrity functions.

In addition to above, the following are some more disadvantages.

1. Complexity
2. Size
3. Performance
4. Higher impact of failure.

Database Users : In large organizations, many people are involved in the design, use and maintenance of large database. Based on the functionalities these users are grouped into different categories as

1. Database Administrators (DBA)
2. Database Designers
3. End Users
4. System Analysts and Application Programmers

- **DBA** : One of the main reasons for having DBMS is to have control of both data and programs accessing that data. The person having such control over the system is called DBA. The DBA is responsible for authorizing access to database, coordinating and monitoring its use and acquiring software and hardware resources as needed.
- **Database Designers** : These are the users whose responsibility is to identify the data to be stored in the database. They can choose appropriate structures to represent and store this data. It is the responsibility of database designers to communicate with all users in order to understand their requirements and to create the design that meets those requirements. So the final database design must be capable of supporting the requirements of all user groups.
- **End Users** : These are the people whose jobs require access to database for querying, updating and generating reports. There are several categories of end users,
    - **Casual End Users** : These occasionally access the database, but they may need different information each time.
    - **Naïve or Parametric End Users** : These are unsophisticated who interact with system by involving one of application program. Their main job function revolves around constantly querying an updating the database.
    - **Sophisticated End Users** : These include engineers, scientists who interact with the system without writing programs.
    - **Standalone Users** : They maintain personal databases by using readymade program package that provide easy to use menu based or graphics based interfaces.

- **System Analysts & Application Programmers** : System analyst determine the requirements of end users, especially naïve and parametric users. They can develop some specifications. Application programmers implement these specifications as programs then they test, debug, document and maintain them.

Database Administrator :

A person who has central control over the system of both data and programs that access the data is called Database Administrator

The function of DBA are :

1. creation and modification of conceptual Schema definition
2. Implementation of storage structure and access method.
3. schema and physical organization modifications .
4. granting of authorization for data access.
5. Integrity constraints specification.
6. Execute immediate recovery procedure in case of failures
7. ensure physical security to database

Database language :

Data definition language(DDL) :

DDL is used to define database objects .The conceptual schema is specified by a set of definitions expressed by this language. It also give some details about how to implement this schema in the physical devices used to store the data. This definition includes all the entity sets and their associated attributes and their relation ships. The result of DDL statements will be a set of tables that are stored in special file called data dictionary.

Data manipulation language(DML) :

A DML is a language that enables users to access or manipulate data stored in the database. Data manipulation involves retrieval of data from the database, insertion of new data into the database and deletion of data or modification of existing data.

There are basically two types of DML:

Procedural: Which requires a user to specify what data is needed and how to get it.

non-Procedural: which requires a user to specify what data is needed with out specifying how to get it.

Data control language(DCL):

This language enables user to grant authorization and canceling authorization of database objects.

Elements of DBMS:

DML pre-compiler:

It converts DML statement embedded in an application program to normal procedure calls in the host language. The pre-complier must interact with the query processor in order to generate the appropriate code.

DDL compiler:

The DDL compiler converts the data definition statements into a set of tables. These tables contains information concerning the database and are in a form that can be used by other components of the dbms.

File manager:

File manager manages the allocation of space on disk storage and the data structure

used to represent information stored on disk.

Database manager:

A database manager is a program module which provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.

The responsibilities of database manager are:

Interaction with file manager: The data is stored on the disk using the file system which is provided by operating system. The database manager translate the the different DML statements into low-level file system commands. so The database manager is responsible for the actual storing,retrieving and updating of data in the database.

Integrity enforcement: The data values stored in the database must satisfy certain constraints(eg: the age of a person can't be less then zero).These constraints are specified by DBA. Data manager checks the constraints and if it satisfies then it stores the data in the database.

Security enforcement: Data manager checks the security measures for database from unauthorized users.

Backup and recovery: Database manager detects the failures occurs due to different causes (like disk failure, power failure,deadlock,s/w error) and restores the database to original state of the database.

Concurrency control: When several users access the same database file simultaneously, there may be possibilities of data inconsistency. It is

responsible of database manager to control the problems occurs for concurrent transactions.

Query processor:

The query processor used to interpret to online user's query and convert it into an efficient series of operations in a form capable of being sent to the data manager for execution. The query processor uses the data dictionary to find the details of data file and using this information it create query plan/access plan to execute the query.

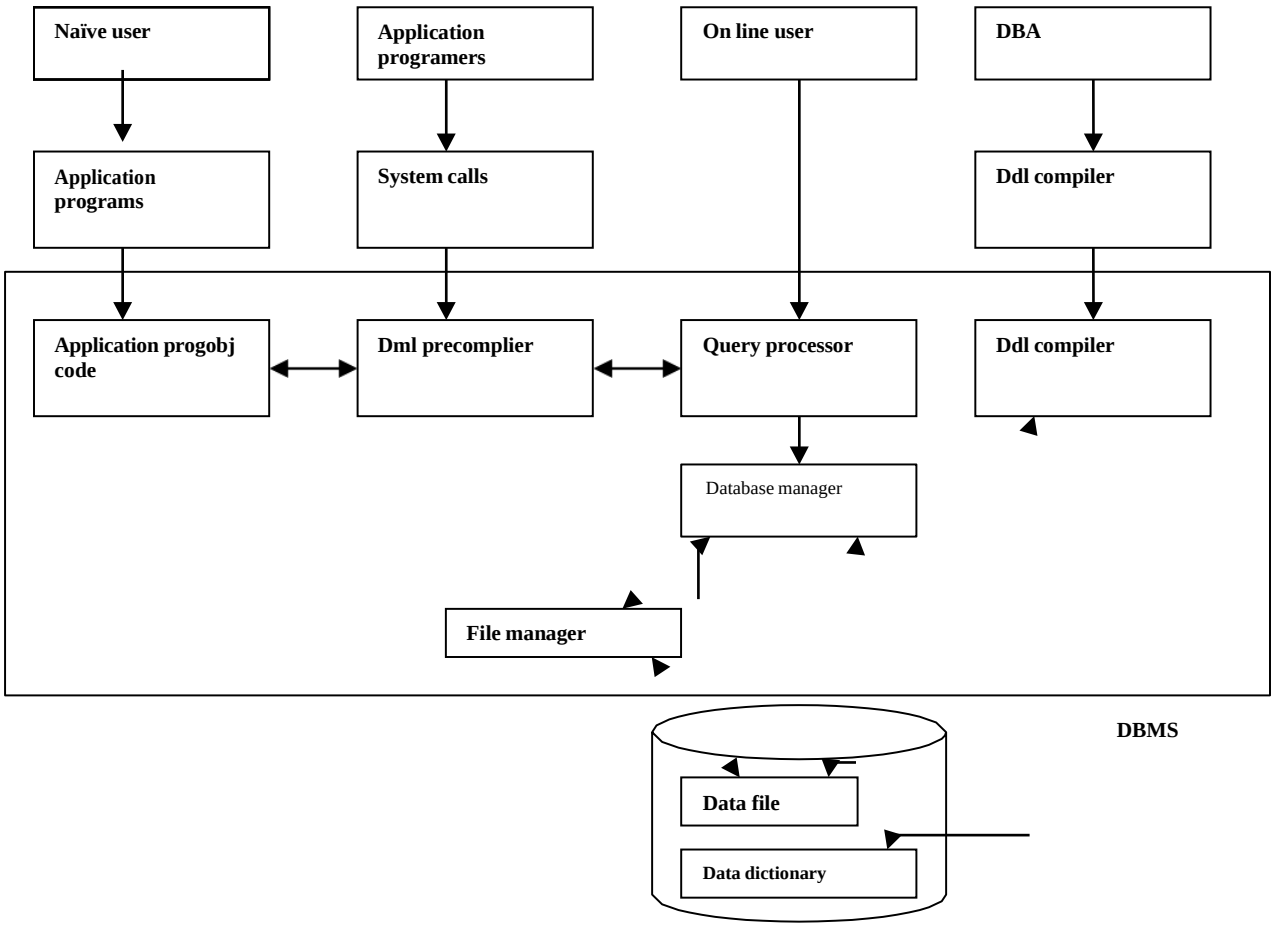
Data Dictionary:

Data dictionary is the table which contains the information about database objects. Itcontains information like

1. external, conceptual and internal database description
2. description of entities , attributes as well as meaning of data elements
3. synonyms, authorization and security codes
4. database authorization

The data stored in the data dictionary is called *meta data.*

DBMS ARCHITECTURE :



DATABASE SYSTEM CONCEPTS & ARCHITECTURE

Data Model : It is a collection of concepts that can be used to describe the structure of database, relationships among data, data constraints and semantics. Actually a data model is abstraction process that suppresses inner details of data storage and organization and highlightingan essential feature for understanding of data. Most data models include the set of basic operations for specifying retrievals and updates on the database.

These data models are classified into different categories like

1. High level or conceptual data models
2. Low level or physical data models
3. Representative or implementation data models

1. **High Level Models** : These models use the concepts like entities, attributes and relationships. Entities represent a real world object or concept such as employee or project that is described in the database. An attributes represent some property of entity such as employee name, salary etc. The relationship is an association among two or more entities.
2. **Low Level Models** : These can provide the concepts that describe the details of how data is stored in computer. These concepts are generally meant for specialists not for typical end users. These describe how data is stored as files in computer by representing information such a record formats, orderings and access paths.
3. **Implementation Models** : These are the models that are used most frequently in commercial DBMSs. These include Hierarchical models, Network models and Relational models. Sometimes there are referred as Record Based Data Models.

SCHEMA : In any data model, the description of database is different from the database itself. This description of database is called database schema. So the overall design of database is called database schema. The displayed schema is called schema diagram. This diagram displays only structure of each record type but not actual instance of records. So schema will remain same and does't changed for instance to instance. The student subschema is given below :

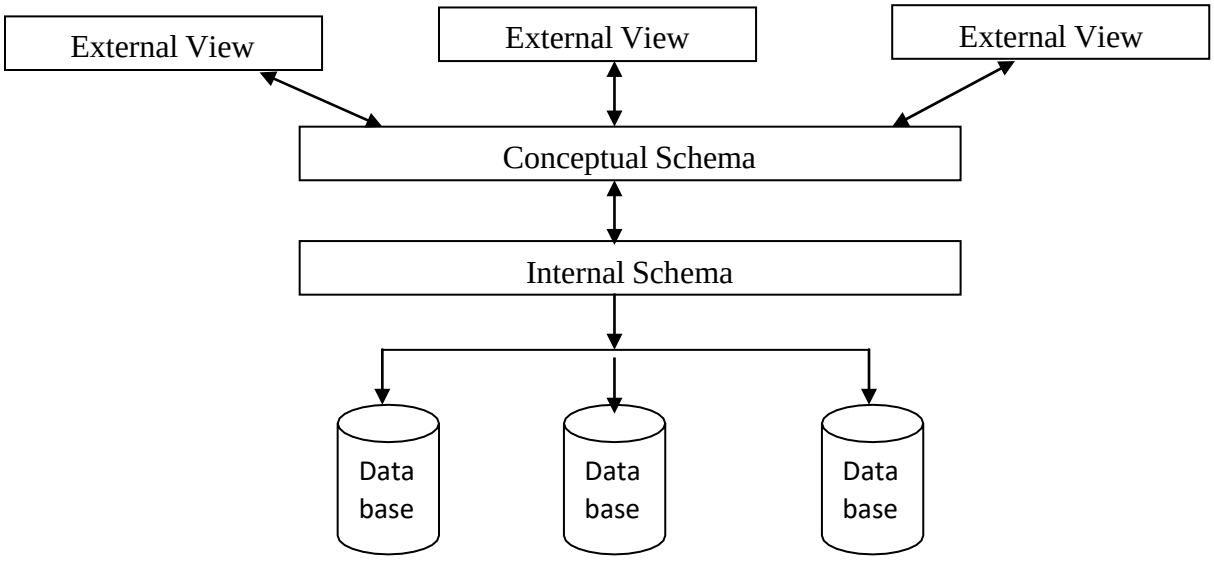
Stu_schema :

No	Name	Class	Course

INSTANCES : Usually database changes over time when some information is inserted or deleted. So the collection of information stored in the database at a particular moment is called an Instance of database. It is also referred as database state or snapshot. It is also called the current set of occurrences or instances in the database. At a given database state, each schema construct has its own current set of instances.

Three Schema Architecture :

An important purpose of database system is to provide users with an abstract view of data. i.e system hides certain details of how the data is stored and maintained. The DBMS provides 3 levels of abstraction for the data which is said to be 3 schema architecture of database system. The view at each level is described by a schema. A schema is outline or a plan that describes the way in which entities at one level of abstraction cab be mapped to the next level. The overall design of database is called the database schema. The main goal of the 3 schema architecture is to separate the user applications and physical database. This 3 schema architecturecan be depicted as follows :



Because each level is defined by schema, 3 schemas exist in database, which are described as follows :

- **Internal Schema** : The Internal level has internal schema which describes the physical storage structure of database. It is also referred as physical schema which provides the lowest level of abstraction. It actually represents how the data will be stored and described the data structures and access methods to be used by database. It uses aphysical data model and describes complete details of data storage and access paths for database.
- **Conceptual Schema** : The Conceptual level has conceptual schema that describes the structure of whole database for a community of users. This schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, operations and constraints. A representational model is used to describe this schema when a database system is implemented.
- **External Schema** : This is at the highest level of abstraction where only those portions of concern to user or application programmers are included. This is also referred as subschema and any number of sub schemas exist for a given conceptual schema. This contains the definitions of logical records and relationships in the external view.

In general one physical schema, one conceptual schema and several sub-schemas exist for a single database system.

Data Independence : Data Independence is one of main advantages of DBMS. The ability to modify the schema definition at one level without affecting schema definition in the next higher level is called Data Independence. We can have two types of Data Independences.

- **Logical Data Independence** : It is the ability to change the conceptual schema without having to change external schemas or application programs. The change would be absorbed by the mapping between the external and conceptual levels. We may change conceptual schema to expand database, to change constraints or to reduce the database.So only mapping is need to be changed in DBMS that supports logical data independence. It is achieved by providing external level or user view of the database.
- **Physical Data Independence** : It is the ability to change the internal schema without having to change the conceptual schema. External schemas need not be changed as well. These changes to internal schema may be needed because some physical files are reorganized. The change would be absorbed by the mapping between the conceptual and

internal levels. It is achieved by the presence of mapping or transformation from conceptual level of database to internal level. So no changes are required in the application programs to access data from new physical organization.

Database Languages & Interfaces : DBMS must provide appropriate languages and interfaces for each category of users. Actually when design of database is completed then DBMS is chosen to implement it. First step is to specify conceptual and internal schemas.When no strict separation is maintained between them then designers use DDL to define both schemas. When clear separation is maintained between them then the designers use DDL for conceptual schema and SDL (Storage Definition Language) for internal schema. The mapping between two schemas may be specified in either one of these languages. They use VDL (View Definition Language) to specify user views and their mapping to conceptual schema. However in relational DBMS, SQL is used in the role of VDL to define user or application views. Once the database schemas are completed, users need to manipulate the database. These manipulations include retrieval, insertions, deletions and modifications of data. The DBMS provides a language called DML for these purposes.

DBMS Interfaces : DBMS must provide user friendly interfaces which include

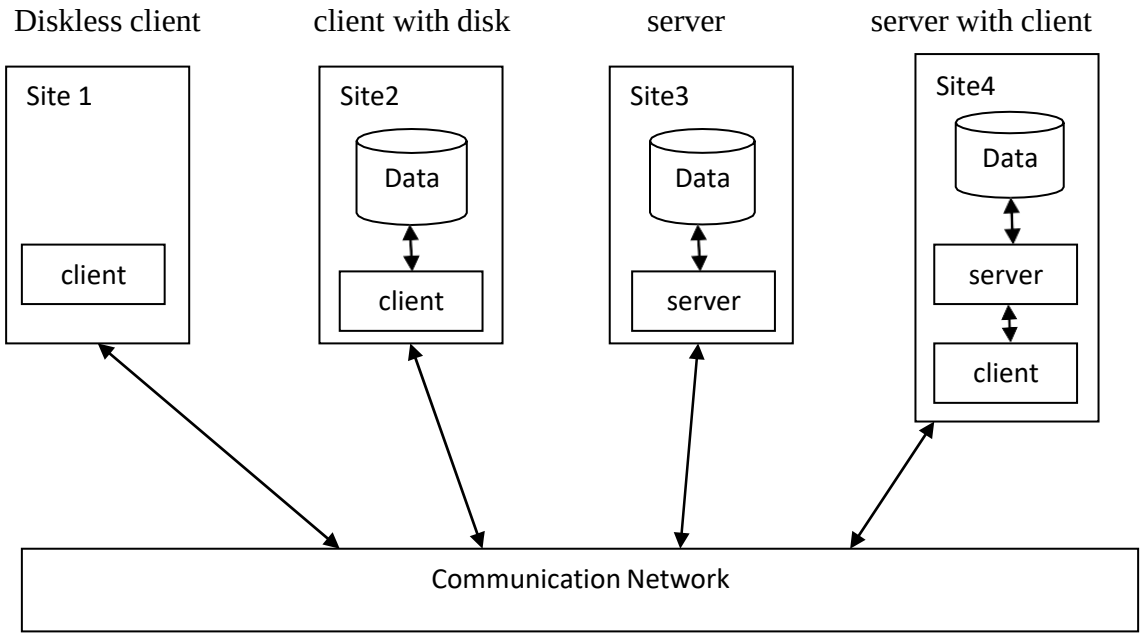
- **Menu based Interfaces** : These interfaces present the user with lists of options that lead the user through the formation of request.
- **Forms based Interfaces** : This interface displays a form to each user. Users can fill out all the form entries to insert new data.
- **GUI (Graphical User Interface)** : This interface displays diagrammatic form to the user. So user can specify a query by manipulating the diagram. GUI utilizes both menus and form. These can use pointing device such as mouse to pick certain parts of diagram.
- **Natural Language Interface** : This interface accepts the request written in English or some other language. This interface has its own schema as well as dictionary of important words.
- **Speech Input & Output** : This interface use speech as input query and speech as answer to query.

- **Interface for Parametric users** : This interface allows the parametric users like bank tellers to proceed with minimal number of keystrokes. The goal of this interface is to minimize the number of keystrokes required for each request.
- **Interfaces for DBA** : This interface is used by DBA itself. This can be used for creating accounts, setting parameters, granting account authorization, changing schema and reorganizing storage structures of database.

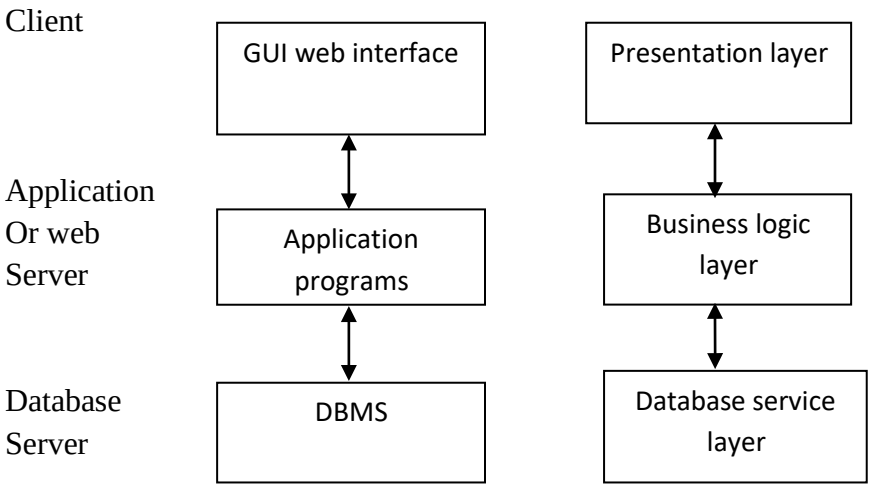
Centralized Architecture for DBMSs : In this architecture, the database system use computers as display terminals and a centralized DBMS will do all DBMS functionality, application program execution and user interface processing were carried out on one machine.

Client/Server Architecture : This architecture divides the total processing into two units called Clients and Servers. A Client is typically user machine that provides user interface capabilities and local processing. A Server is a system containing both hardware and software that can provide services to the client machines such as file access, printing, database access. This Client/Server architecture was developed to deal with computing environment in whichlarge number of PCs, workstations, file servers, printers, web servers and other equipments are connected via network. There are two main types of basic architectures for DBMS under Client/Server framework. They are given below.

Two Tier Architecture : This is the architecture in which the software components are distributed over two systems called Clients and Servers. That's why it is called Two Tier. The advantage of this architecture is simplicity and compatibility with existing systems. In this architecture, the user interface and application programs can run on client side. However the server can do the query and transaction processing functions along with data storage. Thecommunication between clients and server can be done through ODBC API.



Three Tier and n-Tier Architecture : The introduction of web can change the client/server architecture to 3 tier architecture. So many web applications use an architecture called 3 tier architecture. This is the architecture which adds an intermediate layer between client and server (database server). This intermediate layer or middle tier is sometimes called application server and sometimes web server depending on application. This server plays an intermediate role by storing business rules (procedures or constraints) that are used to access data from database server. Generally in this architecture clients contain GUI interface and additional application specific business rule. The intermediate server accepts request from client, process that request and sends database commands to database server. It then sends the processed data from the database server to clients. Thus, the user interface, application rules and data access act as 3 tiers. However sometimes the business logic layer may be divided into multiple layers based on application which is called as n-tier. This technology gives higher levels of data security and network security issues remain a major concern.



Classification of DBMS : The DBMSs are classified according to the following criteria.

- **Data Model** : According to this, DBMS is classified as Hierarchical, Network, Relational and Object Oriented DBMSs. However RDBMS is used in many current commercial systems.
- **Number of Users** : This criterion can be used to classify DBMS as single user or multi user systems. Single user system allows only one user at a time and mostly used with PCs. But multiuser support multiple users at a time.
- **Number of Sites** : This criterion can be used to classify DBMS as centralized or distributed DBMS. A DBMS is centralized if data is stored at a single computer site. In distributed DBMS can have the database and software distributed over sites which connected by network.
- **Type of Access Paths** : DBMS can also be classified on the basis of types of accesspaths. One such DBMS is based on inverted file structure.
- **Generality** : This criterion can be used to classify DBMS as general purpose or special purpose. The special purpose can be designed for specific application like airline reservation. These can't be used for other applications.

BASIC FILE STRUCTURES AND INDEX STRUCTURES

Types of File Organizations : Generally data is organized on secondary storage in terms of files. Each file has been stored in terms of records. Because huge data can't be stored in the main memory, it has to be stored on secondary storage like disk. However for processing, the data is to be accessed in to main memory. The unit of information being transferred between main memory and the disk is called a page. Buffer management can be used for reading and writing the data between main memory and disks. Disk space manager is a software that allocates space for records on the disk. When DBMS requires an additional space then it calls disk manager to allocate the space. Also DBMS informs the disk manager when it is not going to use the space.

Most widely used file organizations are

1. Heap (Unordered) file
2. Sequential (Ordered) file
3. Hash file.

- **Heap File** : It is also called unordered file because it stores records in file in the order as they arrive. This is the simplest organization .
  - **Insert Record** : Records are inserted in the same order as they arrive.
  - **Delete Record** : The page in which record is to be deleted is accessed first and marked as deleted. After deletion of record the entire page is then loaded onto the disk.
  - **Access Record** : A linear search is performed on file starting from the first record until the desired record is found.
- **Sequential File** : It is also called Ordered file because it stores the records in a sequential order. The records are ordered based on value of one field called ordering or key field. The main advantage of this file is that we can use binary search as file is sorted.
  - **Insert Record** : This is difficult task because we need to identify the place to insert. If space is available then record can directly inserted. If space is not sufficient then successive records must be moved to next pages which is tiresome job.

- **Delete Record** : This is also difficult job as insertion because the pages must move back to remove the empty space of deleted records.
- **Access Record** : This task becomes simple because we can use binary search as file is sorted.

- **Hash File** : In these files, records are stored randomly instead of sequentially. A function called hash function can be used to store and retrieve the records from these files.

INDEXES : Indexes enhance the performance of DBMS. They enable us to go to the desired record directly without scanning each record in the files. This is similar to an index page of a book. i.e., the index page enables us to find the desired keyword in the book by avoiding the need of sequential scan through the complete book. So an index can be defined as data structure that allows faster retrieval of data. Each index is based on certain attribute of field. This is given in search key. We can have several indexes based on search keys.

Types of single level ordered indexes : There are several types of ordered indexes. They are

1. Primary Index
2. Clustering Index
3. Secondary Index.

- **Primary Index** : Index on set of fields that include primary key is called primary index. A primary index is an ordered file whose records are of fixed length with two fields. The first field is called key field which is called primary key of data file and second field is a pointer to disk block. There is only one index entry in index file for each block of data file. We will refer two field values of index entry 'i' as <K(i),P(i)>. This primary index can further divided into two types called
  - **Dense Index** : It has an index entry for every search key value in the data file.The index record contains search key value and a pointer to first data record with that search key value.
  - **Sparse Index** : This index has entries for only some of the search values. So each index entry record contains a search key value and a pointer to the first data record with the largest search key value that is less than or equal to search key

value for which we are looking. We start at record pointed to by index entry and follow the pointers in the file until we find the desired record.

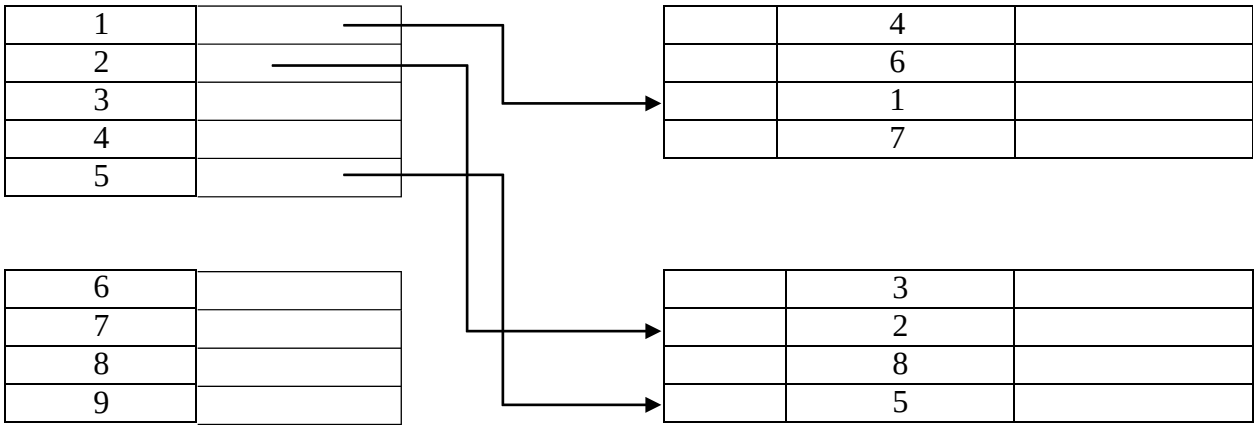
- **Clustering Index** : If the file records are physically ordered on a nonkey field which does not have distinct value for each record, that field is called clustering field. The indexcreated for such field is called clustering index. This clustering index is different from primary index which requires that ordering field of data file have a distinct value for eachrecord.

  This clustering index is also an ordered file with 2 fields. The first field isclustering field and second field is block pointer. There is one entry in clustering indexfor each distinct value of clustering field and containing a pointer to the first block in datafile that has record with clustering field value.
- **Secondary Index** : An index that is not primary key index is called secondary index. This index provides secondary access which means that primary access already exist for that file. This secondary index may be on a field which is candidate key and has unique value in every record or a non key with duplicate values. This index is also an orderedfile with 2 fields. The first field is indexing field and second field is either block pointer or record pointer. In general secondary index is different form primary index if searchkey is not primary key. i.e., a secondary index must contain pointers to all records because records are ordered by search key of primary index but not by search key of secondary key index. So the records with same search key value could be anywhere in the file. Therefore this index needs more space and more search time than primary index. However it improves the search time that use keys other than the search key of primary index.

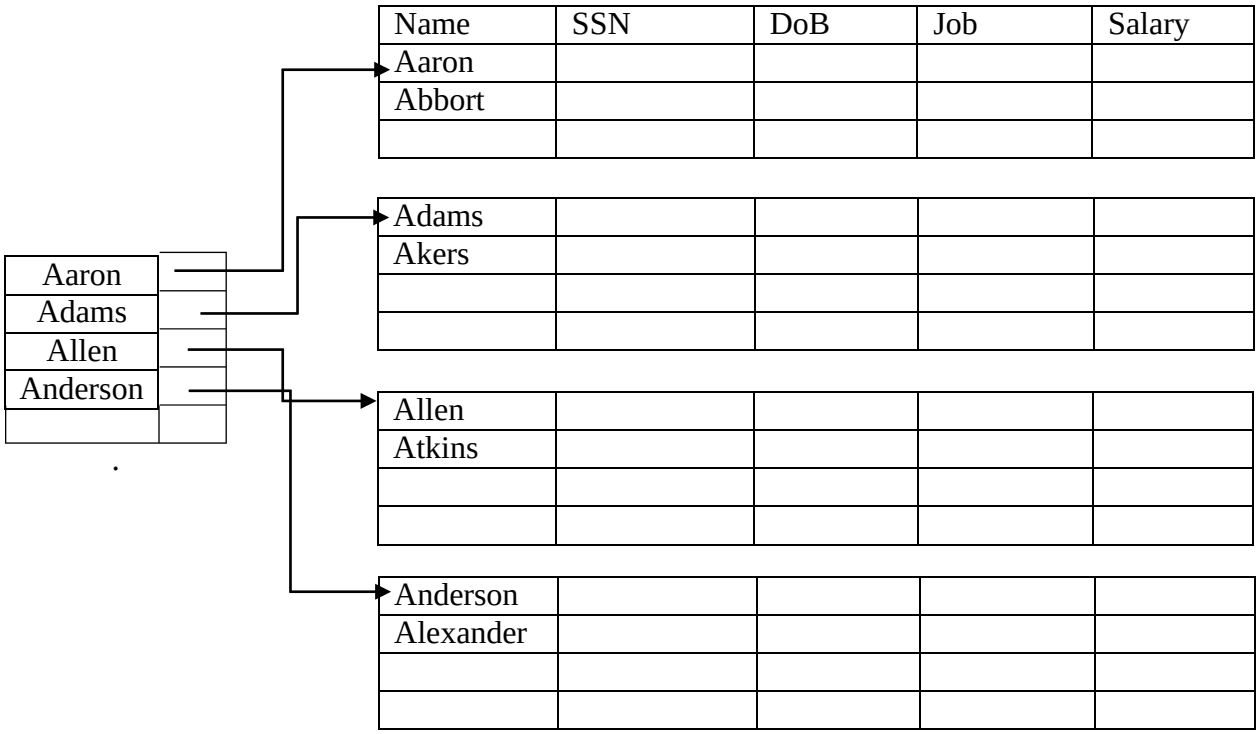
Index Data Structures : There are two methods that can be used to organize the index files. They are

1) **Hash based Indexing** : In this, a hash function is used to find which block contains the desired record. So file records are grouped into blocks, which contain a primary page along with other pages that are chained together.
2) **Tree based Indexing** : In this, records are arranged in tree like structure. The records are arranged according to the search key values and they are in a hierarchical manner.

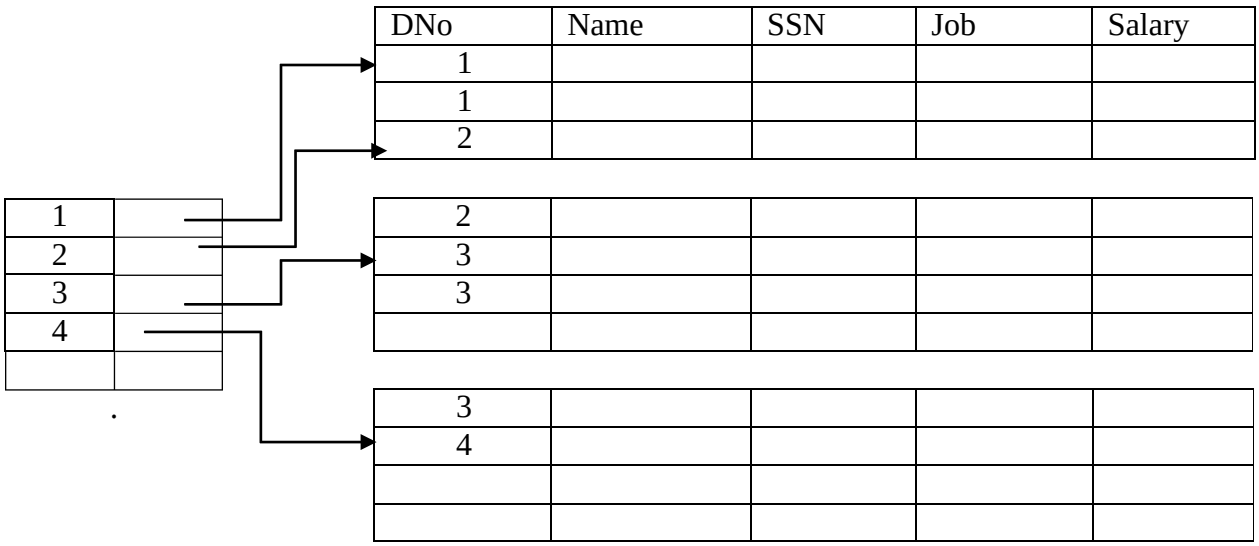
Secondary Index :



Primary index :



Clustering Index :



DATA MODELING USING ER MODEL

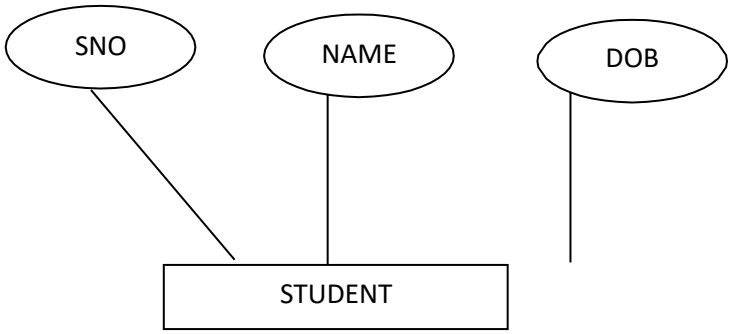
Introduction : Conceptual Modeling is very important phase in design of successful database application. The database application refers to database and associated programs that implement database queries and updates. The ER is advanced and dominant model which is closest to the conceptual model of database. So this model is frequently used for the conceptual design of database application. In this model, the overall logical structure of database can be expressed by a diagram known as ER diagram. This ER model describes the data as Entities, Relationships and Attributes.

Entities : The basic object that ER model represents is an Entity. It is a thing in real world with independent existence. It may be an object with physical existence like person, car, house, employee etc or it may be object with conceptual existence like company, job and course etc. These entities are represented by a rectangle containing the entity type name in it.

Ex :

Entity Type & Entity Set : Entity type defined as a collection of entities that have the same attributes. The collection of all entities of particular type in the database at any point in time is called Entity Set.

Attributes : Each entity has certain characteristics known as attributes. An attribute is a characteristic property of entity. In ER diagram these attributes are represented by ovels or ellipses enclosing the attribute name and they are attached to their entity type by straight lines. These attributes have domains. A domain is a set of all possible values for attributes. Sometimesan attributes share a domain. For example student and professor can share the domain of all possible addresses.

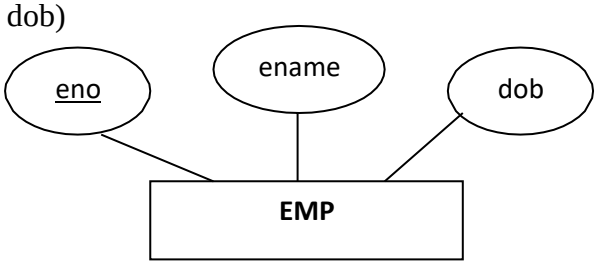


Classification of Attributes : Attributes cab be classified as

1. Key Attributes
2. Simple & Composite Attributes
3. Single valued & Multi-valued Attributes
4. Stored & Derived Attributes

Key Attributes : A key attribute is the attribute whose values are distinct for each individual entity in entity set. So this is used as primary key. This attribute is underlined inside the ovel.

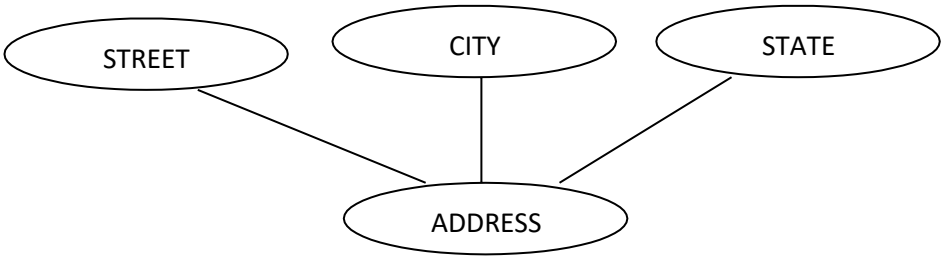
Ex : EMP(eno, ename, dob)



Simple Attribute (Atomic) : It is an attribute that cannot be further subdivided is called simple attribute.

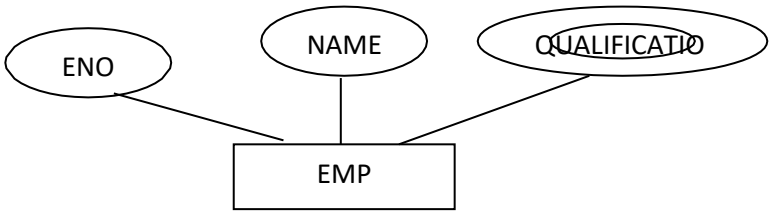
Ex : Age

Composite Attributes : It is an attribute that can be further subdivided to yield additional attributes. Ex: address can be subdivided into street, city, state and zip code etc.



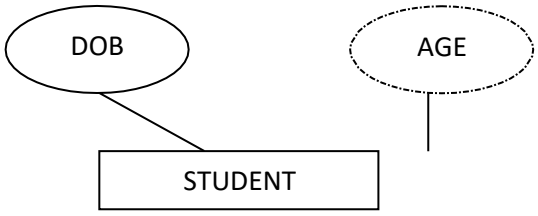
Single-Valued Attribute : This is an attribute which can have single value. For example age is single valued attribute of a person. However single valued attribute is need not be a simple. For example parts serial number is SE080219326 is a single valued but not simple because it can be subdivided into region in which part was produced, part number etc.

Multi-valued Attribute : It is an attribute which can have many values. In ER diagram this attribute is represented by a double line ovels. For example color of a car and qualification of an employee etc.

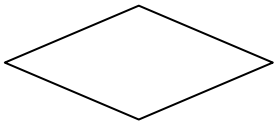


Stored Attribute : It is an attribute whose value is physically stored in the database.

Derived Attribute : It is an attribute whose value is determined from the value of existing attribute. i.e., this need not be physically stored with in database. This is represented by a dotted ovel in ER diagram. For example age attribute is a derived attribute.



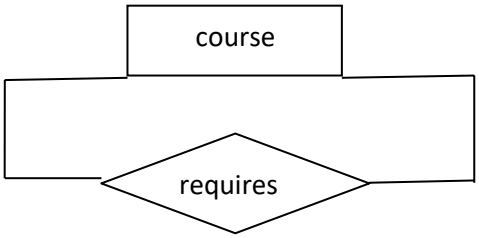
Relationships : A relationship is an association between entities. This is represented by diamond symbol in ER diagram. A relationship type R among n entities E1,E2,E3,……,En defines a set of associations. This collection of associations is also called a Relationship set.



Degree of Relationship : It is the number of participating entity types in that relationship. i.e., it indicates number of entitites that are associated. Based on this degree the relationships can be categorized as

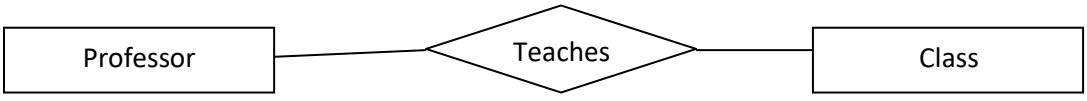
1. Unary Relationships
2. Binary Relationships
3. Ternary Relationships
4. Higher order degree relationships.

- **Unary Relationship** : This is relationship which exists when an association is maintained within a single entity type.

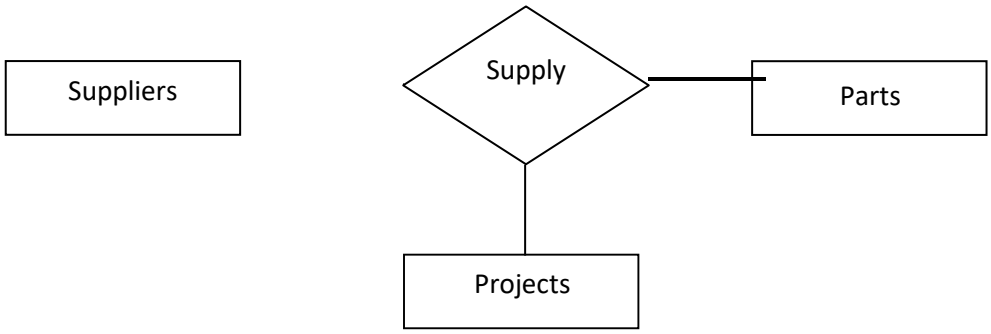


In the above example course within a course is prerequisite for another course. i.e., course required a course. i.e., course has a relationship with itself. Such relationships are also called as +Recursive relationships.

- **Binary Relationship**: It is a relationship which exists when association is maintained among two entities. In fact in order to simplify the conceptual design, most higher order relationships are decomposed into appropriate binary relationships whenever possible.



- **Ternary Relationship** : This is one which exists when association is maintained among 3 entities.



Constraints on Relationship Types : Relationship types usually have certain constraints that limit the combinations of entities that participate in relationship set. There are two main types of relationship constraints which are

1. Cardinality ratio.
2. Participation constraints

Both of these are called **Structural Constraints**.

1. **Cardinality Ratio** : It specifies the maximum number of entity instances associated with one occurrence of related entity. For example in Teaches relationship the cardinality ratio for professor : class is 1:N, means that each professor can teach any number of classes. But each class is associated to only one professor. i.e., 1:1
2. **Participation Constraint** : It specifies whether the existence of an entity on its being related to another entity through relationship type. i.e., this constraint specifies the minimum number of instances that entity can participate in relationship. So sometimes it is also called minimum cardinality constraint.

   There are two types of participation constraints. They are total and partial. The total participation means that every entity must be related to the another type entity in the relationship. Thus it is also called existence dependency. However partial participation means that some entities are related to some other entities in the relationship but not necessary all.

Weak Entity : An entity that do not have key attribute of its own is called weak entity. However in contrast entity that have a key attribute is called Regular or Strong Entity. Usually key attribute of this weak entity is formed by combining the attribute of an entity to which it is related. So this weak entity always has total participation. i.e., it is always existence dependent. This can be represented by double line rectangle in ER diagram.

Example:

Consider the entity type dependent related to employee entity, which is used to keep track of the dependents of each employee. The attributes of dependents are : name
,birthrate, sex and relationship. Each employee entity set is said to its own thedependent entities that are related to it. How ever, not that the 'dependent' entity doesnot exist of its own., it is dependent on the employee entity. In other words we can saythat in case an employee leaves the organization all dependents related to without the entity 'employee'. Thus it is a weak entity.

Keys:

Super key:

A super key is a set of one or more attributes that taken collectively, allow us toidentify uniquely an entity in the entity set.

For example , customer-id,(cname,customer-id),(cname,telno)

Candidate key:

In a relation R, a candidate key for R is a subset of the set of attributes of R, whichhave the following properties:

Uniqueness: no two distinct tuples in R have the same values for the candidate key

Irreducible: No proper subset of the candidate key has the uniqueness property that is the candidate key.

Eg: (cname,telno)

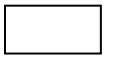
Primary key:

The primary key is the candidate key that is chosen by the database designer as the principal means of identifying entities with in an entity set. The remaining candidate keys if any, are called *alternate key*.

ER-DIAGRAM:

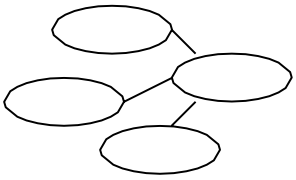
The overall logical structure of a database using ER-model graphically with the help of an ER-diagram.

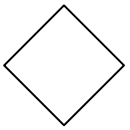
Symbols use ER- diagram:


entity


Weak entity

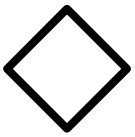

attribute

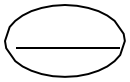

composite attribute

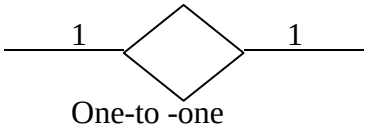

Relationship


Multi valued attribute

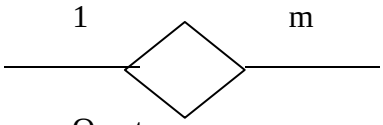

Derived attribute


Identifying Relationship

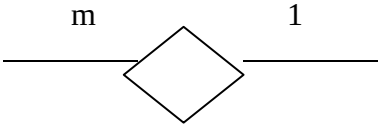

Key attribute



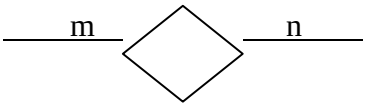
One-to -one



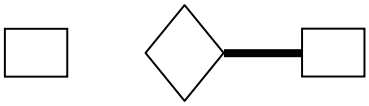
One-to -many

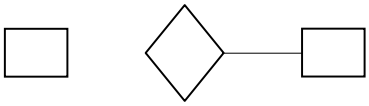


many-to -one



many-to -many


Total participation


Partial participation

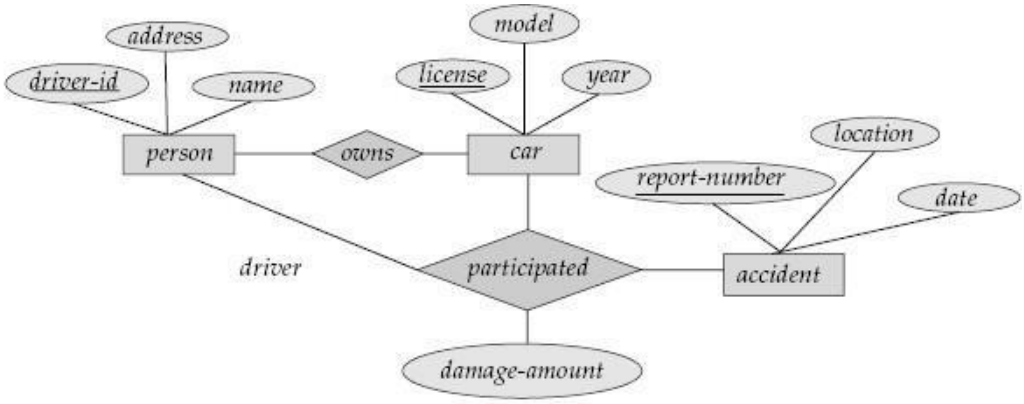


Figure 2.1 E-R diagram for a Car-insurance company.

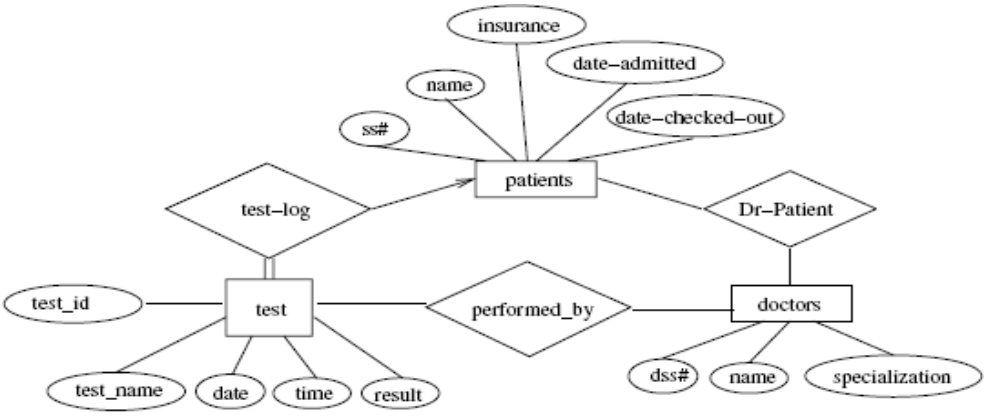


Figure 2.2 E-R diagram for a hospital.

A university registrar's office maintains data about the following entities: (a) courses, including number, title, credits, syllabus, and prerequisites; (b) course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom; (c) students, including student-id, name, and program; and (d) instructors, including identification number, name, department, and title. Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled.

Construct an E-R diagram for the registrar's office. Document all assumptions that you make about the mapping constraints.

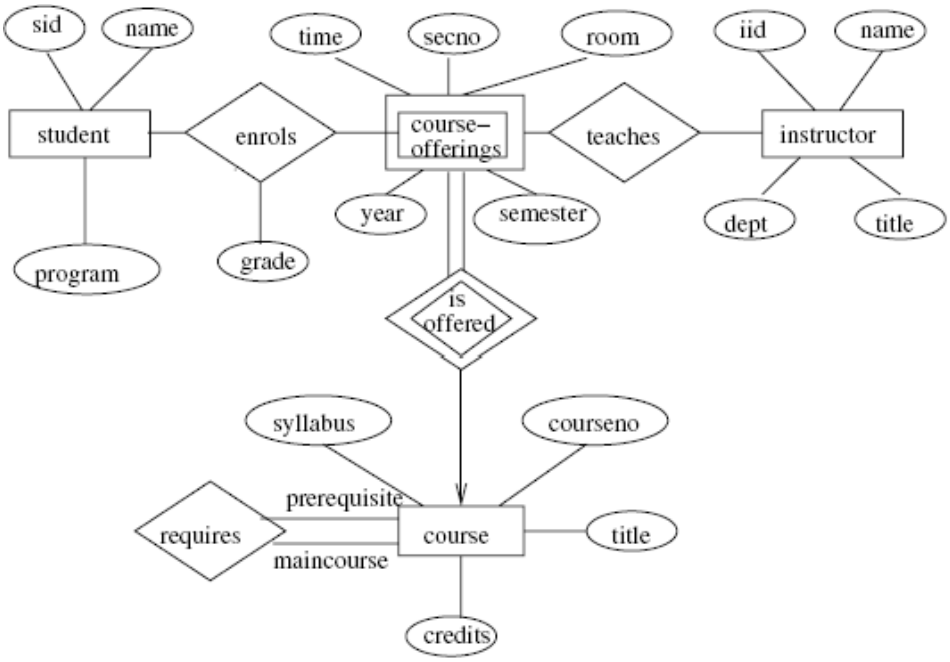


Figure 2.3 E-R diagram for a university.

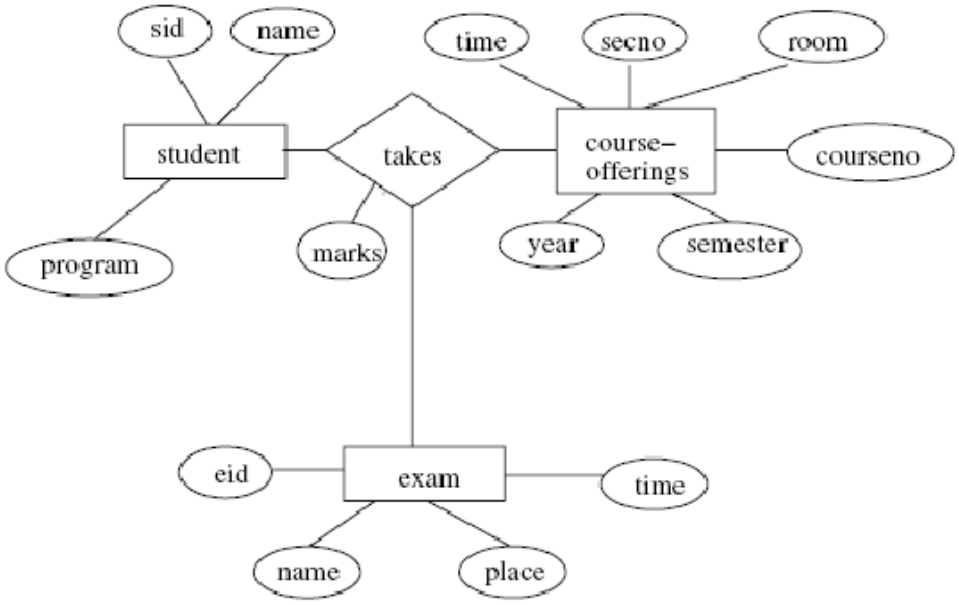


Figure 2.4 E-R diagram for marks database.

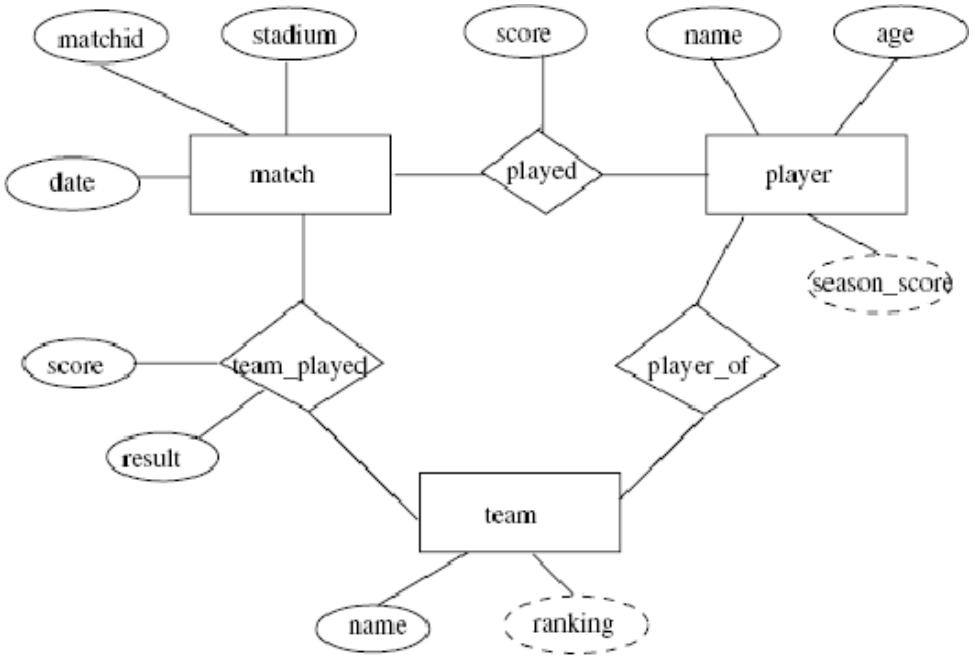


Figure 2.7 E-R diagram for all teams statistics.

Consider a university database for the scheduling of classrooms for final exams. This database could be modeled as the single entity set *exam*, with attributes *course-name*, *section-number*, *room-number*, and *time*. Alternatively, one or more additional entity sets could be defined, along with relationship sets to replace some of the attributes of the *exam* entity set, as

- *course* with attributes *name*, *department*, and *c-number*
- *section* with attributes *s-number* and *enrollment*, and dependent as a weak entity set on *course*
- *room* with attributes *r-number*, *capacity*, and *building*

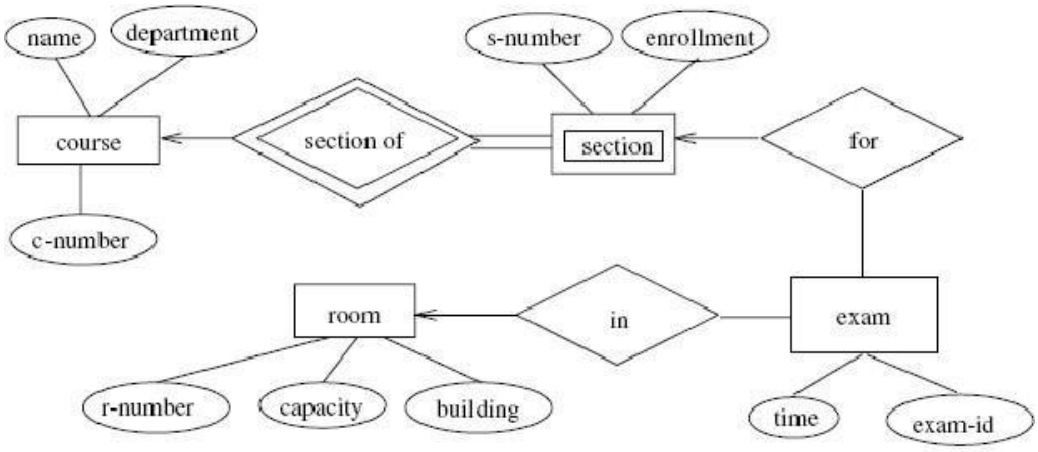
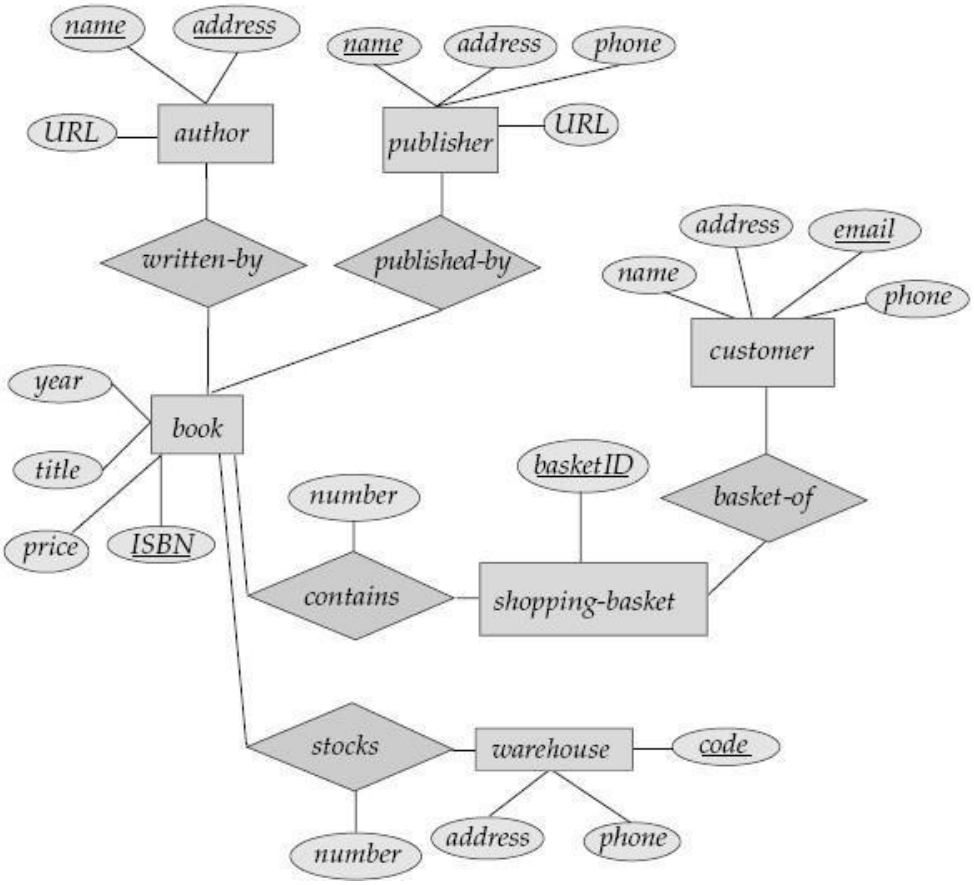


Figure 2.12 E-R diagram for exam scheduling.



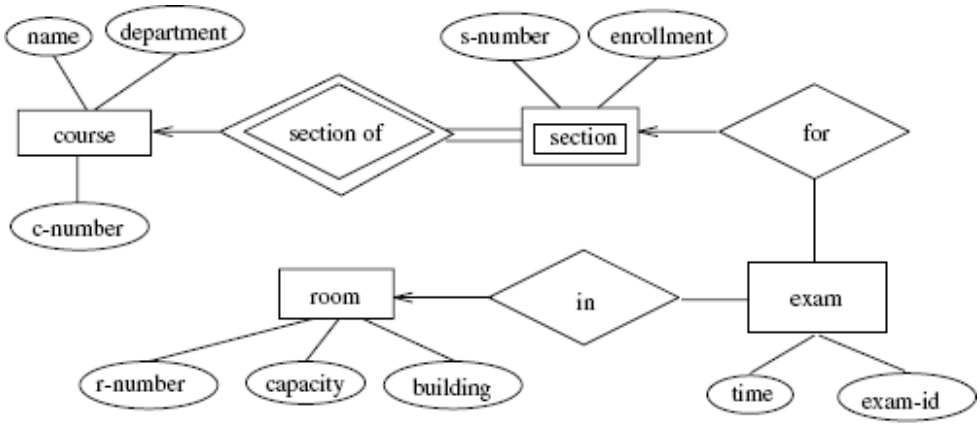
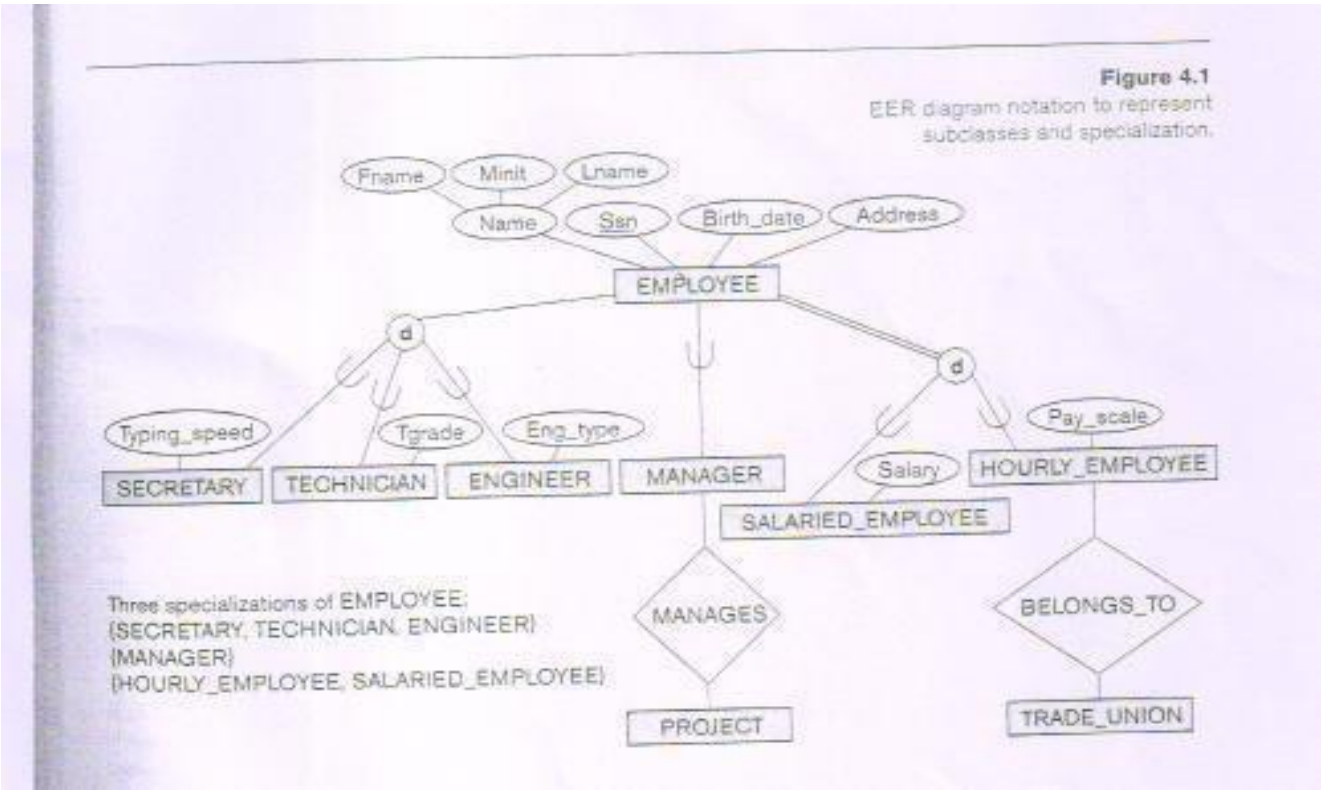


Figure 2.12 E-R diagram for exam scheduling.

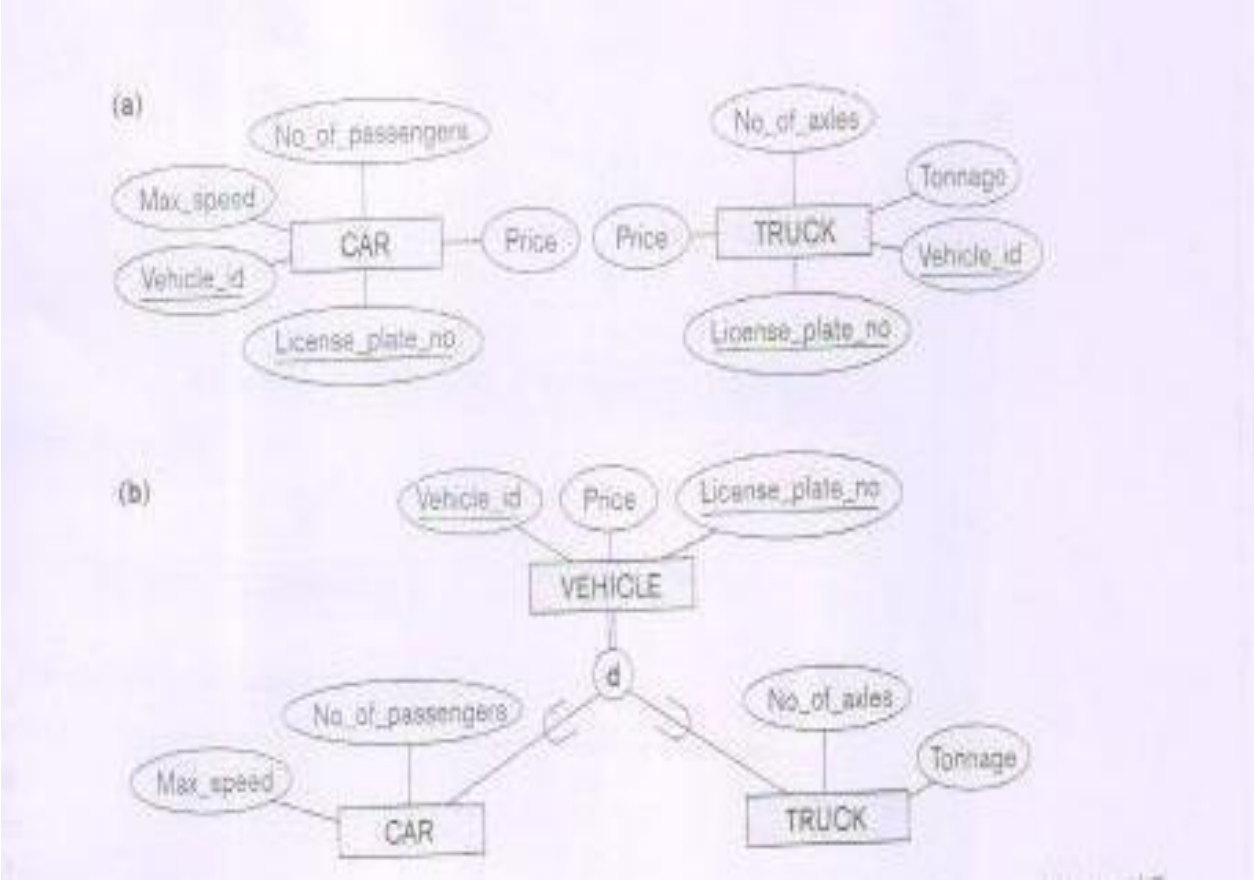
Enhanced Entity Relationship Model(EER Model)

This model includes the concepts like Specialization and Generalization. This EER is a complex model which includes semantic concepts that give a more complex picture of the precise nature of the data in a given system.

Specialization : It is a process of defining a set of subclasses of an entity type which is called super class of Specialization. The set of subclasses can be defined on the basis of some distinguishing characteristics of entities in super class. For example the following diagram shows the specialization of Employee based on job characteristics. The lower entities are called subclass entities. Subclass entities inherits all the attributes of super class entities. The relationship between super and sub class entities is called Class/Subclass relationship or IS-A relationship. So the set of sub classes secretary, engineering, technician is specialization of super class EMP based on job type. Another specialization of EMP may yield a set of subclasses salaried and hourly based on method of pay.



Generalization : Actually it is a reverse process of specialization. i.e., it is a process of defining generalized entity from the given entity types. i.e.,. it is a process by which the subclassentities are generalized into a single super class entity type. For example consider entity types CAR & TRUCK. They can be generalized in to entity type VEHICLE because they have some common attributes. So CAR & TRUCK are the subclasses of generalized super class VEHICLE.



Specialization & Generalization Hierarchies & Lattices : A subclass itself may have further subclasses specified on it, i.e., forming a hierarchy or a lattice of specialization. For example Engineering is a sub class of EMP and is a super class of Eng-Manger. A specialization hierarchy has a constraint that every subclass has only one parent, which results a tree structure. However in specialization lattice, a subclass can have more than one parent which results lattice.

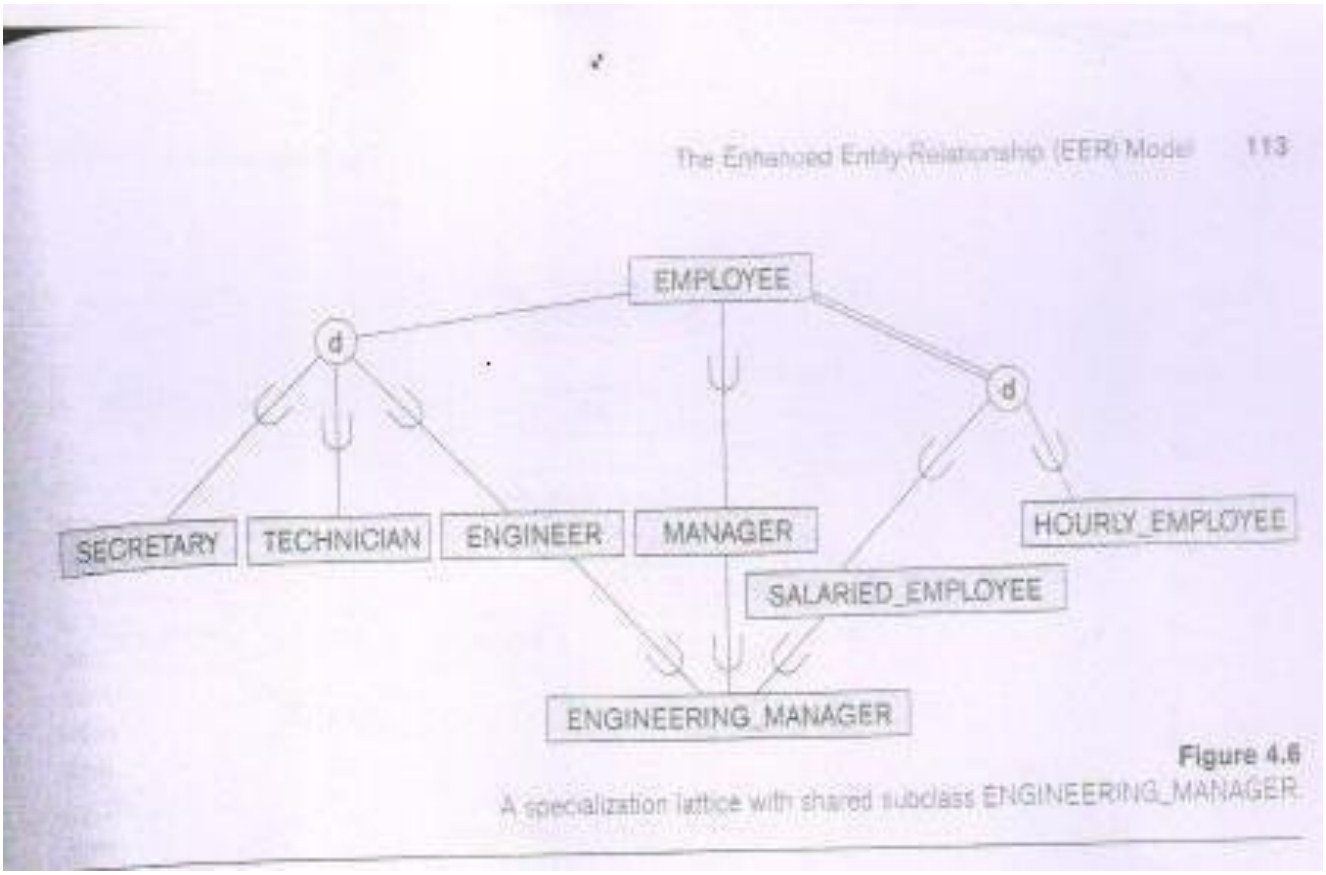


Figure 4.6
A specialization lattice with shared subclass ENGINEERING_MANAGER.

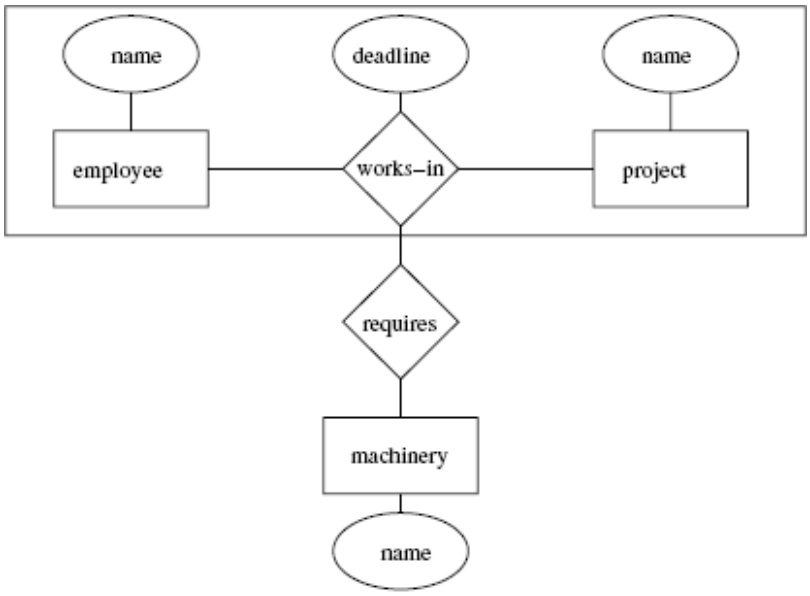


Figure 2.8 E-R diagram Example 1 of aggregation.

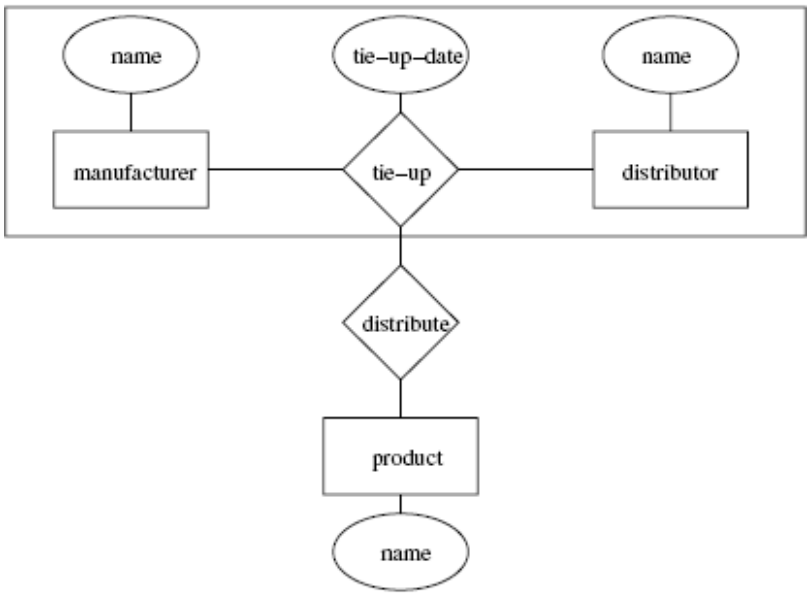
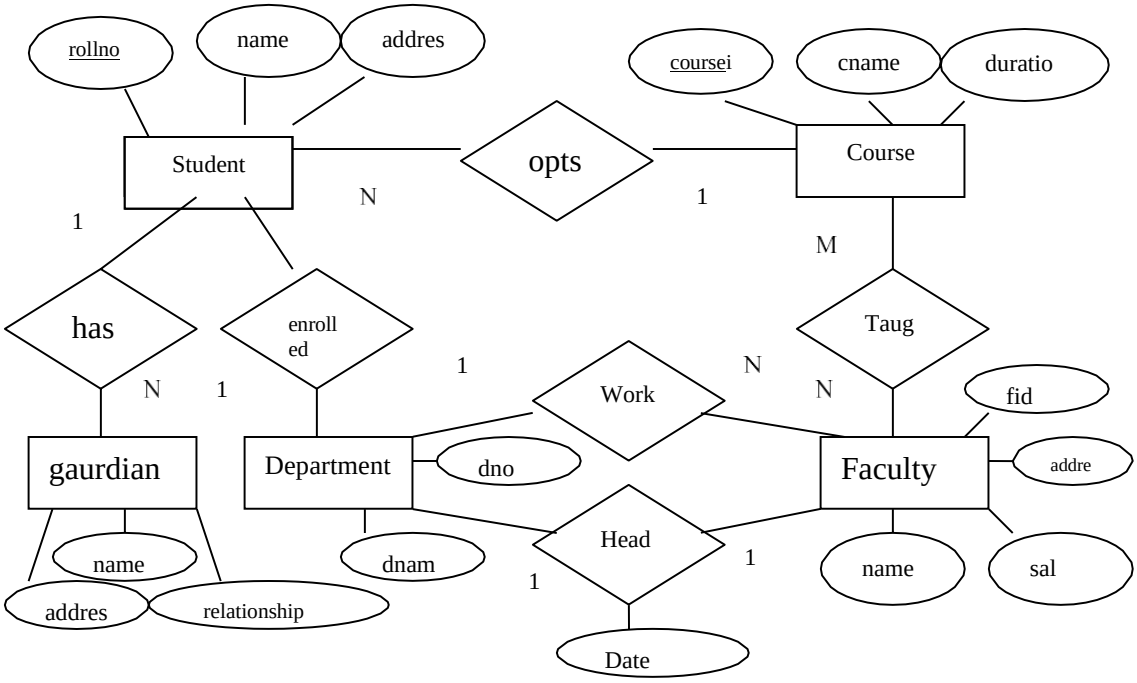


Figure 2.9 E-R diagram Example 2 of aggregation.

ER- Diagram For College Database



Conversion of ER-diagram to relational database

Conversion of entity sets:

1. For each strong entity type E in the ER diagram, we create a relation R containing all the single attributes of E. The primary key of the relation R will be one of the key attribute of R.

   STUDENT(rollno (primary key),name, address)
   FACULTY(id(primary key),name ,address, salary)
   COURSE(course-id,(primary key),course_name,duration)
   DEPARTMENT(dno(primary key),dname)

2. for each weak entity type W in the ER diagram, we create another relation R that contains all simple attributes of W. If E is an owner entity of W then key attribute of E is also include In R. This key attribute of R is set as a foreign key attribute of
R. Now the combination of primary key attribute of owner entity type and partial key of the weak entity type will form the key of the weak entity type

GUARDIAN((rollno,name) (primary key),address,relationship)

Conversion of relationship sets:Binary Relationships:

- **One-to-one relationship:**

For each 1:1 relationship type R in the ER-diagram involving two entities E1 and E2 we choose one of entities(say E1) preferably with total participation and add primary key attribute of another E as a foreign key attribute in the table ofentity(E1). We will also include all the simple attributes of relationship type R in E1 if any, For example, the department relationship has been extended tp include head-id and attribute of the relationship.

DEPARTMENT(D_NO,D_NAME,HEAD_ID,DATE_FROM)

- **One-to-many relationship:**

For each 1:n relationship type R involving two entities E1 and E2, we identify the entity type (say E1) at the n-side of the relationship type R and include primary key of the entity on the other side of the relation (say E2) as a foreign key attribute in the table of E1. We include all simple attribute(or simple components of a composite attribute of R(if any) in he table E1)
For example:
The works in relationship between the DEPARTMENT and FACULTY. For this relationship choose the entity at N side, i.e, FACULTY and add primary key attribute of another entity DEPARTMENT, ie, DNO as a foreign key attribute in FACULTY.
FACULTY(CONSTAINS WORKS_IN RELATIOSHIP)
(ID,NAME,ADDRESS,BASIC_SAL,DNO)

- **Many-to-many relationship:**

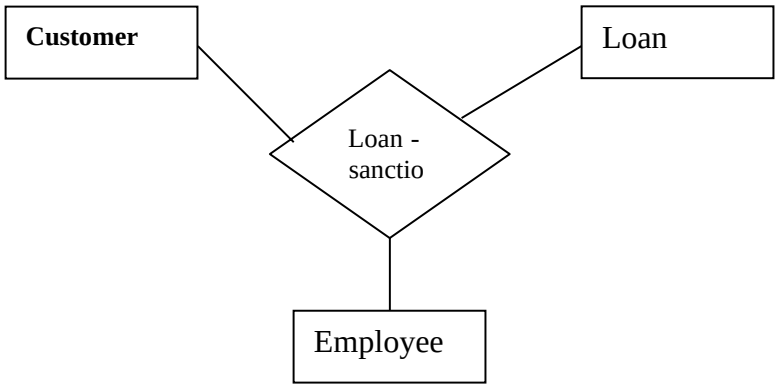
For each m:n relationship type R, we create a new table (say S) to represent R, Wealso include the primary key attributes of both the participating entity types as a foreign key attribute in s. Any simple attributes of the m:n relationship type(or simple components as a composite attribute) is also included as attributes of S. For example:

The M:n relationship taught-by between entities COURSE; and FACULTY shod be represented as a new table. The structure of the table will include primary key of COURSE and primary key of FACULTY entities.

TAUGHT-BY(ID (primary key of FACULTY table),course-id (primary key of COURSE table)

- **N-ary relationship:**

For each n-anry relationship type R where n>2, we create a new table S torepresent R, We include as foreign key attributes in s the primary keys of the relations that represent the participating entity types. We also include any simple attributes of the n-ary relationship type(or simple components of complete attribute) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types.



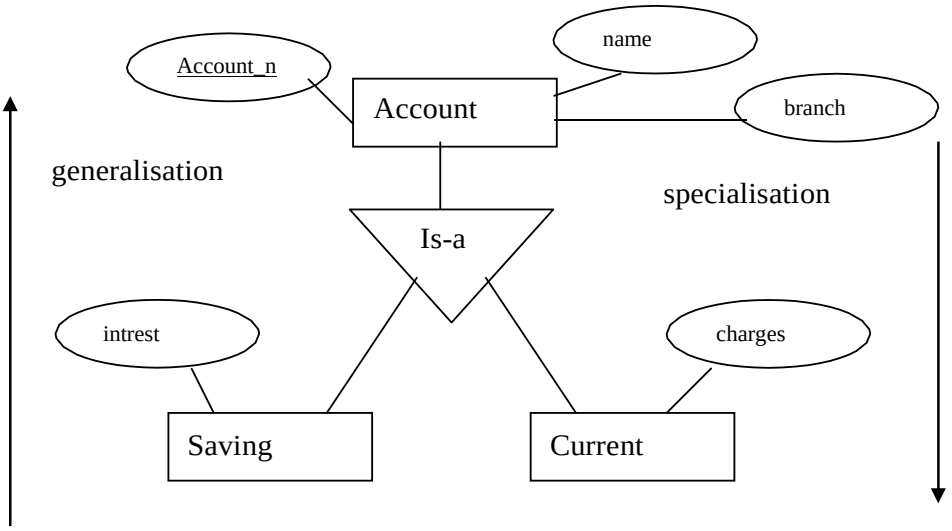
LOAN_SANCTION(customer_empno,loan_sanction,loan_date,loan_amount)

- **Multi-valued attributes:**

For each multivalued attribute ‘A’, we create a new relation R that includes an attribute corresponding to plus the primary key attributes k of the relation that represents the entity type or relationship that has as an attribute. The primary key of R is then combination of A and k.

For example, if a STUDENT entity has rollno,name and phone number where phone numer is a multivalued attribute the we will create table PHONE(rollno,phoneno) where primary key is the combination,In the STUDENTtable we need not have phone number, instead if can be simply (rollno,name) only.

PHONE(rollno,phoneno)



- **Converting Generalisation /specification hierarchy to tables:**

  A simple rule for conversion may be to decompose all the specialized entities into table in case they are disjoint, for example, for the figure we can create the two table as:

  Account(account_no,name,branch,balance)

  Saving account(account-no,intrest)

  Current_account(account-no,charges)

RELATIONAL DATA MODEL

Introduction : This model uses the concept of mathematical relation that looks like a table. This relation is a basic building block of this model. So this relational model represents a database as a collection of relations. In the formal relational model, row is called a tuple, a column is called an attribute and a table is called as a relation.

Domain, attributes tuples and relational:

Tuple:

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

Attributes:

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

In the above table, account_number, branch name, balance are the attributes.

Domain : A domain D is a set of atomic values. It means that an each value in the domain is indivisible as far as relational model is concerned. A common method to specify a domain is to specify a data type. For example emp_age, phone_num, etc.

Relation:

A relation consist of

- **Relational schema**
- **Relation instance**

Relation : A relation r of a relation schema R(A1, A2, A3,……,An) is denoted by r(R), is a set of n-tuples. That is r = {t1, t2, t3,…..,tn}. Each tuple t is an ordered list of n values i.e t = <v1, v2, v3,…..,vn> where vi is element of dom (Ai) where $1 \leq I \leq n$.

Relational schema:

A relational schema specifies the relation' name, its attributes and the domain of each attribute. If R is the name of a relation and A1,A2,… and is a list of attributes representing R then R(A1,A2,…,an) is called a relational schema. Each attribute in this relational schema takes a value from some specific domain called domain(Ai).

Example:

PERSON(PERSON_IDinteger,NAME:

STRING,AGE:INTEGER,ADDRESS:string)

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

Relational Database Schema : A relational database schema S is a set of relation schemas S={R1, R2, R3,…..,Rn} and set of integrity constraints (ICs). For example

COMPANY ={ EMP, DEPT, PROJECT, WORKS_ON, DEPENDENT}

Relation Instance:

A relational instance denoted as r is a collection of tuples for a given relational schema at a specific point of time.

A relation state r to the relations schema R(A1,A2…,An) also denoted by r® is a set of n-tuples

R{t1,t2,…tm}

Where each n-tuple is an ordered list of n values T=<v1,v2,….vn>

Where each vi belongs to domain (Ai) or contains null values.

The relation schema is also called ‘intension’ and the relation state is also called ‘extension’.

Eg:

Relation schema for student:

STUDENT(rollno:strinhg,name:string,city:string,age:integer)

Relation instance:
Student:

Rollno	Name	City	Age
101	Sujit	Bam	23
102	kunal	Bbsr	22

Relational Model Constraints : In relational database, there may be many relations and tuples of those relations are related in various ways. Generally there are many restrictions on actual values in the tuples of database. They are called as constraints. These constraints on databases can generally divided into 3 types. They are

- **Inherent model based or Implicit constraints** : These are the constraints that are inherent in the data model.
- **Schema based or Explicit constraints** : These are the constraints that can be directly expressed in schemas of data model. These can be specified in DDL.
- **Application based or Semantic constraints** : These are the constraints that can’t be expressed directly in schemas of data model. So these must be expressed and enforced by application programs.

RELATIONAL CONSTRAINTS (SCHEMA based):

There are three types of constraints on relational database of schema based constraints that include

- DOMAIN CONSTRAINTS
- KEY CONSTRAINTS
- INTEGRITY CONSTRAINTS

These include

- **Domain Constraints** : These constraints can specify that within each tuple, the value of each attribute A must be atomic from the dom(A).
- **Key Constraints** : These constraints can specify that no two tuples can have the same value for their attributes. So there must be an attribute or set of attributes with property that they should have unique values for all the tuples. i.e t1[sk] ≠ t2[sk] where sk is a subset of attributes. Any such set of attributes sk is called super key. There are certain keys that we can specify for a relation. They are
  - **Super key** : It is an attribute or set of attributes that uniquely identifies a tuple within the relation.
  - **Candidate Key** : it is a minimal super key i.e., it is a super key with no proper subset itself a super key.
  - **Primary Key** : It is a candidate key that was selected to identify tuples uniquely within a relation.
  - **Alternative Key** : It is also a candidate key that was not selected to be primary key. So these are also referred as secondary key.
  - **Foreign Key** : It is an attribute or set of attributes within one relation that should match a candidate key of some other relation.
  - **Integrity Constraints** : There are two types of Integrity Constraints in this model andthese constraints include
  - **Entity Integrity Constraint** : It states that every relation should have a primary

key and its value can not be NULL. This is because Primary Key value is used to identify individual tuples in a relation. So having NULL value for Primary Key we can't identify some typles.

- **Referential Integrity Constraint** : It is specified between tow relations and it is used to maintain consistency among tuples in the relations. However entity integrity constraint can be specified on a single relation. This referential constraint states that a tuple in one relation must refer to an existing tuple in another relation. To define this constraint we use foreign key concept. So when we set a foreign key between R1 and R2 then foreigh key of R1 refers to Primary key of R2 if it satisfies the following conditions.
  - Attributes of Foreign key have the same domain of Primary Key.
  - The value of Foreign Key is either a value of Primary Key or NULL.

RELATIONAL ALGEBRA & RELATIONAL CALCULUS

Introduction : These are the two formal languages for relational model of data. The language by which we can communicate with database is known as query language. This query language can be either non-procedural or Procedural. Relational algebra is procedural language and relational calculus is non-procedural language.

Relational Algebra : It is a method of specifying a sequence of operations to be performed on the database to compute the desired results. It consists of a set of operations that take one or two relations as input and produce a new relation as result. **Relational calculus** provides a higher level declarative notation for specifying relation queries. It specifies what information the result should contain i.e., there is no order of operations to specify how to retrieve the query result. A relational calculus produces a new relation which is specified in terms of variables that range over rows (tuple calculus) or over columns (domain calculus). This is the main distinguishing feature between relational algebra and relational calculus.

Relational Algebra : Actually it is considered to be an integral part of relational data model. It is a collection of operations to manipulate the relations. It specifies the operations to be performed on existing relations to derive resultant relations. The set of operations can be classified into two types. They are

- **Traditional Set Operations** : These can be further categorized into unary and binary operations. Unary operations are the operations that can be operated on only one (single) relation. However binary operations are those operations that can be operated on two relations.
- **Unary Operations** : These operations are
- **SELECT Operation** : It is used to select subset of tuples from a relation that satisfies a selection condition i.e., this operation results only those tuples that satisfy specified condition. So it will produce horizontal partition (subset) of a relation. In general, this operation is denoted by σ as follows :

$$\sigma_{\text{<select condition>}}(R)$$

where the symbol **σ** (sigma) is used to denote SELECT operation and <select condition> is a Boolean expression specified on the attributes of relation R.

For Example : Select emps whose deptno is 4 or those whose salary is greater than 30,000

Query : $\sigma_{(dno = 4 \text{ or } sal > 30,000)}(EMP)$

- **PROJECT Operation** : This is used to select certain attributes from a relation while discarding other attributes. Therefore the result of this operation can be visualized as vertical partition of a relation. In general, this operation can be denoted by **π** as follows :

$$\Pi_{\text{<attribute list>}}(R)$$

Where π is the symbol used to represent PROJECT operation and attribute list is the desired list of attributes from the relation R. The result has attribute list in the same order they appear in the list.

For Example : To list each employee's first name, last name and salary, we use following

Query : $\Pi_{lname, fname, sal}(EMP)$

- **RENAME Operation** : This can be used to rename a relation or attribute name or both. This is denoted by **ρ** as follows:

1. $\rho_{S(B1,B2,B3,\ldots,Bn)}(R)$
2. $\rho_{S}(R)$
3. $\rho_{(B1,B2,B3,\ldots,Bn)}(R)$

where **ρ** (read as rho) is the symbol used to denote rename operation. S is new relation name and B1, B2, B3,…,Bn are new attribute names. The first form renames both relation and attributes, 2nd form renames only the relation and 3rd form renames only the attribute name of the relation R.

❖ **Binary Operations** : These are the operations that can be applied on two relations. These are related to mathematical set operations. The relations must be union compatible to

apply these operations. Two relations R(A1, A2, A3,,….,An) and S(B1, B2, B3,….,Bn) are said to be union compatible if they have same degree 'n' and dom(Ai)=dom(Bi) for all i, $1 \leq i \leq n$. These operations are given below :

- **UNION (U) :** This operation between two relations R and S can be denoted by RUS and it results all the tuples that are either in R or in S or in both R and S. However duplicate tuples are eliminated.

  For Example : Find the emps who have either salary >2000 or belong to computer science dept.

  Query : Π ENAME **($\sigma_{(sal>2,000)}$ (EMP))** UNION **Π** ENAME **($\sigma_{(DNAME='CS')}$ (EMP_DEPT))**

- **INTERSECTION (∩)** : This operation between two relations R and S can be denoted by R ∩ S and it results all the tuples that are in both R and S only. However duplicate tuples are eliminated.

  For Example : Find the emps whose salary >2000 and belong to computer science dept.

  Query : Π ENAME **($\sigma_{(sal>2,000)}$ (EMP))** INTERSECTION **Π** ENAME **($\sigma_{(DNAME='CS')}$ (EMP_DEPT))**

- **SET DIFFERENCE (MINUS**) : This operation is also referred as MINUS operation. This operation between two relations R and S can be denoted by R - S and it results all the tuples that are in R and but not in S.

  For Example : Find the department numbers that don't have employees.

  Query : Π_{dno} (DEPT) MINUS **Π_{dno} (EMP)**

- **CARTESIAN PRODUCT (X) :** This operation between two relations R and S is denoted by R X S and it results a relation that contains all possible combination of tuples of both relations R and S. If 'n' is the number of tuples in R and 'm' is the number of tuples in S then the resultant relation has n X m tuples. Usually this operation is used along with a selection condition.

  For Example : Find emp numbers who are in computer science department.

  Π_{eno} ($\sigma_{(DNAME='CS')}$ (EMP X EMP_DEPT))

- **SPECIAL RELATIONAL OPERATIONS** : In addition to traditional set operations, relational algebra also supports some special operations which include

❖ **JOIN Operation(⋈)** : This operation can be used to combine related tuples from two relations into single tuples. This operation between two relations R and S is denoted by R ⋈ S. This operation not only include cartisian product but also includes a selection operation on the result of cartisian product. This can be represented as follows :

Example : R ⋈ <selection condition> S.

❖ **DIVISION Operation (÷) :** This is special kind of operation which is denoted by R÷S**.** In general this operation is applied on two relations R(Z)S(X) where X ⊆ Z, then it results the set of attributes Y of R that are not arributes of S (Y= Z – X).

Ex : Table P Table Q

A	B
a1	b1
a1	b1
a2	b1
a2	b3
a3	b2
a4	b1

B
b1
b2

Then P ÷ Q is given as

A
a1

Example Queries

- Find all customers who have an account from at least the "Downtown" and the Uptown" branches.

Query 1

$$\Pi_{CN}(\sigma_{BN="Downtown"}(depositor \bowtie account)) \cap \Pi_{CN}(\sigma_{BN="Uptown"}(depositor \bowtie account))$$

where *CN* denotes customer-name and *BN* denotes *branch-name*.

Query 2

$$\Pi_{customer\text{-}name,\ branch\text{-}name}(depositor \bowtie account) \div \rho_{temp(branch\text{-}name)}(\{("Downtown"), ("Uptown")\})$$

Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.

$$\Pi_{customer\text{-}name,\ branch\text{-}name}(depositor \bowtie account) \div \Pi_{branch\text{-}name}(\sigma_{branch\text{-}city = "Brooklyn"}(branch))$$

Banking Example

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-only)

account (account-number, branch-name, balance)

loan (loan-number, branch-name, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

Example Queries

n Find all loans of over $1200

$$\sigma_{amount > 1200}(loan)$$

n Find the loan number for each loan of an amount greater than $1200

$$\prod_{loan\text{-}number}(\sigma_{amount > 1200}(loan))$$

Example Queries

- Find the names of all customers who have a loan, an account, or both, from the bank

$$\Pi_{customer\text{-}name}(borrower) \cup \Pi_{customer\text{-}name}(depositor)$$

- Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer\text{-}name}(borrower) \cap \Pi_{customer\text{-}name}(depositor)$$

Example Queries

Find the names of all customers who have a loan at the Perryridge branch.

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name=\text{“}Perryridge\text{”}}(\sigma_{borrower.loan\text{-}number = loan.loan\text{-}number}(borrower\ x\ loan)))$$

Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name = \text{“}Perryridge\text{”}}(\sigma_{borrower.loan\text{-}number = loan.loan\text{-}number}(\text{borrower x loan}))) - \Pi_{customer\text{-}name}(\text{depositor})$$

Example Queries

- Find the names of all customers who have a loan at the Perryridge branch.
  - Query 1

$$\Pi_{\text{customer-name}}(\sigma_{\text{branch-name} = \text{“Perryridge”}} (\sigma_{\text{borrower.loan-number} = \text{loan.loan-number}}(\text{borrower x loan})))$$

  - Query 2

$$\Pi_{\text{customer-name}}(\sigma_{\text{loan.loan-number} = \text{borrower.loan-number}}((\sigma_{\text{branch-name} = \text{“Perryridge”}}(\text{loan})) \text{ x borrower}))$$

Example Queries

Find the largest account balance n Rename *account* relation as *d* n The query is:

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

Aggregate functions:

Aggregation function takes a collection of values and returns a single value as a result.

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

Aggregate operation in relational algebra

G1, G2, …, Gn g F1(A1), F2(A2),…, Fn(An) (*E*)

E is any relational-algebra expression
*G*1, *G*2 …, *G*n is a list of attributes on which to group (can be empty)
Each *Fi* is an aggregate function.
Each *Ai* is an attribute name

Find sum of sal fo all emp records
$G_{sum(sal)}$ *(emp)*

Find maximum salalry from emp table
$G_{max(salary)}$ *(emp)*

Find branch name and maximum salalry from emp table
Branch_name $G_{,max(salary)}$ *(emp)*

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

$g_{sum(c)}(r)$

sum-C
27

OUTER JOIN:

The outer join operation is an extension of the join operation to deal with missing information.

There are three forms of outer join

- left outer join
- right outer join
- full outer join

employee:

Empname	Street	City
Coyote	Toon	Hollywood
Rabbit	Tunnel	carrot
Smith	Revolver	Death valley
William	Seaview	Seattle

Ft_works:

Empname	Branch name	Salary
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
William	Redmond	1500

Employee ⋈ ft_works

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500

Left outer join:

It takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes. The right relation and adds them to the result of the natural join. In tuple (smith, Revolcer, Death valley, null, null) is such a tuple. All information from the left relation is present in the result of the left outer join.

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death valley	Null	null

Result of Employee ⟗ ft_works

Right outer join:

It is symmetric with the left outer join. It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join. tuple(Gates,null,null,Redmond,5300) is such a tuple. Thus, all information from the right relation is present in the result of the right outer join.

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500
gates	Null	null	Redmond	5300

Full outer join:

It does both of those operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join. Figure 3.35 shows the result of a full outer join.

Since outer join operations may generate results containing null values, we need to specify how the different relation-algebra operations deal with null values. It is interesting to note that the outer join operations can be expressed by the basic relational algebra operations. For instance the left outer join operation

Employee ⟕ ft_works

Empname	Street	City	Branch name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	carrot	Mesa	1300
William	Seaview	Seattle	Redmond	1500
Gates	Null	null	Redmond	5300

RELATIONAL CALCULUS

Introduction : It is also formal query language for relational model. This is non procedural language because it specifies what is to be retrieved rather than how to retrieve it. In this, query is written as formula that comprises variables. Generally Tuple calculus and Domain calculus are collectively called as Ralational Calculus.

Tuple Calculus : It is based on specifying a number of tuple variables. Each type variable ranges over a particular relation. It means that variable may take its value as any individual tuple from that relation. A simple tuple relational calculus query is in the form of

$$\{t \mid COND(t)\}$$

Where 't' is a tuple variable and COND(t) is conditional expression involving 't'. The result is set of tuples that satisfy COND(t).

Example : Find the employee whose salary is > 5000

We can write a tuple calculus query for this as

$$\{t \mid EMP(t) \text{ and } t.sal>5000\}$$

Example : Find the employee's first and last names whose salary > 5000.

$$\{t.fname, t.lname \mid EMP(t) \text{ and } t.sal>5000\}$$

Quantifiers : Sometimes we need to use two special symbols in the formulas, called Quantifiers. They are

1. Existential quantifier, denoted by symbol (∃)
2. Universal quantifier, denoted by symbol (∀)

Existential Quantifier : If 'F' is a formula then (∃t) (F) is true if 'F' evaluates true for some (atleast) tuple assigned to free occurrence of t in F, otherwise (∃t) (F) is false.

For Example : Find employees whose salary >5000

$$\{ t \mid (\exists t)(EMP(t) \text{ and } t.sal>5000\}$$

Universal Quantifier: If 'F' is a formula then (∀t) (F) is true if F evaluates to true for every tuple assigned to free occurrence of t in F, otherwise (∀t)(F) is false.

For Example : Find employees in department number 10.

$$\{ t \mid (\forall t)\ (EMP(t) \text{ and } t.dno = 10)\ \}$$

Free and Bound Variables : A tuple variable is said to be bound variable when it is quantified by a quantifier i.e., when it is used as (∃t) or (∀t), otherwise it is free variable.

Some Example Queries in Tuple Calculus :

Example 1 : List name, address of all employee who work for the 'research' department.

{ t.fname, t.lname, t.address | EMP(t) and (∃d) (DEPT(d) and d.dname='research' and d.dno=t.dno)}

Example 2 : List the employee names with their manager names.

{ e.ename, m.ename | EMP(e) and EMP(m) and e.mgr=m.eno}

Example 3 : List the name of employees who works on some project controlled by the dept = 5.

{ e.ename | EMP(e) and ((∃x) (∃w)(PROJECT (x) and WORK_ON(w) and x.dno= 5 and w.eno= e.no and x.pno = w.pno))}

Domain Calculus : There is another type of relational calculus, called Domain Relational Calculus, simply Domain Calculus. This domain calculus differ from tuple calculus in only the type of variable used in the formulas i.e., the variables in domain calculus range over the values of domain of attributes rather than over tuple values. A simple domain calculus query is in the form of

$$\{ x_1, x_2, x_3, \ldots\ldots\ldots, x_n \mid COND(x_1, x_2, x_3, \ldots\ldots, x_n, x_{n+1}, x_{n+2}, \ldots\ldots, x_{n+m}\}$$

Where $x_1, x_2, x_3, \ldots\ldots, x_n, x_{n+1}, x_{n+2}, \ldots\ldots, x_{n+m}$ are the domain variables in which the desired attributes are $x_1, x_2, x_3, \ldots\ldots\ldots, x_n$, COND is a condition or formula of domain calculus.

For Example 1 : List birth date and address of employees whose name is John B Smith.

{ uv | (∃q) (∃r) (∃s) (∃t) (∃w) (∃x) (∃y) (∃z) (EMP(qrstuvwxyz) and q=’John’ and r=’B’ and s =’Smith’)}

Example 2 : Retrieve the name, address of all employees who work in the research dept.

{ qsv | (∃z) (∃l) (∃m) (EMP(qrstuvwxyz) and DEPT (lmno) and l= ‘research’ andm = z) }

Here the condition relating two domain variables from two relations such as m = z is referred a join condition and condition that relates a domain variable to constant like l= ‘research’ is called selection condition.

NORMALIZATION

Introduction : **Normalization** is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency.

Actually it is a part of design process. This is used to design tables in which data redundancies are minimized and memory is saved. It is also defined by step by step decomposition of complex records into simple records. This normalization works through a series of stages or forms called Normal Forms. Hence it is a theory built around the concept of normal forms. The most commonly used normal forms are 1NF, 2NF, and 3NF. From the structural point of view higher normal form is better than lower normal form because those forms yield relatively few data redundancies in the database. So 3NF is better than 2NF which is better than 1NF. Almost all business design used 3NF as ideal normal form.

Unnormalized Relation : It is a relation or table in which each row has some repeating information. The relational model does not support such unnormalized relations in the database. For example consider a construction company that manages several building projects. Each project has its own project number, project name, employees assigned to the project and so on. The table looks like as follows:

PNO	PNAME	EMPNO	ENAME	JOB	CH HOUR	HOURS WORKED
1	Evergreen	103	JE Arbough	Analyst	84.5	23
		101	JG News	Designer	105.00	19
		105	A. Johnson	Programer	35.75	12
2	Rolling tide	114	A. Jones	Designer	105.00	24
		104	K. Ramoras	Analyst	84.5	32
3	Star flight	114	A. Jones	Designer	105.00	33
		101	JG News	Designer	105.00	56
		112	M. Smith	Analyst	84.5	41

Functional Dependency (FD) : Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.

If R is a relation with attributes X and Y, a functional dependency between the attributes is represented as X → Y which specifies Y is functionally dependent on X. Here X is a determinant set and Y is a dependent attribute. Each value of X is associated precisely with one Y value.

Functional dependency in a database serves as a constraint between two sets of attributes. Defining functional dependency is an important part of relational database design and contributes to aspect normalization.

First Normal Form(1NF) : In order to maintain a relation in relational database it must be at least in First Normal Form (1NF). A relational table is said to be in 1NF when it does not contain repeating groups and every value should be atomic value. It can be defined in various ways as

Definition 1 : A table is in 1NF if and only if it satisfies the following rules.

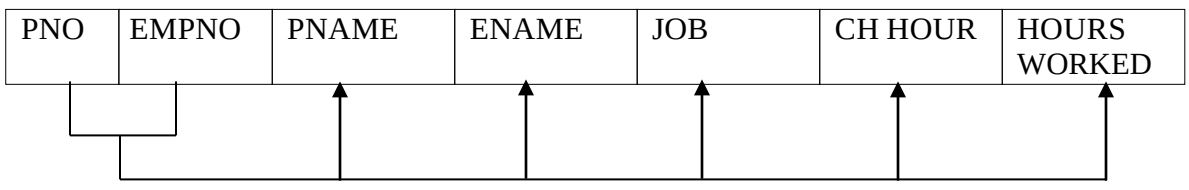
a) All the key attributes are defined.
b) There are no repeating groups in the table. i.e., each row/column intersection can contain one and only one value rather than set of values.
c) All attributes are dependent on the primary key.

Definition 2 : A table is in 1NF if and only if

a) It should contain only atomic values
b) Every non key attribute is functionally dependent on key attribute.

After converting the above unnormalized table into 1NF it will become

Table name : PROJECTS (PNO, EMPNO, PNAME, ENAME, CH HOUR,HOURS WORKED)



PNO	PNAME	EMPNO	ENAME	JOB	CH HOUR	HOURS WORKED
1	Evergreen	103	JE Arbough	Analyst	84.5	23
1	Evergreen	101	JG News	Designer	105.00	19
1	Evergreen	105	A. Johnson	Programer	35.75	12
2	Rolling tide	114	A. Jones	Designer	105.00	24
2	Rolling tide	104	K.Ramoras	Analyst	84.5	32
3	Star flight	114	A. Jones	Designer	105.00	33
3	Star flight	101	JG News	Designer	105.00	56
3	Star flight	112	M. Smith	Analyst	84.5	41

After converting the above table into 1NF, it has a primary key of (PNO, EMPNO) which is a composite key. Even though the table in 1NF it also has some redundancies which yield the following anomalies.

1. **Insertion Anomaly** : We can't insert an employee until employee was assigned to at least one project.
2. **Deletion Anomaly** : If we delete an employee from a project, such deletions may also cause the lost of other vital data. For example worked hours of employee in that project.
3. **Updation Anomaly** : Modifying an attributes for a particular employee requires many alterations, one for each entry.

So even though a table is in 1NF, it can still contain both partial and transitive dependencies.

Partial Dependency : It is the dependency in which an attribute is functionally dependent on only a part of multi attribute primary key. For example pname attribute is dependent on pno only which is a part of primary key in the above table. This partial dependency can't be tolerated because a table that contains such dependency is still subject to data redundancies and various anomalies. Therefore this dependency should be removed. After removing this dependency 1NF becomes 2NF.

Fully Functional Dependency (FFD) : If R is a relation with attributes X and Y, a fully functional dependency between the attributes is represented as X→Y, which specifies Y is fully

functionally dependent on X when Y value is dependent on entire set value of X but not depending on any proper subset of X.

Alternative Examples for First Normal Form :

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab

12	Sam	8589830302	Punjab

Second Normal Form(2NF) : It should be noted that for a table to be in 2NF, it should also be in 1NF and every non key attribute should functionally dependent on primary key. This can be defined in various ways.

Definition 1 : A table is said to be in 2NF if and only if

a) It is in 1NF.

b) It does not have any partial dependencies. i.e., no attribute is dependent on a portion of primary key.

Definition 2 : A table is said to be in 2NF if and only if

a) It is in 1NF.

b) Every non key attribute is fully functionally dependent on primary key.

The conversion of 1NF to 2NF is so simple and it includes the following steps :

1. Write each key component on separate line and then write the original key on the last line.
2. Write the dependent attributes for each key component.

Then each line represents a new table which satisfies the requirements of 2NF. For example by converting the above 1NF table in to 2NF, the following tables are formed.

Table 1 : PROJECTS(PNO, PNAME)

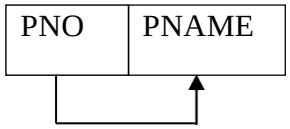


Table 2 : EMP(EMPNO, ENAME, JOB, CH HOUR)

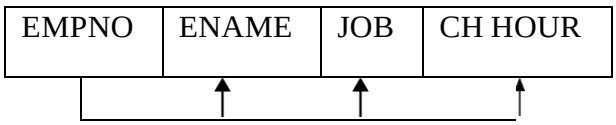
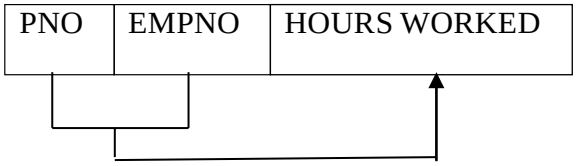


Table 3 : ASSIGN(PNO, EMPNO, HOURS WORKED)



Even though the 2NF removes the partial dependency from the database, the table EMP of database can still contain transitive dependency. i.e., JOB determines the CH HOUR attribute which can still leads to the data anomalies and hence it should be converted into next higher normal form i. e 3NF.

Alternative Examples for Second Normal Form :

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table:

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Transitive Dependency(TD) : It is one type of dependency in which a non key attribute is functionally dependent on another non key attribute. For example in the above table JOB is a non key attribute which will determine another non key attribute i.e., CH HOUR.

Third Normal Form(3NF) : The data anomalies caused by the transitive dependencies in the table are easily eliminated by converting it into 3NF. The 3NF can also be defined in various

ways.

Definition 1 : A table is said to be in 3NF if and only if

1. It is in 2NF
2. It contains no transitive dependencies.

Definition 2 : A table is said to be in 3NF when

1. It is in 2NF
2. Every non key attribute is non transitively dependent on key attribute only.

In the above given example EMP table shows the transitive dependency between JOB AND CH HOUR and hence by eliminating the transitive dependency, the database is having the following tables.

Table 1 : PROJECTS(PNO, PNAME)

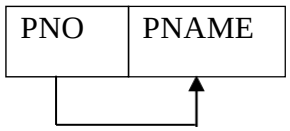


Table 2 : EMP(EMPNO, ENAME, JOB)

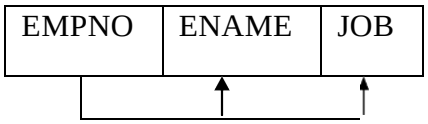


Table 3 : ASSIGN(PNO, EMPNO, HOURS WORKED)

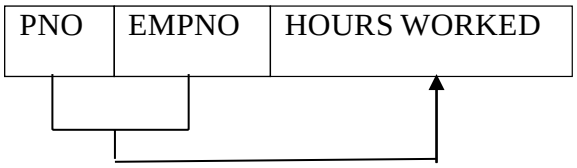
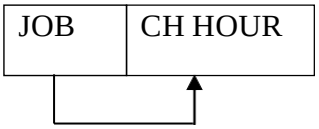


Table 4 : JOB (JOB, CH HOUR)



Alternative Examples for Third Normal Form :

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}. so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

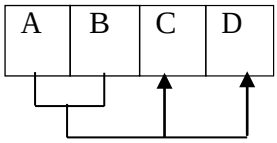
EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

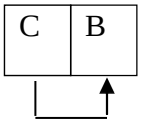
EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

BOYCE CODD Normal Form (BCNF) :

Consider a table with 4 attributes like A, B, C, D. and the functional dependencies as given below :





In the second statement, a non key attribute C determines a part of key attribute. There is no name to this type of dependency and BOYCE discussed about this case with CODD and introduced a new normal form which is BOYCE-CODD Normal Form(BCNF).

Definition : A table is said to be in BCNF if and only if

1. It is in 3NF
2. Every determinant in the table should be a candidate key.

BCNF is more stronger than 3NF. For example consider the following table which is in 3NF but not in BCNF.

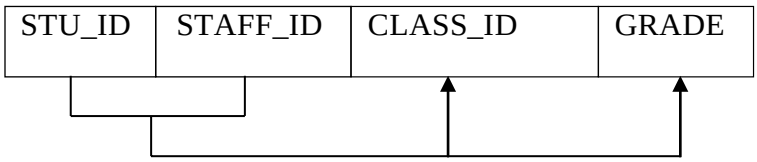
Table : (STU_ID, STAFF_ID, CLASS_ID, GRADE)

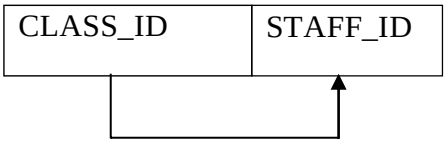
STU_ID	STAFF_ID	CLASS_ID	GRADE
125	25	21334	A
125	20	32455	C
135	20	28458	B
144	25	27563	C
144	20	32455	B

We can observe the following things from the above table.

1. Each course might generate many classes and each class is identified by its unique class code.
2. A student can take many classes. For example student 125 has taken both classes 21334 and 32455 and earning the grades A and C respectively.
3. A staff member can take or teach many classes. But each class is taught by only one staff member. For example teacher 20 teaches the classes 32455 and 28458.

Therefore from the above points we have observed the following dependencies.





The second FD shows that the given table is in 3NF but not in BCNF and hence by converting the given table into BCNF, then the given table can be splitted into the following tables.

Table 1 : (CLASS_ID, STAFF_ID)

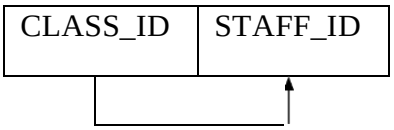
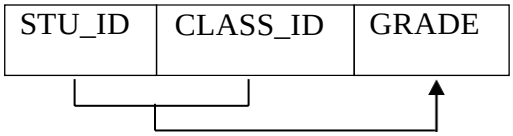


Table 2 : (STU_ID, CLASS_ID, GRADE)



Alternative Examples for BCNF :

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

Candidate key: {EMP-ID, EMP-DEPT}

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232

D283	549

Functional dependencies:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

Forthe first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Given a table with 3 attributes like A, B, C. The multi valued dependency A→→B holds in the table if and only if the set of B values matching a given (A-value, C-value) pair in that table depends only on A-value and independent of C-value or depends on C-value and independent of A-value.

Fourth Normal Form (4NF) : It is defined as follows :

A table is in 4NF if and only if

1. It is in BCNF.
2. It does not have multi valued dependencies.

For example consider the unnormalized relation containing information about COURSES, TEACHERS, TEXT BOOKS. Each record in that relation consists COURSE name, a repeating group of teachers and a repeating group of text books. It is given as follows :

COURSE	TEACHER	TEXT
Physics	Prof. Brown	Basic mechanics
	Prof. Green	Principles of Optics

Maths	Prof. White	Modern Algebra Projective Geometry

After converting it into normalized form, it will be looked like as follows :

COURSE	TEACHER	TEXT
Physics	Prof. Brown	Basic Mechanics
Physics	Prof. Brown	Principles of Optics
Physics	Prof. Green	Basic Mechnics
Physics	Prof. Green	Principles of Optics
Maths	Prof. White	Modern Algebra
Maths	Prof. White	Projective Geometry

The meaning of normalized relation is that each tuple represents that the course can be taught by the teacher and uses the text books as reference and also the teacher uses all the indicated text books.

Even though the above normalized table contains good deal of redundancy but leading to problems over update operations. This is because of multi valued dependencies. For example to add information that Physics uses a new text book, then it is necessary to create two new tuples, one for each of two teachers. Hence it is clear that for course C and text book X, the set of teachers T matching the pair (C, X) depends on C alone but not X. i.e., it makes no difference whatever the value of X. Hence it shows multi valued dependency. To eliminate this type of dependency the above table should be converted into next higher normal form i.e., 4NF.

By converting the above table into 4NF, it will become as

COURSE	TEACHER
Physics	Prof. Brown
Physics	Prof. Green
Maths	Prof. White

COURSE	TEXT
Physics	Basic Mechanics
Physics	Principles of Optics
Maths	Modern Algebra
Maths	Projective Geometry

Join Dependency (JD) : It is denoted by JD (R1,R2,….,Rn) specified on relation schema R, specifies a constraint on states r of R. The constraint states that every r of R should have a non additive join decomposition into R1,R2,….Rn. i.e., for every such r we have

$$(\pi_{R1}(r), \pi_{R2}(r), \pi_{R3}(r), \ldots\ldots \pi_{Rn}(r))=r$$

- Join decomposition is a further generalization of Multivalued dependencies.
- If the join of R1 and R2 over C is equal to relation R, then we can say that a join dependency (JD) exists.
- Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.
- A JD ⋈ {R1, R2,..., Rn} is said to hold over a relation R if R1, R2,…. , Rn is a lossless-join decomposition.
- The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R.
- Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Fifth Normal Form (5NF) : A relation schema R is in 5NF with respect to a set F of functional, multivalued, and join dependencies for every nontrivial join dependency JD(R1,R2,…,Rn) in F and every Ri is a super key of R.

Below are the points :

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

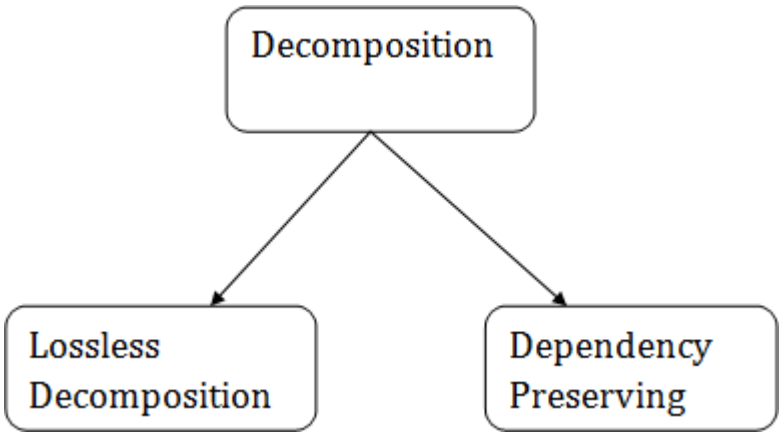
P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Relational Decomposition

- When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
- In a database, it breaks the table into multiple tables.
- If the relation has no proper decomposition, then it may lead to problems like loss of information.
- Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Properties of Relational Decomposition



Lossless Decomposition

- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Example:

EMPLOYEE_DEPARTMENT table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing

46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

DEPARTMENT table

DEPT_ID	EMP_ID	DEPT_NAME
827	22	Sales
438	33	Marketing
869	46	Finance
575	52	Production
678	60	Testing

Now, when these two relations are joined on the common column "EMP_ID", then the resultant relation will look like:

Employee ⋈ Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

Hence, the decomposition is Lossless join decomposition.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.

- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

Multivalued Dependency

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

1. BIKE_MODEL → → MANUF_YEAR
2. BIKE_MODEL → → COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

Inclusion Dependency

- Multivalued dependency and join dependency can be used to guide database design although they both are less common than functional dependencies.
- Inclusion dependencies are quite common. They typically show little influence on designing of the database.
- The inclusion dependency is a statement in which some columns of a relation are contained in other columns.
- The example of inclusion dependency is a foreign key. In one relation, the referring relation is contained in the primary key column(s) of the referenced relation.
- Suppose we have two relations R and S which was obtained by translating two entity sets such that every R entity is also an S entity.
- Inclusion dependency would be happen if projecting R on its key attributes yields a relation that is contained in the relation obtained by projecting S on its key attributes.
- In inclusion dependency, we should not split groups of attributes that participate in an inclusion dependency.
- In practice, most inclusion dependencies are key-based that is involved only keys.

OTHER DEPENDENCIES :

- There are two types of templates:
- tuple-generating templates and constraint-generating templates.

- A template consists of a number of hypothesis tuples that are meant to show an example of the tuples that may appear in one or more relations. The other part of the template is the template conclusion.
- For tuple-generating templates, the conclusion is a set of tuples that must also exist in the relations if the hypothesis tuples are there. For constraint-generating templates, the template conclusion is a condition that must hold on the hypothesis tuples.

OTHER NORMAL FORMS :-

Domain Key Normal Forms(DKNF) :

- The process of normalization and the process of discovering undesirable dependencies was carried through 5NF as a meaningful design activity, but it has been possible to define stricter normal forms that take into account additional types of dependencies and constraints.
- The idea behind domain-key normal form (DKNF) is to specify (theoretically, at least) the "ultimate normal form" that takes into account all possible types of dependencies and constraints.
- A relation is said to be in DKNF if all constraints and dependencies that should hold on the relation can be enforced simply by enforcing the domain constraints and key constraints on the relation. For a relation in DKNF, it becomes very straightforward to enforce all database constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint is enforced.

TRANSACTION PROCESSING CONCEPTS

Introduction : A transaction is an executing program that forms a logical unit of database processing. A transaction includes one or more database operations. The basic database operations that a transaction can perform are

1. Read (x) : This operation reads a database item x into a program variable.
2. Write (x) : This operation writes a value into the item x.

Therefore a transaction includes read, write operations to access and update the database.

Example :

T1
Read (x);
X:=x-n;
Write(x);
Read(y);
Y:=y+m;
Write(y);

Need of Concurrency Control : Concurrency control is most important when transactions can be made on same database i.e., several problem can occur when concurrent transactions execute in uncontrolled manner. Some of these problems are given below

- **Lost Update problem** : This problem occurs when two interleaved transactions that access the same item and may make the value of some database item incorrect. Suppose consider two transactions T1 and T2 as

T1	T2

Read(x) X=x - n; Write(x)	 Read(x); X=x + m; Write(x);

The final value of x is incorrect because T2 reads before T1 changes in database. Hence updated value resulting from T1 is lost. This is also called Write Write(WW) conflict, because one transaction overwrites item which is modified by another transaction.

- **Temporary Update(Dirty Read) Problem** : This is also called as Write Read(WR) conflict. This conflict occurs when an updated item is read before committing i.e., the problem occurs when one transaction updates database item and it fails for some reason. The updated item is accessed by another transaction before it is changed back to original value. This problem is called Dirty Read problem.

T1	T2
Read(x) X=x - n; Write(x);	 Read(x); X=x + m; Write(x);

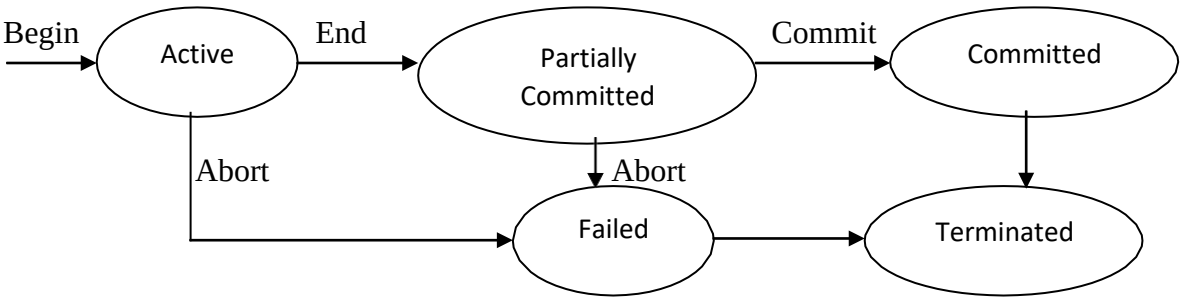
When T1 fails, the value of x must back to its original value, meanwhile T2 has read temporary incorrect value of x i.e., called dirty read.

- **Unrepeatable Read (RW) Problem** : This is another problem that occurs when a transaction T reads an item twice and it is modified by another transaction between two reads. So T can receive two different values for two reads of same item.

Types of Failures : There are several reasons for a transaction to fail in the middle of execution. The important reasons are

- **A Computer Failure (System Crash)** : This problem occurs in the computer because of hardware, software or network error.
- **A Transaction Error** : Some operations like integer overflow or division by 0 may cause transaction to fail.
- **Logical Errors** : During transaction execution certain conditions may lead it to cancelation. For example data was not found.
- **Concurrency Control Enforcement** : Sometimes concurrency control method may decide to abort transaction because it violates serilizability.
- **Disk Failure** : This failure may occur because of read/write head crash.
- **Physical Problems** : These problems include fire, theft and overwriting disks.

Transaction States : The sates of a transaction are given in the following diagram.



- **Active State** : It is initial state and transaction is said to be in active state while it is in execution.
- **Partially Committed** : When last statement of transaction is executed then it enters into partially committed state. From this state it can be finished its execution or it may get aborted due to some failure.
- **Failed State** : When the execution of transaction can't be processed then it enters into this state and system can perform rollback operation.
- **Abort State** : The rollback transaction enters into abort state where the transaction may be restarted or terminated.

Transaction Properties : Transaction should posses several properties called ACID properties. They are

- **Atomicity (A)** : A transaction is consistency preserving if its execution takes database from one consistent state to another consistent state.
- **Consistency Preservation (C)** : A transaction is consistency preserving if its execution takes database from one consistent state to another consistent state.
- **Isolation (I)** : Transactions are isolated one another i.e., the execution of one transaction should not be interfered by any other transactions executing concurrently.
- **Durability or Permanency (D)** : 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.

Schedule of Transactions : A schedule S of n transactions T1, T2,T3,…,Tn is an ordering of operations of transactions in such a manner that operations of each Ti must appear in the same order in which they occur in Ti.

- **Complete Schedule** : A schedule S of n transactions T1, T2,T3,…,Tn is said to be complete schedule if it satisfies the following conditions.
    1. The operations in S are exactly those operations in T1, T2,T3,…,Tn including commit or abort operation as last operation for each transaction in schedule.
    2. For any pair of operations from some transaction Ti, the order in S is same as their order in Ti.
    3. For any two conflict operations, one of two must occur before other in the schedule.
    4.

Characterizing Schedule Based on Recoverability :

If any transaction that performs a dirty read operation from an uncommitted transaction and also its committed operation becomes delayed till the uncommitted transaction is either committed or rollback such type of schedules is called as Recoverable Schedules.

Types of recoverable schedules

There are three types of recoverable schedules which are explained below with relevant examples –

- Cascading schedules
- Cascadeless Schedules
- Strict Schedules.

Cascading Schedules:

A cascading schedule is classified as a recoverable schedule. A recoverable schedule is basically a schedule in which the commit operation of a particular transaction that performs read operation is delayed until the uncommitted transaction either commits or roll backs.

A cascading rollback is a type of rollback in which if one transaction fails, then it will cause rollback of other dependent transactions. The main disadvantage of cascading rollback is that it can cause CPU time wastage.

Cascading less Schedules:

When a transaction is not allowed to read data until the last transaction which has written it is committed or aborted, these types of schedules are called cascadeless schedules.

Given below is an example of a cascadeless schedule −

T1	**T2**
R(X)	
W(X)	
	W(X)
commit	
	R(X)
	Commit

Here, the updated value of X is read by transaction T2 only after the commit of transaction T1. Hence, the schedule is cascadeless schedule.

Strict Schedule :

It is the schedule in which transactions can neither read nor write an item x until the last transaction that wrote x has committed or aborted. These schedules can simplify the recovery process.

Given below is an example of a strict schedule −

T1	**T2**
R(X)	
	R(X)

T1	T2
W(X)	
Commit	
	W(X)
	R(X)
	Commit

Here, transaction T2 reads and writes the updated or written value of transaction T1 only after the transaction T1 commits. Hence, the schedule is strict schedule.

Schedule Based on Serializability : A schedule is said to be serializable if the interleaved transactions can produce the same result which if equivalent to the result produced by executing individual transactions separately. So a schedule is consistent if and only if it is seriablizable. Hence serializability is a standard to ensure the consistency of a schedule.

There are two types of serializabilities, which include

- **Serial and Non-Serial Schedule** : A schedule is said to be serial when the operations of each transaction are executed serially or consecutively i.e., execute all operations of T1 then all the operations of T2 and so on in a sequence. The schedule which is not serial called Non Serial schedule. In Serial schedule, the transactions are executed in serial order and it has the following properties.

1. Only one transaction is active at a time.
2. The commit or abort of active transaction initiates the execution of next transaction.
3. No interleaving occurs in a serial schedule.

Example for Serial Schedule :

T1	T2
Read(x) X=x-n; Write(x);	 Read(x); X=x+m; Write(x);

T1	T2
   Read(x) X=x-n; Write(x);	Read(x); X=x+m; Write(x);

Example for Non-Serial Schedule :

T1	T2
Read(x) X=x-n; Write(x)	 Read(x); X=x+m; Write(x);

- **Conflict Serializability** : We can define a schedule S to be Conflict Serializable if it is conflict equivalent to some serial schedule S1. The two schedules S and S1 are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules. Two operations in a schedule are said to be conflict if they satisfy the conditions.
    - They belong to different transactions.
    - They can access the same item x.
    - Atleast one of the operations in Write(x).

- **Conflict Equivalence** : Consider read and write operations of T1 and T2 as

1. r1(x), r2(x) 2. r1(x), w2(x) 3. w1(x), r2(x) 4. w1(x), w2(x)

In the above cases 2 and 3 are conflict each other and 1 and 4 are non conflict operations.

A conflict equivalent schedule can be obtained by swapping the operations which can satisfy conditions as

1. They belong to different transactions
2. They are not conflict operations.

For Example : Consider the following schedules

Schedule S

T1	T2
Read(x) X=x-n;	
	Read(x); Write(x);
read(y); write(y)	
	Read(y) Write(y);

Schedule S1 :

T1	T2
Read(x) X=x-n; read(y); write(y)	
	Read(x); Write(x); Read(y) Write(y);

Here in schedule S, write(x) of T1 conflicts with read(x) of T2. But write(x) of T2 does't conflict with read(y) of T1 because they perform on two data items.

So S1 schedule can be obtained by swapping

1. Read(y) of T1 with read(x) of T2
2. Write(y) of T1 with write(x) of T2

Therefore we can say that the schedules S and S1 are conflict equivalent. In general two schedules are said to be conflict equivalent when it is possible to generate a schedule (S1) from the given schedule(S) by performing series of swap operations on non conflicting pair of operations.

- **View Serializability** : Two schedules S1 and S2 consisting of same transactions are said to be view equivalent, if they satisfy the following conditions

  - If a transaction T1 in schedule S1 performs read operation on the initial value of item x, then the same transaction in schedule S2 must also perform read operation on the initial value of item x.
  - If a transaction T1 in S1 reads a value x, which was written by another transaction T2 then in S2 also T1 reads value x which was written by another transaction T2.
  - If a transaction T1 in S1 performs final write operation on x then same T1 in

S2 must also performs final write operation on x.

For example consider the following schedules.

Schedule S1

T1	T2

Read(x)
X=x-10;
write(x);
read(y);
y=y+10;
write(y);	

	Read(x); X=x*20; Write(x); Read(y); Y=y/10; Write(y);

Schedule S2

T1	T2
read(x) X=x-10; Write(x);	
	read(x); X=x*20; Write(x);

read(y);
y=y+10;
write(y);	

	read(y); Y=y/10; Write(y);

Here the schedule S1 is view equivalent to S2. This view equivalence leads to another notion called view serializability. A schedule S is said to be view serializable, if it is view equivalent with the serial schedule.

Note : Every conflict serializable schedule is view serializable but reverse is not true always.

CONCURRENCY CONTROL TECHNIQUES

Introduction : Concurrency control methods allows many transactions to access same data item at the same time without interfering each other. The concurrency control is the management of concurrent transaction execution. DBMS implement these techniques to ensure seriablizability and isolation of transactions. These techniques can use set of protocols to ensure concurrent execution of transactions. These protocols include

- Locking Protocols
- Multiversion Concurrency Control Protocols
- Timestamp Protocols
- Optimization Protocol

Locking Techniques for Concurrency Control : This is one of the main technique used to control concurrent execution of transactions. These techniques are based on the concept of locking data item. A lock is a variable associated with a data item that describes status of data item with respect to possible operations that can be applied to it.

Type of Locks : There are different type of locks, they are

1. Binary lock 2.Shared (Read) lock 3. Exclusive (Write) lock

1. **Binary Lock** : It can have two values or states, locked and unlocked. For simplicity these can be represented by '0' and '1'. A distinct lock is associated with each data item x. If lock on x is 1 then x can't be accessed by operation that request item. If lock on x is 0 then it can be accessed. This scheme is too restrictive for database items because atmost one transaction can hold a lock on given item. So another locking scheme is introduced i.e., shared/exclusive or read/write locks.
2. **Shared (Read) Lock** : A transaction that acquires this lock on item x then it can execute read operation only, but not write operation. So these locks support only read integrity.
3. **Exclusive (Write) Lock** : A transaction that acquires this lock on item x then it can perform both read and write operations. So these locks support read and write integrity. Moreover if a transaction gets this lock on data item then no other transaction can get any lock on that item until the previous transaction releases its lock on item.

T1	T2

Readlock(x);
read(x);
unlock(x);
writelock(y);
read(y);
x:=x+y;
write(y);
unlock(y);
	readlock(y);
	read(y);
	unlock(y);
	writelock(x);
	read(x);
	x:=x+y;
	write(x);
	unlock(x);

Two Phase Locking Protocol (2PL) : This is a concurrency control protocol that used for serializability between schedules. A transaction obeys 2PL protocol when read as well as write operations are executed before the execution of first unlock operation. In 2PL transactions can be divided into two phases. They are

1. Growing Phase (Lock acquisition) : In this phase, transaction may acquires new locks but can't release them.

2. Shrinking Phase (Lock releasing) : In this phase, transaction release the existing locks, but can't acquire any new lock.

There are number of variations in 2PL, they are

1. Basic 2PL
2. Static or Conservative 2PL
3. Strict 2PL
4. Rigorous 2PL

1. Basic 2PL : The above said 2PL is called basic 2PL.

2. Static 2PL : In this scheme all the data items are locked before any operations on them and they are released only after last operation performed on any data item.

For example

T1	T2
Writelock(p); Read(p); P:=p+10; Write(p); Unlock(p);	Writelock(p); Read(p); Readloack(x); Read(x); P:=p+x; Write(p); Unlock(x); Unlock(p);

3. Strict 2PL : It is most popular variation of 2PL. It is compatible with schedules following 2PL that posses strictness property i.e., A transaction follows strict 2PL if

1. It is compatible with 2PL and
2. It does't release write locks until transaction either commits or abort.

Therefore this protocol prevents other transaction from reading and writing data items until uncommitted transaction is committed.

4. Rigorous 2PL : It is another restrictive variation of 2PL. A transaction follows rigorous 2PL if

1. It is compatible with 2PL and
2. It does't release write and read locks until transaction either commits or abort.

Note : The main disadvantage of locking scheme is that it can lead to deadlocks for sometimes.

2. Concurrency Control Based on Time Stamp Ordering : This is another technique that ensures serializability. This technique is based on timestamp ordering but don't use locks. So deadlocks can't occur in this scheme.

Time Stamp Ordering (TO) : The idea for this scheme is to order the transactions based on their timestamps. This is called Time Stamp Ordering(TO). Time Stamp is an identifier that specifies start time of transaction and it is generated by DMBS. It uniquely identifies transaction in schedule. The timestamp of a transaction T is denoted by TS(T). In this two timestamp values are associated with each item.

1. **Read-TS(x)** : This is read timestamp. This is the largest timestamp among all timestamp of transactions that successfully read item x. i.e., read-TS(x) = TS(T) where T is youngest transaction that has read x successfully.
2. **Write-TS(x)** : This is write timestamp. This is the largest timestamp among all timestamps of transactions that successfully write item x. i.e., write-TS(x) = TS(T) where T is youngest transaction that has write x successfully.

Basic Time Stamp Ordering Algorithm : Whenever some transaction T tries to issue read (x) or write (x) operation then basic TO compares the timestamp of T with read TS(x) or write TS(x), to ensure that timestamp order is not violate. If this order is violated then T is aborted and resubmitted as new transaction with new timestamp. For this following two cases can be

considered.

Case 1 : Transaction T issues a write (x) operation.

a) If read TS(x) > TS(T) or if write TS(x) > TS(T) then abort and rollback T and reject operation. This is because there is some younger transaction with timestamp greater than TS(T), which has already read or write on x before T had chance to write (x).
b) If condition a) does't occur then execute write(x) on T and set write TS(x) to TS(T).

Case 2 : Transaction T issues a read (x) operation

a) If write TS(x) >TS(T) then abort and rollback T and reject operation.
b) If write TS(x)<TS(T) then execute read (x) operation of T and set read TS(x) as TS(T).

Strict TimeStamp Ordering : A variation of basic TO called Strict TimeStamp Ordering, which ensures that the schedules are both strict and seriablizable. In this variation, a transaction T that issues a read(x) or write(x) such that TS(T)>write TS(x), has its read or write operation delayed until T1 that wrote(x) has committed or aborted. To implement this algorithm, it is necessary to simulate locking of x that has been written by T1 until T1 is either committed or aborted.

Thoma's Write Rule : A modification of basic TO algorithm is known as Thoma's Write Rule. It will enhance TO algorithm by rejecting few write operations by modifying checks for write operation as follows :

1. If read TS(x) > TS(T) then abort, rollback T and reject operation
2. If write TS(x) > TS(T) then don't execute write operation but continue the operation, because some transaction with timestamp greater than TS(T) that has already written x.
3. If neither condition 1 nor condition 2 occurs then execute write (x) operation of T and set write TS(x) to TS(T).

3. Multiversion Concurrency Control Techniques : The protocols used in this technique must keep old values of data item when item is updated. These are called as multiversion because several versions (values) of item are maintained. The idea is that some read operations that would be rejected in other techniques can still be accepted by reading older version of item to maintain serializability. When transaction writes an item, it write new version and old version of item is retained. So the main drawback of this technique is that more storage is needed to maintain multiple versions of database items.

Several schemes are proposed for multiversion concurrency control, which include

1. Multiversion based on TimeStamp Ordering
2. Multiversion based on 2PL.

1. Based on TimeStamp Ordering : In this method, several versions x1,x2,x3,….xk of each item x are maintained. For each version, two timestamp values are kept for xi.

a) read TS(xi) : It is larger of all timestamp of transactions that have successfully read version xi.

b) write TS(xi) : It is the timestamp of transaction that wrote the value of version xi.

When transaction T is allowed to execute write (x), a new version xk+1 is created with write TS(xk+1) and read TS(xk+1) set to TS(T). To ensure serializability the following rules are used.

1. If T issues write (x) and xi has the highest write TS(xi) of all version of x that is less than or equal to TS(T) and read TS(xi) > TS(T) then abort and rollback T, otherwise create new version xj of x with read TS(xj)=write TS(xj)=TS(T).
2. If T issues read (x), find the version xi, that has highest write TS(xi) of all versions xi that has highest write TS(xi) of all version of x that <= TS(T) then return value of xi to T and set the value of read TS(xi) to larger of TS(T) and current read TS(xi).

2. Multiversion based on 2PL : In this scheme, there are 3 locking modes for item. i.e read, write and certify, are used instead of two locking modes. i.e read and write. The idea behind this technique is that it allows T2 to read an item x while single transaction T1 holds write lock on x. This is accomplished by allowing two versions for each item x, one version must always have been written by some committed transaction. The second version is created when T acquires write lock on item. So other transaction can continue to read committed version of x while T holds write lock. However once T is ready to commit, it must obtained certify lock on all items that it holds write lock. So once T acquired certify lock then committed version of x is set as new version and certify lock are then released. The lock compatible tables for this scheme is given as follows :

	Read	Write
Read	YES	NO
Write	NO	NO

	Read	Write	Cerify
Read	YES	YES	NO
Write	YES	NO	NO
Certify	NO	NO	NO

The entry YES means that if a transaction holds type of lock specified in the column header on item x then any other transaction T1 requests the type of lock specified in the row header on the same item x. Similarly NO indicates that locks are not compatible. So T1 must wait until T releases the lock.

4. Validation Technique for Concurrency Control (Optimistic)

This optimistic concurrency control technique is also known as validation or certification techniques. These techniques don't require checks while transaction is executing. That's why this technique can be used for several methods. In this technique, updates in transaction are not applied directly to database item until transaction reaches its end. So during transaction execution, all updates are made to local copies of items that are kept for transaction. At the end of transaction, a validation phase checks when any transaction update violates the seriablzability. If serializability is not violated then transaction is committed and database is updated from local copies, otherwise transaction is aborted and restarted later. So certain information needed by validation phase must be kept by the system.

The basic idea of this technique is to do all the checks at once. So transaction execution proceeds with minimum overhead until it reach validation phase. This is called optimistic because they assume little interference will occur and hence there is no need to check during transaction execution.

There are 3 phases for this concurrency control, they are

1. **Read Phase** : In this phase, transaction can read values of committed data items from the database. But updates are applied to only local copies of items.
2. **Validation Phase** : This phase follows read phase where checking is done to ensure that serializability will not be violated. If conflict occur between transactions then it is aborted and restarted.
3. **Write Phase** : The successful completion of validation phase lead to this write phase in which transaction updates are applied to the database.

DISTRIBUTED DATABASES

Definition : A Distributed Database (DDB) is a collection of multiple logically interrelated databases distributed over a computer network. A distributed DBMS (DDBMS) is software system that can manage Distributed Database.

Advantages of DDB : DDB has several advantages, some of them are

1. Management Distributed data with different levels of Transparency : DBMS should be distributed transparent i.e it hides the details of where each file is physically stored within the system. Different types of transparencies are possible which include

*** Network (distribution) Transparency** : This hides the operational details of network from the user. It is divided into two types like location transparency and naming transparency. The location transparency refers to the command used to perform task which is independent of location of data and location of the system. Naming transparency implies that the named object can be accessed unambiguously.

*** Replication Transparency** : The copies of data may be stored at multiple sites for better availability, performance and reliability.

*** Fragmentation Transparency** : Two types of fragmentation are possible. Horizontal fragmentation, distributes a relation into sets of tuples (rows). Vertical fragmentation, distributes a relation into columns of relation.

*** Other Transparencies** : This include design and execution transparencies.

2. Increased Reliability & Availability : Reliability is defined as probability that a system is running at a certain time point. Availability is the probability that the system is continuously available during the time interval. i.e what data and software are distributed over sites and one site may fail while other sites continue to operate. So it improves both reliability and availability.

3. Improved Performance : When large database is distributed over sites, smaller databases exist at each site. As a result local queries and transactions have better performance.

4. Easier Expansion : In a distributed environment, expansion of system in terms of more data i.e., increasing database size or adding more processors is much easier.

Below are the Techniques for Distributed Database design

DataFragmentation, Replication and Allocation

Data Fragmentation

In distributed databases, a database is breakup into logical units called Fragments which are stored at various sites. The same fragment is stored at more than one site then it is called Data Fragmentation. The simplest logical units are the relations themselves. i.e., each whole relation is to be stored at a particular site. However a relation can be divided into smaller logical units for distribution. There are three types of such fragmentations. They are

1. Horizontal Fragmentation
2. Vertical Fragmentation
3. Mixed Fragmentation

1. Horizontal Fragmentation : A Horizontal Fragmentation of a relation is a subset of tuples in that relation. Generally the tuples that belong to horizontal fragments are specified by a condition on one or more attributes of the relation. For example employees working for particular department. So it divides a relation horizontally by grouping the rows to create subset of tuples, where each subset has logical meaning. These fragments can be assigned to different sites in distributed system.

2. Vertical Fragmentation : A vertical fragment of relation is a subset of attributes of the relation. So vertical fragmentation divides the relation vertically by columns. These fragments can be assigned to different sites because each site may not need all attributes of a relation. For example Name, DOB, address of employee may need at one site.

3. Mixed Fragmentation (Hybrid) : It is the mixed of above two. For example Name, DOB and address of employees for a particular department.

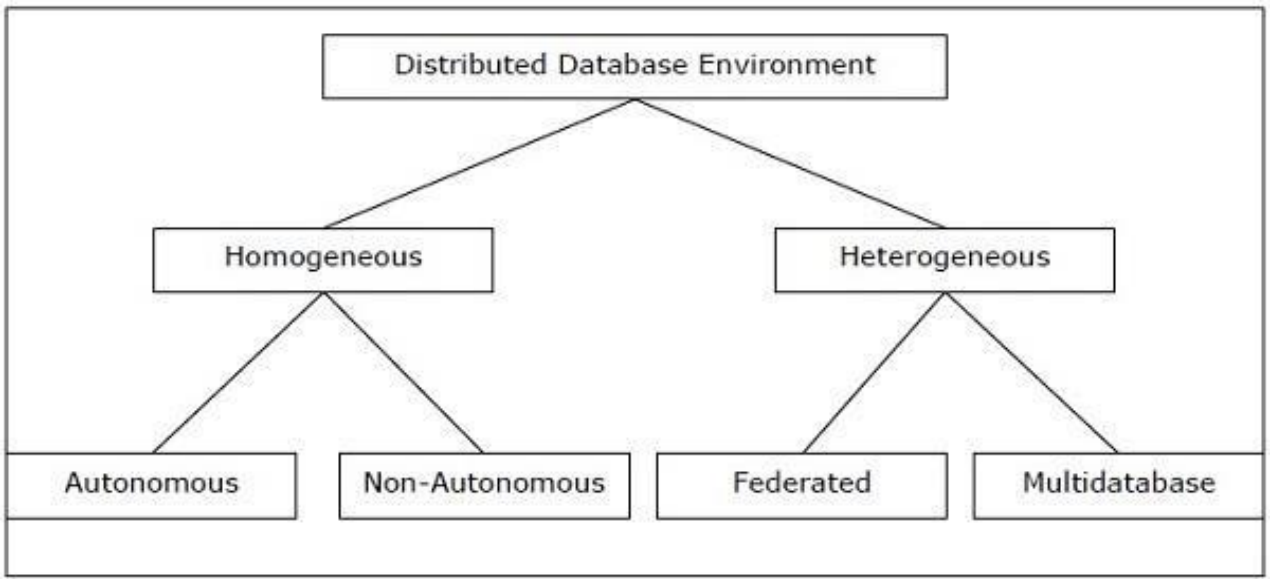
Data Replication : Replication means duplication which is useful in improving availability of data. Suppose the replication of whole database is maintained at every site in distributed system then it is referred as fully replicated distributed database. On the other hand of full replication, there is no replication where each fragment is stored at exactly one site only. In this case all

fragments must be disjoint. This is called non redundant allocation. Between these two types, we have another type that is partial replication of data i.e., some fragments may be replicated and others may not.

Data Allocation : Each fragment must be assigned to particular site in distributed system, which is called data distribution (or) data allocation. The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site.

For example,if high availability is required and transactions can be submitted at any site and if most transactions are retrieval only, a fully replicated database is a good choice.

Type of Distributed Database Systems :



Homogeneous Distributed Databases

In a homogeneous distributed database, all the sites use identical DBMS and operating systems. Its properties are −

- The sites use very similar software.
- The sites use identical DBMS or DBMS from the same vendor.
- Each site is aware of all other sites and cooperates with other sites to process user requests.
- The database is accessed through a single interface as if it is a single database.

Types of Homogeneous Distributed Database

There are two types of homogeneous distributed database −

- **Autonomous** − Each database is independent that functions on its own. They are integrated by a controlling application and use message passing to share data updates.
- **Non-autonomous** − Data is distributed across the homogeneous nodes and a central or master DBMS co-ordinates data updates across the sites.

Heterogeneous Distributed Databases

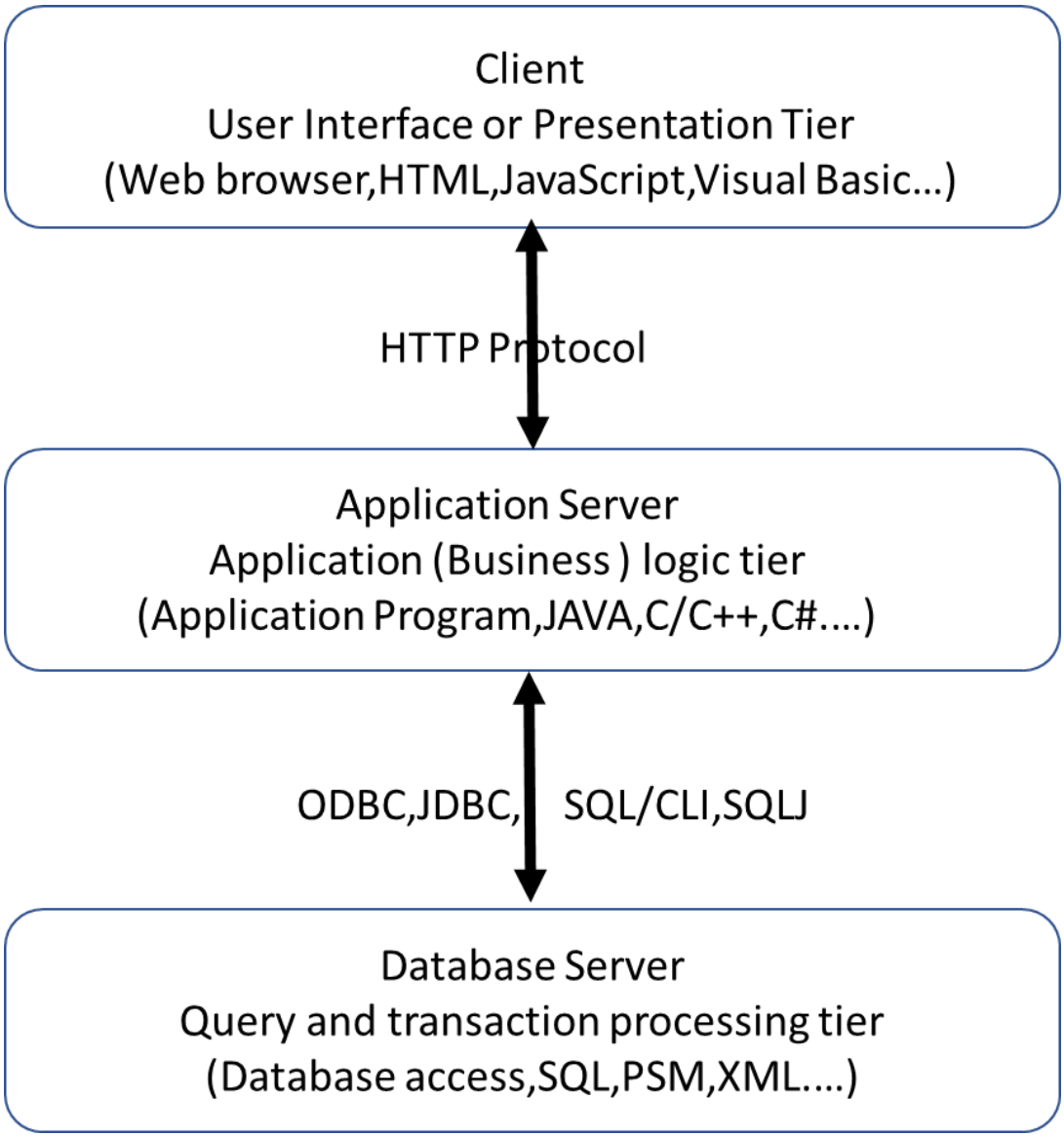
In a heterogeneous distributed database, different sites have different operating systems, DBMS products and data models. Its properties are −

- Different sites use dissimilar schemas and software.
- The system may be composed of a variety of DBMSs like relational, network, hierarchical or object oriented.
- Query processing is complex due to dissimilar schemas.
- Transaction processing is complex due to dissimilar software.
- A site may not be aware of other sites and so there is limited co-operation in processing user requests.

Types of Heterogeneous Distributed Databases:

- Federated − The heterogeneous database systems are independent in nature and integrated together so that they function as a single database system.
- Un-federated − The database systems employ a central coordinating module through which the databases are accessed.

3-Tier Client Server Architecture



1.Presentation Layer :

This provides the user interface and interacts with the user. The programs at this layer present Web interfaces or forms to the client in order to interface with the application.

Web browsers are often utilized and the languages used include HTML,JAVA,JavaScript…..and so on .

This layer handles user input, output and navigation by accepting user commands and displaying the needed information,usually in the form of static or dynamic Web Pages.

2.Application Layer (Business Logic):

This layer programs the application logic. For example, queries can be formulated based on user input from the client or query results can be formatted and sent to the client for presentation.

The application layer can interact with one or more databases or data sources as needed by connecting to the database using ODBC,JDBC,SQL/CLI or other database techniques.

3.Database Server :

This layer handles query and update requests from the application layer, processs the request, and send the results.

SQL is used to access the database if it is relational or object relatational and stored procedures may also be invoked. Query results may be formatted into XML when transmitted between application server and database server.