# QUESTION BANK
# Database Management System(18CS53)
**Faculty: Mrs.ANJALI GUPTA**

## MODULE-1

**Q1- Discuss the main characteristics of database approach. How it differ from traditional database.**

Ans-

**FILE ORIENTEDAPPROACH:**A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storagedevices.

Data Integrity and Security:

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

**Concurrent Access and Crash Recovery:**

A database system allows several users to access the database concurrently. Answering different questions from different users with the same (base) data is a central aspect of an information system. Such concurrent use of data increases the economy of a system.

An example for concurrent use is the travel database of a bigger travel agency. The employees of different branches can access the database concurrently and book journeys for their clients. Each travel agent sees on his interface if there are still seats available for a specific journey or if it is already fully booked.

A DBMS also protects data from failures such as power failures and crashes etc. by the recovery schemes such as backup mechanisms and log filesetc.

**Data Administration:**

When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals, who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and fine-tuning the storage of the data to make retrieval efficient.

**Reduced Application Development Time:**

The earliest business computer systems were used to process business records and produce information. They were generally faster and more accurate than equivalent manual systems. These systems stored groups of records in separate files, and so they were called **file processing systems.**

1. File system is a collection of data. Any management with the file system, user has to write theprocedures.

2. File system gives the details of the data representation and Storage ofdata.
3. In File system storing and retrieving of data cannot be doneefficiently.
4. Concurrent access to the data in the file system has many problems like a Reading the file while other deleting some information, updating someinformation
5. File system doesn't provide crash recoverymechanism.

**Eg**. While we are entering some data into the file if System crashes then content of the file is lost.

6. Protecting a file under file system is verydifficult.

The typical file-oriented system is supported by a conventional operating system. Permanent records are stored in various files and a number of different application programs are written to extract records from and add records to the appropriate files.

## DISADVANTAGES OF FILE-ORIENTED SYSTEM:

The following are the disadvantages of File-Oriented System:

### Data Redundancy and Inconsistency:

Since files and application programs are created by different programmers over a long period of time, the files are likely to be having different formats and the programs may be written in several programming languages. Moreover, the same piece of information may be duplicated in several places. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency.

### Difficulty in Accessing Data:

The conventional file processing environments do not allow needed data to be retrieved in a convenient and efficient manner. Better data retrieval system must be developed for general use.

### Data Isolation:

Since data is scattered in various files, and files may be in different formats, it is difficult to write new application programs to retrieve the appropriate data.

### Concurrent Access Anomalies:

In order to improve the overall performance of the system and obtain a faster response time, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data.

### Security Problems:

Not every user of the database system should be able to access all the data. For example, in banking system, payroll personnel need only that part of the database that has information about various bank employees. They do not need access to information about customer accounts. It is difficult to enforce such security constraints.

## **<u>Integrity Problems:</u>**

The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount. These constraints are enforced in the system by adding appropriate code in the various application programs. When new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items for different files.

## **<u>Atomicity Problem</u>**:

A computer system like any other mechanical or electrical device is subject to failure. In many applications, it is crucial to ensure that once a failure has occurred and has been detected, the data are restored to the consistent state existed prior to the failure

## **<u>Example:</u>**

Consider part of a savings-bank enterprise that keeps information about all customers and savings accounts. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system hasa number of application programs that manipulate the files,including:

A program to debit or credit an account
A program to add a new account
A program to find the balance of an account
A program to generate monthly statements

Programmers wrote these application programs to meet the needs of the bank. New application programs are added to the system as the need arises. For example, suppose that the savings bank decides to offer checkingaccounts.

As a result, the bank creates new permanent files that contain information about all the checking accounts maintained in the bank, and it may have to write new application programs to deal with situations that do not arise in savings accounts, such as overdrafts. Thus, as time goes by, the system acquires more files and more application programs. The system stores permanent records in various files, and it needs different

Application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMS) came along, organizations usually stored information in such systems. **Organizational information in a file-processing system has a number of majordisadvantages:**

## **<u>1. Data Redundancy andInconsistency:</u>**

The address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. In, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.

**2. Difficulty in AccessingData:**

Suppose that one of the bank officers needs to find out the names of all customers who live within a particular postal-code area. The officer asks the data-processing department to generate such a list. Because there is no application program to generate that. The bank officer has now two choices: either obtain the list of all customers and extract the needed information manually or ask a system programmer to write the necessary application program. Both alternatives are obviouslyunsatisfactory.

**3. DataIsolation:**

Because data are scattered in various files and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

**4. Integrity Problems:**

The balance of a bank account may never fall below a prescribed amount (say, $25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

**5. Atomicity Problems:**

A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account *A* to account *B*. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account *A* but was not credited to account *B*, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be *atomic*—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processingsystem.

**6. Concurrent-AccessAnomalies:**

For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data. Consider bank account *A*, containing $500. If two customers withdraw funds (say $50 and $100 respectively) from account *A* at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $500, and write back $450 and $400, respectively. Depending on which one writes the value last, the account may contain $450 or $400, rather than the correct value of $350. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

**7. Security Problems:**

Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult. These difficulties, among others, prompted the development of databasesystems.

## ADVANTAGES OF A DBMS OVER FILE SYSTEM:

Using a DBMS to manage data has many advantages:

**Data Independence:**

Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

**Efficient Data Access:**

DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick development of applications. Such applications are also likely to be more robust than applications developed from scratch because many important tasks are handled by the DBMS instead of being implemented by theapplication.

## Q2-Describe the three schema architecture. What are the problems associated with three schema architecture.

**Three Schema Architecture:**

The goal of this architecture is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following threelevels:

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

2. The **conceptual level** has a **conceptual schema,** which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. A high-level data model or an implementation data model can be used at thislevel.

3. The **external** or **view level** includes a number of **external schemas** or **user views.** Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model can be used at thislevel.

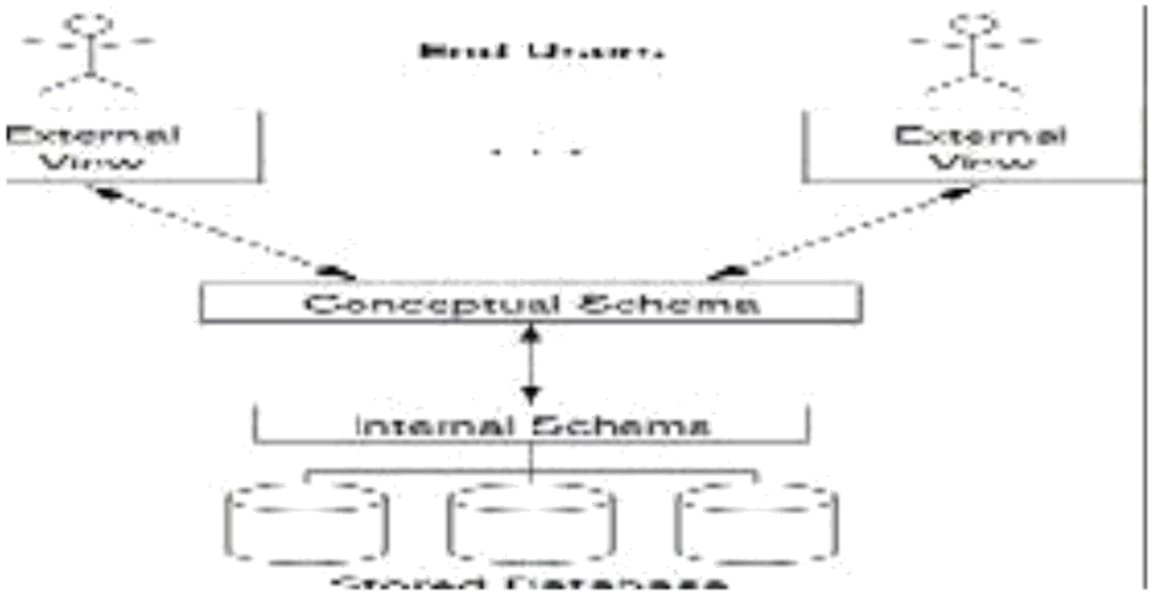

End Users
External View
. . .
External View
Conceptual Schema
Internal Schema
Stored Database

Fig: Three-Schema Architecture

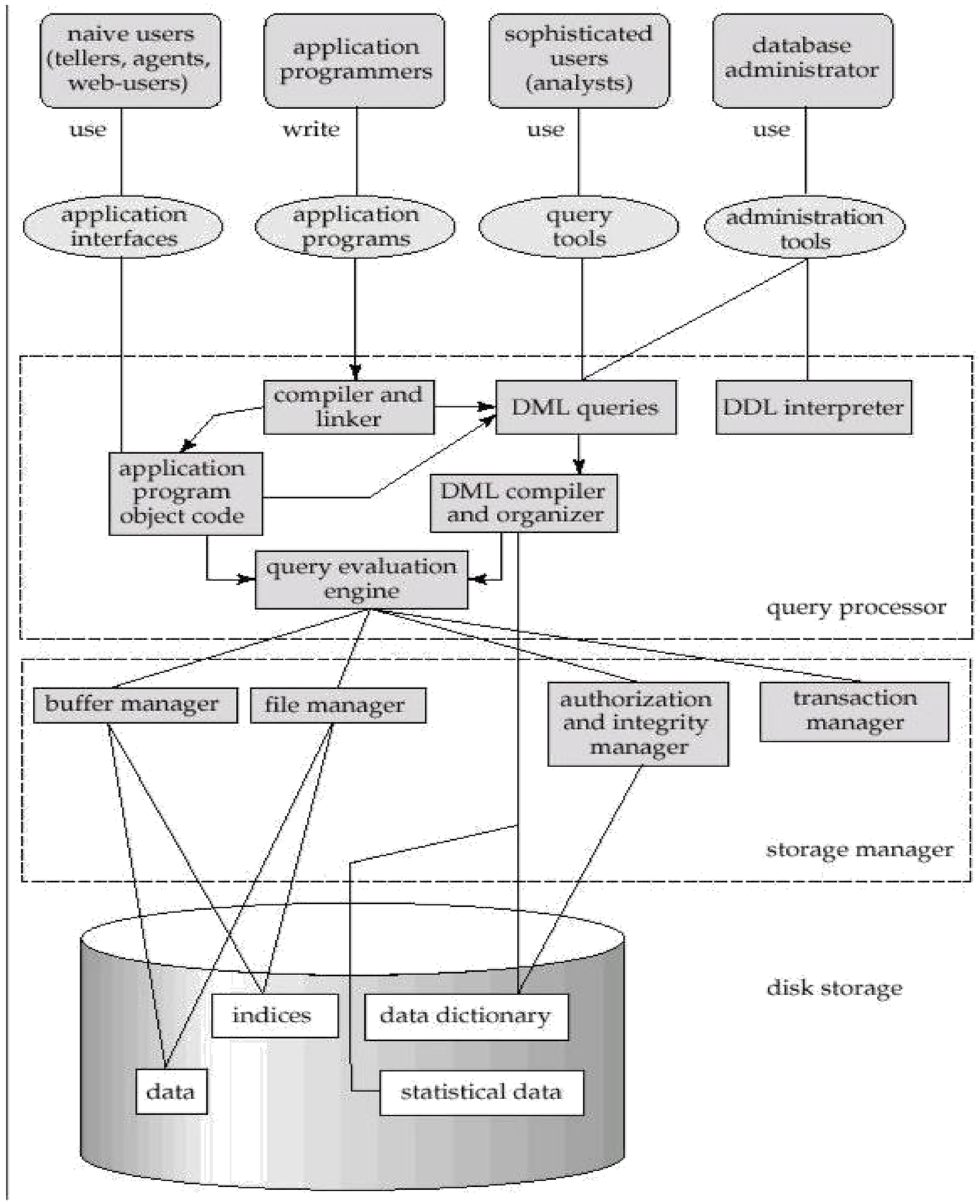
## Q3- Explain the component modules of DBMS and their interaction, with the help of a diagram.

**Ans-DATABASE SYSTEM ENVIRONMENT ( DBMS COMPONENT MODULES)**

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processorcomponents.

The storage manager is important because databases typically require a large amount of storage space. Some Big organizations Database ranges from Giga bytes to Tera bytes. So the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory asneeded.

The query processor also very important because it helps the database system simplify and facilitate access to data. So quick processing of updates and queries is important. It is the job of the database system to translate updates and queries written in a nonprocedurallanguage,


naive users (tellers, agents, web-users)
application programmers
sophisticated users (analysts)
database administrator
use
write
use
use
application interfaces
application programs
query tools
administration tools
compiler and linker
DML queries
DDL interpreter
application program object code
DML compiler and organizer
query evaluation engine
query processor
buffer manager
file manager
authorization and integrity manager
transaction manager
storage manager
disk storage
indices
data dictionary
data
statistical data

## StorageManager:

A storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.

## Storage Manager Components:

**Authorization and integrity manager** which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

**Transaction manager** which ensures that the database itself remains in a consistent state despite system failures, and that concurrent transaction executions proceed withoutconflicting.

**File manager:** which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

**Buffer manager** which is responsible for fetching data from disk storage into main memory. Storage manager implements several data structures as part of the physical system implementation. Data files are used to store the database itself. Data dictionary is used to stores metadata about the structure of the database, in particular the schema of the database.

## Query Processor Components:

**DDL interpreter:** It interprets DDL statements and records the definitions in the data dictionary.

**DML compiler:** It translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.

**Query evaluation engine:** It executes low-level instructions generated by the DML compiler.

## Application Architectures:

Most users of a database system today are not present at the site of the database system, but connect to it through a network. We can therefore differentiate between client machines,on which remote database users' work, and server machines, on which the database system runs. Database applications are usually partitioned into two or three parts. They are:

1-Two Tier architecture

2-Three Tier Architecture

## Q4-What is the difference between logical independence and physical data independence ?

**Ans-<u>DATA INDEPENDENCE:</u>**

A very important advantage of using DBMS is that it offers Data Independence.

The ability to modify a scheme definition in one level without affecting a scheme definition in a higher level is called **data independence**.

There are two kinds:

1. Physical Data Independence
2. Logical DataIndependence

**<u>Physical Data Independence:</u>**

The ability to modify the physical schema without causing application programs to be rewritten

 Modifications at this level are usually to improve performance.

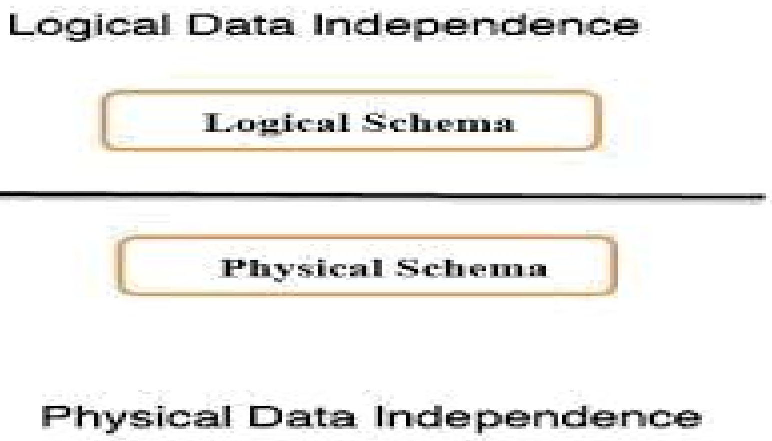


Fig: Data Independence

**Logical Data Independence:**

- The ability to modify the conceptual schema without causing application programs to be rewritten
- Usually done when logical structure of database is altered.
- Logical data independence is harder to achieve as the application programs are usually heavily dependent on the logical structure of the data.

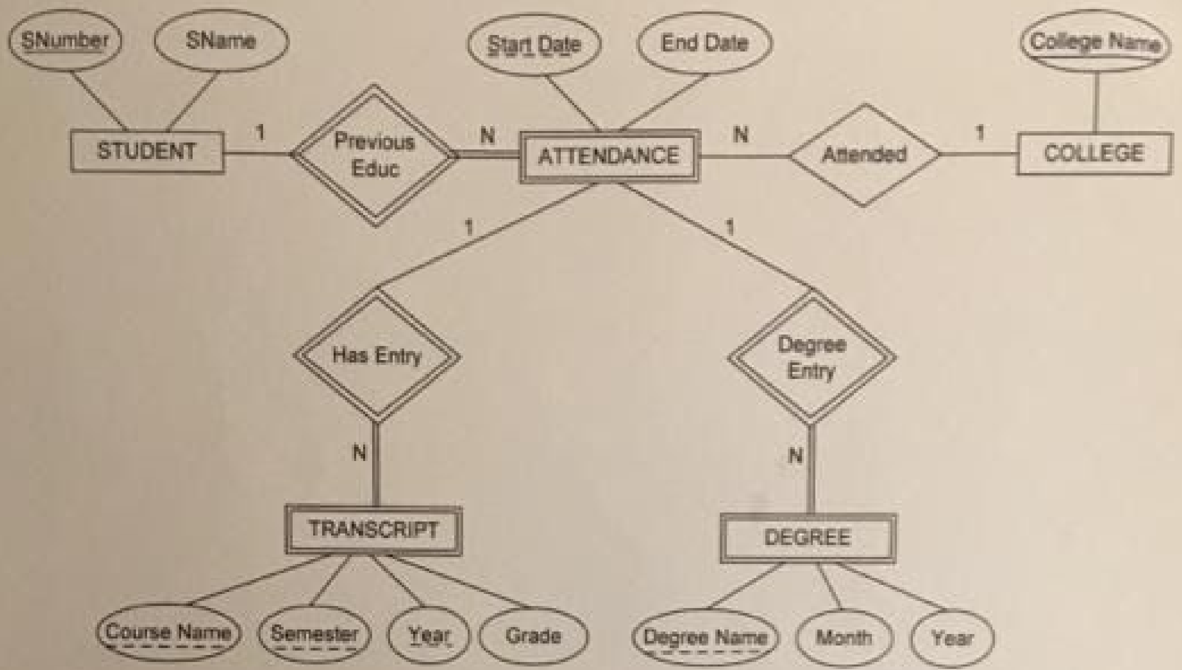
## Q5- Entity –Relationship Diagrams

**(i) University Database**

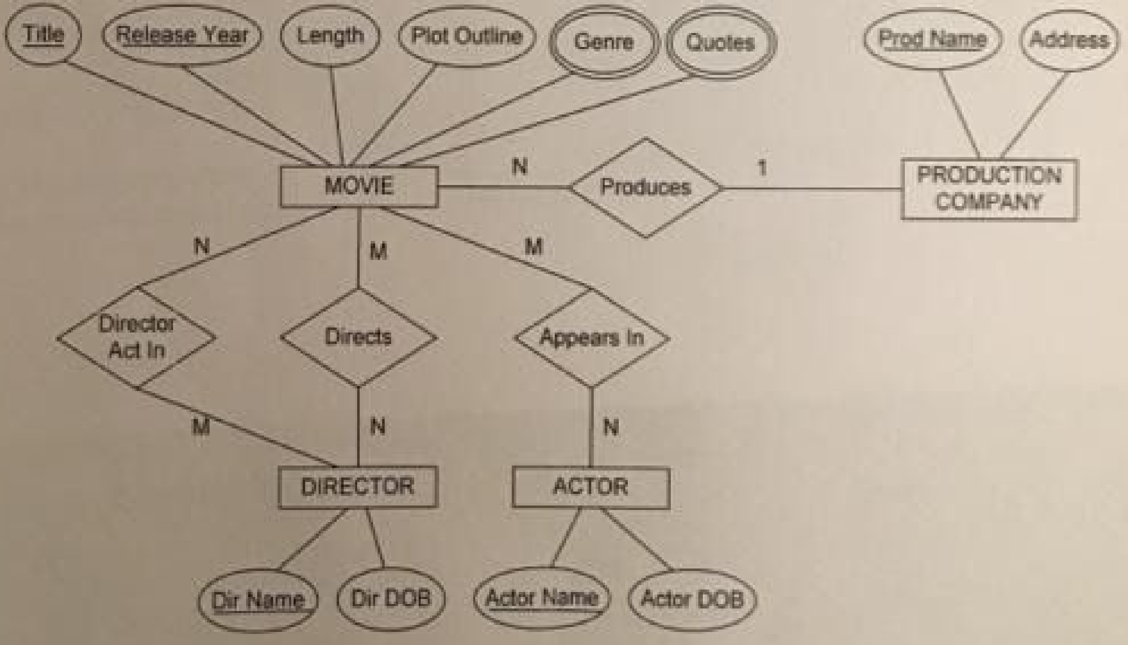
**(ii) Movie Database**

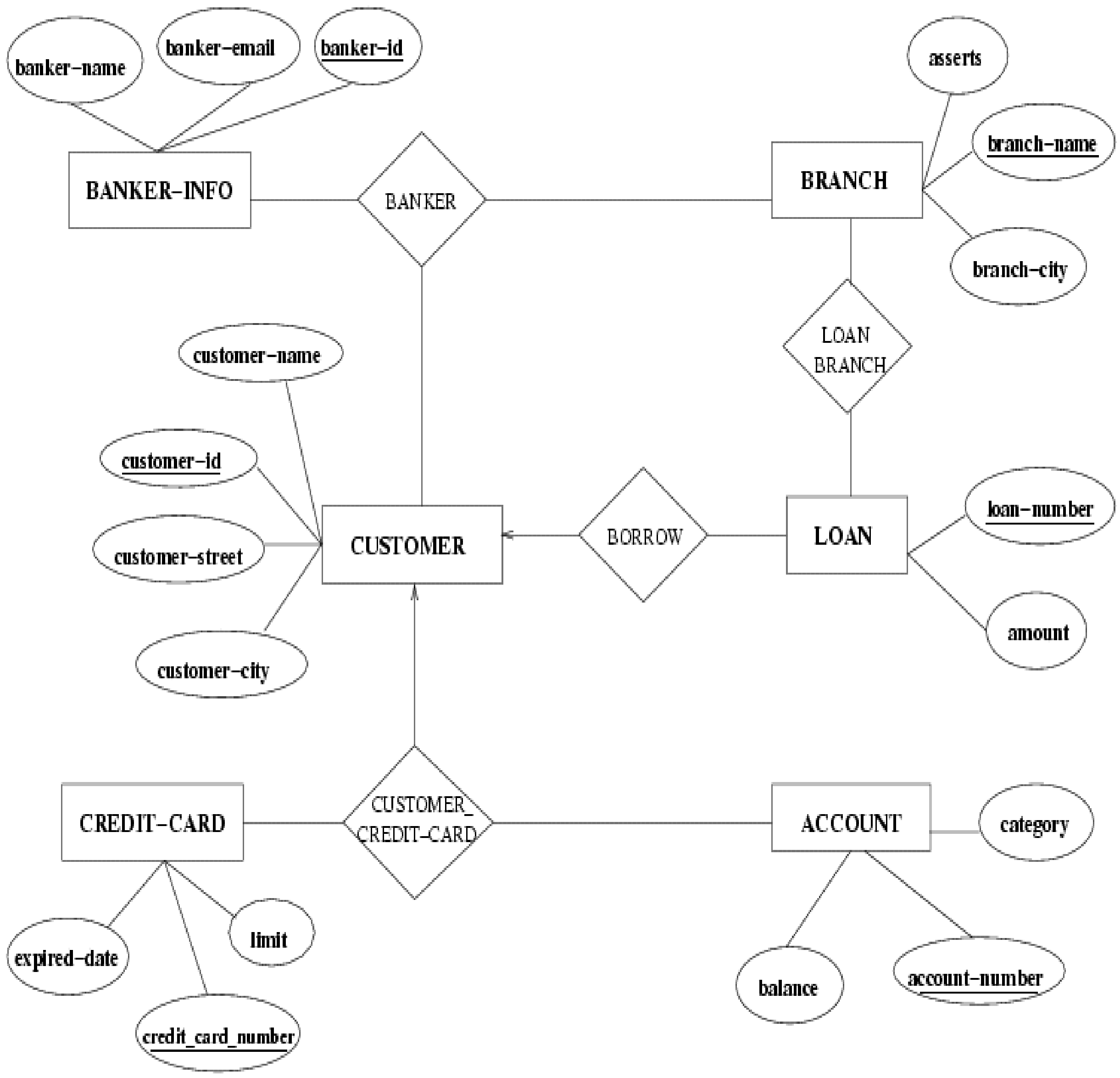
**(iii) Bank Database**

1. The ER diagram below represents the UNIVERSITY database that is used to keep track of students' transcripts. Map the ER diagram into a relational schema.



2. The ER diagram below represents the MOVIE database that is used to keep track of movies and actors. Map the ER diagram into a relational schema.





## Q6-Explain the Basic concepts of ER models(Diagrams) in DBMS

**Ans-**Entity-relationship model is a model used for design and representation of relationships between data.

The main data objects are termed as **Entities,** with their details defined as **attributes,** some of these attributes are important and are used to identify the entity, and different entities are related using **relationships**.

In short, to understand about the ER Model, we must understand about:

- Entity and Entity Set
- What are Attributes? And Types of Attributes.
- Keys
- Relationships

For a **School Management Software**, we will have to store **Student** information, **Teacher** information, **Classes**, **Subjects** taught in each class etc.**ER Model: Entity and Entity Set**

Considering the above example, **Student** is an entity, **Teacher** is an entity, similarly, **Class**, **Subject** etc are also entities.

**An Entity is generally a real-world object which has characteristics and holds relationships in a DBMS.**

If a Student is an Entity, then the complete dataset of all the students will be the **Entity Set**.

## ER Model: Attributes

If a Student is an Entity, then student's **roll no.**, student's **name**, student's **age**, student's **gender** etc will be its attributes.

An attribute can be of many types, here are different types of attributes defined in ER database model:

1. **Simple attribute:** The attributes with values that are atomic and **cannot be broken down further are simple attributes**. For example, student's **age**.
2. **Composite attribute:** A composite attribute is made up of more than one simple attribute. For example, student's **address** will contain, **house no.**, **street name**, **pin code** etc.
3. **Derived attribute:** These are the attributes which are not present in the whole database management system, but are derived using other attributes. For example**, *average age of students in a class*.**
4. **Single-valued attribute:** As the name suggests, they have a single value.
5. **Multi-valued attribute:** they can have multiple values.

## ER Model: Keys

If the attribute **roll no.** can uniquely identify a student entity, amongst all the students, then the attribute **roll no.** will be said to be a key.

Following are the types of Keys:

1. **Super Key**
2. **Candidate Key**
3. **Primary Key**

## ER Model: Relationships

**When an Entity is related to another Entity**, they are said to have a relationship. For example, A **Class** Entity is related to **Student** entity, because students study in classes, hence this is a relationship.

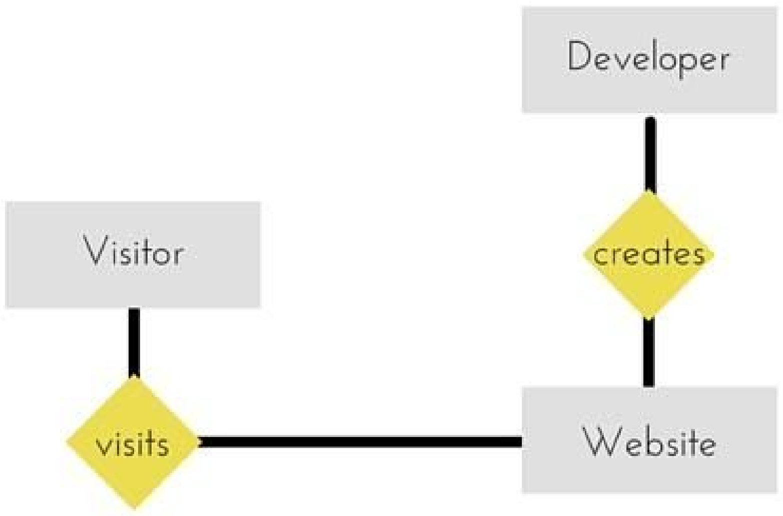
Depending upon the number of entities involved, a **degree** is assigned to relationships.

For example, if 2 entities are involved, it is said to be **Binary relationship**, if 3 entities are involved, it is said to be **Ternary** relationship, and so on.

## Working with ER Diagrams

ER Diagram is a visual representation of data that describes how data is related to each other. In ER Model, we disintegrate data into entities, attributes and setup relationships between entities, all this can be represented visually using the ER diagram.

For example, in the below diagram, anyone can see and understand what the diagram wants to convey: *Developer develops a website, whereas a Visitor visits a website*.



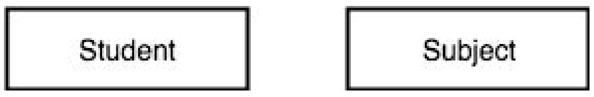
### Components of ER Diagram

Entitiy, Attributes, Relationships etc form the components of ER Diagram and there are defined symbols and shapes to represent each one of them.

Let's see how we can represent these in our ER Diagram.

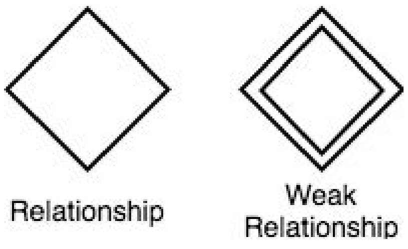
### Entity

Simple rectangular box represents an Entity.



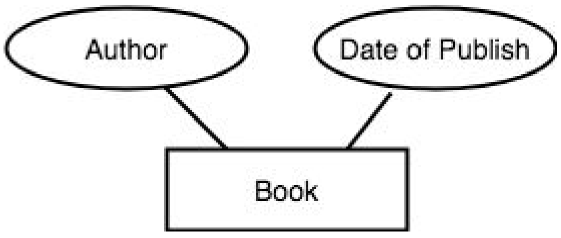
### Relationships between Entities - Weak and Strong

Rhombus is used to setup relationships between two or more entities.



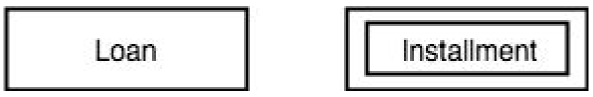
## Attributes for any Entity

Ellipse is used to represent attributes of any entity. It is connected to the entity.



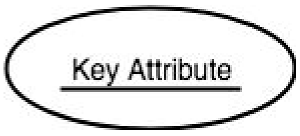
## Weak Entity

A weak Entity is represented using double rectangular boxes. It is generally connected to another entity.



## Key Attribute for any Entity

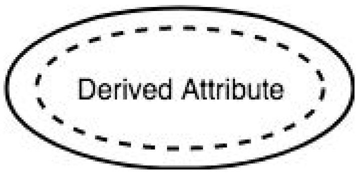
To represent a Key attribute, the attribute name inside the Ellipse is underlined.



## Derived Attribute for any Entity

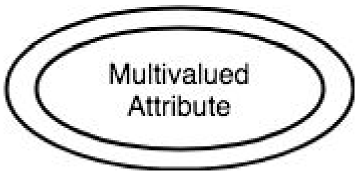
Derived attributes are those which are derived based on other attributes, for example, age can be derived from date of birth.

To represent a derived attribute, another dotted ellipse is created inside the main ellipse.



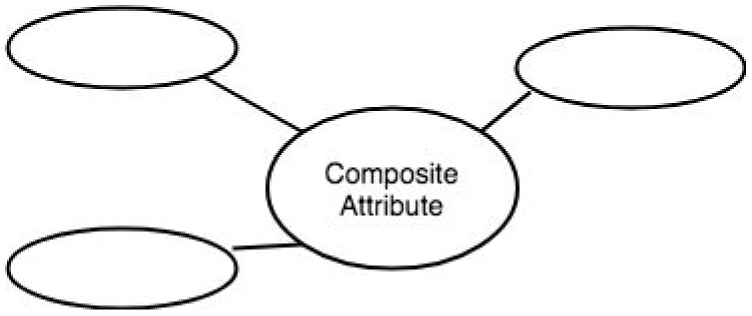
## Multivalued Attribute for any Entity

Double Ellipse, one inside another, represents the attribute which can have multiple values.



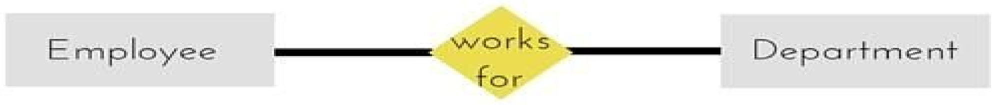
## Composite Attribute for any Entity

A composite attribute is the attribute, which also has attributes.



## ER Diagram: Entity

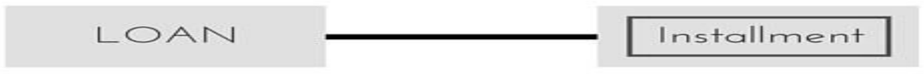
An **Entity** can be any object, place, person or class. In ER Diagram, an **entity** is represented using rectangles. Consider an example of an Organisation- Employee, Manager, Department, Product and many more can be taken as entities in an Organisation.



The yellow rhombus in between represents a relationship.

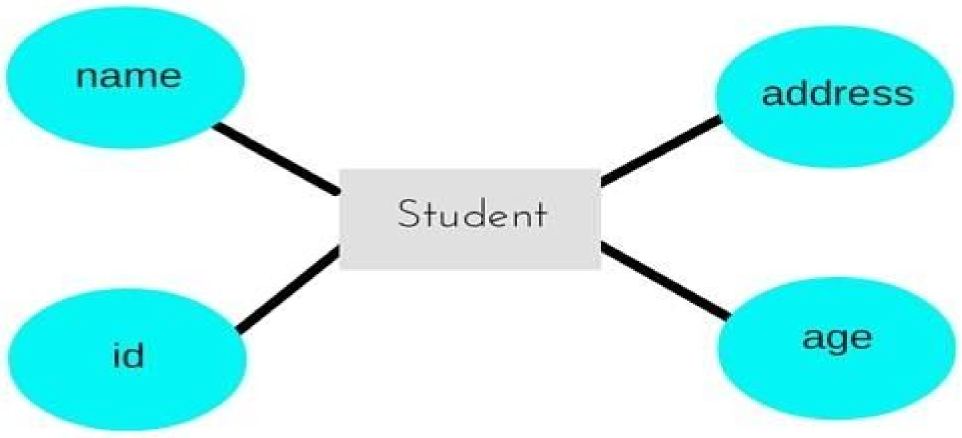
## ER Diagram: Weak Entity

Weak entity is an entity that depends on another entity. Weak entity doesn't have any key attribute of its own. Double rectangle is used to represent a weak entity.



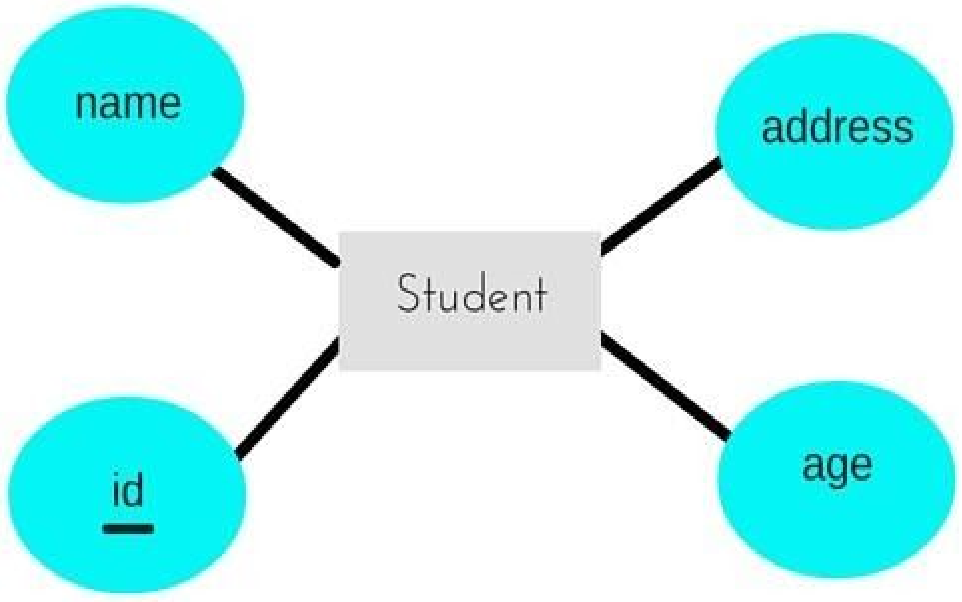
## ER Diagram: Attribute

An **Attribute** describes a property or characteristic of an entity. For example, **Name**, **Age**, **Address** etc can be attributes of a **Student**. An attribute is represented using eclipse.



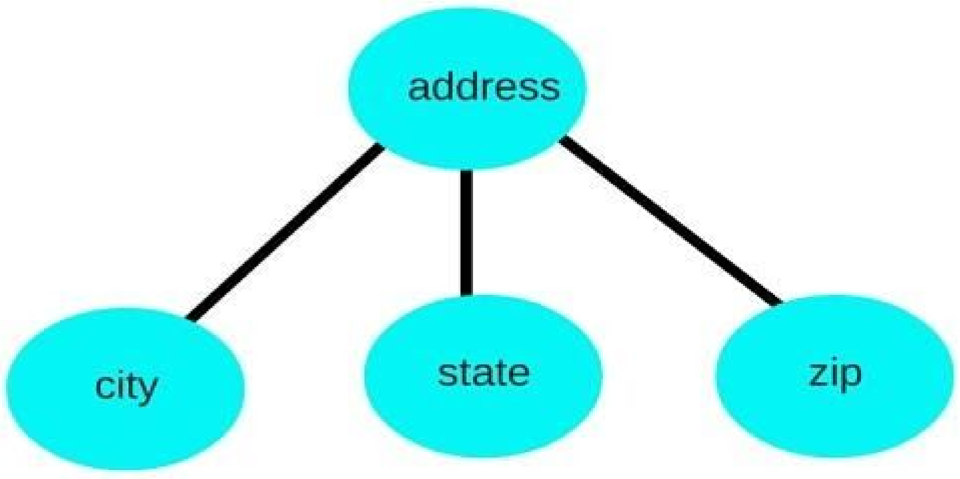
### ER Diagram: Key Attribute

Key attribute represents the main characteristic of an Entity. It is used to represent a Primary key. Ellipse with the text underlined, represents Key Attribute.



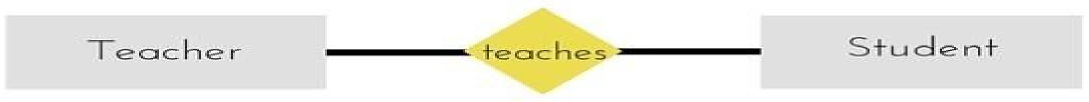
## ER Diagram: Composite Attribute

An attribute can also have their own attributes. These attributes are known as **Composite** attributes.



## ER Diagram: Relationship

A Relationship describes relation between **entities**. Relationship is represented using diamonds or rhombus.



There are three types of relationship that exist between Entities.

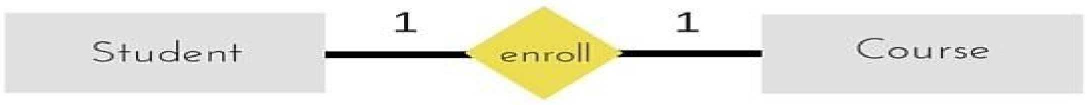
1. Binary Relationship
2. Recursive Relationship
3. Ternary Relationship

## ER Diagram: Binary Relationship

Binary Relationship means relation between two Entities. This is further divided into three types.

***One to One Relationship***

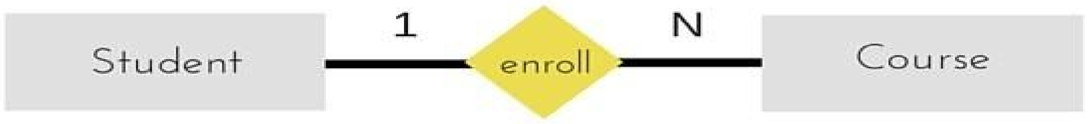
This type of relationship is rarely seen in real world.



The above example describes that one student can enroll only for one course and a course will also have only one Student. This is not what you will usually see in real-world relationships.

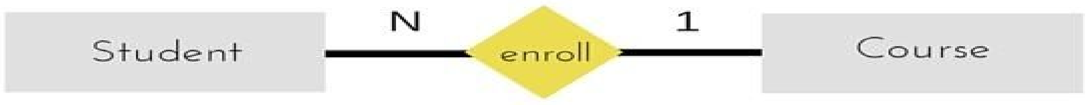
One to Many Relationship

The below example showcases this relationship, which means that 1 student can opt for many courses, but a course can only have 1 student. Sounds weird! This is how it is.

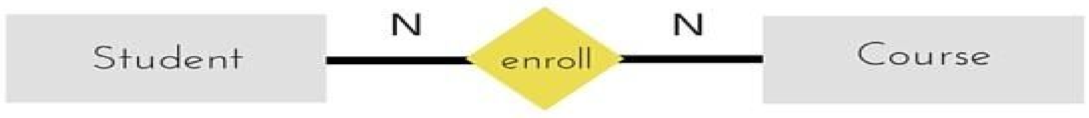


Many to One Relationship

It reflects business rule that many entities can be associated with just one entity. For example, Student enrolls for only one Course but a Course can have many Students.



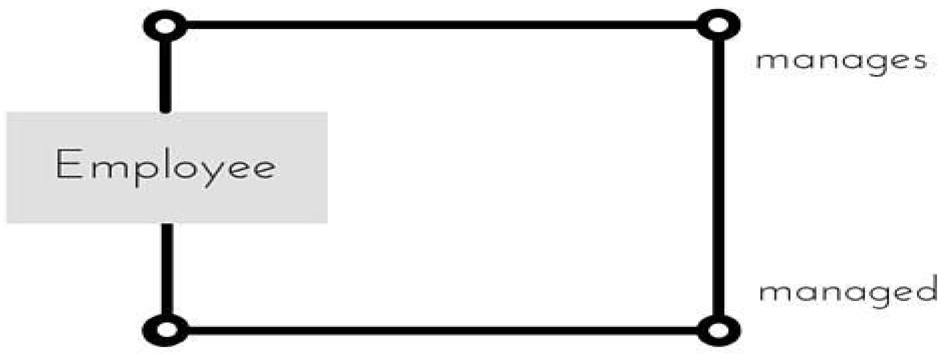
**Many to Many Relationship**



The above diagram represents that one student can enroll for more than one courses. And a course can have more than 1 student enrolled in it.

**ER Diagram: Recursive Relationship**

When an Entity is related with itself it is known as **Recursive** Relationship.

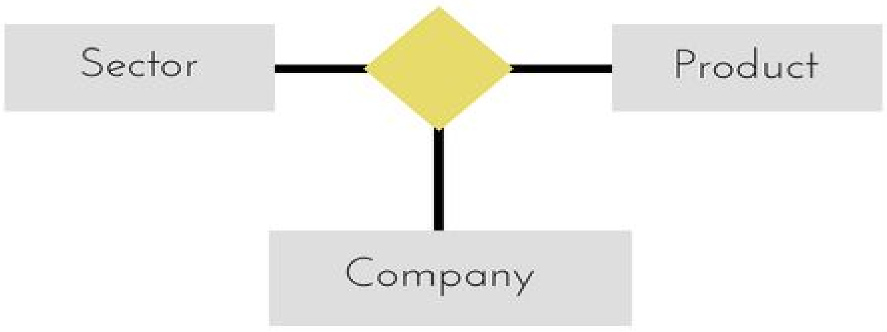


---

**ER Diagram: Ternary Relationship**

Relationship of degree three is called Ternary relationship.

A Ternary relationship involves three entities. In such relationships we always consider two entites together and then look upon the third.



- The above relationship involves 3 entities.
- Company operates in Sector, producing some Products.

For example, in the diagram above, we have three related entities, **Company**, **Product**and **Sector**. To understand the relationship better or to define rules around the model, we should relate two entities and then derive the third one.

A **Company** produces many **Products**/ each product is produced by exactly one company.

A **Company** operates in only one **Sector** / each sector has many companies operating in it.

Considering the above two rules or relationships, we see that although the complete relationship involves three entities, but we are looking at two entities at a time.

**The Enhanced ER Model**

As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modelling. Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better.

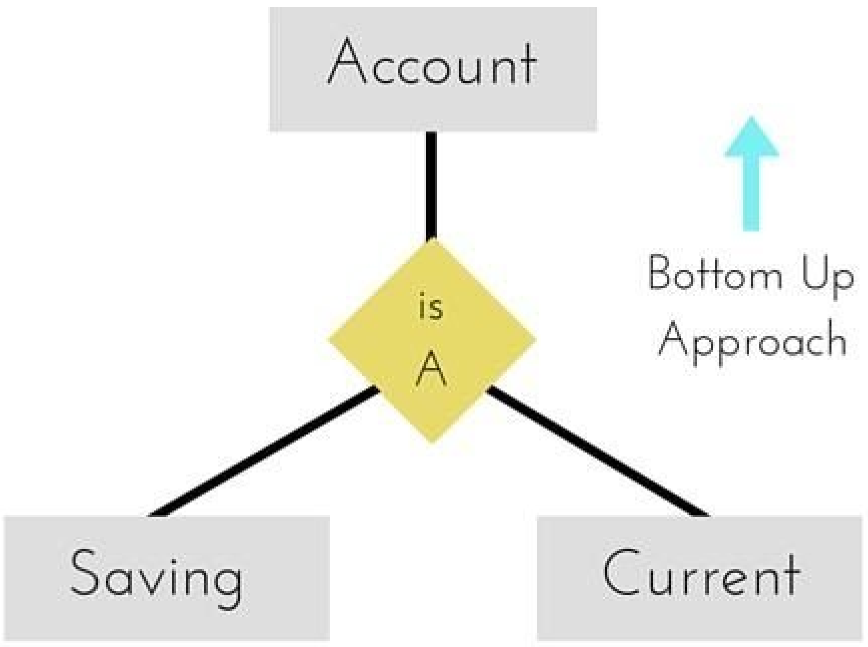
Hence, as part of the **Enhanced ER Model**, along with other improvements, three new concepts were added to the existing ER Model, they were:

1. Generalization
2. Specialization
3. Aggregation

## Generalization

Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entities to make further higher level entity.

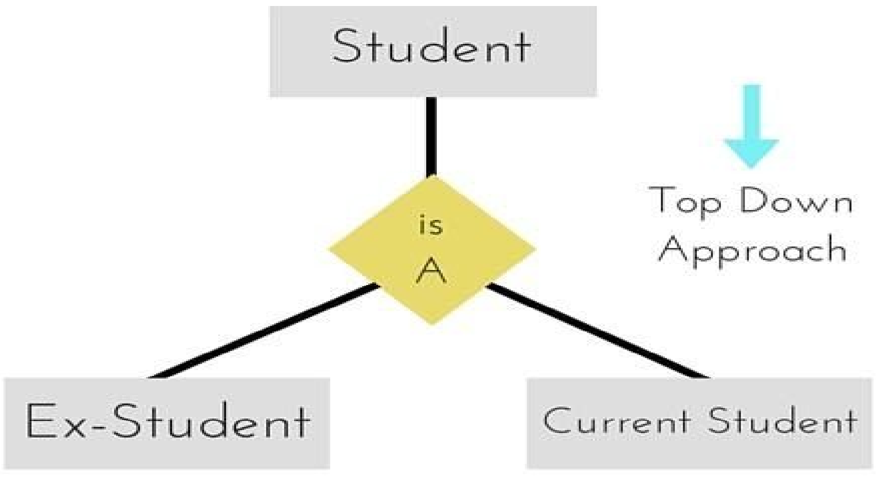
It's more like Super class and Subclass system, but the only difference is the approach, which is bottom-up. Hence, entities are combined to form a more generalised entity, in other words, sub-classes are combined to form a super-class.



For example, **Saving** and **Current** account types entities can be generalised and an entity with name **Account** can be created, which covers both.

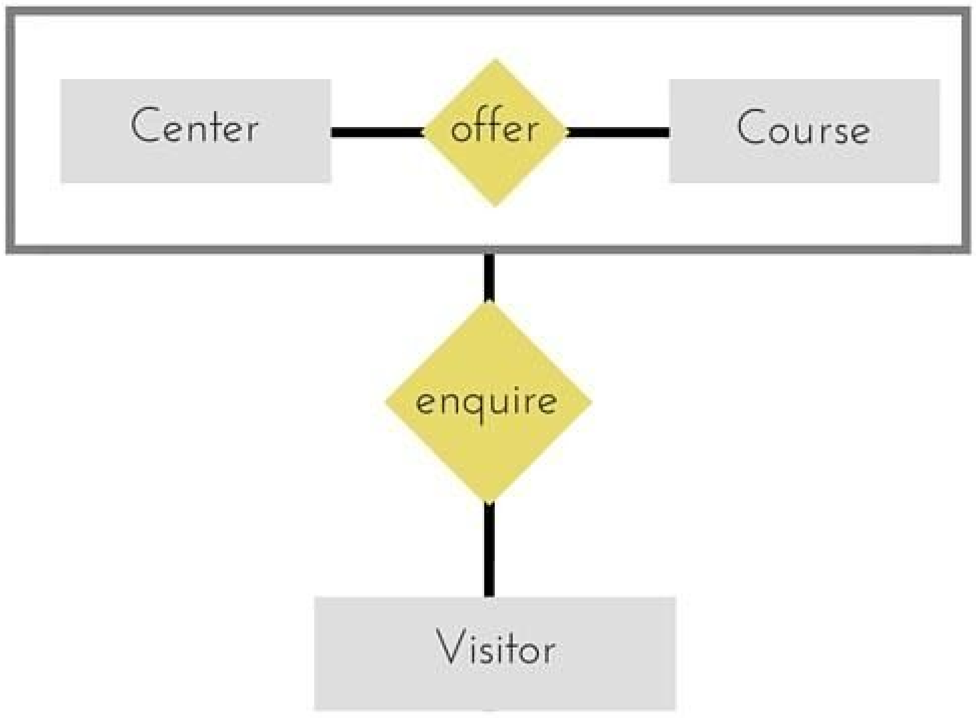
### Specialization

**Specialization** is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, a higher level entity may not have any lower-level entity sets, it's possible.



## Aggregation

Aggregation is a process when relation between two entities is treated as a **single entity**.



In the diagram above, the relationship between **Center** and **Course** together, is acting as an Entity, which is in relationship with another entity **Visitor**. Now in real world, if a Visitor or a Student visits a Coaching Center, he/she will never enquire about the center only or just about the course, rather he/she will ask enquire about both.

## Q7-Difference between Traditional file system and DBMS Approach.

**Ans-** **<u>Difference between File system & DBMS:</u>**

| File system | DBMS |
|---|---|
| Filesystemis acollection ofdata. Any 1.management withthefilesystem,userhastowritetheprocedures | 1. DBMS is a collection of data and user is not required to write the procedures for managing the database. |
| File system gives the details of 2. the data representation and Storage of data. | 2. DBMS provides an abstract view of data that hides the details. |

| | |
|---|---|
| InFile system storing and retrieving of data 3.cannot be done efficiently. | 3. DBMS is efficient use since there are to wide varieties of sophisticated and techniquesto store retrieve the data. |
| Concurrent access to the data in the file system 4.has many problems like : Reading the file while other deleting some information, updatingsome information | 4. DBMS takes care of Concurrent access using some form of locking. |
| File system doesn't provide crash 5. recovery mechanism. Eg. While we are entering some data into the file if System crashes then content of the file is lost | 5. DBMS has crash recovery mechanism, DBMS protects user from the effects of system failures. |
| 6.Protectingafileunderfilesystemisverydifficult. | 6. DBMS has a good protection mechanism. |

## Q8- Explain the classification of DBMS.

**Ans-**The Evolution of Database systems are as follows:
1.File Management System 2.Hierarchical database System 3.Network Database System 4.Relational database System

**<u>File Management System:</u>**

The file management system also called as FMS in short is one in which all data is stored on a single large file. The main disadvantage in this system is searching a record or data takes a long time. This lead to the introduction of the concept, of indexing in this system. Then also the FMS system had lot of drawbacks to name a few like updating or modifications to the data cannot be handled easily, sorting the records took long time and so on. All these drawbacks led to the introduction of the Hierarchical DatabaseSystem.

**<u>Hierarchical Database System:</u>**

The previous system FMS drawback of accessing records and sorting records which took a long time was removed in this by the introduction of parent-child relationship between records in database. The origin of the data is called the root from which several branches have data at different levels and the last level is called the leaf. The main drawback in this was if there is any modification or addition made to the structure then the whole structure needed alteration which made the task a tedious one. In order to avoid this next system took its origin which is called as the Network DatabaseSystem.

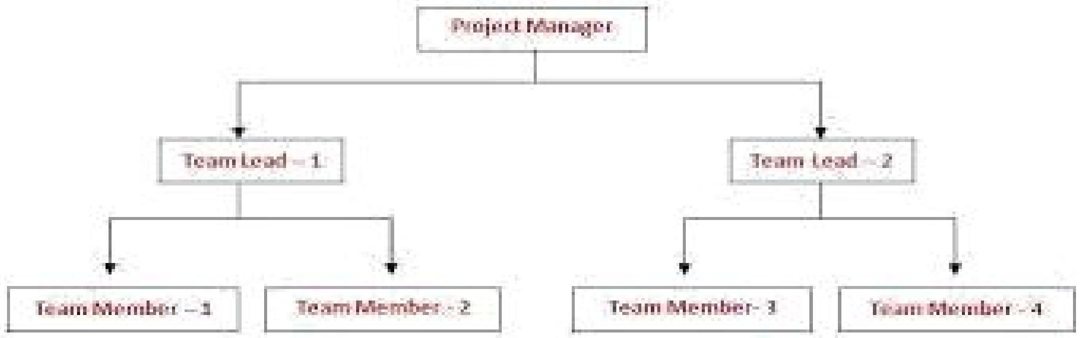
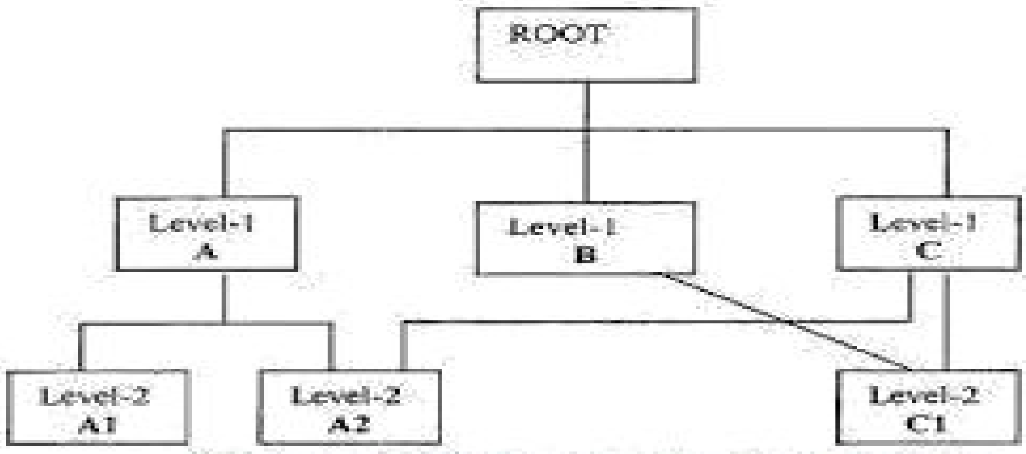


Fig: Hierarchical Database System

**Network Database System**:

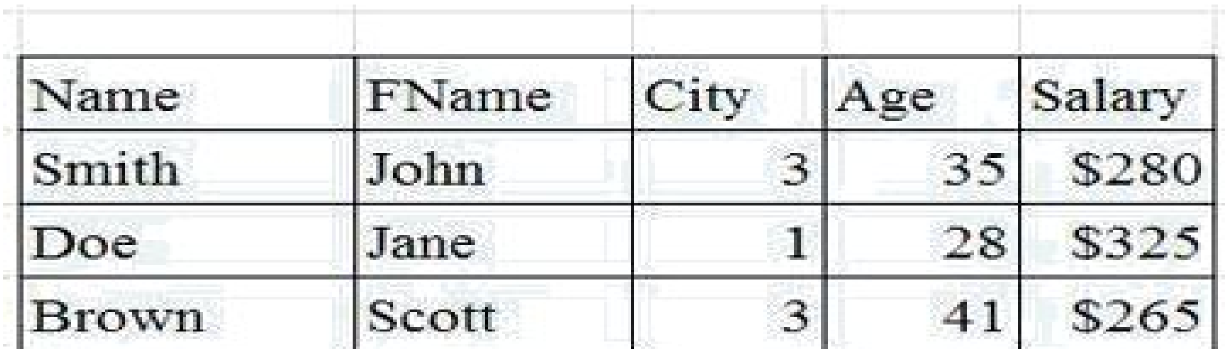
In this the main concept of many-many relationships got introduced. But this also followed the same technology of pointers to define relationships with a difference in this made in the introduction if grouping of data items as sets.



Network Database Model Diagram

**Relational Database System:**

In order to overcome all the drawbacks of the previous systems, the Relational Database System got introduced in which data get organized as tables and each record forms a row with many fields or attributes in it. Relationships between tables are also formed in this system.



| Name | FName | City | Age | Salary |
|---|---|---|---|---|
| Smith | John | 3 | 35 | $280 |
| Doe | Jane | 1 | 28 | $325 |
| Brown | Scott | 3 | 41 | $265 |

**Q9.What are different database schema languages and interfaces?**
**Ans-**

**Database Language.**

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.

Types of Database Language:

(i)DDL

(ii)DML

(iii)DCL

(iv)TCL

**1.Data Definition Language**

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

**2. Data Manipulation Language**

**DML** stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data.
- **Lock Table:** It controls concurrency.

**3. Data Control Language**

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.

  (But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

**4. Transaction Control Language**
TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

## Interfaces in DBMS

**A database management system (DBMS) interface is a user interface which allows for the ability to input queries to a database without using the query language itself.**

User-friendly interfaces provide by DBMS may include the following:

1. **Menu-Based Interfaces for Web Clients or Browsing** –
   These interfaces present the user with lists of options (called menus) that lead the user through the formation of a request. Basic advantage of using menus is that they removes the tension of remembering specific commands and syntax of any query language, rather than query is basically composed step by step by collecting or picking options from a menu that is basically shown by the system. Pull-down menus are a very popular technique in *Web based interfaces*. They are also often used in *browsing interface* which allow a user to look through the contents of a database in an exploratory and unstructured manner.
2. **Forms-Based Interfaces** –
   A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert a new data, or they can fill out only certain entries, in which case the DBMS will redeem same type of data for other remaining entries. This type of forms are usually designed or created and programmed for the users that have no expertise in operating system. Many DBMSs have *forms specification languages* which are special languages that help specify such forms.
   Example: SQL* Forms is a form-based language that specifies queries using a form designed in conjunction with the relational database schema.
3. **Graphical User Interface** –
   A GUI typically displays a schema to the user in diagrammatic form.The user then can specify a query by manipulating the diagram. In many cases, GUI's utilize both menus and forms. Most GUIs use a pointing device such as mouse, to pick certain part of the displayed schema diagram.
4. **Natural language Interfaces** –
   These interfaces accept request written in English or some other language and

attempt to understand them. A Natural language interface has its own schema, which is similar to the database conceptual schema as well as a dictionary of important words.
The natural language interface refers to the words in its schema as well as to the set of standard words in a dictionary to interpret the request.If the interpretation is successful, the interface generates a high-level query corresponding to the natural language and submits it to the DBMS for processing, otherwise a dialogue is started with the user to clarify any provided condition or request. The main disadvantage with this is that the capabilities of this type of interfaces are not that much advance.

5. **Speech Input and Output –**
   There is an limited use of speech say it for a query or an answer to a question or being a result of a request it is becoming commonplace Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure, and bank account information are allowed speech for input and output to enable ordinary folks to access this information.
   The Speech input is detected using a predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech take place.

**Interfaces for DBA –**
Most database system contains privileged commands that can be used only by the DBA's staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, reorganizing the storage structures of a databases.

# MODULE-2

## Q1.What is Relational algebra. Explain different relational algebra operations with example.

## Ans-Relational Algebra

Relational algebra is the basic set of operations for the relational model .

➲ These operations enable a user to specify basic retrieval requests (or queries)

➲ The result of an operation is a new relation, which may have been formed from one or more input relations .This property makes the algebra "closed" .

The algebra operations thus produce new relations .These can be further manipulated using operationsλ of the same algebra

➲ A sequence of relational algebra operations forms a relational algebra expression .The result of a relational algebra expression is alsoλ a relation that represents the result of a database query (or retrieval request)

➲ Relational Algebra consists of several groups of operations

**Unary Relational Operations**

SELECT (symbol: σ (sigma))

PROJECT (symbol: π (pi))

RENAME (symbol: ρ (rho))

**Relational Algebra Operations From Set Theory**

UNION ( ∪ ), INTERSECTION ( ∩ ), DIFFERENCE (or MINUS, – ), CARTESIAN PRODUCT ( x )

**Binary Relational Operations**

JOIN (several variations of JOIN exist) , DIVISION

**Additional Relational Operations**: OUTER JOINS, OUTER UNION, AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

## The SELECT Operation

The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a selection condition.

It is a filter that keeps only those tuples that satisfy a qualifying condition.

The SELECT operation can also be visualized as a horizontal partition of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.

For example, to select the EMPLOYEE tuples whose department is 4, or those whose salary is greater than $30,000, we can individually specify each of these two conditions with a SELECT operation as follows:

$$\sigma_{Dno=4}(\text{EMPLOYEE})$$
$$\sigma_{Salary>30000}(\text{EMPLOYEE})$$

In general, the SELECT operation is denoted by

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

where the symbol σ (sigma) is used to denote the SELECT operator and the selection condition is a Boolean expression (condition) specified on the attributes of relation R.

The Boolean expression specified in <selection condition> is made up of a numberof clauses of the form

<attribute name><comparison op><constant value>

or

<attribute name><comparison op><attribute name>

where <attribute name> is the name of an attribute of R, <comparison op> is normally one of the operators {=, <, ≤, >, ≥, ≠}, and <constant value> is a constant value from the attribute domain.

the tuples for all employees who either work in department 4 and make over $25,000 per year, or work in department 5 and make over $30,000, we can specify the following SELECT operation:

$$\sigma_{(\text{Dno}=4\ \textbf{AND}\ \text{Salary}>25000)\ \textbf{OR}\ (\text{Dno}=5\ \textbf{AND}\ \text{Salary}>30000)}(\text{EMPLOYEE})$$

The SELECT operator is unary; that is, it is applied to a single relation. Moreover, the selection operation is applied to each tuple individually; hence, selection conditions cannot involve more than one tuple. The degree of the relation resulting from a SELECT operation—its number of attributes—is the same as the degree of R.

The number of tuples in the resulting relation is always less than or equal to the number of tuples in R. That is, $|\sigma c\ (R)| \leq |R|$ for any condition C. The fraction of tuples selected by a selection condition is referred to as the selectivity of the condition.

Notice that the SELECT operation is **commutative**; that is,

$$\sigma_{<\text{cond1}>}(\sigma_{<\text{cond2}>}(R)) = \sigma_{<\text{cond2}>}(\sigma_{<\text{cond1}>}(R))$$

Hence, a sequence of SELECTs can be applied in any order. In addition, we can always combine a **cascade** (or **sequence**) of SELECT operations into a single SELECT operation with a conjunctive (AND) condition; that is,

$$\sigma_{<\text{cond1}>}(\sigma_{<\text{cond2}>}(\ldots(\sigma_{<\text{cond}n>}(R))\ \ldots)) = \sigma_{<\text{cond1}>\ \textbf{AND}<\text{cond2}>\ \textbf{AND}\ldots\textbf{AND}\ <\text{cond}n>}(R)$$

In SQL, the SELECT condition is typically specified in the WHERE clause of a query. For example, the following operation:

$$\sigma_{\text{Dno}=4\ \textbf{AND}\ \text{Salary}>25000}\ (\text{EMPLOYEE})$$

would correspond to the following SQL query:

```
SELECT    *
FROM      EMPLOYEE
WHERE     Dno=4 AND Salary>25000;
```

## <u>The PROJECT Operation</u>

The PROJECT operation selects certain columns from the table and discards the other columns.

The result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations: one has the needed columns (attributes) and contains the result of the operation, and the other contains the discarded columns.

to list each employee's first and last name and salary, we can use the PROJECT operation as follows:

$$\pi_{\text{Lname, Fname, Salary}}(\text{EMPLOYEE})$$

The general form of the PROJECT operation is

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

where π (pi) is the symbol used to represent the PROJECT operation, and <attribute list> is the desired sublist of attributes from the attributes of relation *R*. ~~Again~~

The result of the PROJECT operation has only the attributes specified in <attribute list> in the same

order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute list>.

The PROJECT operation removes any duplicate tuples, so the result of the PROJECT operation is a set of distinct tuples.

The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in R.

In SQL, the PROJECT attribute list is specified in the SELECT clause of a query. For example, the following operation:

$$\pi_{\text{Sex, Salary}}(\text{EMPLOYEE})$$

would correspond to the following SQL query:

```
SELECT      DISTINCT Sex, Salary
FROM        EMPLOYEE
```

## **<u>Sequences of Operations and the RENAME Operation</u>**

In general, for most queries, we need to apply several relational algebra operations one after the other. Either we can write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations. In the latter case, we must give names to the relations that hold the intermediate results.

For example, to retrieve the first name, last name, and salary of all employees who work in department

number 5, we must apply a SELECT and a PROJECT operation.We can write a single relational algebra expression, also known as an in-line expression, as follows:

$\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno}=5}(\text{EMPLOYEE}))$

Figure 6.2(a) shows the result of this in-line relational algebra expression. Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation, as follows:

$$\text{DEP5\_EMPS} \leftarrow \sigma_{\text{Dno}=5}(\text{EMPLOYEE})$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Salary}}(\text{DEP5\_EMPS})$$

We can also define a formal **RENAME** operation—which can rename either the relation name or the attribute names, or both—as a unary operator. The general RENAME operation when applied to a relation $R$ of degree $n$ is denoted by any of the following three forms:

$$\rho_{S(B1,\, B2,\, \ldots,\, Bn)}(R) \quad \text{or} \quad \rho_S(R) \quad \text{or} \quad \rho_{(B1,\, B2,\, \ldots,\, Bn)}(R)$$

where the symbol ρ (rho) is used to denote the RENAME operator, $S$ is the new relation name, and $B_1, B_2, \ldots, B_n$ are the new attribute names. The first expression renames both the relation and its attributes, the second renames the relation only, and the third renames the attributes only. If the attributes of $R$ are $(A_1, A_2, \ldots, A_n)$ in that order, then each $A_i$ is renamed as $B_i$.

**Relational Algebra Operations from Set Theory**

Several set theoretic operations are used to merge the elements of two sets in various ways, including UNION, INTERSECTION, and SET DIFFERENCE (also called MINUS or EXCEPT). These are binary operations; that is, each is applied to two sets(of tuples).

When these operations are adapted to relational databases, the two relations on which any of these three operations are to be union compatible (or type compatible) if they have the same degree n and if dom(Ai) = dom(Bi) for 1 ▯i▯ n. This means that the two relations have the same number of attributes and each corresponding applied must have the same type of tuples; this condition has been called **union compatibility** or type compatibility.

Two relations R(A1, A2, ..., An) and S(B1, B2, ..., Bn) are said pair of attributes has the same domain.

We can define the three operations UNION, INTERSECTION, and SET DIFFERENCE

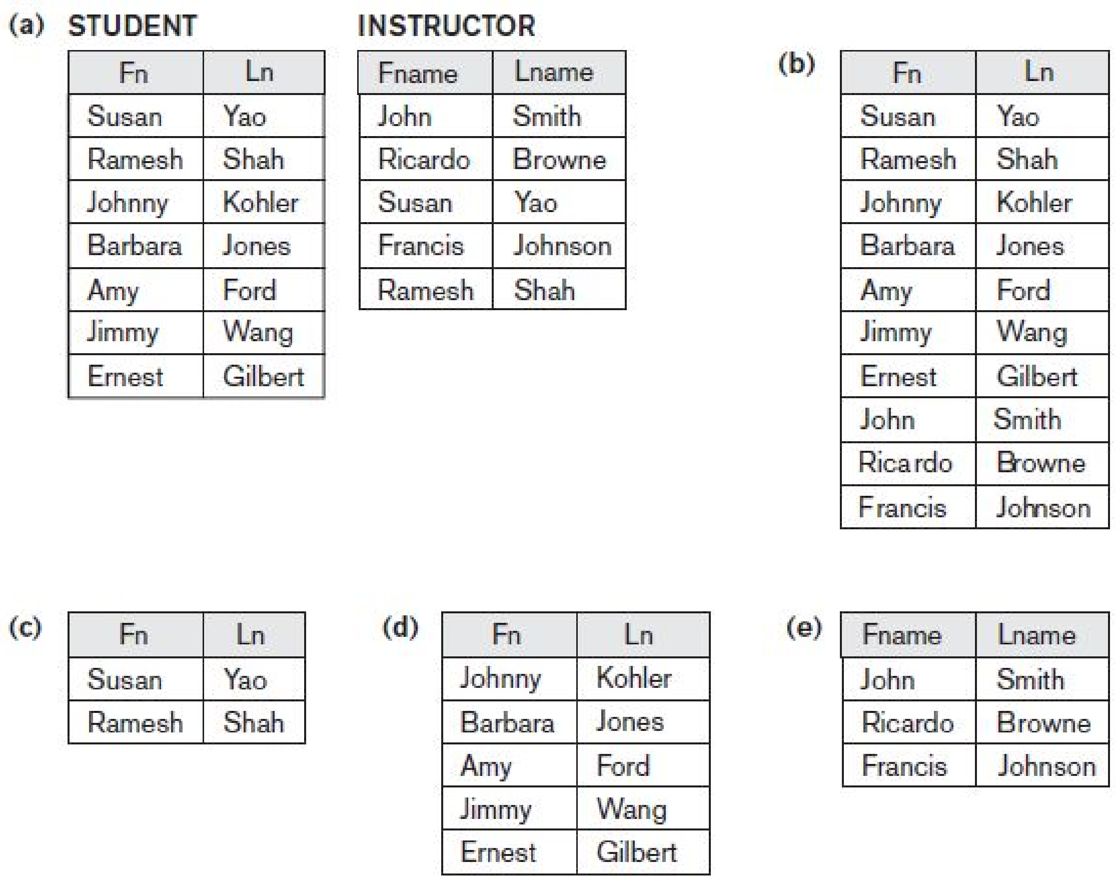
on two union-compatible relations R and S as follows:

■ UNION: The result of this operation, denoted by R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

■ INTERSECTION: The result of this operation, denoted by R ∩ S, is a relation that includes all tuples that are in both R and S.

■ SET DIFFERENCE (or MINUS): The result of this operation, denoted by R – S, is a relation that includes all tuples that are in R but not in S.

**Figure 6.4**
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.



(a) **STUDENT**

| Fn | Ln |
|---|---|
| Susan | Yao |
| Ramesh | Shah |
| Johnny | Kohler |
| Barbara | Jones |
| Amy | Ford |
| Jimmy | Wang |
| Ernest | Gilbert |

**INSTRUCTOR**

| Fname | Lname |
|---|---|
| John | Smith |
| Ricardo | Browne |
| Susan | Yao |
| Francis | Johnson |
| Ramesh | Shah |

(b)

| Fn | Ln |
|---|---|
| Susan | Yao |
| Ramesh | Shah |
| Johnny | Kohler |
| Barbara | Jones |
| Amy | Ford |
| Jimmy | Wang |
| Ernest | Gilbert |
| John | Smith |
| Ricardo | Browne |
| Francis | Johnson |

(c)

| Fn | Ln |
|---|---|
| Susan | Yao |
| Ramesh | Shah |

(d)

| Fn | Ln |
|---|---|
| Johnny | Kohler |
| Barbara | Jones |
| Amy | Ford |
| Jimmy | Wang |
| Ernest | Gilbert |

(e)

| Fname | Lname |
|---|---|
| John | Smith |
| Ricardo | Browne |
| Francis | Johnson |

INTERSECTION can be expressed in terms of union and set difference as follows:

$$R \cap S = ((R \cup S) - (R - S)) - (S - R)$$

## <u>The CARTESIAN PRODUCT (CROSS PRODUCT) Operation</u>

The CARTESIAN PRODUCT operation—also known as CROSS PRODUCT or CROSS JOIN is denoted by ×. This is also a binary set operation, but the relations on which it is applied do not have to be union compatible. In its binary form, this set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set). In general, the result of R(A1, A2, … , An) × S(B1, B2, … , Bm) is a relation Q with degree n + m attributes Q(A1, A2, … , An, B1, B2, … , Bm), in that order.

The resulting relation Q has one tuple for each combination of tuples—one from R and one from S. Hence, if R has nR tuples (denoted as |R| = nR), and S has nS tuples, then R × S will have nR * nS tuples.

- Example: relations $r$, $s$:

$r$

| A | B |
|---|---|
| $\alpha$ | 1 |
| $\beta$ | 2 |

$s$

| C | D | E |
|---|---|---|
| $\alpha$ | 10 | $+$ |
| $\beta$ | 10 | $+$ |
| $\beta$ | 20 | $-$ |
| $\gamma$ | 10 | $-$ |

$r \times s$

| A | B | C | D | E |
|---|---|---|---|---|
| $\alpha$ | 1 | $\alpha$ | 10 | $+$ |
| $\alpha$ | 1 | $\beta$ | 10 | $+$ |
| $\alpha$ | 1 | $\beta$ | 20 | $-$ |
| $\alpha$ | 1 | $\gamma$ | 10 | $-$ |
| $\beta$ | 2 | $\alpha$ | 10 | $+$ |
| $\beta$ | 2 | $\beta$ | 10 | $+$ |
| $\beta$ | 2 | $\beta$ | 20 | $-$ |
| $\beta$ | 2 | $\gamma$ | 10 | $-$ |

| OPERATION | PURPOSE | NOTATION |
|---|---|---|
| SELECT | Selects all tuples that satisfy the selection condition from a relation $R$. | $\sigma_{<\text{selection condition}>}(R)$ |
| PROJECT | Produces a new relation with only some of the attributes of $R$, and removes duplicate tuples. | $\pi_{<\text{attribute list}>}(R)$ |
| THETA JOIN | Produces all combinations of tuples from $R_1$ and $R_2$ that satisfy the join condition. | $R_1 \bowtie_{<\text{join condition}>} R_2$ |
| EQUIJOIN | Produces all the combinations of tuples from $R_1$ and $R_2$ that satisfy a join condition with only equality comparisons. | $R_1 \bowtie_{<\text{join condition}>} R_2$, OR $R_1 \bowtie_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$ |
| NATURAL JOIN | Same as EQUIJOIN except that the join attributes of $R_2$ are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all. | $R_1 *_{<\text{join condition}>} R_2$, OR $R_1 *_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$ OR $R_1 * R_2$ |
| UNION | Produces a relation that includes all the tuples in $R_1$ or $R_2$ or both $R_1$ and $R_2$; $R_1$ and $R_2$ must be union compatible. | $R_1 \cup R_2$ |
| INTERSECTION | Produces a relation that includes all the tuples in both $R_1$ and $R_2$; $R_1$ and $R_2$ must be union compatible. | $R_1 \cap R_2$ |
| DIFFERENCE | Produces a relation that includes all the tuples in $R_1$ that are not in $R_2$; $R_1$ and $R_2$ must be union compatible. | $R_1 - R_2$ |
| CARTESIAN PRODUCT | Produces a relation that has the attributes of $R_1$ and $R_2$ and includes as tuples all possible combinations of tuples from $R_1$ and $R_2$. | $R_1 \times R_2$ |
| DIVISION | Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in $R_1$ in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$. | $R_1(Z) \div R_2(Y)$ |

.38 x 9.13 in

**Q2.Write the following expressions in Relational algebra.**

**Query 1.** Retrieve the name and address of all employees who work for the 'Research' department.

$\text{RESEARCH\_DEPT} \leftarrow \sigma_{\text{Dname='Research'}}(\text{DEPARTMENT})$
$\text{RESEARCH\_EMPS} \leftarrow (\text{RESEARCH\_DEPT} \bowtie_{\text{Dnumber=Dno}} \text{EMPLOYEE})$
$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Address}}(\text{RESEARCH\_EMPS})$

As a single in-line expression, this query becomes:

$\pi_{\text{Fname, Lname, Address}} (\sigma_{\text{Dname='Research'}}(\text{DEPARTMENT} \bowtie_{\text{Dnumber=Dno}}(\text{EMPLOYEE}))$

**Query 2.** For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

$\text{STAFFORD\_PROJS} \leftarrow \sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})$
$\text{CONTR\_DEPTS} \leftarrow (\text{STAFFORD\_PROJS} \bowtie_{\text{Dnum=Dnumber}} \text{DEPARTMENT})$
$\text{PROJ\_DEPT\_MGRS} \leftarrow (\text{CONTR\_DEPTS} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE})$
$\text{RESULT} \leftarrow \pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}(\text{PROJ\_DEPT\_MGRS})$

In this example, we first select the projects located in Stafford, then join them with their controlling departments, and then join the result with the department managers. Finally, we apply a project operation on the desired attributes.

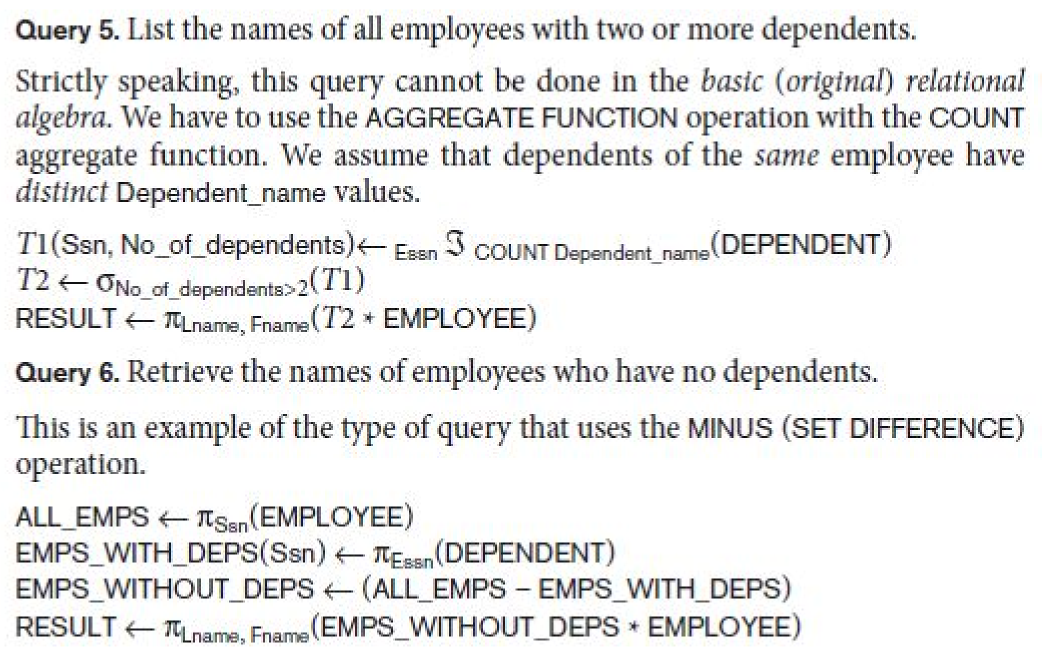
**Query 3.** Find the names of employees who work on *all* the projects controlled by department number 5.

$\text{DEPT5\_PROJS} \leftarrow \rho_{(\text{Pno})}(\pi_{\text{Pnumber}}(\sigma_{\text{Dnum=5}}(\text{PROJECT})))$
$\text{EMP\_PROJ} \leftarrow \rho_{(\text{Ssn, Pno})}(\pi_{\text{Essn, Pno}}(\text{WORKS\_ON}))$
$\text{RESULT\_EMP\_SSNS} \leftarrow \text{EMP\_PROJ} \div \text{DEPT5\_PROJS}$
$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{RESULT\_EMP\_SSNS} * \text{EMPLOYEE})$

In this query, we first create a table DEPT5_PROJS that contains the project numbers of all projects controlled by department 5. Then we create a table EMP_PROJ that holds (Ssn, Pno) tuples, and apply the division operation. Notice that we renamed the attributes so that they will be correctly used in the division operation. Finally, we join the result of the division, which holds only Ssn values, with the EMPLOYEE table to retrieve the Fname, Lname attributes from EMPLOYEE.

**Query 4.** Make a list of project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

$\text{SMITHS(Essn)} \leftarrow \pi_{\text{Ssn}} (\sigma_{\text{Lname='Smith'}}(\text{EMPLOYEE}))$
$\text{SMITH\_WORKER\_PROJS} \leftarrow \pi_{\text{Pno}}(\text{WORKS\_ON} * \text{SMITHS})$
$\text{MGRS} \leftarrow \pi_{\text{Lname, Dnumber}}(\text{EMPLOYEE} \bowtie_{\text{Ssn=Mgr\_ssn}} \text{DEPARTMENT})$
$\text{SMITH\_MANAGED\_DEPTS(Dnum)} \leftarrow \pi_{\text{Dnumber}} (\sigma_{\text{Lname='Smith'}}(\text{MGRS}))$
$\text{SMITH\_MGR\_PROJS(Pno)} \leftarrow \pi_{\text{Pnumber}}(\text{SMITH\_MANAGED\_DEPTS} * \text{PROJECT})$
$\text{RESULT} \leftarrow (\text{SMITH\_WORKER\_PROJS} \cup \text{SMITH\_MGR\_PROJS})$


**Query 5.** List the names of all employees with two or more dependents.

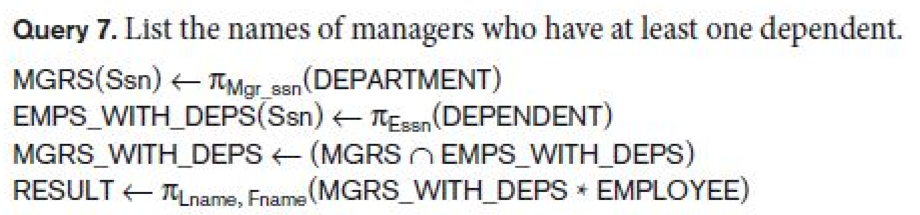
Strictly speaking, this query cannot be done in the *basic (original) relational algebra*. We have to use the AGGREGATE FUNCTION operation with the COUNT aggregate function. We assume that dependents of the *same* employee have *distinct* Dependent_name values.

$T1(\text{Ssn, No\_of\_dependents}) \leftarrow {}_{\text{Essn}}\mathfrak{I}_{\text{COUNT Dependent\_name}}(\text{DEPENDENT})$
$T2 \leftarrow \sigma_{\text{No\_of\_dependents}>2}(T1)$
$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(T2 * \text{EMPLOYEE})$

**Query 6.** Retrieve the names of employees who have no dependents.

This is an example of the type of query that uses the MINUS (SET DIFFERENCE) operation.

$\text{ALL\_EMPS} \leftarrow \pi_{\text{Ssn}}(\text{EMPLOYEE})$
$\text{EMPS\_WITH\_DEPS(Ssn)} \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$
$\text{EMPS\_WITHOUT\_DEPS} \leftarrow (\text{ALL\_EMPS} - \text{EMPS\_WITH\_DEPS})$
$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{EMPS\_WITHOUT\_DEPS} * \text{EMPLOYEE})$


**Query 7.** List the names of managers who have at least one dependent.

$\text{MGRS(Ssn)} \leftarrow \pi_{\text{Mgr\_ssn}}(\text{DEPARTMENT})$
$\text{EMPS\_WITH\_DEPS(Ssn)} \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$
$\text{MGRS\_WITH\_DEPS} \leftarrow (\text{MGRS} \cap \text{EMPS\_WITH\_DEPS})$
$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{MGRS\_WITH\_DEPS} * \text{EMPLOYEE})$

## Q3. Explain how the different update operations deal with constraint violations.

Ans-There are three basic operations that can change the states of relations in the database:
Insert, Delete, and Update (or Modify). There are three basic operations that can change the states of relations in the database:Insert, Delete, and Update (or Modify). They insert new data, delete old data,or modify existing data records, respectively. **Insert** is used to insert one or more new tuples in a relation, **Delete** is used to delete tuples, and **Update** (or **Modify**) is used to change the values of some attributes in existing tuples.

**1.Insert Operation**

Insert can violate any of the four types of constraints. Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type. Key constraints can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R). Entity integrity can be violated if any part of the primary key of the new tuple t is NULL. Referential integrity can be violated if the value of

any foreign key in t refers to a tuple that does not exist in the referenced relation.

**Option to deal with violation.**

If an insertion violates one or more constraints, the default option is to reject the insertion. Another option is to attempt to correct the reason for rejecting the insertion, but this is typically not used for violations caused by Insert.

**2.The Delete Operation**

The **Delete** operation can violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database. To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted.

**Option to deal with violation.**

Several options are available if a deletion operation causes a violation. The first option, called **restrict,** is to reject the deletion. The second option, called **cascade,** is to attempt to cascade (or propagate) the deletion by deleting tuples that reference the tuple that is being deleted. A third option, called **set null** or **set default,** is to modify the referencing attributevalues that cause the violation; each such value is either set to NULL or changed to reference another default valid tuple. Notice that if a referencing attribute that causes a violation is part of the primary key, it cannot be set to NULL; otherwise, it would violate entity integrity. Combinations of these three options also possible.

**3.The Update Operation**

Updating an attribute that is neither part of a primary key nor part of a foreign key usually causes no problems; the DBMS need only check to confirm that the newvalue is of the correct data type and domain. Modifying a primary key value is similarto deleting one tuple and inserting another in its place because we use the primarykey to identify tuples.So all violations that can happen in insert and delete operations apply in case of update also. If a foreign key attribute is modified, the DBMS must make sure that the new value refers to an existing tuple in the referencedrelation (or is set to NULL).

**Options to deal with violation**

Same as the options available if a insertion operation and deletion operation causes violation.

## Q4.Explain the basic constraints that can be specified in SQL as part of table creation with example.

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

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

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Uniquely identifies a row/record in another table
- CHECK - Ensures that all values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column when no value is specified
- INDEX - Used to create and retrieve data from the database very quickly
- UNIQUE -Used to create and retrieve data from the database very quickly

**NOT NULL Constraint**

NOT NULL constraint restricts a column from having a NULL value. Once NOT NULL constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.

One important point to note about this constraint is that it cannot be defined at table level.

Example using NOT NULL constraint

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

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

**UNIQUE Constraint**

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

Using UNIQUE constraint when creating a Table (Table Level)

Here we have a simple CREATE query to create a table, which will have a column s_idwith unique values.

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

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

Using UNIQUE constraint after Table is created (Column Level)

```
ALTER TABLE Student ADD UNIQUE(s_id);
```

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

**Primary Key Constraint**

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

Using PRIMARY KEY constraint at Table Level

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

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

Using PRIMARY KEY constraint at Column Level

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

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

**Foreign Key Constraint**

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see its use, with help of the below tables:

Customer_Detail Table

| c_id | Customer_Name | Address |
|---|---|---|
| 101 | Adam | Noida |
| 102 | Alex | Delhi |
| 103 | Stuart | Rohtak |

Order_Detail Table

| Order_id | Order_Name | c_id |
|---|---|---|
| 10 | Order1 | 101 |
| 11 | Order2 | 103 |

|  |  |  |
|---|---|---|
| 12 | Order3 | 102 |

In Customer_Detail table, c_id is the primary key which is set as foreign key in Order_Detail table. The value that is entered in c_id which is set as foreign key in Order_Detail table must be present in Customer_Detail table where it is set as primary key. This prevents invalid data to be inserted into c_id column of Order_Detail table.

If you try to insert any incorrect data, DBMS will return error and will not allow you to insert the data.

Using FOREIGN KEY constraint at Table Level

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

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

Using FOREIGN KEY constraint at Column Level

```
ALTER table Order_Detail ADD FOREIGN KEY (c_id) REFERENCES
Customer_Detail(c_id);
```

**CHECK Constraint**

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

Using CHECK constraint at Table Level

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

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

Using CHECK constraint at Column Level

# MODULE-3

**Question 1.**

**How trigger and assertion can be defined in SQL? Explain with example.**

Answer:

**TRIGGER**

Triggers are stored procedures for the actions to be performed on certain condition. It is defined in such a way that it will run automatically, before or after some events such as INSERT,UPDATE OR DELETE.

Syntax:

**Create trigger <trigger_name>**

**[before/after]**

**{insert/update/delete}**

**On [table_name]**

**[for each row]**

**[trigger-body]**

Example:

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
ON EMPLOYEE
FOR EACH ROW
WHEN ( NEW.SALARY> ( SELECT SALARY FROM EMPLOYEE
WHERE SSN = NEW.SUPERVISOR_SSN ) )
INFORM_SUPERVISOR(NEW.Supervisor_ssn,
NEW.Ssn );
```

A typical trigger which is regarded as an ECA (Event,Condition, Action) rule has three components:

1. The **event(s)**: These are usually database update operations that are explicitly applied to the database. In this example the events are: inserting a new employee record, changing an employee's salary, or changing an employee's supervisor . These events are specified after the keyword **BEFORE** in this example, which means that the triggers should be executed before the triggering operation is executed. An alternative is to use the keyword **AFTER**, which specifies that the trigger should be executed after the operation specified in the event is completed.
2. The **condition** that determines whether the rule action should be executed: Once the triggering event has occurred, an *optional* condition may be evaluated.If*no condition* is specified, the action will be executed once the event occurs. If a condition is specified, it is first evaluated, and only *if it evaluatesto true* will the rule action be executed. The condition is specified in the  WHEN clause of the trigger.
3. The **action** to be taken: The action is usually a sequence of SQL statements, but it could also be a database transaction or an external program that will be automatically executed.

Triggers can be used in various applications, such as maintaining database consistency, monitoring database updates, and updating derived data automatically.

**ASSERTION**

Assertion can be used to specify additional types of constraints, other than primary key, unique key, entity integrity and referential integrity. In SQL,users can specify general constraints via declarative assertions, using the CREATE ASSERTION statement.Assertions are database objects of their own right and are not defined within a create table or alter table statement.It is a condition or constaint which we wish the database to always satisfy. Assertion always check for the condition specified to be true.

Syntax:

**Create assertion <assertion_name> CHECK <condition>**

Example:

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS ( SELECT *
FROM EMPLOYEE E, EMPLOYEE M,
DEPARTMENT D
WHERE E.Salary>M.Salary
AND E.Dno = D.Dnumber
AND D.Mgr_ssn = M.Ssn ) );
```

The constraint name(assertion name) SALARY_CONSTRAINT is followed by the keyword CHECK, which is followed by a **condition** in parentheses that must hold true on every database state for the assertion to be satisfied. The constraint name can be used later to disable the constraint or to modify or drop it. Any WHERE clause condition can be used, but many constraints can be specified using the EXISTS and NOT EXISTS style of SQL conditions. Whenever some tuples in the database cause the condition of an ASSERTION statement to evaluate to FALSE, the constraint is **violated**. The constraint is **satisfied** by a database state if *no combination of tuples* in that database state violates the constraint.

**Question 2:**

**What is a view? Explain the implementation of views.**

A view in SQL terminology is a single table that is derived from other tables. it is considered to be a virtual table. in contrast to base tables, whose tuples are always physically stored in the database. This limits thepossible update operations that can be applied to views, but it does not provide anylimitations on querying a view. the command to specify a view is CREATE VIEW. The view is given a (virtual)table name (or view name), a list of attribute names, and a query to specify thecontents of the view.
Example:

```
CREATE VIEW WORKS_ON1
AS SELECT Fname, Lname, Pname, Hours
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE Ssn = EssnAND Pno = Pnumber;
```

We can specify SQL queries on a view—or virtual table—in the same way we specify queries involving base tables. For example, to retrieve the last name and first name of all employees who work on the 'ProductX' project, we can utilize the WORKS_ON1 view and specify the query.

```
SELECT Fname, Lname
FROM WORKS_ON1;
```

**Question3:**

**Write note on Aggregate function in SQL with example.**

Aggregate Functions are all about

- Performing calculations on multiple rows
- Of a single column of a table
- And returning a single value.

five (5) main aggregate functions are:

1)COUNT
2)SUM

3)AVG
4)MIN
5) MAX

**COUNT Function**

The COUNT function returns the total number of values in the specified field. It works on both numeric and non-numeric data types .COUNT (*) is a special implementation of the COUNT function that returns the count of all the rows in a specified table. COUNT (*) also considers Nulls and duplicates.

MIN function

The MIN function returns the smallest value in the specified table field.

MAX function

Just as the name suggests, the MAX function is the opposite of the MIN function. It returns the largest value from the specified table field.

SUM function

Suppose we want a report that gives total amount of payments made so far. We can use the MySQL SUM function which returns the sum of all the values in the specified column. SUM works on numeric fields only. Null values are excluded from the result returned.

AVG function

MySQL AVG function returns the average of the values in a specified column. Just like the SUM function, it works only on numeric data types.

**Question 4:**

**Explain nested query and co-related nested query with example**.

**Nested Query**

Nested queries, are complete select-from-where blocks within another SQL. That other query is called the outer query. These nested queries can also appear in the WHERE clause or the FROM clause or the SELECT clause or other SQL clauses as needed. The comparison
operator IN, which compares a value *v* with a set (or multiset) of values *V* and evaluates to TRUE if *v* is one of the elements in *V*.

Example:

```
SELECT DISTINCT Essn
FROM WORKS_ON
WHERE (Pno, Hours) IN ( SELECTPno, Hours
FROM WORKS_ON
WHERE Essn = '123456789' );
```

This query will select the Essns of all employees who work the same (project, hours) combination on some project that employee 'John Smith' (whose Ssn = '123456789') works on. In this example, the IN operator compares the subtuple of values in parentheses (Pno, Hours) within each tuple in WORKS_ON with the set of type-compatible tuples produced by the nested query. In general, we can have several levels of nested queries. We can once again be faced with possible ambiguity among attribute names if attributes of the same name exist—one in a relation in the FROM clause of the *outer query,* and another in a relation in the FROM clause of the *nested query.* The rule is that a reference to an *unqualified attribute* refers to the relation declared in the innermost nested query.

### Correlated Nested Queries

Whenever a condition in the WHERE clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be correlated.We can understand a correlated query better by considering that the *nested query isevaluated once for each tuple (or combination of tuples) in the outer query.*

```
ELECT E.Fname, E.Lname
FROM EMPLOYEE AS E
WHERE E.SsnIN ( SELECT D.Essn
FROM DEPENDENT AS D
WHERE E.Fname = D.Dependent_name
AND E.Sex = D.Sex );
```

For *each* EMPLOYEE tuple, evaluate the nested query, which retrieves the Essn values for all DEPENDENT tuples with the same sex and name as that EMPLOYEE tuple; if the Ssn value of the EMPLOYEE tuple is *in* the result of the nested query, then select that EMPLOYEE tuple. In general, a query written with nested select-from-where blocks and using the = or IN comparison operators can *always* be expressed as a single block query.

**Question 5:**
**Explain EXISTS and NOT EXISTS operators in SQL.**

EXISTS and NOT EXISTS are Boolean functions that return TRUE or FALSE; hence, they can be used in a WHERE clause condition. The EXISTS function in SQL is used to check whether the result of a nested query is *empty* (contains no tuples) or not. The result of EXISTS is a Boolean value TRUE if the nested query result contains at least one tuple, or FALSE if the nested query result contains no tuples.

Example for EXISTS:

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE AS E
WHERE EXISTS ( SELECT*
FROM DEPENDENT AS D
WHERE E.Ssn = D.EssnAND E.Sex = D.Sex
AND E.Fname = D.Dependent_name);
```

EXISTS and NOT EXISTS are typically used in conjunction with a *correlated* nested Query. The nested query references the Ssn, Fname, and Sex attributes of the EMPLOYEE relation from the outer query. For each EMPLOYEE tuple, evaluate the nested query, which retrieves all DEPENDENT tuples with the same Essn, Sex, and Dependent_name as the EMPLOYEE tuple; if at least one tuple EXISTS in the result of the nested query, then select that EMPLOYEE tuple. EXISTS(Q) returns TRUE if there is *at least one tuple* in the result of the nested query Q, and returns FALSE otherwise. On the other hand, NOT EXISTS(Q) returns TRUE if there are *no tuples* in the result of nested query Q, and returns FALSE otherwise.

Example for NOT EXISTS:

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE NOT EXISTS ( SELECT*
FROM DEPENDENT
WHERE Ssn = Essn );
```

The correlated nested query retrieves all DEPENDENT tuples related to a particular EMPLOYEE tuple. If *none exist,* the EMPLOYEE tuple is selected because the WHERE-clause condition will evaluate to TRUE in this case. For *each* EMPLOYEE tuple, the correlated nested query selects all DEPENDENT tuples whose Essn value matches the EMPLOYEE Ssn; if the result is empty, no dependents are related to the employee, so we select that EMPLOYEE tuple and retrieve its Fname and Lname

**Question6:**
**Draw and explain 3 tier architecture.**

A 3-tier architecture is a type of software architecture which is composed of three “tiers” or “layers” of logical computing. They are often used in applications as a specific type of client-server system.

- **Presentation Tier-** The presentation tier is the front end layer in the 3-tier system and consists of the user interface. This user interface is often a graphical one accessible through a web browser or web-based application and which displays content and information useful to an end user. This tier is often built on web technologies such as HTML5, JavaScript, CSS, or through other popular web development frameworks, and communicates with others layers through API calls.
- **Application Tier-** The application tier contains the functional business logic which drives an application’s core capabilities. It’s often written in Java, .NET, C#, Python, C++, etc.
- **Data Tier-** The data tier comprises of the database/data storage system and data access layer. Examples of such systems are MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB, etc. Data is accessed by the application layer via API calls.

**The main benefits of the 3-tier architectural style are:**

- **Maintainability.** Because each tier is independent of the other tiers, updates or changes can be carried out without affecting the application as a whole.
- **Scalability.** Because tiers are based on the deployment of layers, scaling out an application is reasonably straightforward.
- **Flexibility.** Because each tier can be managed or scaled independently, flexibility is increased.
- **Availability.** Applications can exploit the modular architecture of enabling systems using easily scalable components, which increases availability.

# MODULE-4

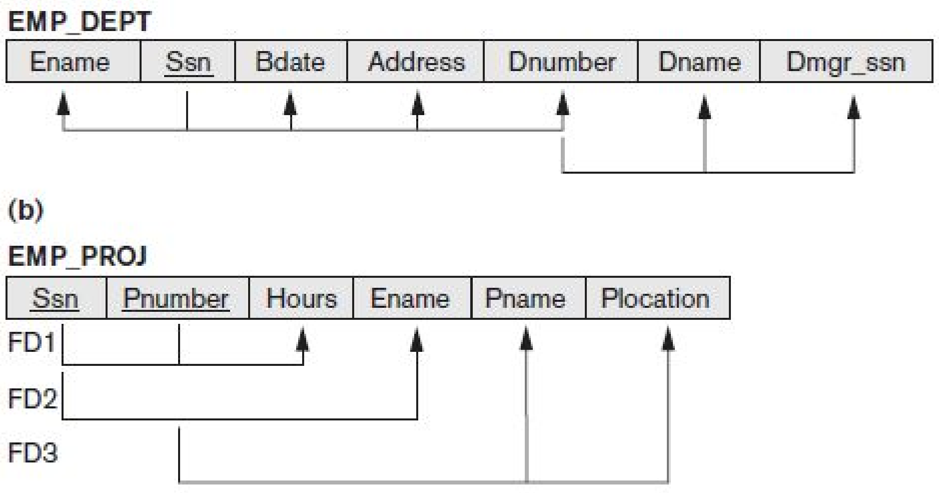
**QUESTION 1.**
**What is a functional dependency?**

A functional dependency is a constraint between two sets of attributes from the
Database.

**Definition.** A **functional dependency**, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t1 and t2 in r that have $t1[X] = t2[X]$, they must also have $t1[Y] = t2[Y]$.

The values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; alternatively, the values of the X componentof a tuple uniquely (or functionally) determine the values of the Y component. The abbreviation for functional dependency is FD or f.d.The set ofattributes X is called the left-hand side of the FD, and Y is called the right-hand side. A functional dependency is a property of the semantics or meaning of theattributes.

**Diagrammatic notation** for displaying FDs: Each FD is displayed as a horizontal line. The left-hand-side attributes of the FD are connected by vertical lines to the line representing the FD, whereas the right-hand-side attributes are connected by the lines with arrows pointing toward the attributes.



.

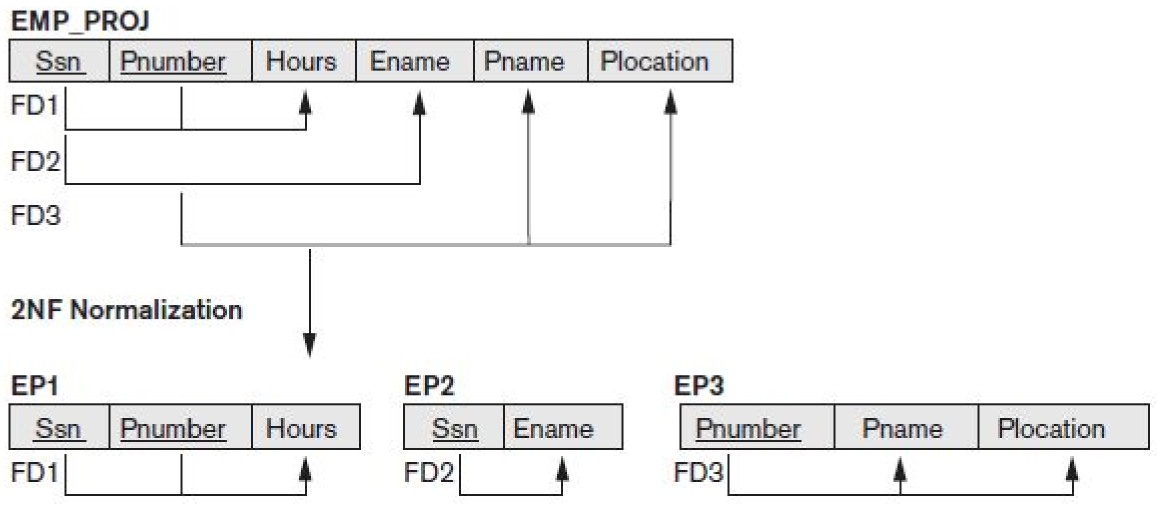
**QUESTION 2.**
**Which normal form is based on the concept of full functional dependency?Explain the same with example.**

2NF (Second Normal Form)

Second normal form (2NF) is based on the concept of *full functional dependency*. A functional dependency *X* → *Y* is a full functional dependency if removal of anyattribute *A* from *X* means that the dependency does not hold anymore.

Definition. A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains a single attribute, the test need not be applied at all. If a relation schema is not in 2NF, it can be second normalized or 2NF normalized into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent.



**QUESTION 3.**
**Explain informal design guidelines for relation schemas.**

Answer:

Four *informal guidelines* that may be used as *measures to determine the quality* of relation schema design:
a.Making sure that the semantics of the attributes is clear in the schema
b.Reducing the redundant information in tuples
c. Reducing the NULL values in tuples
d.Disallowing the possibility of generating spurious tuples.

**(a)Imparting Clear Semantics to Attributes in Relations**
The semantics of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple.While designing a schema, assign simple but meaningful name for the schema and attributes so that schema can be easily explained.Semantics of attributes should be easy to interpret.Name should convey maximum meaning as possible.

**Guideline 1.**Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation. Intuitively, if a relation schema corresponds to one entity type or one relationship type, it is straightforward to explain its meaning.

**(b)Redundant Information in Tuples and Update Anomalies**
When information is stored redundantly,
 -Wastes storage space
-Causes problems with update anomalies(insertion ,deletion and updation anomalies
)
In EMP_DEPT, the attribute values pertaining to a particular department (Dnumber, Dname, Dmgr_ssn) are repeated for *every employee who works for that department.* Storing natural

joins of base relations leads to an additional problem referred to as update anomalies. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies. Insertion Anomalies. Insertion anomalies can be differentiated into two types,

a.To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs (if the employee does not work for a department as yet). For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are *consistent* with the corresponding values for department 5 in other tuples in EMP_DEPT

b. It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes for employee. This violates the entity integrity for EMP_DEPT because its primary key Ssn cannot be null.

**Deletion Anomalies**

If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost inadvertently from the database.

**Modification Anomalies**

In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of *all* employees who work in that department.

**Guideline 2**. Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present,4 note them clearly and make sure that the programs that update the database will operate correctly.

**(c)NULL Values in Tuples**

If many of the attributes do not apply to all tuples in the relation, we end upwith many NULLs in those tuples.NULLs can have multiple interpretations,such as the following:

■ The attribute *does not apply* to this tuple.
■ The attribute value for this tuple is *unknown*.
■ The value is *known but absent*; that is, it has not been recorded yet.

**Guideline 3**. As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

**(d)Generation of Spurious Tuples**

**Guideline 4**. Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated. Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

**QUESTION 4.**

**What is normalization?Explain 1NF .**

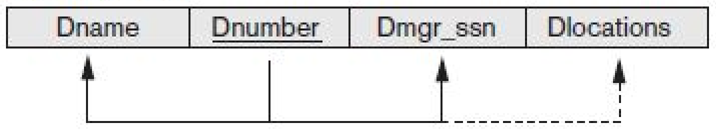
Answer:

**Normalization of data** can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of (1) minimizing redundancy and (2) minimizing the insertion, deletion, and update anomalies

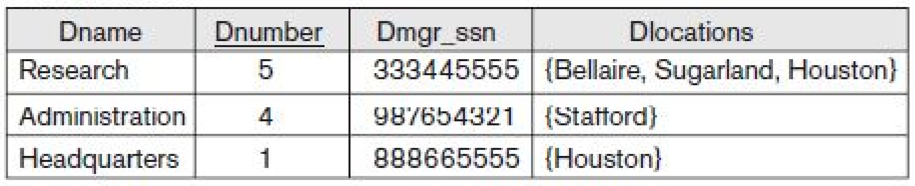
## First Normal Form

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

(a)
DEPARTMENT

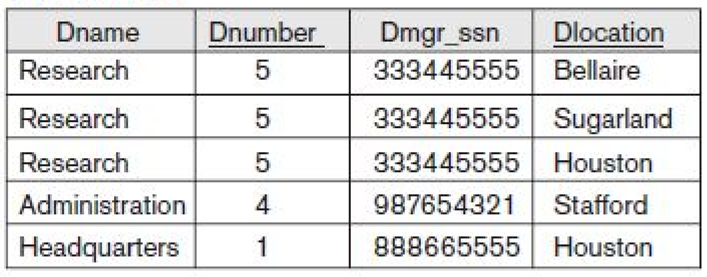


(b)
DEPARTMENT



| Dname | Dnumber | Dmgr_ssn | Dlocations |
|---|---|---|---|
| Research | 5 | 333445555 | {Bellaire, Sugarland, Houston} |
| Administration | 4 | 987654321 | {Stafford} |
| Headquarters | 1 | 888665555 | {Houston} |

(c)
DEPARTMENT

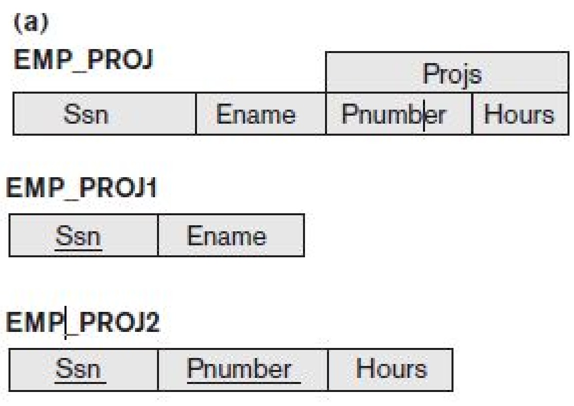


| Dname | Dnumber | Dmgr_ssn | Dlocation |
|---|---|---|---|
| Research | 5 | 333445555 | Bellaire |
| Research | 5 | 333445555 | Sugarland |
| Research | 5 | 333445555 | Houston |
| Administration | 4 | 987654321 | Stafford |
| Headquarters | 1 | 888665555 | Houston |

There are three main techniques to achieve first normal form for such a relation:

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

Figure 14.2. A distinct tuple in DEPT_LOCATIONS exists for *each location* of a department. This decomposes the non-1NF relation into two 1NF relations.
2. Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT, as shown in Figure .In this case, the primary key becomes the combination {Dnumber, Dlocation}. This solution has the disadvantage of introducing *redundancy* in the relation and hence is rarely adopted.
3. If a *maximum number of values* is known for the attribute—for example, if it is known that *at most three locations* can exist for a department—replace the Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3. This solution has the disadvantage of introducing *NULL values* if most departments have fewer than three locations.
The schema of this EMP_PROJ relation can be represented as follows: EMP_PROJ(Ssn, Ename, {PROJS(Pnumber, Hours)})



The existence of more than one multivalued attribute in one relation must be handled carefully. As an example, consider the following non-1NF relation:
PERSON (Ss#, {Car_lic#}, {Phone#})

The right way to deal with the two multivalued attributes in PERSON is to decompose it into two separate relations.

P1(Ss#, Car_lic#) and P2(Ss#, Phone#).