DATABASE MANAGEMENT SYSTEM [DBMS]

PLACEMENT PREPARATION [EXCLUSIVE NOTES]

SAVE AND SHARE

Curated By- HIMANSHU KUMAR(LINKEDIN)

https://www.linkedin.com/in/himanshukumarmahuri

TOPICS COVERED: -

- **Introduction and Advantages**
- **Architectures**
- **ER - Model**
- **Relational Model**
- **Keys in Relational Model**
- **Database Normalization Introduction**
- **Normal Forms**
- **Concurrency Control - Introduction and ACID Properties**
- **Indexing in Database**

Terminologies

Database: Database is a collection of inter-related data which helps in efficient retrieval, insertion and deletion of data from database and organizes the data in the form of tables, views, schemas, reports etc. For Example, university database organizes the data about students, faculty, and admin staff etc. which helps in efficient retrieval, insertion and deletion of data from it.

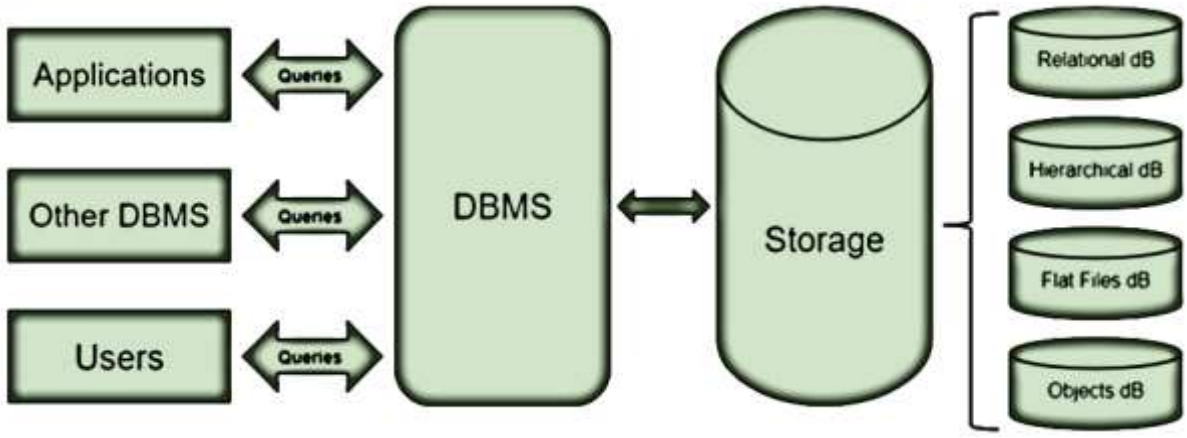
Database Management System: The software which is used to manage the database is called Database Management System (DBMS). For Example, MySQL, Oracle, etc. are popular commercial DBMS used in different applications. DBMS allows users the following tasks:

Data Definition: It helps in creation, modification, and removal of definitions that define the organization of data in the database.

Data Updation: It helps in insertion, modification and deletion of the actual data in the database.

Data Retrieval: It helps in retrieval of data from the database which can be used by applications for various purposes.

User Administration: It helps in registering and monitoring users, enforcing data security, monitoring performance, maintaining data integrity, dealing with concurrency control and recovering information corrupted by unexpected failure.



The history of DBMS evolved in three primary phases:

1. File-based System
2. Relational DBMS
3. NoSQL

Paradigm Shift from File System to DBMS

A File Management system is a DBMS that allows access to single files or tables at a time. In a File System, data is directly stored in a set of files. It contains flat files that have no relation to other files (when only one table is stored in a single file, then this file is known as flat file). File System manages data using files in the hard disk. Users are allowed to create, delete, and update the files according to their requirements. Let us consider the example of a file-based University Management System. Data of students is available to their respective Departments, Academics Section, Result Section, Accounts Section, Hostel Office, etc. Some of the data is common for all sections like Roll No, Name, Father Name, Address and Phone number of students but some data is available to a particular section only like Hostel allotment number which is a part of hostel office. Let us discuss the issues with this system:

- **Redundancy of data:** Data is said to be redundant if same data is copied at many places. If a student wants to change a Phone number, he has to get it updated at various sections. Similarly, old records must be deleted from all sections representing that student.

- **Inconsistency of Data:** Data is said to be inconsistent if multiple copies of the same data do not match with each other. If a Phone number is different in the Accounts Section and Academics Section, it will be inconsistent. Inconsistency may be because of typing errors or not updating all copies of same data.

- **Difficult Data Access:** A user should know the exact location of the file to access data, so the process is very cumbersome and tedious. If a user wants to search student hostel allotment number of a student from 10000 unsorted students' records, how difficult it can be.
- **Unauthorized Access:** File System may lead to unauthorized access to data. If a student gets access to file having his marks, he can change it in unauthorized way.
- **No Concurrent Access:** The access of same data by multiple users at same time is known as concurrency. File system does not allow concurrency as data can be accessed by only one user at a time.
- **No Backup and Recovery:** File system does not incorporate any backup and recovery of data if a file is lost or corrupted.

These are the main reasons which made a shift from file system to Relational DBMS.

Relational DBMS

Relational database means the data is stored as well as retrieved in the form of relations (tables). The table below shows the relational database with only one relation called **STUDENT** which stores **ROLL_NO**, **NAME**, **ADDRESS**, **PHONE** and **AGE** of students.

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

These are some important terminologies that are used in terms of relationships that we will learn later. The relational DBMS provides us with Structured Query Language or SQL which is a standard

Database language that is used to create, maintain and retrieve the relational database. Following are some interesting facts about SQL.

- SQL is case insensitive. But it is a recommended practice to use keywords (like SELECT, UPDATE, CREATE, etc) in capital letters and use user-defined things (liked table name, column name, etc) in small letters.

- We can write comments in SQL using "–" (double hyphen) at the beginning of any line.

- SQL is the programming language for relational databases (explained below) like MySQL, Oracle, Sybase, SQL Server, Postgre, etc. Other non-relational databases (also called NoSQL) databases like MongoDB, DynamoDB, etc do not use SQL

- Although there is an ISO standard for SQL, most of the implementations slightly vary in syntax. So we may encounter queries that work in SQL Server but do not work in MySQL.

NoSQL

A NoSQL originally referring to non-SQL or non-relational is a database that provides a mechanism for storage and retrieval of data. This data is modelled in means other than the tabular relations used in relational databases. NoSQL databases are used in real-time web applications and big data and their use are increasing over time. NoSQL systems are also sometimes called Not only SQL to emphasize the fact that they may support SQL-like query languages. A NoSQL database includes simplicity of design, simpler horizontal scaling to clusters of machines and finer control over availability. The

data structures used by NoSQL databases are different from those used by default in relational databases which makes some operations faster in NoSQL. The suitability of a given NoSQL database depends on the problem it should solve. Data structures used by NoSQL databases are sometimes also viewed as more flexible than relational database tables. Types of NoSQL databases and the name of the databases system that falls in that category are:

1. MongoDB falls in the category of NoSQL document-based database.

2. Key value store: Memcached, Redis, Coherence

3. Tabular: Hbase, Big Table, Accumulo

4. Document based: MongoDB, CouchDB, Cloudant

Advantages of DBMS

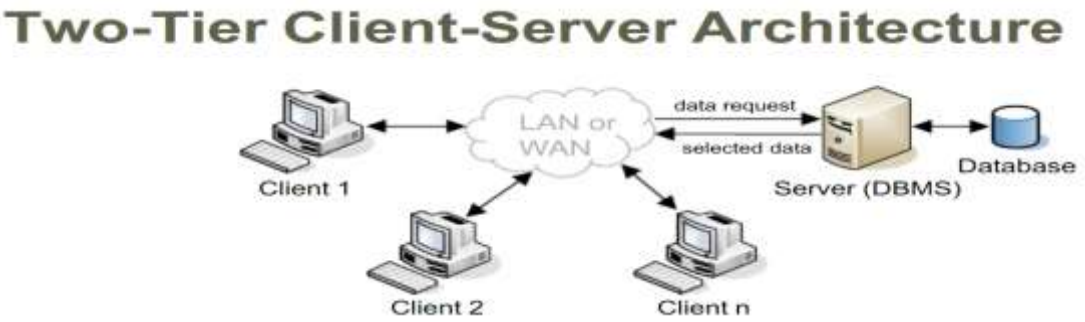
DBMS helps in efficient organization of data in database which has following advantages over typical file system.

- **Minimized redundancy and data consistency:** Data is normalized in DBMS to minimize the redundancy which helps in keeping data consistent. For Example, student information can be kept at one place in DBMS and accessed by different users.
- **Simplified Data Access:** A user need only name of the relation not exact location to access data, so the process is very simple.

- **Multiple data views:** Different views of same data can be created to cater the needs of different users. For Example, faculty salary information can be hidden from student view of data but shown in admin view.
- **Data Security:** Only authorized users are allowed to access the data in DBMS. Also, data can be encrypted by DBMS which makes it secure.
- **Concurrent access to data:** Data can be accessed concurrently by different users at same time in DBMS.
- **Backup and Recovery mechanism:** DBMS backup and recovery mechanism helps to avoid data loss and data inconsistency in case of catastrophic failures.

Two Level Architecture

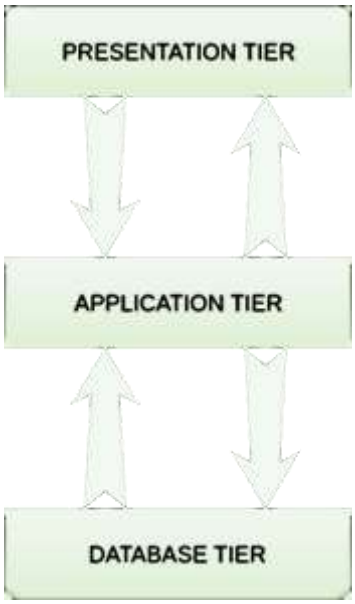
Two level architecture is similar to a basic **client-server** model. The application at the client end directly communicates with the database at the server side. API's like ODBC,JDBC are used for this interaction. The server side is responsible for providing query processing and transaction management functionalities. On the client side, the user interfaces and application programs are run. The application on the client side establishes a connection with the server side in order to communicate with the DBMS.



An **advantage** of this type is that maintenance and understanding are easier, compatible with existing systems. However, this model gives poor performance when there are a large number of users.

DBMS 3-tier Architecture

DBMS 3-tier architecture divides the complete system into three inter-related but independent modules as shown in Figure below.



Presentation or User Tier: This tier is presented before the user of who knows nothing regarding the other existing complex databases underneath. This tier can provide multiple views of the databases, generated by the application residing in the application tier.

Application Tier: This is the middle lined tier between the Database and the Presentation tier and acts as a connection between the end-user and the database. This tier holds the programs and the application server along with the programs that could access the database. This is the last layer that the users are made aware of. It provides a complete abstraction to the database layer.

Database Tier: This tier holds the primary database along with all the data and the query languages for processing. The relations of data with there constraints are also defined in this level.

Advantages:

- **Enhanced scalability** due to distributed deployment of application servers. Now,individual connections need not be made between client and server.
- **Data Integrity** is maintained. Since there is a middle layer between client and server, data corruption can be avoided/removed.
- **Security** is improved. This type of model prevents direct interaction of the client with the server thereby reducing access to unauthorized data.

Disadvantages:
Increased complexity of implementation and communication. It becomes difficult for this sort of interaction to take place due to presence of middle layers.

Data Independence

Data independence means a change of data at one level should not affect another level. Two types of data independence are present in this architecture:

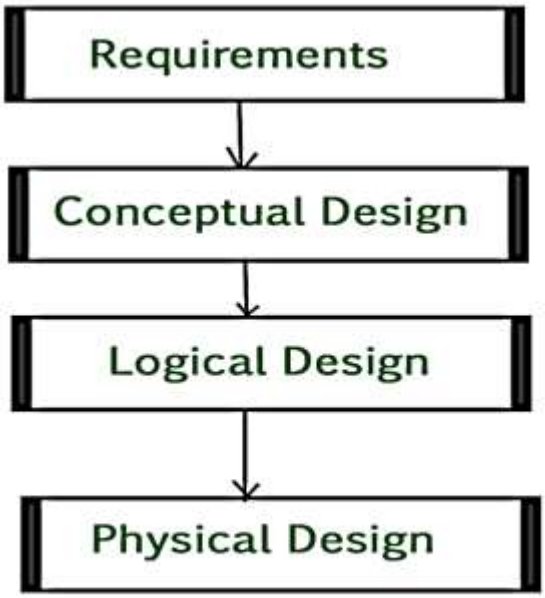
Physical Data Independence: Any change in the physical location of tables and indexes should not affect the conceptual level or external view of data. This data independence is easy to achieve and implemented by most of the DBMS.

Conceptual Data Independence: The data at conceptual level schema and external level schema must be independent. This means, change in conceptual schema should not affect external schema. e.g.; Adding or deleting attributes of a table should not affect the user's view of table. But this type of independence is difficult to achieve as compared to physical data independence because the changes in

conceptual schema are reflected in user's view.

Phases of database design

Database designing for a real world application starts from capturing the requirements to physical implementation using DBMS software which consists of following steps shown in Figure.



Conceptual Design: The requirements of database are captured using high level conceptual data model. For Example, ER model is used for conceptual design of database.

Logical Design: Logical Design represents data in the form of relational model. ER diagram produced in the conceptual design phase is used to convert the data into the Relational Model.

Physical Design: In physical design, data in relational model is implemented using commercial DBMS like Oracle, DB2.

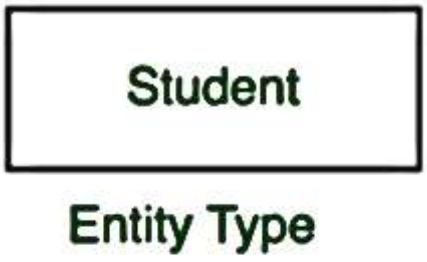
ER-MODEL

ER Model is used to model the logical view of the system from the data perspective which consists of these components:

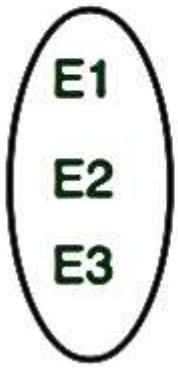
Entity, Entity Type, Entity Set

An **Entity** may be an object with a physical existence - a particular person, car, house, or employee - or it may be an object with a conceptual existence - a company, a job, or a university course.

An Entity is an object of **Entity Type** and set of all entities is called as **Entity Set**. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set. In ER diagram, Entity Type is represented as:



Entity Type

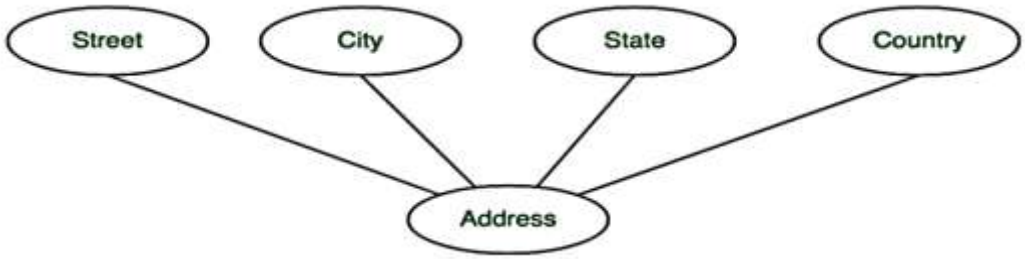


Entity Set

Attribute(s) Attributes are the **properties which define the entity type**. For example, Roll_No, Name, DOB, Age, Address, Mobile_No are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.

1. **Key Attribute -** The attribute which **uniquely identifies each entity** in the entity set is called key attribute.For example, Roll_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

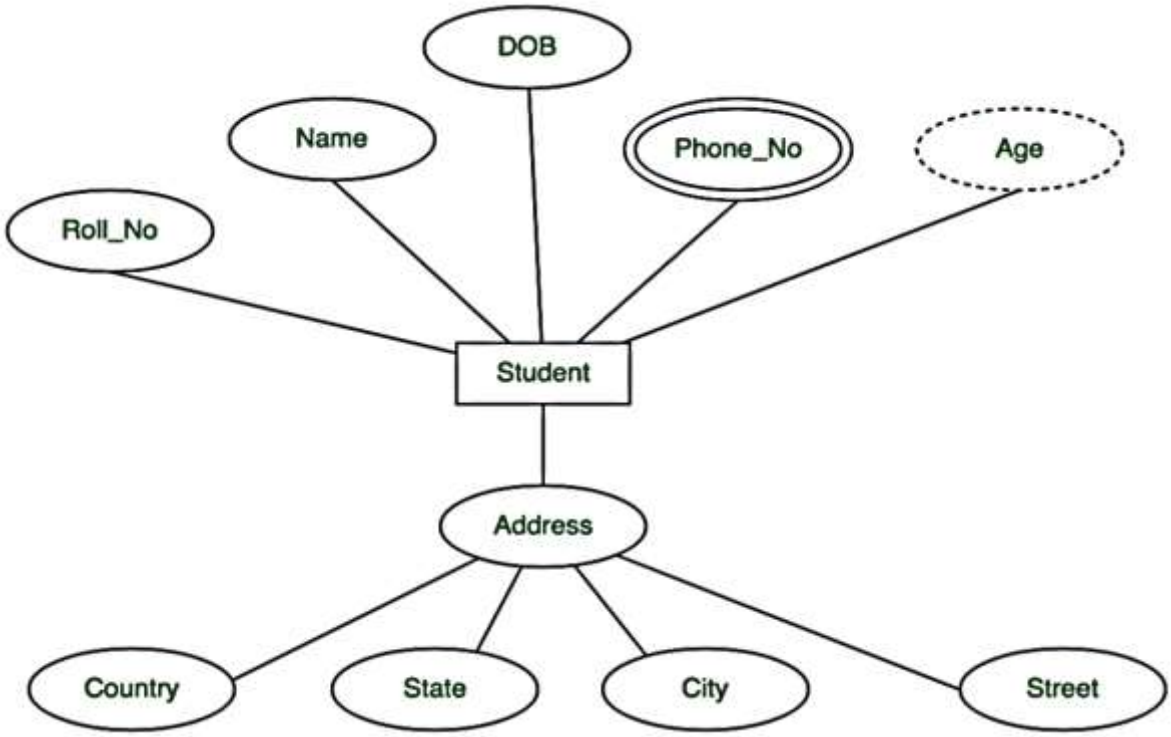
2. **Composite Attribute -** An attribute **composed of many other attribute** is called as composite attribute. For example, Address attribute of student Entity type consists of Street, City, State, and Country. In ER diagram, composite attribute is represented by an oval comprising of ovals.



3. **Multivalued Attribute -** An attribute consisting **more than one value** for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, multivalued attribute is represented by double oval.

4. **Derived Attribute -** An attribute which can be **derived from other attributes** of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, derived attribute is represented by dashed oval.

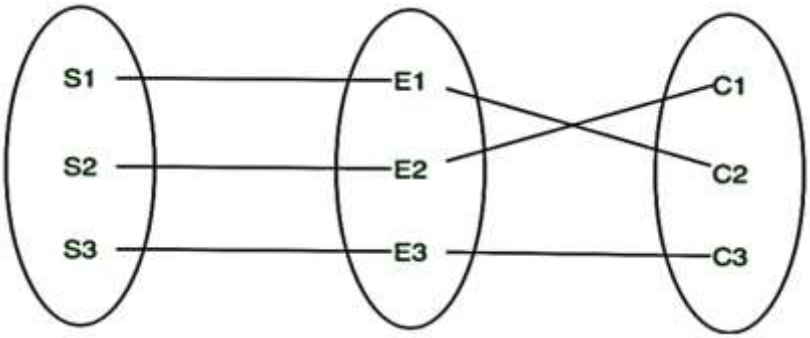
5. The complete entity type **Student** with its attributes can be represented as:



Relationship Type and Relationship Set

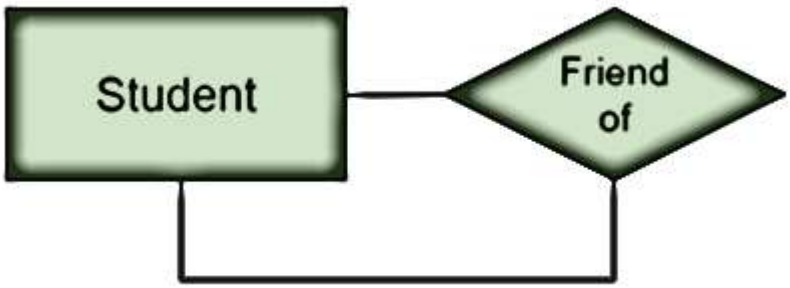
A relationship type represents the **association between entity types**. For example,'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.

A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.

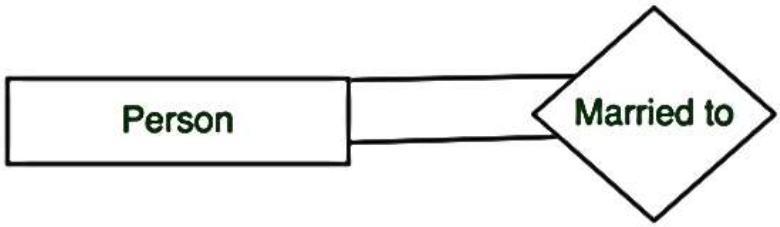


Degree of a relationship set: The number of different entity sets **participating in a relationship** set is called as degree of a relationship set.

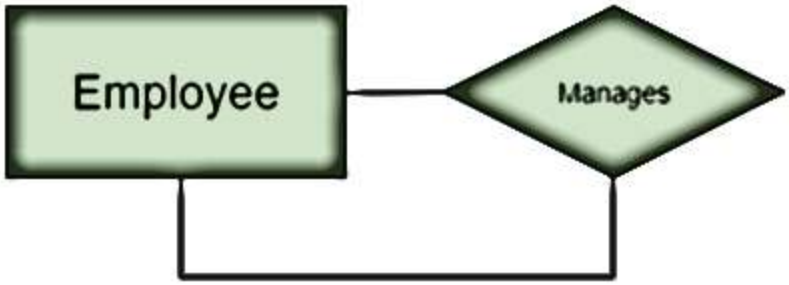
1. **Unary Relationship -** When there is **only ONE entity set participating in a relation**, the relationship is called as unary relationship.
   Example 1: A student is a friend of itself.



   Example 2: A person is married to a person.



Example 3: An employee manages an employee.



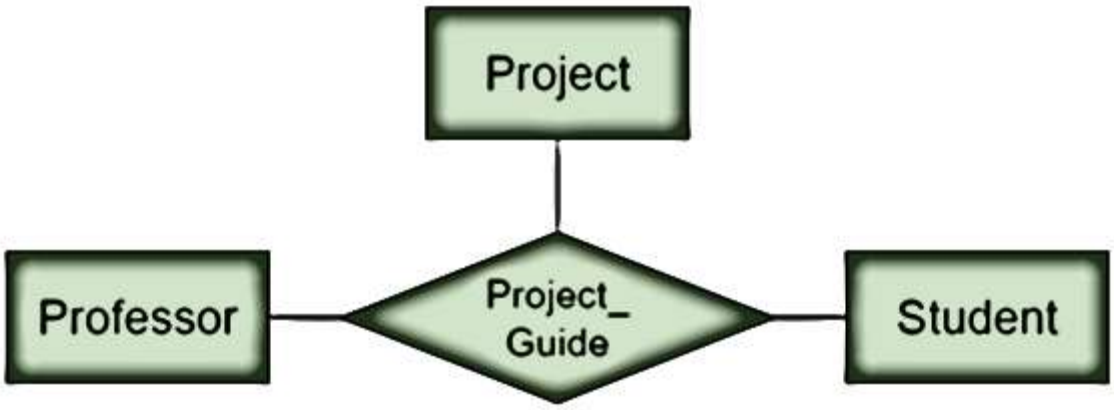
2. **Binary Relationship -** When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship.For example, Student is enrolled in Course.

Example 1: A student attends a course.

Example 2: A supplier supplies item.

Example 3: A Profssor teaches subject.

3. **n-ary Relationship -** When there are n entities set participating in a relation, the relationship is called as n-ary relationship.
   Example: A Professor, student and Project is related to a Project_Guide.



Cardinality

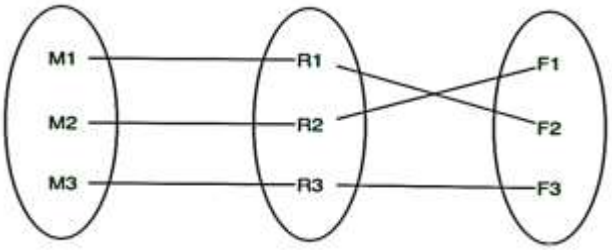
The **number of times an entity of an entity set participates in a relationship** set is known as cardinality. Cardinality can be of different types:

1. **One to One -** When each entity in each entity set can take part **only once in the relationship**, the cardinality is one to one.
   Example 1: Let us assume that a driver can drive one carand a cat can be driven by the same driver. So the relationship will be one to one.

   Example 2: A person can have only one Aadhar card and one Aadhar card can belong to only one person.

Example 3: Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.

Using Sets, it can be represented as:

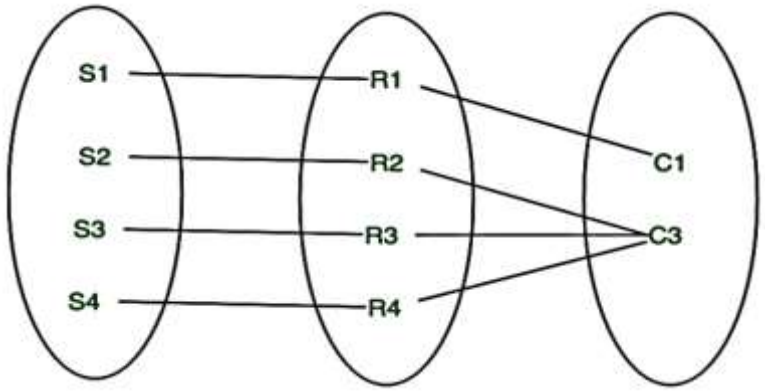


2. **One to Many -** When entities in one entity set **can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set,** cardinalities is one to many. Many to one also come under this category.
Example 1: A professor teaching many courses

Example 2: Many employees working for one department.

Example 3: Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.

Using Sets, it can be represented as:



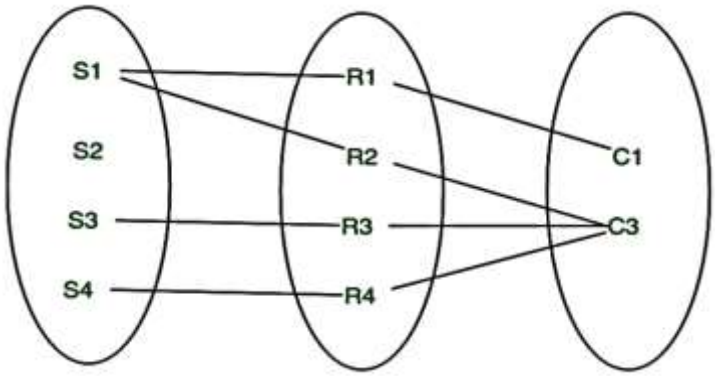
In this case, each student is taking only 1 course but 1 course has been taken by many students.

3. **Many to many -** When entities in all entity sets can **take part more than once in the relationship** cardinality is many to many.
   Example 1: Any number of student can take any number of subjects.

Example 2: Any number of customer can order any number of products.

Example 3: Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

Using sets, it can be represented as:



In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3, and S4. So it is many to many relationships.

Participation Constraint: Participation Constraint is applied on the entity participating in the relationship set.

1. **Total Participation -** Each entity in the entity set **must participate** in the relationship.

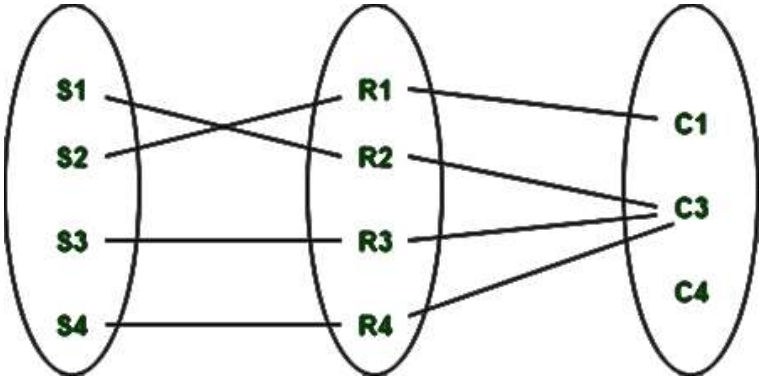
   Example 1: If each student must attend a course, the participation of student will be total. Total participation is shown by double line in ER diagram.

   Example 2: Each employee must join a department.

2. **Partial Participation -** The entity in the entity set **may or may NOT participat**e in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.

   The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.

   Using set, it can be represented as,



   Every student in Student Entity set is participating in a relationship but there exists a course C4 which is not taking part in the relationship.

Weak Entity Type and Identifying Relationship

As discussed before, an entity type has a key attribute that uniquely identifies each entity in the entity set. But there exists **some entity type for which key attribute can't be defined**. These are called the Weak Entity type. A weak entity type is represented by a double

rectangle. The participation of a weak entity type is always total. The relationship between a weak entity type and its identifying strong entity type is called an identifying relationship and it is represented by a double diamond.

Example 1: a school might have multiple classes and each class might have multiple sections. The section cannot be identified uniquely and hence they do not have any primary key. Though a class can be identified uniquely and the combination of a class and section is required to identify each section uniquely. Therefore the section is a weak entity and it shares total participation with the class.

Example 2: A company may store the information of dependants (Parents, Children, Spouse) of an Employee. But the dependents don't have existed without the employee. So Dependent will be a weak entity type and the Employee will be Identifying Entity type for Dependant.

Example 3: In case of multiple hosts, the login id cannot become primary key as it is dependent upon the hosts for its identity.

RELATIONAL MODEL

KEYS IN RELATIONAL MODEL

We have considered a Student and Course Relational Model for our Reference Examples.

Example Student Course Relational Model

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

Candidate Key: The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For **Example**, STUD_NO in STUDENT relation.

- The value of Candidate Key is unique and non-null for every tuple.
- There can be more than one candidate key in a relation. For Example, STUD_NO as well as STUD_PHONE both are candidate keys for relation STUDENT.
- The candidate key can be simple (having only one attribute) or composite as well. For **Example**, {STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.

Note - In Sql Server a unique constraint that has a nullable column, **allows** the value '**null**' in that column **only once**. That's why STUD_PHONE attribute as candidate here, but can not be 'null' values in primary key attribute.

Super Key: The set of attributes that can uniquely identify a tuple is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME), etc.

- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a superkey but vice versa is not true.

Primary Key: There can be more than one candidate key in relation out of which one can be chosen as the primary key. For **Example**, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

Alternate Key: The candidate key other than the primary key is called an alternate key. For **Example**, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_PHONE will be alternate key (only one out of many candidate keys).

Foreign Key: If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute and the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute. A referenced attribute of the referenced relation should be the primary key for it. For **Example**, STUD_NO in STUDENT_COURSE is a foreign key to STUD_NO in STUDENT relation.

It may be worth noting that unlike, Primary Key of any given relation, Foreign Key can be NULL as well as may contain duplicate tuples i.e. it need not follow uniqueness constraint.

For **Example**, STUD_NO in STUDENT_COURSE relation is not unique. It has been repeated for the first and third tuple. However, the STUD_NO in STUDENT relation is a primary key and it needs to be always unique and it cannot be null.

NORMAL FORMS

Normalization is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updating anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

1. First Normal Form

If a relation contains a composite or multi-valued attribute, it violates first normal form or relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

Let's understand the concept of 1NF using this exercise:

Customer_ID	Name	Mob No
101	ABC	8860033933, 9899868189
102	BCD	8960133933
103	XYZ	8681899900, 9681888800
104	PQR	8189399888

As we can see that the above table contains multi-valued attributes, which violates the principle of first normal form and hence must be reduced. There are various methods of doing this:

Method 1: This involves the creation of multiple columns, distributing the values alongside the new attributes. We can distribute the multivalued attributes to new columns by naming them as Mob No 1, Mob No 2, Mob No 3 etc, depending on the number of multivalued attributes till the table gets reduced to single-valued attribute.

Customer_ID	Name	Mob No 1	Mobile No 2	Mobile No 3
101	ABC	8860033933	9899868189	
102	BCD	8960133933	-	
103	XYZ	8681899900	9681888800	
104	PQR	8189399888	-	

But this method got few problems like various fields are needed to be left blank, resulting in wastage of space.

Method 2: This method involves the creation of multiple instead of columns, with copying up of the non-repeated attribute values for each repeated attribute values.

Customer_ID	Name	Mob No
101	ABC	8860033933
102	BCD	8960133933
103	XYZ	8681899900
104	PQR	8189399888
101	ABC	9899868189
103	XYZ	9681888800

In this table we have made a copy of the values of the repeated

attributes, keeping other attributes the same. This saves our space but introduces another problem of repetition of Customer_ID, which must remain unique.

Method 3: This method involves the creation of another table and shifting the repeated attributes to that table and linking it with the previous table by using any type of ID.

Customer_ID	Name
101	ABC
102	BCD
103	XYZ
104	PQR

ID	Customer_ID	Mob No
1	101	8860033933
2	102	.
.	.	.
.	.	.

- **Example 1 -** Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

- **Example 2 -**

ID	Name	Courses
1	A	c1, c2
2	E	c3
3	M	C2, c3

In the above table, Course is a multi-valued attribute so it is not in 1NF.

Below Table is in 1NF as there is no multi valued attribute

ID	Name	Course
1	A	c1
1	A	c2
2	E	c3
3	M	c1
3	M	c2

2. Second Normal Form

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has **No Partial Dependency,** i.e.**,** no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

User_ID	Course_ID	Course_Fee
1	CS101	5000
2	CS102	2000
1	CS102	2000
3	CS101	5000
4	CS102	2000
2	CS103	7000

In the above table, the candidate key is the combination of (User_ID, Course_ID) . The non-prime attribute is Course_Fee. Since this non-prime attribute depends partially on the candidate key, this table is not in 2NF.

To convert this into 2NF, one needs to split the table into two and creating a common link between two. As we have done with the above table after splitting them into two and keeping Course_ID as the common key.

User_ID	Course_ID

Course_ID	Course_Fee

Example

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computers Network
1	C2	Computers Network

Table 3 :

Partial Dependency –

If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

- **Example 1 -** In relation STUDENT_COURSE given in Table 3,

```
FD set: {COURSE_NO->COURSE_NAME}
Candidate Key: {STUD_NO, COURSE_NO}
```

In FD COURSE_NO->COURSE_NAME, COURSE_NO (proper subset of candidate key) is determining COURSE_NAME (non-prime attribute). Hence, it is partial dependency and relation is not in second normal form.
To convert it to second normal form, we will decompose the relation STUDENT_COURSE (STUD_NO, COURSE_NO, COURSE_NAME) as :

```
STUDENT_COURSE (STUD_NO, COURSE_NO)
COURSE (COURSE_NO, COURSE_NAME)
```

Note - This decomposition will be lossless join decomposition as well as dependency preserving.

- **Example 2 -** Consider following functional dependencies in relation R (A, B , C, D)

- AB -> C [A and B together determine C]

BC -> D [B and C together determine D]

In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute.

3. Third Normal Form

Def 1: It follows two rules:

1. The table must be in 2NF.
2. No non-prime attribute must define another non-prime attribute.

Let's understand these rules using the following table:

Stud_No	Stud_Name	Stud_State	Stud_Country
101	Ram	Haryana	India
102	Ramesh	Punjab	India
103	Suresh	Punjab	India

Checking the first condition of 3NF: The candidate key for the table is Stud_No. It has the following dependency on other attributes:
Stud_No → [Stud_Name, Stud_State, State_Country]
Student_State → Stud_Country
In both the above cases, we see that there is only one candidate key and it has no other parts, and also there are no non-prime attributes. Hench the condition of 2NF holds.

Checking the second condition of 3NF: As we can see that Student_State → Stud_Country occurs which means that one non-prime attribute is defining another non-prime attribute, it violates the second rule of 3NF.

This problem can be fixed by splitting the table into two.
T1: Stud_No, Stud_Name, Stud_State
T2: Stud_State, Stud_Country
As in this case, no non-prime attribute is defining another non-prime attribute, hence the condition of 3NF holds.

Let's see another table and try to understand the concept of 3NF:

Exam_Name	Exam_Year	Topper_Name	Topper_DOB
ABC	2016	Ram	15-06-1992
BCD	2017	Ramesh	16-07-1994
EFG	2017	Suresh	10-06-1991
ABC	2017	Kamlesh	11-11-1993

Let's first find out the dependencies:
(Exam_Name, Exam_Year) → (Topper_Name, Topper_DOB)
Topper_Name → Topper_DOB

We can very well see that since there are no repetitions in the columns, 1NF holds. Also, 2NF holds, as no non-prime attribute is partially dependent on the candidate key. But the second dependency violates the 3NF.

To solve this we need to split the table into two as we did in an earlier case:
T1: Exam_Name, Exam_Year, Topper_Name
T2: Topper_Name, Topper_DOB

Let's look at a few other definitions of 3NF:
Def 2:
This also defines two sets of rules which are mostly similar to the Def 1.

1. In 2NF and
2. Non-prime attributes are not transitively dependent on prime attributes.

Def 3:
It says, for every X → A, one of the following should be true:

1. X is a Superkey
2. A-X is a prime attribute
3. X → is a trivial functional dependency.

Example:

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

Transitive dependency - If A->B and B->C are two FDs then A->C is called transitive dependency.

- **Example 1 -** In relation STUDENT given in Table 4,

  FD set: {STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE, STUD_STATE -> STUD_COUNTRY}
  Candidate Key: {STUD_NO}

  For this relation in table 4, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:
  STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
  STATE_COUNTRY (STATE, COUNTRY)

- **Example 2 -** Consider relation R(A, B, C, D, E)
  A -> BC,
  CD -> E,
  B -> D,
  E -> A
  All possible candidate keys in above relation are {A, E, CD, BC} All attribute are on right sides of all functional dependencies are prime.

4. Boyce-Codd Normal Form (BCNF)

A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF if in every non-trivial functional dependency X --> Y, X is a superkey.
Let's understand this in a more basic form. We can say that a table is in BCNF if it satisfies the BCNF condition along with all other normalization condition including 1NF, 2NF and 3NF.
For 1NF, there must not be any repeated values in any attributes.
For 2NF, no non-prime attribute must be defined by any prime attributes.
For 3NF, no non-prime attribute must be defined by any non-prime attributes.
For BCNF, no prime or non-prime attribute must be define any prime attributes
Let's look at the following table and try to understand how this works:

STUD_ID	Subject	Prof_id
1001	DBMS	103
1001	OS	110
1002	DBMS	111

Functional Dependencies:
(Stud_ID, Subject) → Prof_id
Prof_id → Subject

Candidate Keys:
(Stud_ID, Subject), Prof_id, (Prof_id, Stud_ID)

The second condition in Functional Dependencies does not follow the rule of BCNF.
Let's try to analyse whether the relation satisfies all type of Normal Forms.
1NF: Since there are no repeated attributes, this holds true.
2NF: Since there are no non-prime attributes, this also holds true.
3NF: Since there are no non-prime attributes, this also holds true.
BCNF: In the second dependency, Prof_id being a prime attribute defines another prime attribute Subject. Hence the condition fails to satisfy BCNF.

Now let's modify the table by adding another row to it:

STUD_ID	Subject	Prof_id
1001	DBMS	103
1001	OS	110
1002	DBMS	111
1003	DBMS	103

Then break the table into a BCNF compliance database.

STUD_ID	Prof_id
1001	103
1001	110
1002	111
1003	103

Prof_id	Subject
103	DBMS
110	OS
111	DBMS
103	DBMS

Although this decomposition satisfies with the conditions of BCNF, but the DBMS might not accept this split because while splitting the table the first Functional Dependency i.e., (Subject_ID, Subject) → Prof_ID does not exist anymore.

- **Example 1 -** Find the highest normal form of a relation R(A,B,C,D,E) with FD set as -

```
BC->D,
AC->BE,
B->E
```

Step 1. As we can see, (AC)+ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.
Step 2. The prime attributes are those attribute which are part of candidate key {A,C} in this example and others will be non-prime {B,D,E} in this example.
Step 3. The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.

Step 4. The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not a proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).
Step 5.The relation is not in 3rd normal form because in BC->D (neither BC is a superkey nor D is a prime attribute) and in B->E (neither B is a superkey nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute.
So the highest normal form of relation will be 2nd Normal form.

Key Points -

1. BCNF is free from redundancy.
2. If a relation is in BCNF, then 3NF is also also satisfied.
3. If all attributes of relation are prime attribute, then the relation is always in 3NF.
4. A relation in a Relational Database is always and at least in 1NF form.
5. Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
6. If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
7. Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
8. There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

Exercise 1: Find the highest normal form in R (A, B, C, D, E) under following functional dependencies.

```
ABC --> D
CD --> AE
```

Important Points for solving above type of question.
1) It is always a good idea to start checking from BCNF, then 3 NF and so on.
2) If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, ABC --> D is in BCNF (Note that ABC is a super key), so no need to check this dependency for lower normal forms.

Candidate keys in a given relation are {ABC, BCD}

BCNF: ABC -> D is in BCNF. Let us check CD -> AE, the CD is not a superkey so this dependency is not in BCNF. So, R is not in BCNF.

3NF: ABC -> D we don't need to check for this dependency as it already satisfied BCNF. Let us consider a CD -> AE. Since E is not a prime attribute, so the relation is not in 3NF.

2NF: In 2NF, we need to check for partial dependency. CD which is a proper subset of a candidate key and it determine E, which is a non prime attribute. So, the given relation is also not in 2 NF. So, the highest normal form is 1 NF.

FUNCTIONAL DEPENDENCIES

Database normalization is the process of organizing the attributes of the database to reduce or eliminate **data redundancy (having the same data but at different places)** .

Problems because of data redundancy: Data redundancy unnecessarily increases the size of the database as the same data is repeated in many places. Inconsistency problems also arise during insert, delete and update operations.

Functional Dependency

Functional Dependency is a constraint between two sets of attributes in a relation from a database. A functional dependency is denoted by an arrow (→). If an attribute A functionally determines B or B is functionally dependent on A, then it is written as A → B.
For example, employee_id → name means employee_id functionally determines the name of the employee. As another example in a time table database, {student_id, time} → {lecture_room}, student ID and time determine the lecture room where the student should be.
Let's look at the tables below:

Enroll_No	Name	Address	Phone_No

In this table we can say that:
Enroll_No → Name
Enroll_No → Address
Enroll_No → Phone No

Subject_Id	Student_Id	Marks	Grade

In this table we can say that:
(Subject_Id, Student_Id) → Marks
(Subject_Id, Student_Id) → Marks, Grades

Examples: For the following table, which of the following functional dependencies hold True.

1. A → B
2. B → A

A	B
x	1
y	1
z	2

Solution: The first condition holds True as for every value of A there is a unique value in B.
The second dependency results in False, as there is no unique identification for every value of B in A.

What does functionally dependent mean?

A function dependency A → B means for all instances of a particular value of A, there is the same value of B.
Let's do an exercise on Functional Dependency:
For the following table define which of the dependency is True or False

Enroll_No	Name	Address
101	Raj	ABC
102	Ravi	ABC
103	Raj	XYZ

1. Enroll_No → Name, Address
2. Name → Address
3. Address → Name

Solution:

1. This holds True as for every Enroll_No, there is a unique representation in Name and Address combined.
2. This is False as there is an anomaly in the Name Raj
3. This is False as there is an anomaly in the Address ABC

For example in the below table A → B is true, but B → A is not true

as there are different values of A for B = 3.

```
A   B
------
1   3
2   3
4   0
1   3
4   0
```

Why do we study Functional Dependency?

We study functional dependency so that we can carve out a few attributes to another table so that the data redundancy can be reduced to a minimum. Let's look at the following table:

Student_ID	Name	Dept_Id	Dept_Name
101	ABC	10	CS
102	BCD	11	ECE
103	ABC	10	CS
104	XYZ	11	ECE
105	CDE	10	CS

In this table let's find out the functional dependency between two attributes.
As we can see that Dept_Id has each unique identification in Dept_Name, so we can say that Dept_Id → Dept_Name
Therefore we can carve out this two table and create another one out of this.

Dept_Id	Dept_Name
10	CS
11	ECE

Trivial Functional Dependency:
X → Y is trivial only when Y is subset of X.
Examples:

```
ABC → AB
ABC → A
ABC → ABC
```

Non Trivial Functional Dependencies X → Y is a non trivial functional dependencies when Y is not a subset of X.

X → Y is called completely non-trivial when X intersect Y is NULL.
Examples:

```
Id → Name,
Name → DOB
```

Semi Non Trivial Functional Dependencies X → Y is called semi non-trivial when X intersect Y is not NULL.
Examples:

```
AB → BC,
AD → DC
```

NORMALISATION

Objectives of Good Database Design:

1. No updation, insertion and deletion anomalies
2. Easily Extendible
3. Good Performance for all query sets
4. More Informative

Anomalies: There are different types of anomalies which can occur in referencing and referenced relation:

1. **Updation Anomaly:** If some particular attribute value has to be updated, then it has to be updated in every tuple consisting of that value, which would be too costly operation as all records have to be first transferred to the main memory, and then again transferred to the secondary memory after updation.

Student_Id	Student_Name	Subject_Id	Subject_Name
1001	ABC	CS101	Database Management System
1002	XYZ	CS101	Database Management System
1001	ABC	CS102	Operating System
1003	BCD	CS102	Operating System

   In the above table, if someone tries to rename the Subject_Name Database Management System to DBMS and while doing so, the system crashes in between, then it would lead to inconsistency of the table. Similarly imposing a bad query can also lead to redundancy issue.
2. **Insertion Anomaly:** If a tuple is inserted in referencing relation and referencing attribute value is not present in referenced

attribute, it will not allow inserting in referencing relation. Suppose in the above table we need to add a student PQR with Student_ID 1004 and we don't know the Subject_name or the Subject_Id the student intends to take. If the fields are mandatory then, the addition of the data to the table will not be possible.

3. **Deletion Anomaly:** Deletion of some data may lead to loss of some other useful data. For example, Suppose some student details have to be deleted. Now when the required tuple is deleted, the subject name details corresponding to that student record also gets deleted, which might lead to loss of the details of the subject.

CONCURRENCY CONTROL

INTRODUCTION TO ACID PROPERTIES

Concurrency Control deals with **interleaved execution** of more than one transaction. In the next article, we will see what is serializability and how to find whether a schedule is serializable or not.

What is Transaction?

A set of logically related operations is known as transaction. The main operations of a transaction are:

Read(A): Read operations Read(A) or R(A) reads the value of A from the database and stores it in a buffer in main memory.

Write (A): Write operation Write(A) or W(A) writes the value back to the database from buffer.

Let us take a debit transaction from an account which consists of following operations:

1. R(A);
2. A=A-1000;
3. W(A);

Assume A's value before starting of transaction is 5000.

- The first operation reads the value of A from database and stores it in a buffer.
- Second operation will decrease its value by 1000. So buffer will contain 4000.
- Third operation will write the value from buffer to database. So A's final value will be 4000.

But it may also be possible that transaction may fail after executing some of its operations. The failure can be because of **hardware, software or power** etc. For example, if debit transaction discussed above fails after executing operation 2, the value of A will remain 5000 in the database which is not acceptable by the bank. To avoid this, Database has two important operations:
Commit: After all instructions of a transaction are successfully executed, the changes made by transaction are made permanent in the database.

Rollback: If a transaction is not able to execute all operations successfully, all the changes made by transaction are undone.

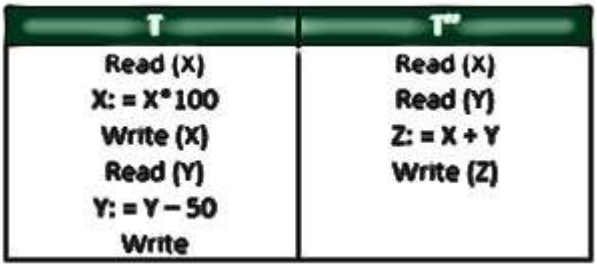
Properties of a transaction

- **Atomicity**: By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.
  --Abort: If a transaction aborts, changes made to database are not visible.
  --Commit: If a transaction commits, changes made are visible.
  Atomicity is also known as the 'All or nothing rule'.
  Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

- **Consistency**: This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,
  The total amount before and after the transaction must be maintained.
  Total **before T** occurs = **500 + 200 = 700**.
  Total **after T occurs** = **400 + 300 = 700**.
  Therefore, database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.
- **Isolation**: This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.
  Let **X**= 500, **Y** = 500.
  Consider two transactions **T** and **T''.**



T	T''
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y – 50 Write	Read (X) Read (Y) Z: = X + Y Write (Z)

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result , interleaving of operations takes place due to which **T''** reads correct value of **X** but incorrect value of **Y** and sum computed by
T'': (X+Y = 50, 000+500=50, 500)
is thus not consistent with the sum at end of transaction: **T: (X+Y = 50, 000 + 450 = 50, 450)**.
This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

- **Durability:** This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

The **ACID** properties, in totality, provide a mechanism to ensure correctness and consistency of a database in a way such that each transaction is a group of operations that acts a single unit, produces consistent results, acts in isolation from other operations and updates that it makes are durably stored.

CONCURRENCY CONTROL PROTOCOLS

Introduction to Concurrency Control: The present-day scenario involves working on a multi-user system where there can be more than one or multiple database or transactions that needs access to. This can raise confusion and conflict within various operations under process. The conflicts can be put under check during the READ operations but the condition might worsen when a single data in memory needs access from multiple databases, during the Write operations. This is where the Concurrency Control comes into play. This is an important procedure for the smooth flow of operations during a database transaction. It allows the simultaneous execution of multiple operations one by one without letting the rise of any deadlock situation. This, in turn, protects the data integrity of every database of every user.
Example: Suppose an online product is needed to be bought by two different persons. However, there is only one product available. If both of them begin there buying process at the same time, this might lead to a conflict. In the absence of concurrency control, both of them might end up buying a single product, which will be a fault. This condition must be avoided and it is checked by the concurrency control which allows both the users to access the database throughout the process of ordering and only allows the one customer to buy who completes the process of the transaction first.

Various Protocols in Concurrency:

- Lock-Based Protocols
- Two Phase Locking
- Timestamp-Based Protocols

1. **Lock-based Protocols**: A lock is a variable associated with a data item that describes the status of the data item with respect to a possible operation that can be applied to it. They synchronize the access by concurrent transactions to the database items. It is required in this protocol that all the data items must be accessed in a mutually exclusive manner. Let's get familiar with the two common locks which are used and some terminology followed in this protocol.

    - **Shared Lock (S):** It is also known as Read-only lock. As the name suggests it can be shared between transactions because while holding this lock the transaction does not have the permission to update data on the data item. S-lock is requested using lock-S instruction.
    - **Exclusive Lock (X):** In this lock, the data items can be both read and written. This is Exclusive and cannot be held simultaneously on the same data item. X-lock is requested using lock-X instruction.
    - **Simplistic Lock Protocol:** This is sort of a pre-lock for the objects which is generally obtained before the beginning of any operation. Once the write operation has been performed, the data may be unlocked by the transaction.
    - **Pre-claiming Locking:** This locking mechanism is used during the execution process for the evaluation of operations and the creation of a list of initiating data items. Everything is unlocked once all the operations are over.

    Starvation: It is possible if the concurrency control manager is badly designed. For example, A transaction may be waiting for an X-lock on an item, while a sequence of other transactions requests and are granted an S-lock on the same item. This may be avoided if the concurrency control manager is properly designed.
    Deadlock: This is the situation when multiple processes are waiting for each other to complete and release a resource. It leads to a circular chain of a wait when neither of them is able to complete as one hold the resource of completion of another.

2. **Two-phase Locking Protocol or 2PL:** A transaction is said to follow a Two-Phase Locking protocol if Locking and Unlocking can be done in two phases.
    - *Growing Phase:* New locks on data items may be acquired but none can be released.
    - *Shrinking Phase:* Existing locks may be released but no new locks can be acquired.

Although the protocol offers serializability, it doesn't avoid the deadlock situation. This can be further divided into a few types:

b. **Strict 2PL:** This method does not release the lock after every usage. It holds the lock until the whole process is over and releases the lock at one go.
c. **Centralized 2PL:** This mechanism has a single lock manager for the management of the entire process.
d. **Primary copy 2PL:** In this type, various copies of a single primary 2PL are distributed to various sites but the data items are controlled by a single primary 2PL. Once the main copy has been updated, the change is reflected on all the associated copies.
e. **Distributed 2PL:** In this type, all the sites are provided with the lock managers to manage the locks of the data of those sites. It resembles primary 2PL when no changes take place in the lock managers and also the communication cost of the former is more than the latter.
f.

3. **Timestamp-based Protocols:** A timestamp is a unique identifier created by the DBMS to identify a transaction. They are usually assigned in the order in which they are submitted to the system, so a timestamp may be thought of as the transaction start time. There may be different ways of generating timestamps such as:

    - A simple counter that increments each time its value is assigned to a transaction. They may be numbered 1, 2, 3…. Though we'll have to reset the counter from time to time to avoid overflow.

- Using the current date/time from the system clock. Just ensuring that no two transactions are given the same value in the same clock tick, we will always get a unique timestamp. This method is widely used.

The main idea for this protocol is to order the transactions based on their Timestamps. The advantages of using Timestamp-based protocols are:

- A schedule in which the transactions participate is then serializable and the only equivalent serial schedule permitted has the transactions in the order of their Timestamp Values.
- The deadlocks can be avoided since there is no waiting time for the transaction.

One disadvantage of such a mechanism is that it might lead to starvation if the same transaction is continuously aborted and restarted.

The advantages of using Concurrency Control Protocols are:

- It helps to maintain serializability.
- To preserve data integrity.
- To fix the read-write issues if any.
- For a smoother workflow of concurrent transactions.

SCHEDULES AND TYPES OF SCHEDULES

What is a Schedule?

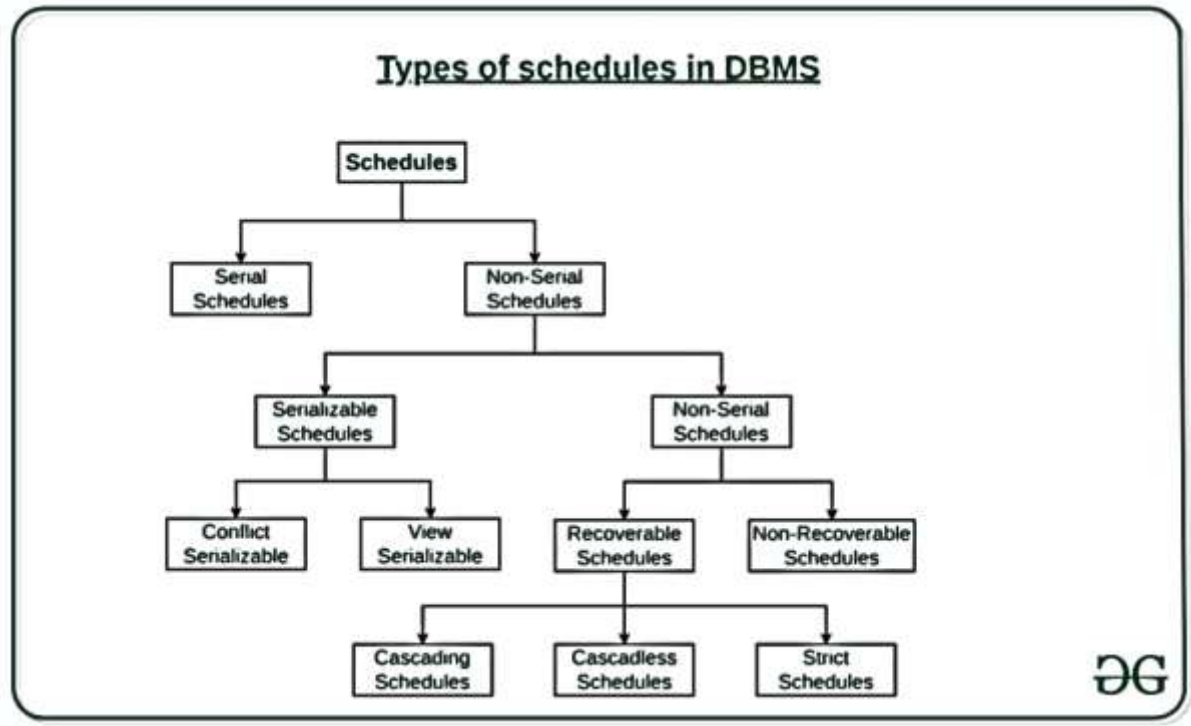
A schedule is a series of operations from one or more transactions. Schedules are used to resolve conflicts between the transactions.

Schedule, as the name suggests, is a process of lining the transactions

and executing them one by one. When there are multiple transactions that are running in a concurrent manner and the order of operation is needed to be set so that the operations do not overlap each other, Scheduling is brought into play and the transactions are timed accordingly.

Types of Schedules?

Lets discuss various types of schedules.



1. **Serial Schedules:**
   Schedules in which the transactions are executed non-interleaved, i.e., a serial schedule is one in which no transaction starts until a running transaction has ended are called serial schedules.

   Example: Consider the following schedule involving two transactions T_1 and T_2.

2. where R(A) denotes that a read operation is performed on some data item 'A'
   This is a serial schedule since the transactions perform serially in the order T_1 —> T_2

3. **Non-Serial Schedule:** This is a type of Scheduling where the operations of multiple transactions are interleaved. This might lead to a rise in the concurrency problem. The transactions are executed in a non-serial manner, keeping the end result correct and same as the serial schedule. Unlike the serial schedule where one transaction must wait for another to complete all its operation, in the non-serial schedule, the other transaction proceeds without waiting for the previous transaction to complete. This sort of schedule does not provide any benefit of the concurrent transaction. It can be of two types namely, Serializable and Non-Serializable Schedule.

   The Non-Serial Schedule can be divided further into Serializable and Non-Serializable.

4. **Serializable:** This is used to maintain the consistency of the database. It is mainly used in the Non-Serial scheduling to verify whether the scheduling will lead to any inconsistency or not. On the other hand, a serial schedule does not need the serializability because it follows a transaction only when the previous transaction is complete. The non-serial schedule is said to be in a serializable schedule only when it is equivalent to the serial schedules, for an n number of transactions. Since concurrency is allowed in this case thus, multiple transactions can execute concurrently. A serializable schedule helps in improving both resource utilization and CPU throughput. These are of two types:

5. **Conflict Serializable:** A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations. Two operations are said to be conflicting if all conditions satisfy:
    - They belong to different transactions
    - They operate on the same data item
    - At Least one of them is a write operation

View Serializable: A Schedule is called view serializable if it is view equal to a serial schedule (no overlapping transactions). A conflict schedule is a view serializable but if the serializability contains blind writes, then the view serializable does not conflict serializable.

Non-Serializable: The non-serializable schedule is divided into two types, Recoverable and Non-recoverable Schedule.

Recoverable Schedule: Schedules in which transactions commit only after all transactions whose changes they read commit are called recoverable schedules. In other words, if some transaction T_j is reading value updated or written by some other transaction T_i, then the commit of T_j must occur after the commit of T_i.
Example 1:

S1: R1(x), **W1(x)**, R2(x), R1(y), R2(y),

W2(x), W1(y), **C1**, **C2**;

Given schedule follows order of **Ti->Tj => C1->C2**. Transaction T1 is executed before T2 hence there is no chances of conflict occur. R1(x) appears before W1(x) and transaction T1 is committed before T2 i.e. completion of first transaction performed first update on data item x, hence given schedule is recoverable.

Example 2: Consider the following schedule involving two transactions T_1 and T_2.

This is a recoverable schedule since T_1 commits before T_2, that makes the value read by T_2 correct.

There can be three types of recoverable schedule:

Cascading Schedule: When there is a failure in one transaction and this leads to the rolling back or aborting other dependent transactions, then such scheduling is referred to as Cascading rollback or cascading abort. Example:

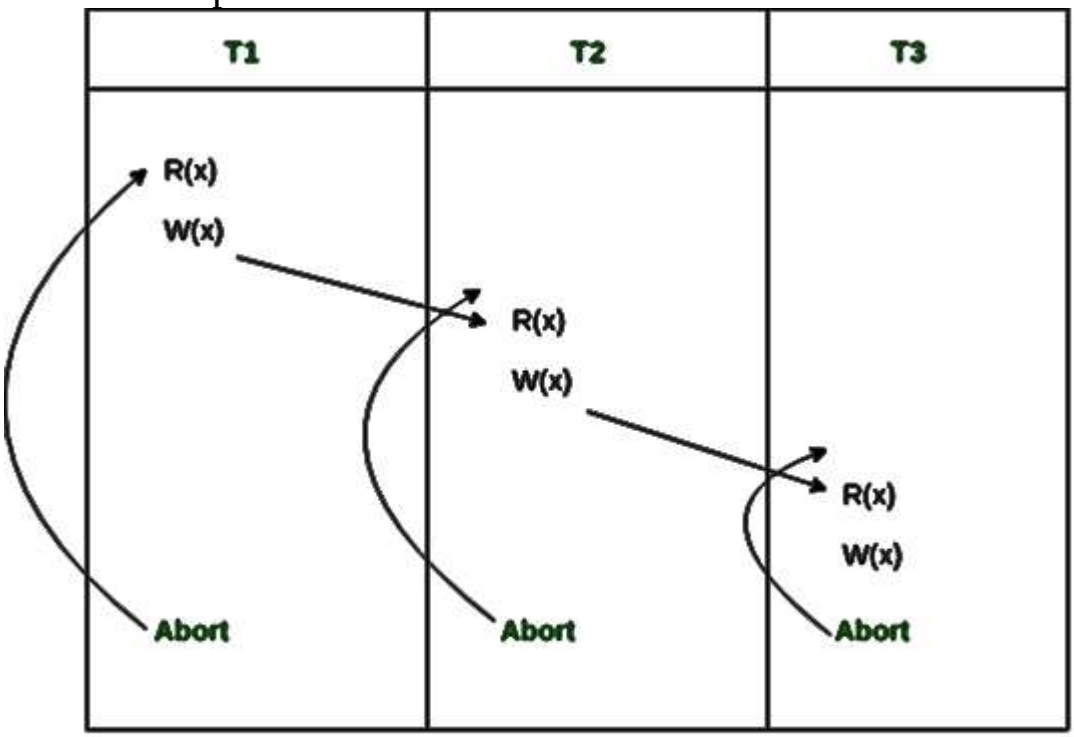


Figure - Cascading Abort

Cascadeless Schedule: Also called Avoids cascading aborts/rollbacks (ACA). Schedules in which transactions read values only after all transactions whose changes they are going to read commit are called cascadeless schedules. Avoids that a single transaction abort leads to a series of transaction rollbacks. A strategy to prevent cascading aborts is to disallow a transaction from reading uncommitted changes from another transaction in the same schedule.

In other words, if some transaction T_j wants to read value updated or

written by some other transaction T_i, then the commit of T_j must read it after the commit of T_i.

Strict Schedule: A schedule is strict if for any two transactions T_i, T_j, if a write operation of T_i precedes a conflicting operation of T_j (either read or write), then the commit or abort event of T_i also precedes that conflicting operation of T_j.
In other words, T_j can read or write updated or written value of T_i only after T_i commits/aborts.

Non-Recoverable Schedule: The table below shows a schedule with two transactions, T1 reads and writes A and that value is read and written by T2. T2 commits. But later on, T1 fails. So we have to rollback T1. Since T2 has read the value written by T1, it should also be rollbacked. But we have already committed that. So this schedule is irrecoverable schedule. When Tj is reading the value updated by Ti and Tj is committed before committing of Ti, the schedule will be irrecoverable.

T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
A=A-100;	A=4000			A=5000
W(A);	A=4000			A=4000
		R(A);	A=4000	A=4000
		A=A+500;	A=4500	A=4000
		W(A);	A=4500	A=4500
		Commit;		
Failure Point				
Commit;				

Note - It can be seen that:

1. Cascadeless schedules are stricter than recoverable schedules or are a subset of recoverable schedules.
2. Strict schedules are stricter than cascadeless schedules or are a subset of cascadeless schedules.
3. Serial schedules satisfy constraints of all recoverable, cascadeless and strict schedules and hence is a subset of strict schedules.

HIMANSHU KUMAR(LINKEDIN)

https://www.linkedin.com/in/himanshukumarmahuri

CREDITS- INTERNET.

DISCLOSURE- ALL THE DATA AND IMAGES ARE TAKEN FROM GOOGLE AND INTERNET.