ER Diagrams

- ER diagram or **Entity Relationship diagram** is a conceptual model that gives the graphical representation of the logical structure of the database.
- It shows all the constraints and relationships that exist among the different components.

Components of ER diagram- An ER diagram is mainly composed of following three components-

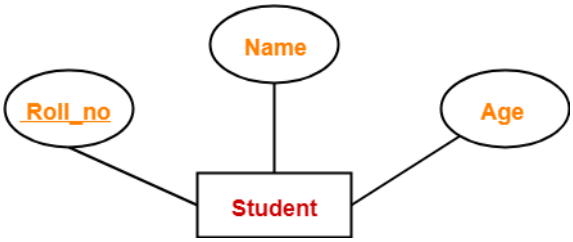
1. Entity Sets
2. Attributes
3. Relationship Set

Example- Consider the following Student table-

Roll_no	Name	Age
1	Akshay	20
2	Rahul	19
3	Pooja	20
4	Aarti	19

This complete table is referred to as "Student Entity Set" and each row represents an "entity".

Representation as ER Diagram- The above table may be represented as ER diagram as-



Here,

- Roll_no is a primary key that can identify each entity uniquely.
- Thus, by using student's roll number, a student can be identified uniquely.

ER Diagram Symbols-

An ER diagram is composed of several components and each component in ER diagram is represented using a specific symbol.

ER diagram symbols are discussed below-

1. For Entity Sets-

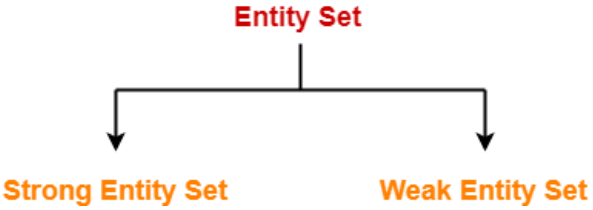
An entity set is a set of same type of entities.

An entity refers to any object having-

- Either a physical existence such as a particular person, office, house or car.

- Or a conceptual existence such as a school or a company.

An entity set may be of the following two types-



1. Strong Entity Set-

- A strong entity set possess its own primary key.
- It is represented using a single rectangle.

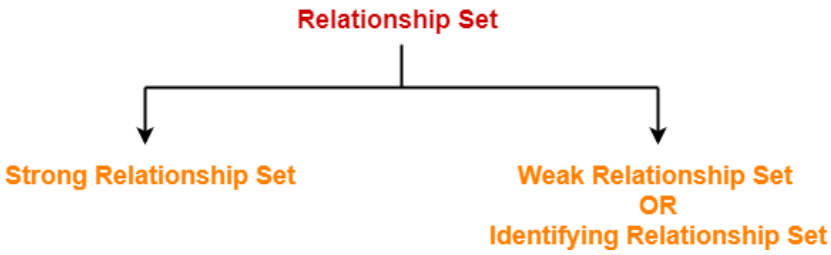
2. Weak Entity Set-

- A weak entity set do not possess its own primary key.
- It is represented using a double rectangle.

2. For Relationship Sets-

- Relationship defines an association among several entities.
- A relationship set is a set of same type of relationships.

A relationship set may be of the following two types-



1. Strong Relationship Set-

- A strong relationship exists between two strong entity sets.
- It is represented using a diamond symbol.

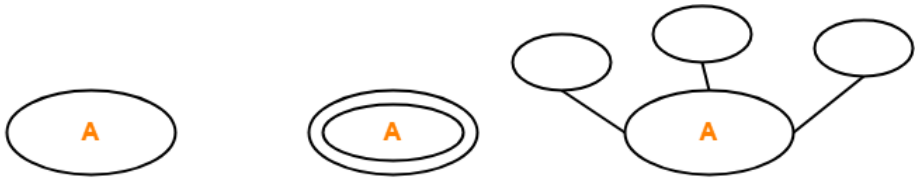
2. Weak Relationship Set-

- A weak or identifying relationship exists between the strong and weak entity set.
- It is represented using a double diamond symbol.

Strong Relationship Set Weak or Identifying Relationship Set

3. For Attributes-

- Attributes are the properties which describes the entities of an entity set.
- There are several types of attributes.



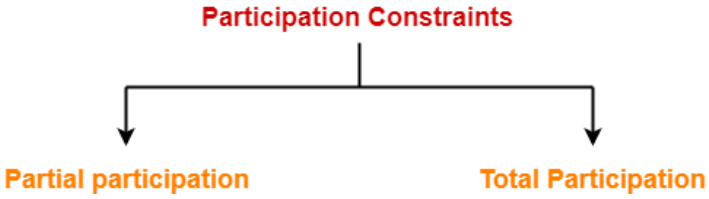
Attribute Multivalued Attribute Composite Attribute

Key Attribute Partial Attribute Derived Attribute

4. For Participation Constraints-

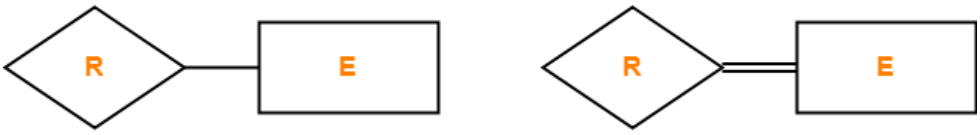
Participation constraint defines the least number of relationship instances in which an entity has to necessarily participate.

There are two types of participation constraints-



1. Partial Participation- Partial participation is represented using a single line between the entity set and relationship set.

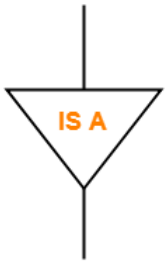
2. Total Participation- Total participation is represented using a double line between the entity set and relationship set.



Partial Participation Total Participation

5. For Specialization and Generalization-

- Generalization is a process of forming a generalized super class by extracting the common characteristics from two or more classes.
- Specialization is a reverse process of generalization where a super class is divided into sub classes by assigning the specific characteristics of sub classes to them.

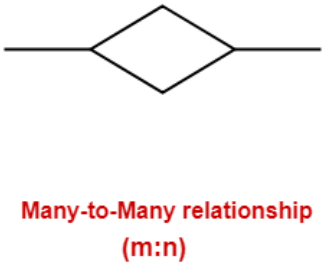


IS A specialization or generalization

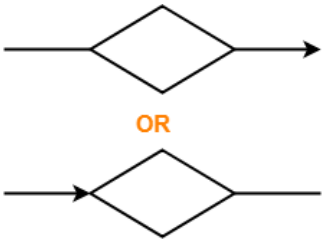
6. For Cardinality Constraints / Ratios- Cardinality constraint defines the maximum number of relationship instances in which an entity can participate.

There are 4 types of cardinality ratios-

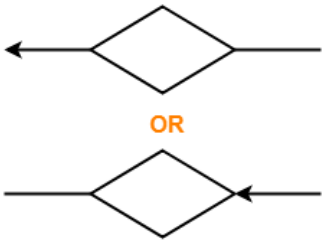
1. Many-to-many cardinality (m : n)
2. Many-to-one cardinality (m : 1)
3. One-to-many cardinality (1 : n)
4. One-to-one cardinality (1 : 1)



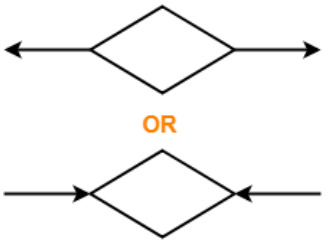
Many-to-Many relationship
(m:n)



Many-to-One relationship
(m:1)



One-to-Many relationship
(1:n)



One-to-One relationship
(1:1)

Entity Sets in DBMS

An entity set is a set of same type of entities.

An entity refers to any object having-

- Either a physical existence such as a particular person, office, house or car.
- Or a conceptual existence such as a school, a university, a company or a job.

In ER diagram,

- Attributes are associated with an entity set.
- Attributes describe the properties of entities in the entity set.
- Based on the values of certain attributes, an entity can be identified uniquely.

Types of Entity Sets- An entity set may be of the following two types-

1. Strong entity set
2. Weak entity set

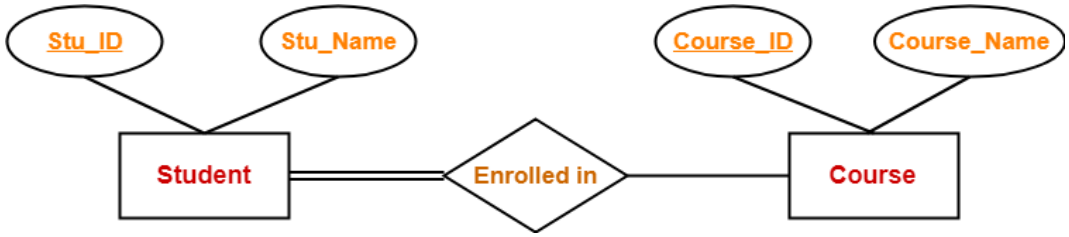
1. Strong Entity Set-

- A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities.
- In other words, a primary key exists for a strong entity set.
- Primary key of a strong entity set is represented by underlining it.

Symbols Used-

- A single rectangle is used for representing a strong entity set.
- A diamond symbol is used for representing the relationship that exists between two strong entity sets.
- A single line is used for representing the connection of the strong entity set with the relationship set.
- A double line is used for representing the total participation of an entity set with the relationship set.
- Total participation may or may not exist in the relationship.

Example- Consider the following ER diagram-



In this ER diagram,

- Two strong entity sets "**Student**" and "**Course**" are related to each other.
- Student ID and Student name are the attributes of entity set "Student".
- Student ID is the primary key using which any student can be identified uniquely.
- Course ID and Course name are the attributes of entity set "Course".
- Course ID is the primary key using which any course can be identified uniquely.

- Double line between Student and relationship set signifies total participation.
- It suggests that each student must be enrolled in at least one course.
- Single line between Course and relationship set signifies partial participation.
- It suggests that there might exist some courses for which no enrollments are made.

2. Weak Entity Set-

- A weak entity set is an entity set that does not contain sufficient attributes to uniquely identify its entities.
- In other words, a primary key does not exist for a weak entity set.
- However, it contains a partial key called as a **discriminator.**
- Discriminator can identify a group of entities from the entity set.
- Discriminator is represented by underlining with a dashed line.

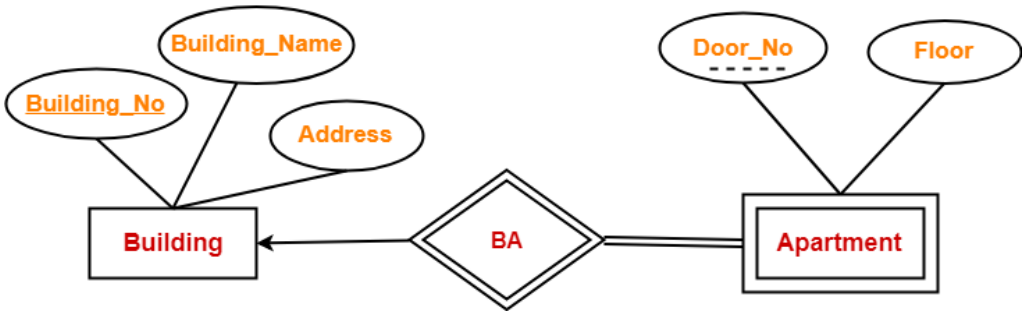
NOTE-

- The combination of discriminator and primary key of the strong entity set makes it possible to uniquely identify all entities of the weak entity set.
- Thus, this combination serves as a primary key for the weak entity set.
- Clearly, this primary key is not formed by the weak entity set completely.
  Primary key of week entity set = Its own discriminator + Primary key of strong entity set

Symbols Used-

- A double rectangle is used for representing a weak entity set.
- A double diamond symbol is used for representing the relationship that exists between the strong and weak entity sets and this relationship is known as **identifying relationship**.
- A double line is used for representing the connection of the weak entity set with the relationship set.
- Total participation always exists in the identifying relationship.

Example- Consider the following ER diagram-



In this ER diagram,

- One strong entity set "**Building**" and one weak entity set "**Apartment**" are related to each other.

- Strong entity set "Building" has building number as its primary key.
- Door number is the discriminator of the weak entity set "Apartment".
- This is because door number alone cannot identify an apartment uniquely as there may be several other buildings having the same door number.
- Double line between Apartment and relationship set signifies total participation.
- It suggests that each apartment must be present in at least one building.
- Single line between Building and relationship set signifies partial participation.
- It suggests that there might exist some buildings which has no apartment.

To uniquely identify any apartment,

- First, building number is required to identify the particular building.
- Secondly, door number of the apartment is required to uniquely identify the apartment.

Thus,

Primary key of Apartment
= Primary key of Building + Its own discriminator
= Building number + Door number

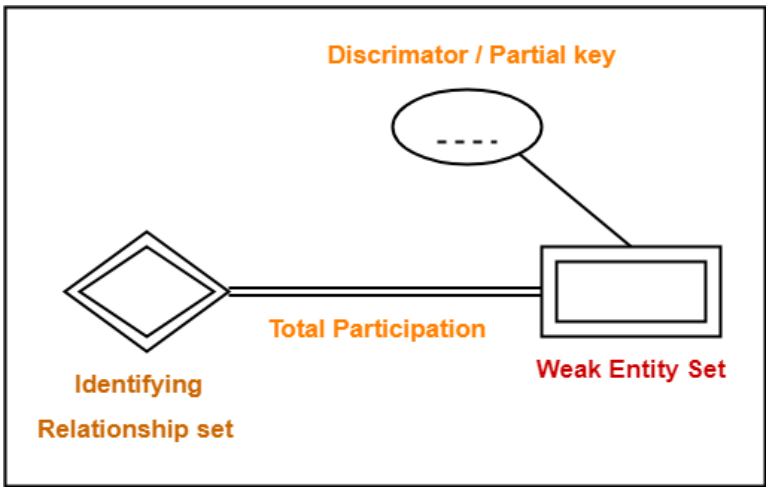
Differences between Strong entity set and Weak entity set-

Strong entity set	Weak entity set
A single rectangle is used for the representation of a strong entity set.	A double rectangle is used for the representation of a weak entity set.
It contains sufficient attributes to form its primary key.	It does not contain sufficient attributes to form its primary key.
A diamond symbol is used for the representation of the relationship that exists between the two strong entity sets.	A double diamond symbol is used for the representation of the identifying relationship that exists between the strong and weak entity set.
A single line is used for the representation of the connection between the strong entity set and the relationship.	A double line is used for the representation of the connection between the weak entity set and the relationship set.
Total participation may or may not exist in the relationship.	Total participation always exists in the identifying relationship.

Important Note-

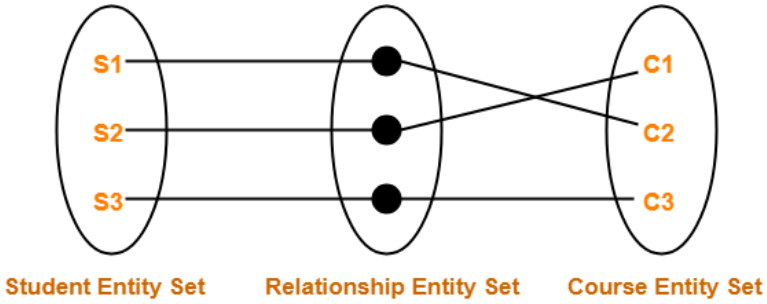
In ER diagram, weak entity set is always present in total participation with the identifying relationship set.

So, we always have the picture like shown here-



Relationship Set- A relationship set is a set of relationships of same type.

Example- Set representation of above ER diagram is-



Set Representation of ER Diagram

Degree of a Relationship Set- The number of entity sets that participate in a relationship set is termed as the degree of that relationship set. Thus,

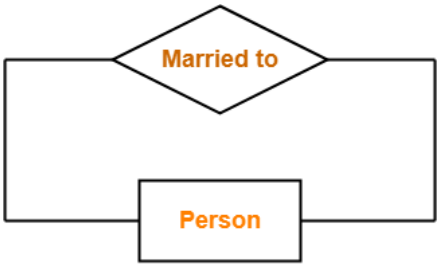
Degree of a relationship set = Number of entity sets participating in a relationship set

Types of Relationship Sets- On the basis of degree of a relationship set, a relationship set can be classified into the following types-

1. Unary relationship set
2. Binary relationship set
3. Ternary relationship set
4. N-ary relationship set

1. Unary Relationship Set- Unary relationship set is a relationship set where only one entity set participates in a relationship set.

Example- One person is married to only one person



Unary Relationship Set

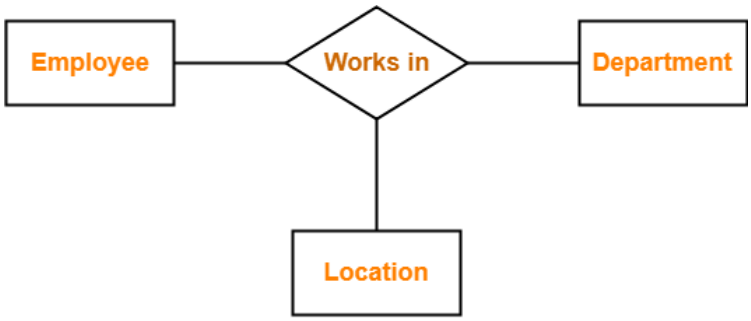
2. Binary Relationship Set- Binary relationship set is a relationship set where two entity sets participate in a relationship set.

Example- Student is enrolled in a Course

Binary Relationship Set

3. Ternary Relationship Set- Ternary relationship set is a relationship set where three entity sets participate in a relationship set.

Example-



Ternary Relationship Set

4. N-ary Relationship Set- N-ary relationship set is a relationship set where 'n' entity sets participate in a relationship set.

Cardinality Constraint- Cardinality constraint defines the maximum number of relationship instances in which an entity can participate.

Types of Cardinality Ratios- There are 4 types of cardinality ratios-

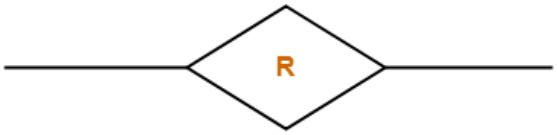
1. Many-to-Many cardinality (m : n)

2. Many-to-One cardinality (m : 1)
3. One-to-Many cardinality (1 : n)
4. One-to-One cardinality (1 : 1)

1. Many-to-Many Cardinality- By this cardinality constraint,

- An entity in set A can be associated with any number (zero or more) of entities in set B.
- An entity in set B can be associated with any number (zero or more) of entities in set A.

Symbol Used-



Cardinality Ratio = m : n

Example- Consider the following ER diagram-

Many to Many Relationship

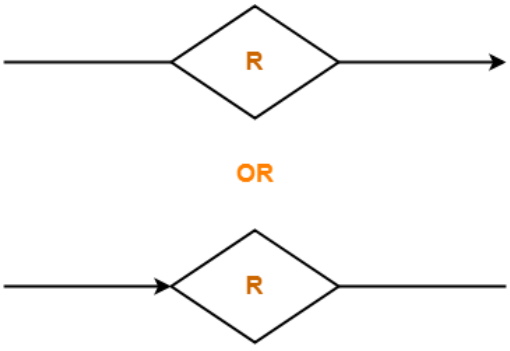
Here,

- One student can enroll in any number (zero or more) of courses.
- One course can be enrolled by any number (zero or more) of students.

2. Many-to-One Cardinality- By this cardinality constraint,

- An entity in set A can be associated with at most one entity in set B.
- An entity in set B can be associated with any number (zero or more) of entities in set A.

Symbol Used-



Cardinality Ratio = m : 1

Example- Consider the following ER diagram-

Many to One Relationship

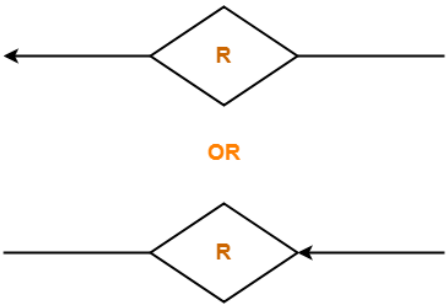
Here,

- One student can enroll in at most one course.
- One course can be enrolled by any number (zero or more) of students.

3. One-to-Many Cardinality- By this cardinality constraint,

- An entity in set A can be associated with any number (zero or more) of entities in set B.
- An entity in set B can be associated with at most one entity in set A.

Symbol Used-



Cardinality Ratio = 1 : n

Example- Consider the following ER diagram-

One to Many Relationship

Here,

- One student can enroll in any number (zero or more) of courses.
- One course can be enrolled by at most one student.

4. One-to-One Cardinality- By this cardinality constraint,

- An entity in set A can be associated with at most one entity in set B.
- An entity in set B can be associated with at most one entity in set A.

Symbol Used-

OR

Cardinality Ratio = 1 : 1

Example- Consider the following ER diagram-

One to One Relationship

Here,

- One student can enroll in at most one course.
- One course can be enrolled by at most one student.

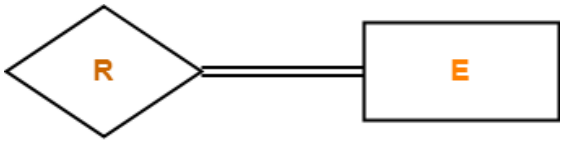
Participation Constraints- Participation constraints define the least number of relationship instances in which an entity must compulsorily participate.

Types of Participation Constraints- There are two types of participation constraints-

1. Total participation
2. Partial participation

1. Total Participation-

- It specifies that each entity in the entity set must compulsorily participate in at least one relationship instance in that relationship set.
- That is why, it is also called as **mandatory participation.**
- Total participation is represented using a double line between the entity set and relationship set.



Total Participation

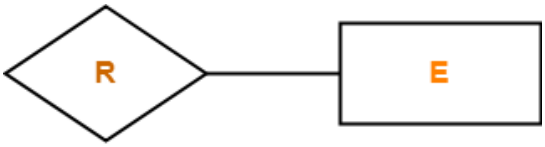
Example-

Here,

- Double line between the entity set "Student" and relationship set "Enrolled in" signifies total participation.
- It specifies that each student must be enrolled in at least one course.

2. Partial Participation-

- It specifies that each entity in the entity set may or may not participate in the relationship instance in that relationship set.
- That is why, it is also called as **optional participation.**
- Partial participation is represented using a single line between the entity set and relationship set.



Partial Participation

Example-

Here,

- Single line between the entity set "Course" and relationship set "Enrolled in" signifies partial participation.
- It specifies that there might exist some courses for which no enrollments are made.

Relationship between Cardinality and Participation Constraints-

Minimum cardinality tells whether the participation is partial or total.

- If minimum cardinality = 0, then it signifies partial participation.
- If minimum cardinality = 1, then it signifies total participation.

Maximum cardinality tells the maximum number of entities that participates in a relationship set.

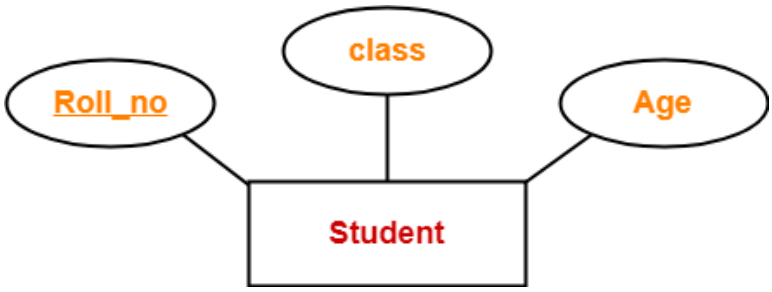
Types of Attributes- In ER diagram, attributes associated with an entity set may be of the following types-

1. Simple attributes
2. Composite attributes

3. Single valued attributes
4. Multi valued attributes
5. Derived attributes
6. Key attributes

1. Simple Attributes- Simple attributes are those attributes which can not be divided further.

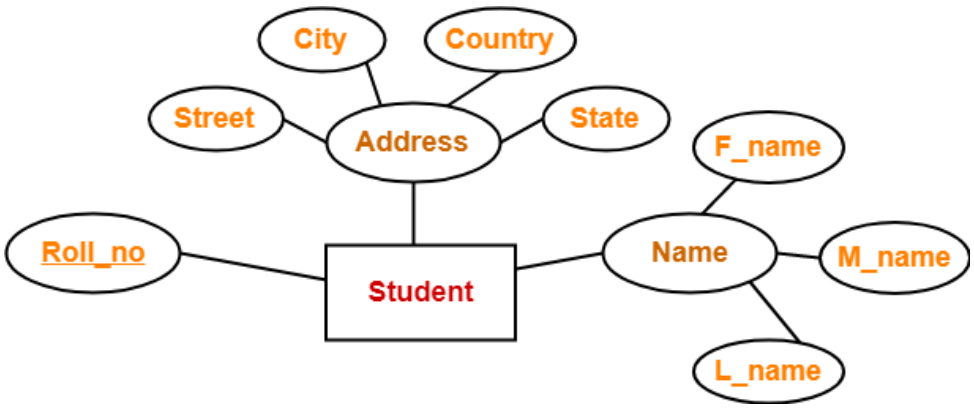
Example-



Here, all the attributes are simple attributes as they can not be divided further.

2. Composite Attributes- Composite attributes are those attributes which are composed of many other simple attributes.

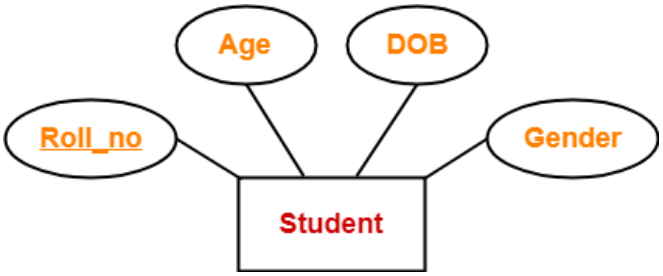
Example-



Here, the attributes “Name” and “Address” are composite attributes as they are composed of many other simple attributes.

3. Single Valued Attributes- Single valued attributes are those attributes which can take only one value for a given entity from an entity set.

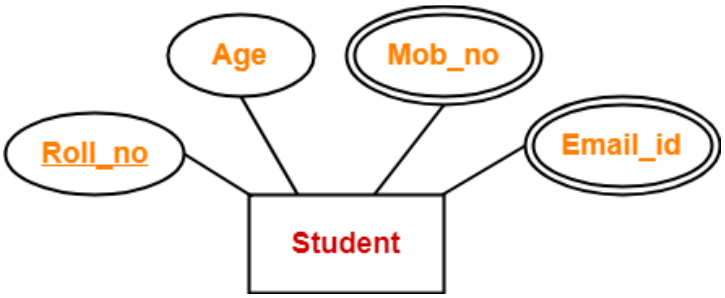
Example-



Here, all the attributes are single valued attributes as they can take only one specific value for each entity.

4. Multi Valued Attributes- Multi valued attributes are those attributes which can take more than one value for a given entity from an entity set.

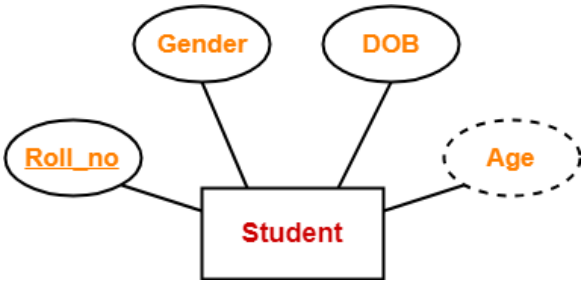
Example-



Here, the attributes "Mob_no" and "Email_id" are multi valued attributes as they can take more than one values for a given entity.

5. Derived Attributes- Derived attributes are those attributes which can be derived from other attribute(s).

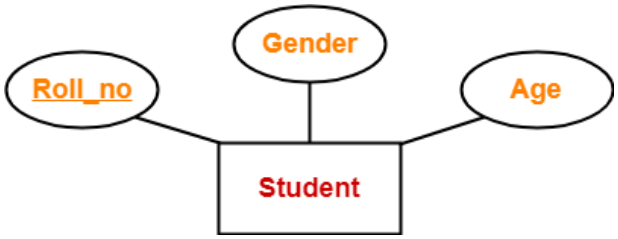
Example-



Here, the attribute "Age" is a derived attribute as it can be derived from the attribute "DOB".

6. Key Attributes- Key attributes are those attributes which can identify an entity uniquely in an entity set.

Example-



Here, the attribute "Roll_no" is a key attribute as it can identify any student uniquely.

Converting ER Diagrams to Tables-

- ER diagram is converted into the tables in relational model.
- This is because relational models can be easily implemented by RDBMS like MySQL , Oracle etc.

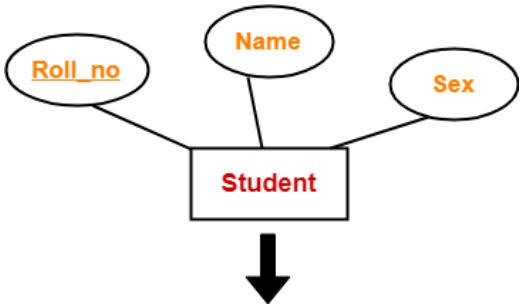
Following rules are used for converting an ER diagram into the tables-

Rule-01: For Strong Entity Set With Only Simple Attributes-

A strong entity set with only simple attributes will require only one table in relational model.

- Attributes of the table will be the attributes of the entity set.
- The primary key of the table will be the key attribute of the entity set.

Example-



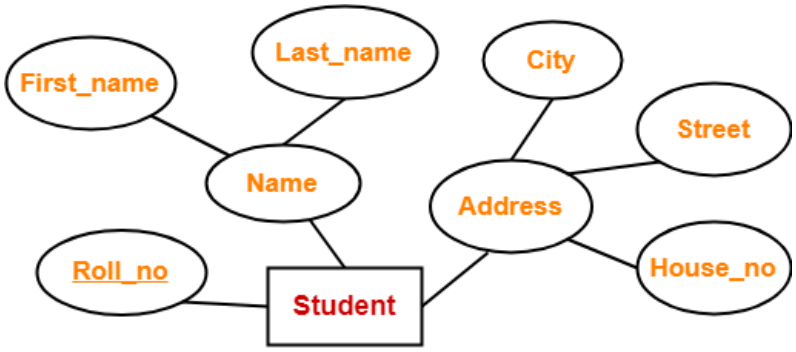
Roll_no	Name	Sex

Schema : Student (Roll_no , Name , Sex)

Rule-02: For Strong Entity Set With Composite Attributes-

- A strong entity set with any number of composite attributes will require only one table in relational model.
- While conversion, simple attributes of the composite attributes are taken into account and not the composite attribute itself.

Example-



Roll_no	First_name	Last_name	House_no	Street	City

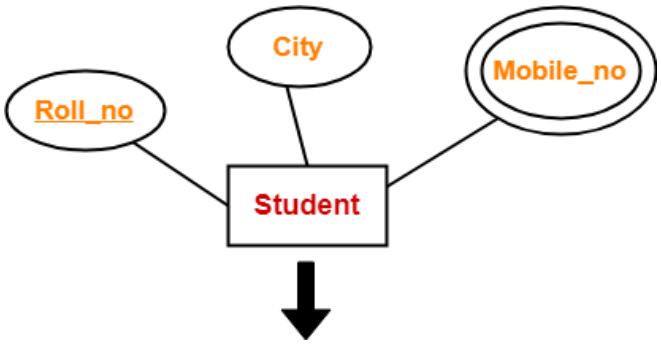
Schema : Student (Roll_no , First_name , Last_name , House_no , Street , City)

Rule-03: For Strong Entity Set With Multi Valued Attributes-

A strong entity set with any number of multi valued attributes will require two tables in relational model.

- One table will contain all the simple attributes with the primary key.
- Other table will contain the primary key and all the multi valued attributes.

Example-



Roll_no	City

Roll_no	Mobile_no

Rule-04: Translating Relationship Set into a Table-

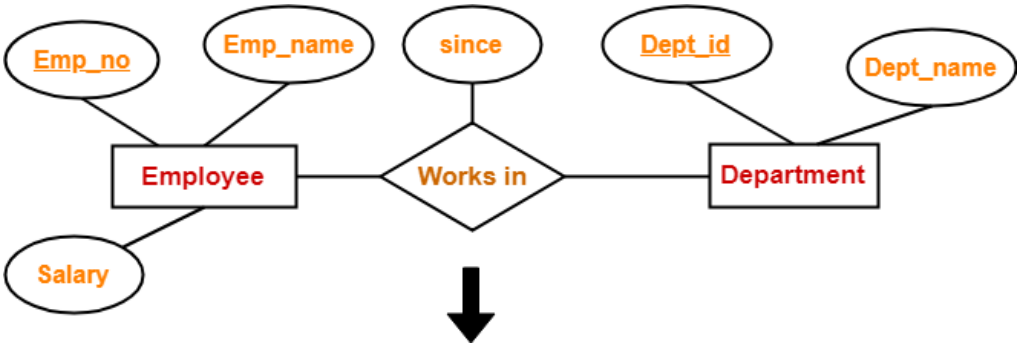
A relationship set will require one table in the relational model.

Attributes of the table are-

- Primary key attributes of the participating entity sets
- Its own descriptive attributes if any.

Set of non-descriptive attributes will be the primary key.

Example-



Emp_no	Dept_id	since

Schema : Works in (Emp_no , Dept_id , since)

NOTE- If we consider the overall ER diagram, three tables will be required in relational model-

- One table for the entity set "Employee"
- One table for the entity set "Department"
- One table for the relationship set "Works in"

Rule-05: For Binary Relationships With Cardinality Ratios-

The following four cases are possible-

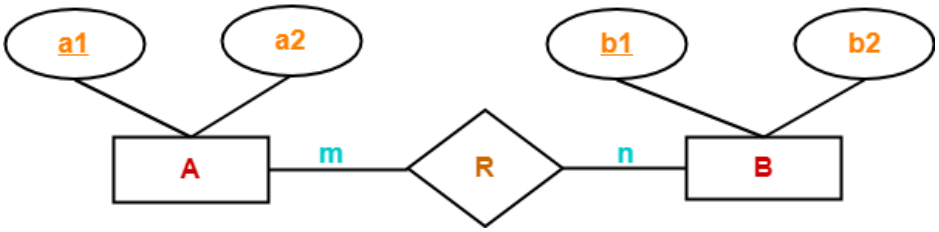
Case-01: Binary relationship with cardinality ratio m:n

Case-02: Binary relationship with cardinality ratio 1:n

Case-03: Binary relationship with cardinality ratio m:1

Case-04: Binary relationship with cardinality ratio 1:1

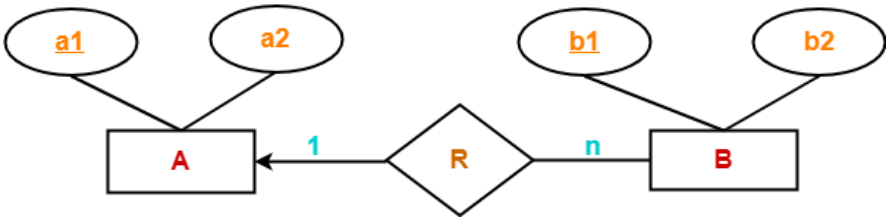
Case-01: For Binary Relationship With Cardinality Ratio m : n



Here, three tables will be required-

1. A (a1 , a2)
2. R (a1 , b1)
3. B (b1 , b2)

Case-02: For Binary Relationship With Cardinality Ratio 1 : n

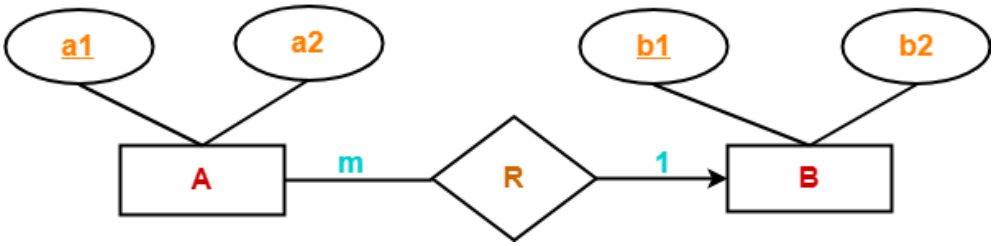


Here, two tables will be required-

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

NOTE- Here, combined table will be drawn for the entity set B and relationship set R.

Case-03: For Binary Relationship With Cardinality Ratio m : 1

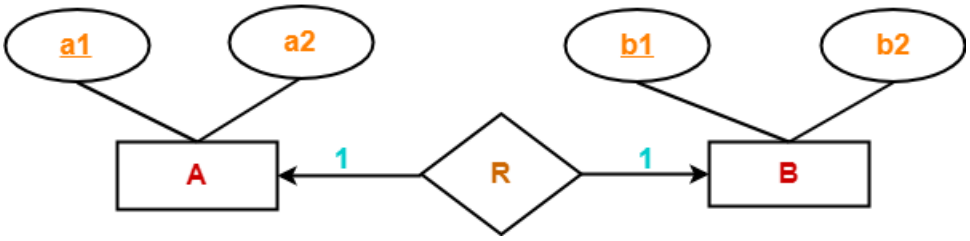


Here, two tables will be required-

1. AR (a1 , a2 , b1)
2. B (b1 , b2)

NOTE- Here, combined table will be drawn for the entity set A and relationship set R.

Case-04: For Binary Relationship With Cardinality Ratio 1 : 1



Here, two tables will be required. Either combine 'R' with 'A' or 'B'

Way-01:

1. AR (a1 , a2 , b1)
2. B (b1 , b2)

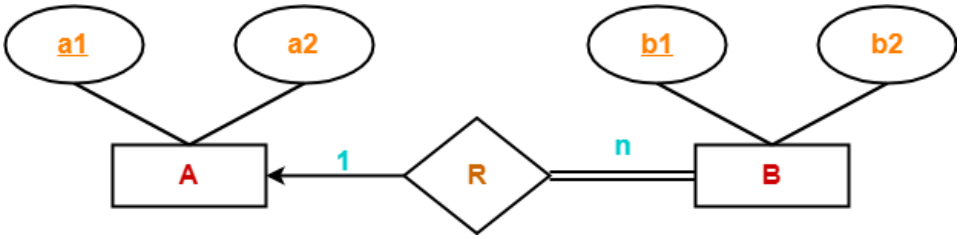
Way-02:

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

Rule-06: For Binary Relationship With Both Cardinality Constraints and Participation Constraints-

- Cardinality constraints will be implemented as discussed in Rule-05.
- Because of the total participation constraint, foreign key acquires **NOT NULL** constraint i.e. now foreign key can not be null.

Case-01: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From One Side-



Because cardinality ratio = 1 : n, so we will combine the entity set B and relationship set R.

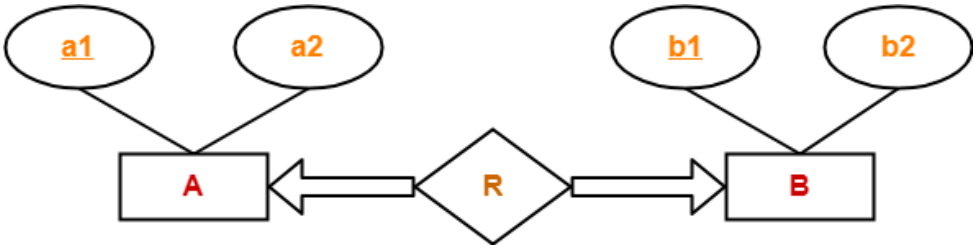
Then, two tables will be required-

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

Because of total participation, foreign key a1 has acquired NOT NULL constraint, so it can't be null now.

Case-02: For Binary Relationship With Cardinality Constraint and Total Participation Constraint From Both Sides-

If there is a key constraint from both the sides of an entity set with total participation, then that binary relationship is represented using only single table.

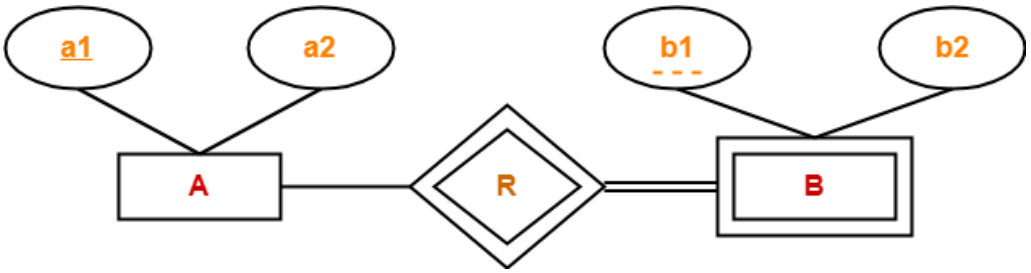


Here, Only one table is required.

- ARB (a1 , a2 , b1 , b2)

Rule-07: For Binary Relationship With Weak Entity Set-

Weak entity set always appears in association with identifying relationship with total participation constraint.

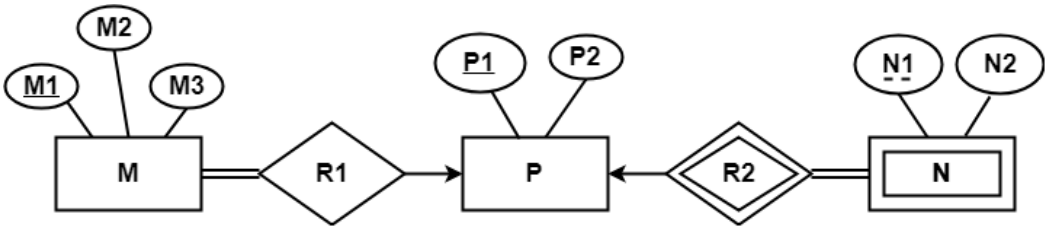


Here, two tables will be required-

1. A (a1 , a2)
2. BR (a1 , b1 , b2)

PRACTICE PROBLEMS BASED ON CONVERTING ER DIAGRAM TO TABLES-

1. Find the minimum number of tables required for the following ER diagram in relational model-



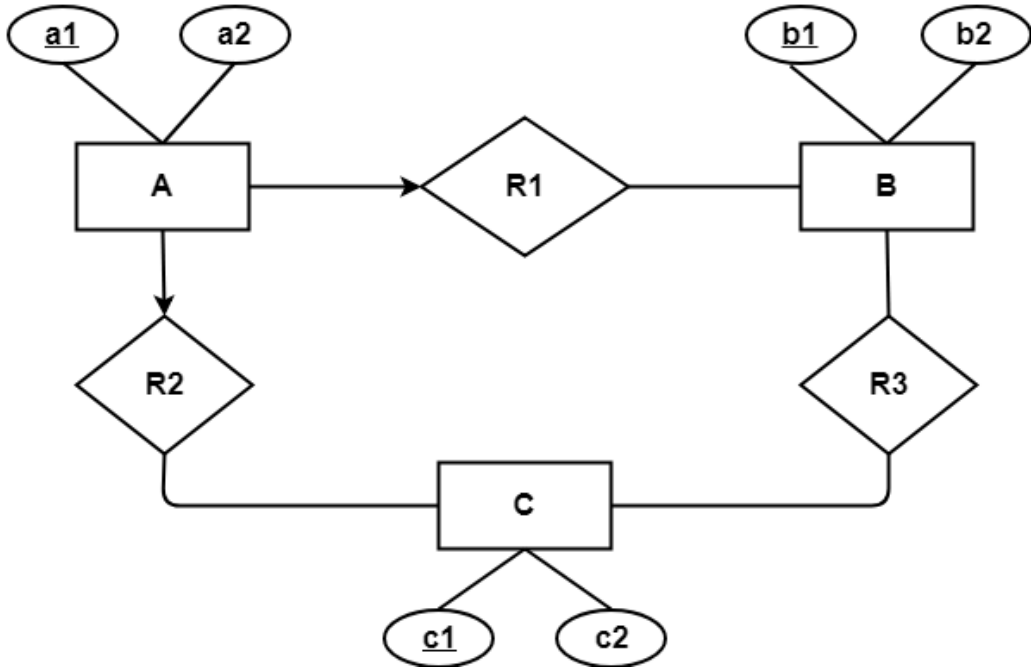
Sol. Applying the rules, minimum 3 tables will be required-

MR1 (M1 , M2 , M3 , P1)

P (P1 , P2)

NR2 (P1 , N1 , N2)

2. Find the minimum number of tables required to represent the given ER diagram in relational model-



Sol. Applying the rules, minimum 4 tables will be required-

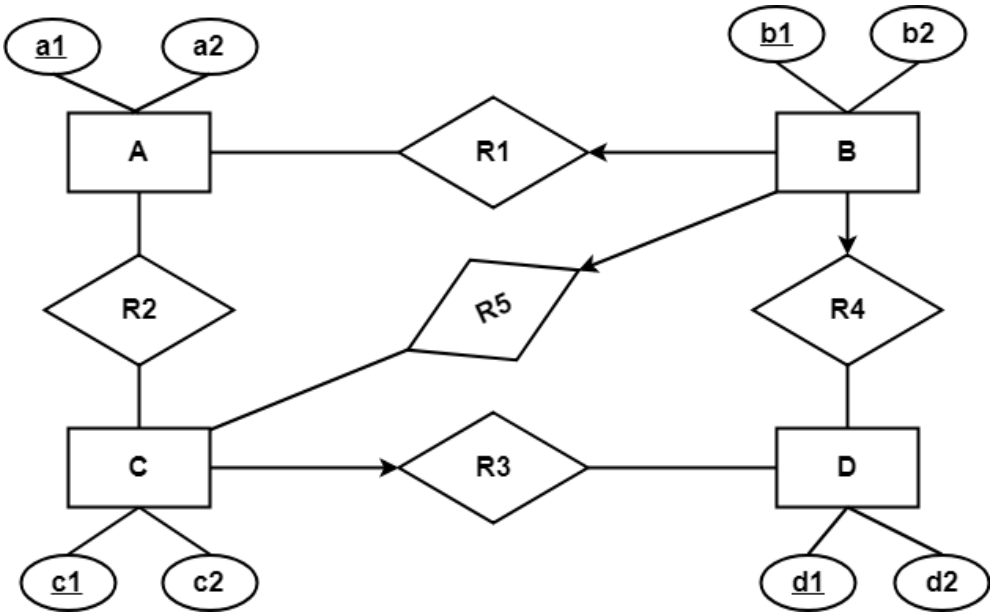
AR1R2 (a1 , a2 , b1 , c1)

B (b1 , b2)

C (c1 , c2)

R3 (b1 , c1)

3. Find the minimum number of tables required to represent the given ER diagram in relational model-

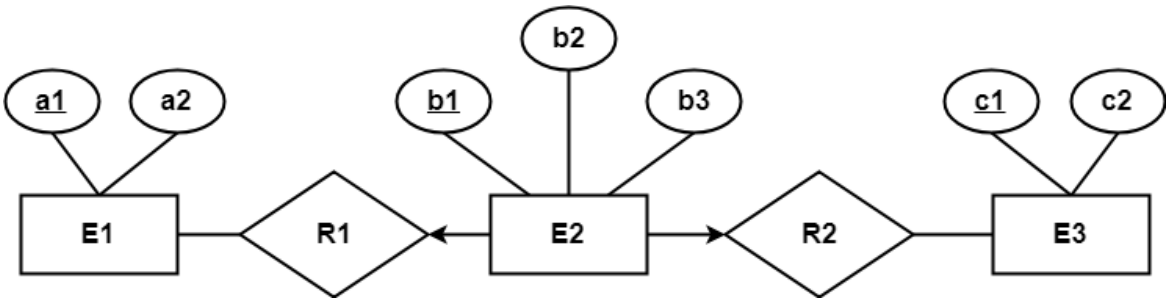


Sol. Applying the rules, minimum 5 tables will be required-

BR1R4R5 (b1 , b2 , a1 , c1 , d1)

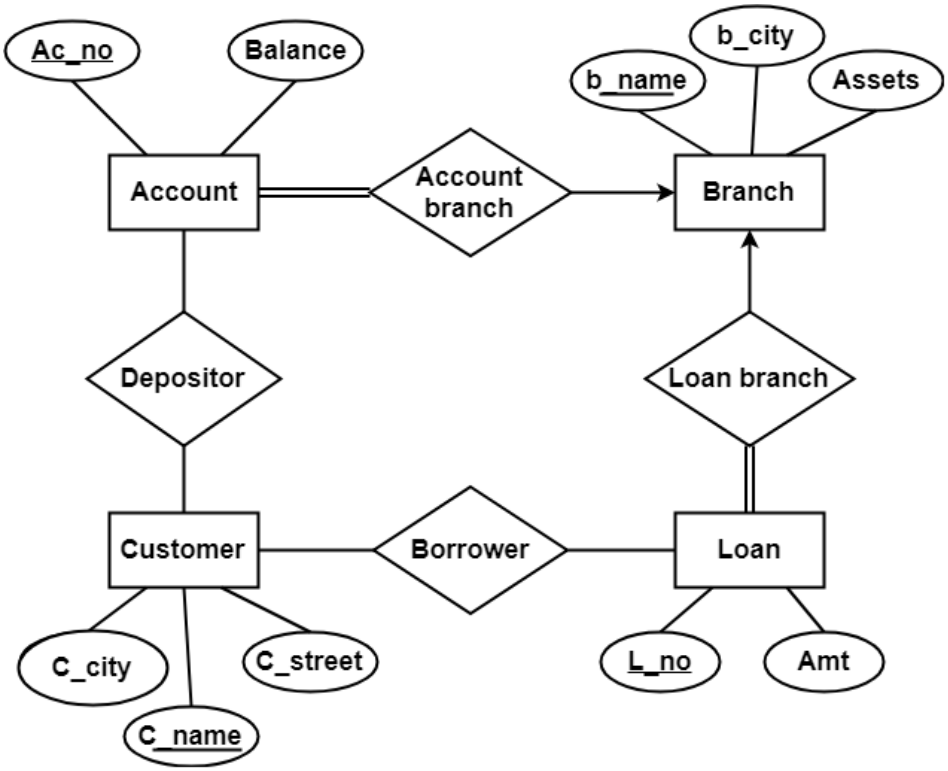
A (a1 , a2)
R2 (a1 , c1)
CR3 (c1 , c2 , d1)
D (d1 , d2)

4. Find the minimum number of tables required to represent the given ER diagram in relational model-



Sol. Applying the rules, minimum 3 tables will be required-
E1 (a1 , a2)
E2R1R2 (b1 , b2 , a1 , c1 , b3)
E3 (c1 , c2)

5. Find the minimum number of tables required to represent the given ER diagram in relational model-



Sol. Applying the rules that we have learnt, minimum 6 tables will be required-

Account (Ac_no , Balance , b_name)
Branch (b_name , b_city , Assets)
Loan (L_no , Amt , b_name)
Borrower (C_name , L_no)
Customer (C_name , C_street , C_city)
Depositor (C_name , Ac_no)

Constraints in DBMS

- Relational constraints are the restrictions imposed on the database contents and operations.
- They ensure the correctness of data in the database.

Types of Constraints in DBMS- In DBMS, there are following 5 different types of relational constraints-

1. Domain constraint
2. Tuple Uniqueness constraint
3. Key constraint
4. Entity Integrity constraint
5. Referential Integrity constraint

1. Domain Constraint-

- Domain constraint defines the domain or set of values for an attribute.
- It specifies that the value taken by the attribute must be the atomic value from its domain.

Example- Consider the following Student table-

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	**A**

Here, value **'A'** is not allowed since only integer values can be taken by the age attribute.

2. Tuple Uniqueness Constraint- Tuple Uniqueness constraint specifies that all the tuples must be necessarily unique in any relation.

Example-01: Consider the following Student table-

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	20

This relation satisfies the tuple uniqueness constraint since here all the tuples are unique.

Example-02: Consider the following Student table-

STU_ID	Name	Age
S001	**Akshay**	**20**
S001	**Akshay**	**20**
S003	Shashank	20
S004	Rahul	20

This relation does not satisfy the tuple uniqueness constraint since here all the tuples are not unique.

3. Key Constraint- Key constraint specifies that in any relation-

- All the values of primary key must be unique.
- The value of primary key must not be null.

Example- Consider the following Student table-

STU_ID	Name	Age
S001	Akshay	20
S001	Abhishek	21
S003	Shashank	20
S004	Rahul	20

This relation does not satisfy the key constraint as here all the values of primary key are not unique.

4. Entity Integrity Constraint-

- Entity integrity constraint specifies that no attribute of primary key must contain a null value in any relation.
- This is because the presence of null value in the primary key violates the uniqueness property.

Example- Consider the following Student table-

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
	Rahul	20

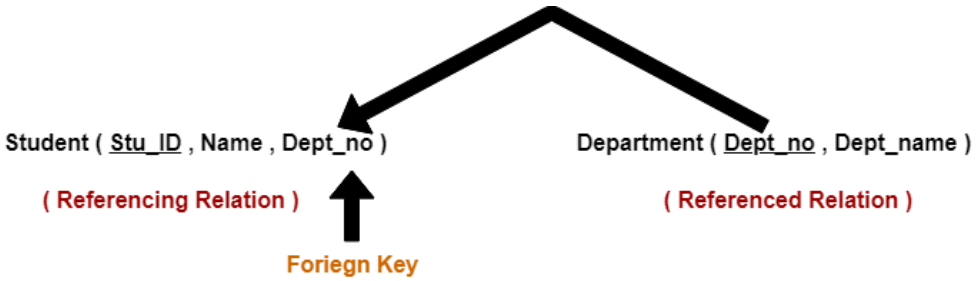
This relation does not satisfy the entity integrity constraint as here the primary key contains a NULL value.

5. Referential Integrity Constraint-

- This constraint is enforced when a foreign key references the primary key of a relation.
- It specifies that all the values taken by the foreign key must either be available in the relation of the primary key or be null.

Example-

Consider the following two relations- 'Student' and 'Department'.
Here, relation 'Student' references the relation 'Department'.



STU_ID	Name	Dept_no
S001	Akshay	D10
S002	Abhishek	D10
S003	Shashank	D11

S004	Rahul	D14

Student

Department

Dept_no	Dept_name
D10	ASET
D11	ALS
D12	ASFL
D13	ASHS

Here,

- The relation 'Student' does not satisfy the referential integrity constraint.
- This is because in relation 'Department', no value of primary key specifies department no. 14.
- Thus, referential integrity constraint is violated.

Handling Violation of Referential Integrity Constraint-

To ensure the correctness of the database, it is important to handle the violation of referential integrity constraint properly.

Referential Integrity Constraint Violation- There are following three possible causes of violation of referential integrity constraint-

Cause-01: Insertion in a referencing relation

Cause-02: Deletion from a referenced relation

Cause-03: Updation in a referenced relation

Cause-01: Insertion in a Referencing Relation-

- It is allowed to insert only those values in the referencing attribute which are already present in the value of the referenced attribute.

- Inserting a value in the referencing attribute which is not present in the value of the referenced attribute violates the referential integrity constraint.

Example- Consider the following two schemas-

Student

Roll_no	Name	Age	Branch_Code
1	Rahul	22	CS
2	Anjali	21	CS
3	Teena	20	IT

Branch

Branch_Code	Branch_Name
CS	Computer Science
EE	Electronics Engineering
IT	Information Technology
CE	Civil Engineering

Here,

- In relation "Student", we can not insert any student having branch code ME (Mechanical Engineering).
- This is because branch code ME is not present in the relation "Branch".

Cause-02: Deletion from a Referenced Relation-

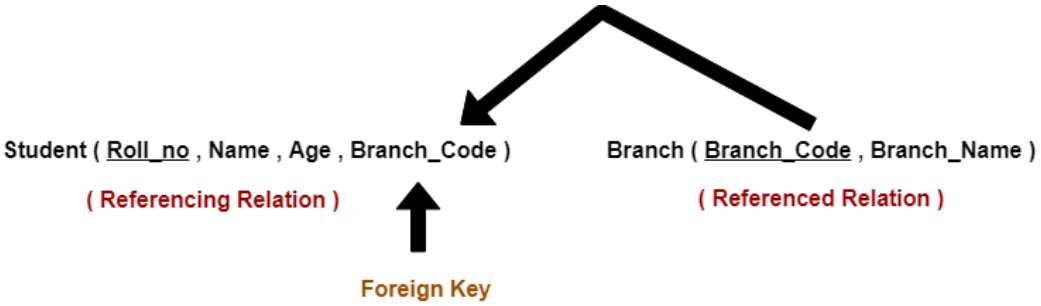
- It is not allowed to delete a row from the referenced relation if the referencing attribute uses the value of the referenced attribute of that row.
- Such a deletion violates the referential integrity constraint.

Example- Consider the above two relations,

- We can not delete a tuple from the relation "Branch" having branch code 'CS'.
- This is because the referencing attribute "Branch_Code" of the referencing relation "Student" references the value 'CS'.

  However, we can safely delete a tuple from the relation "Branch" having branch code 'CE'.
- This is because the referencing attribute "Branch_Code" of the referencing relation "Student" does not uses this value.

Handling the Violation-



Here, relation "Student" references the relation "Branch".

The violation caused due to a deletion from the referenced relation can be handled in the following three ways-

Method-01:

- This method involves simultaneously deleting those tuples from the referencing relation where the referencing attribute uses the value of referenced attribute being deleted.
- This method of handling the violation is called as **On Delete Cascade**.

Method-02:

- This method involves aborting or deleting the request for a deletion from the referenced relation if the value is used by the referencing relation.

Method-03:

- This method involves setting the value being deleted from the referenced relation to NULL or some other value in the referencing relation if the referencing attribute uses that value.

Cause-03: Updation in a Referenced Relation-

- It is not allowed to update a row of the referenced relation if the referencing attribute uses the value of the referenced attribute of that row.
- Such an updation violates the referential integrity constraint.

Example- Consider the above relation.

- We can not update a tuple in the relation "Branch" having branch code 'CS' to the branch code 'CSE
- This is because referencing attribute "Branch_Code" of the referencing relation "Student" references the value 'CS'.

Handling the Violation-

The violation caused due to an updation in the referenced relation can be handled in the following three ways-

Method-01:

- This method involves simultaneously updating those tuples of the referencing relation where the referencing attribute uses the referenced attribute value being updated.
- This method of handling the violation is called as **On Update Cascade**.

Method-02:

- This method involves aborting or deleting the request for an updation of the referenced relation if the value is used by the referencing relation.

Method-03:

- This method involves setting the value being updated in the referenced relation to NULL or some other value in the referencing relation if the referencing attribute uses that value.

Closure of an Attribute Set-

- The set of all those attributes which can be functionally determined from an attribute set is called as a closure of that attribute set.
- Closure of attribute set {X} is denoted as $\{X\}^+$.

Steps to Find Closure of an Attribute Set- Following steps are followed to find the closure of an attribute set-

Step-01: Add the attributes contained in the attribute set for which closure is being calculated to the result set.

Step-02: Recursively add the attributes to the result set which can be functionally determined from the attributes already contained in the result set.

Example- Consider a relation R (A , B , C , D , E , F , G) with the functional dependencies-

$A \rightarrow BC$

$BC \rightarrow DE$

$D \rightarrow F$

$CF \rightarrow G$

Now, let us find the closure of some attributes and attribute sets-

Closure of attribute A-

$A^+ = \{A\}$

$= \{A, B, C\}$ (Using $A \rightarrow BC$)

$= \{A, B, C, D, E\}$ (Using $BC \rightarrow DE$)

$= \{A, B, C, D, E, F\}$ (Using $D \rightarrow F$)

$= \{A, B, C, D, E, F, G\}$ (Using $CF \rightarrow G$)

Thus,

$A^+ = \{A, B, C, D, E, F, G\}$

Closure of attribute D-

$D^+ = \{D\}$

$= \{D, F\}$ (Using $D \rightarrow F$)

We cannot determine any other attribute using attributes D and F contained in the result set.

Thus,

$D^+ = \{D, F\}$

Closure of attribute set {B, C}-

$\{B, C\}^+ = \{B, C\}$

$= \{B, C, D, E\}$ (Using $BC \rightarrow DE$)

$= \{B, C, D, E, F\}$ (Using $D \rightarrow F$)

$= \{B, C, D, E, F, G\}$ (Using $CF \rightarrow G$)

Thus,

$\{B, C\}^+ = \{B, C, D, E, F, G\}$

Finding the Keys Using Closure-

Super Key-

- If the closure result of an attribute set contains all the attributes of the relation, then that attribute set is called as a super key of that relation.
- Thus, we can say-

  "The closure of a super key is the entire relation schema."

Example- In the above example,

- The closure of attribute A is the entire relation schema.
- Thus, attribute A is a super key for that relation.

Candidate Key-

- If there exists no subset of an attribute set whose closure contains all the attributes of the relation, then that attribute set is called as a candidate key of that relation.

Example- In the above example,

- No subset of attribute A contains all the attributes of the relation.
- Thus, attribute A is also a candidate key for that relation.

PRACTICE PROBLEM BASED ON FINDING CLOSURE OF AN ATTRIBUTE SET-

1. Consider the given functional dependencies-

$AB \rightarrow CD$

$AF \rightarrow D$

$DE \rightarrow F$

$C \rightarrow G$

$F \rightarrow E$

$G \rightarrow A$

Which of the following options is false?

(A) $\{CF\}^+ = \{A, C, D, E, F, G\}$ (B) $\{BG\}^+ = \{A, B, C, D, G\}$

(C) $\{AF\}^+ = \{A, C, D, E, F, G\}$ (D) $\{AB\}^+ = \{A, C, D, F, G\}$

Sol. Let us check each option one by one-

Option-(A):

$\{CF\}^+ = \{C, F\}$

$= \{C, F, G\}$ (Using $C \rightarrow G$)

$= \{C, E, F, G\}$ (Using $F \rightarrow E$)

$= \{A, C, E, E, F\}$ (Using $G \rightarrow A$)

$= \{A, C, D, E, F, G\}$ (Using $AF \rightarrow D$)

Since, our obtained result set is same as the given result set, so, it means it is correctly given.

Option-(B):

$\{BG\}^+ = \{B, G\}$

$= \{A, B, G\}$ (Using $G \rightarrow A$)

$= \{A, B, C, D, G\}$ (Using $AB \rightarrow CD$)

Since, our obtained result set is same as the given result set, so, it means it is correctly given.

Option-(C):

$\{AF\}^+ = \{A, F\}$

$= \{A, D, F\}$ (Using $AF \rightarrow D$)

$= \{A, D, E, F\}$ (Using $F \rightarrow E$)

Since, our obtained result set is different from the given result set, so,it means it is not correctly given.

Option-(D):

$\{AB\}^+ = \{A, B\}$

$= \{A, B, C, D\}$ (Using $AB \rightarrow CD$)

$= \{A, B, C, D, G\}$ (Using $C \rightarrow G$)

Since, our obtained result set is different from the given result set, so,it means it is not correctly given.

Thus,

Option (C) and Option (D) are correct.

Keys in DBMS- A key is a set of attributes that can identify each tuple uniquely in the given relation.

Different Types Of Keys in DBMS- There are following 10 important keys in DBMS-

1. Super key
2. Candidate key
3. Primary key
4. Alternate key
5. Foreign key
6. Partial key
7. Composite key
8. Unique key
9. Surrogate key
10. Secondary key

NOTE- Before proceeding further, Kindly note-

- The terms 'relation' and 'table' are used interchangeably.
- The terms 'tuple' and 'record' are used interchangeably.

So, don't get confused!

1. Super Key-

- A super key is a set of attributes that can identify each tuple uniquely in the given relation.
- A super key is not restricted to have any specific number of attributes.
- Thus, a super key may consist of any number of attributes.

Example- Consider the following Student schema-

Student (roll, name, sex, age, address, class, section)

Given below are the examples of super keys since each set can uniquely identify each student in the Student table-

- (roll, name, sex, age, address, class, section)
- (class, section, roll)
- (class, section, roll, sex)
- (name, address)

NOTE- All the attributes in a super key are definitely sufficient to identify each tuple uniquely in the given relation but all of them may not be necessary.

2. Candidate Key- A minimal super key is called as a candidate key.

OR

A set of minimal attribute(s) that can identify each tuple uniquely in the given relation is called
as a candidate key.

Example- Consider the following Student schema-

Student (roll , name , sex , age , address , class , section)

Given below are the examples of candidate keys since each set consists of minimal attributes required to identify each student uniquely in the Student table-

- (class, section, roll)
- (name, address)

NOTES-

- All the attributes in a candidate key are sufficient as well as necessary to identify each tuple uniquely.
- Removing any attribute from the candidate key fails in identifying each tuple uniquely.
- The value of candidate key must always be unique.
- The value of candidate key can never be NULL.
- It is possible to have multiple candidate keys in a relation.
- Those attributes which appears in some candidate key are called as **prime attributes**.

3. Primary Key- A primary key is a candidate key that the database designer selects while designing the database.

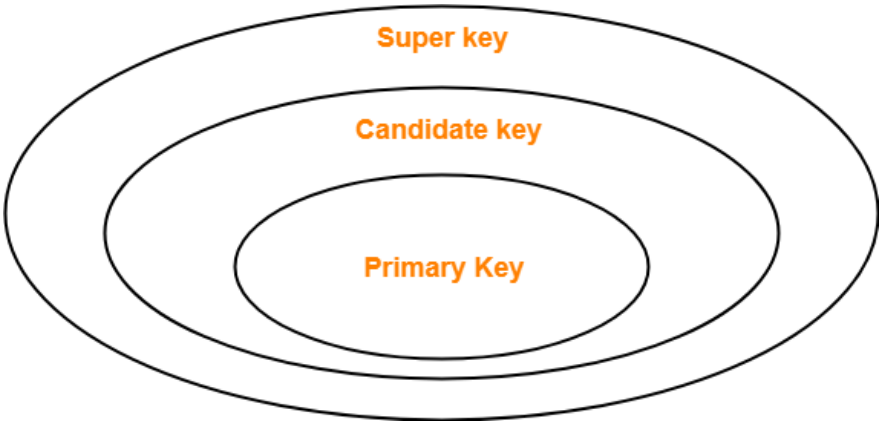
OR

Candidate key that the database designer implements is called as a primary key.

NOTES-

- The value of primary key can never be NULL.
- The value of primary key must always be unique.
- The values of primary key can never be changed i.e. no updation is possible.
- The value of primary key must be assigned when inserting a record.
- A relation is allowed to have only one primary key.

Remember-



4. Alternate Key- Candidate keys that are left unimplemented or unused after implementing the primary key are called as alternate keys.

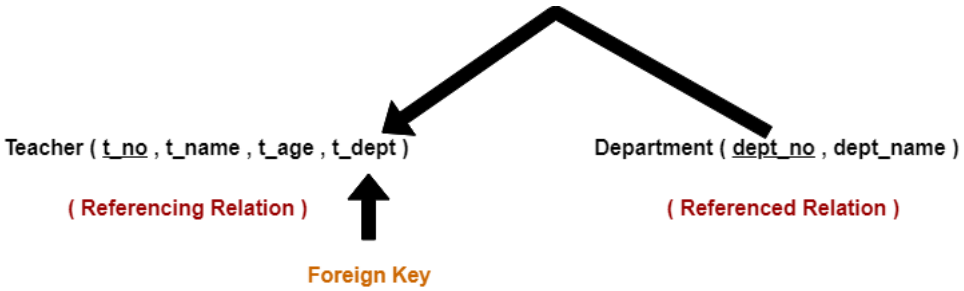
OR

Unimplemented candidate keys are called as alternate keys.

5. Foreign Key-

- An attribute 'X' is called as a foreign key to some other attribute 'Y' when its values are dependent on the values of attribute 'Y'.
- The attribute 'X' can assume only those values which are assumed by the attribute 'Y'.
- Here, the relation in which attribute 'Y' is present is called as the **referenced relation**.
- The relation in which attribute 'X' is present is called as the **referencing relation**.
- The attribute 'Y' might be present in the same table or in some other table.

Example- Consider the following two schemas-



Here, t_dept can take only those values which are present in dept_no in Department table since only those departments actually exist.

NOTES-

- Foreign key references the primary key of the table.
- Foreign key can take only those values which are present in the primary key of the referenced relation.
- Foreign key may have a name other than that of a primary key.
- Foreign key can take the NULL value.
- There is no restriction on a foreign key to be unique.
- In fact, foreign key is not unique most of the time.
- Referenced relation may also be called as the master table or primary table.
- Referencing relation may also be called as the foreign table.

6. Partial Key-

- Partial key is a key using which all the records of the table cannot be identified uniquely.
- However, a bunch of related tuples can be selected from the table using the partial key.

Example- Consider the following schema-

Department (Emp_no , Dependent_name , Relation)

Emp_no	Dependent_name	Relation
E1	Suman	Mother
E1	Ajay	Father
E2	Vijay	Father
E2	Ankush	Son

Here, using partial key Emp_no, we can not identify a tuple uniquely but we can select a bunch of tuples from the table.

7. Composite Key- **A** primary key comprising of multiple attributes and not just a single attribute is called as a composite key.

8. Unique Key- Unique key is a key with the following properties-

- It is unique for all the records of the table.
- Once assigned, its value cannot be changed i.e. it is non-updatable.
- It may have a NULL value.

Example- The best example of unique key is Adhaar Card Numbers.

- The Adhaar Card Number is unique for all the citizens (tuples) of India (table).
- If it gets lost and another duplicate copy is issued, then the duplicate copy always has the same number as before.
- Thus, it is non-updatable.
- Few citizens may not have got their Adhaar cards, so for them its value is NULL.

9. Surrogate Key- Surrogate key is a key with the following properties-

- It is unique for all the records of the table.
- It is updatable.
- It cannot be NULL i.e. it must have some value.

Example- Mobile Number of students in a class where every student owns a mobile phone.

10. Secondary Key- Secondary key is required for the indexing purpose for better and faster searching.

Functional Dependency- In any relation, a functional dependency α → β holds if-

Two tuples having same value of attribute α also have same value for attribute β.

Mathematically,

If α and β are the two sets of attributes in a relational table R where-

- α ⊆ R
- β ⊆ R

Then, for a functional dependency to exist from α to β,

If t1[α] = t2[α], then t1[β] = t2[β]

α	β
t1[α]	t1[β]
t2[α]	t2[β]
…….	…….

f_d : α → β

Types Of Functional Dependencies- There are two types of functional dependencies-

1. Trivial Functional Dependencies
2. Non-trivial Functional Dependencies

1. Trivial Functional Dependencies-

- A functional dependency X → Y is said to be trivial if and only if Y ⊆ X.
- Thus, if RHS of a functional dependency is a subset of LHS, then it is called as a trivial functional dependency.

Examples- The examples of trivial functional dependencies are-

- AB → A
- AB → B
- AB → AB

2. Non-Trivial Functional Dependencies-

- A functional dependency X → Y is said to be non-trivial if and only if Y ⊄ X.
- Thus, if there exists at least one attribute in the RHS of a functional dependency that is not a part of LHS, then it is called as a non-trivial functional dependency.

Examples- The examples of non-trivial functional dependencies are-

- AB → BC
- AB → CD

Inference Rules-

Reflexivity- If B is a subset of A, then A → B always holds.
Transitivity- If A → B and B → C, then A → C always holds.
Augmentation- If A → B, then AC → BC always holds.
Decomposition- If A → BC, then A → B and A → C always holds.
Composition- If A → B and C → D, then AC → BD always holds.
Additive- If A → B and A → C, then A → BC always holds.

Rules for Functional Dependency-

Rule-01: A functional dependency X → Y will always hold if all the values of X are unique (different) irrespective of the values of Y.

Example- Consider the following table-

A	B	C	D	E
5	4	3	2	2
8	5	3	2	1
1	9	3	3	5
4	7	3	3	8

The following functional dependencies will always hold since all the values of attribute 'A' are unique-

- A → B
- A → BC
- A → CD
- A → BCD
- A → DE
- A → BCDE

In general, we can say following functional dependency will always hold-

A → Any combination of attributes A, B, C, D, E

Similar will be the case for attributes B and E.

Rule-02: A functional dependency X → Y will always hold if all the values of Y are same irrespective of the values of X.

Example- Consider the following table-

A	B	C	D	E

5	4	3	2	2
8	5	3	2	1
1	9	3	3	5
4	7	3	3	8

The following functional dependencies will always hold since all the values of attribute 'C' are same-

- A → C
- AB → C
- ABDE → C
- DE → C
- AE → C

In general, we can say following functional dependency will always hold true-

Any combination of attributes A, B, C, D, E → C

Combining Rule-01 and Rule-02 we can say-

In general, a functional dependency α → β always holds-

If either all values of α are unique or if all values of β are same or both.

Rule-03: For a functional dependency X → Y to hold, if two tuples in the table agree on the value of attribute X, then they must also agree on the value of attribute Y.

Rule-04: For a functional dependency X → Y, violation will occur only when for two or more same values of X, the corresponding Y values are different.

Equivalence of Two Sets of Functional Dependencies-

In DBMS,

- Two different sets of functional dependencies for a given relation may or may not be equivalent.
- If F and G are the two sets of functional dependencies, then following 3 cases are possible-

Case-01: F covers G (F ⊇ G)

Case-02: G covers F (G ⊇ F)

Case-03: Both F and G cover each other (F = G)

Case-01: Determining Whether F Covers G- Following steps are followed to determine whether F covers G or not-

Step-01: Take the functional dependencies of set G into consideration.

- For each functional dependency X → Y, find the closure of X using the functional dependencies of set G.

Step-02: Take the functional dependencies of set G into consideration.

For each functional dependency X → Y, find the closure of X using the functional dependencies of set F.

Step-03: Compare the results of Step-01 and Step-02.

If the functional dependencies of set F has determined all those attributes that were determined by the functional dependencies of set G, then it means F covers G.

Thus, we conclude F covers G (F ⊇ G) otherwise not.

Case-02: Determining Whether G Covers F- Following steps are followed to determine whether G covers F or not-

Step-01: Take the functional dependencies of set F into consideration.

For each functional dependency X → Y, find the closure of X using the functional dependencies of set F.

Step-02: Take the functional dependencies of set F into consideration.

For each functional dependency X → Y, find the closure of X using the functional dependencies of set G.

Step-03: Compare the results of Step-01 and Step-02.

If the functional dependencies of set G has determined all those attributes that were determined by the functional dependencies of set F, then it means G covers F.

Thus, we conclude G covers F (G ⊇ F) otherwise not.

Case-03: Determining Whether Both F and G Cover Each Other-

- If F covers G and G covers F, then both F and G cover each other.
- Thus, if both the above cases hold true, we conclude both F and G cover each other (F = G).

PRACTICE PROBLEM BASED ON EQUIVALENCE OF FUNCTIONAL DEPENDENCIES-

1. A relation R (A , C , D , E , H) is having two functional dependencies sets F and G as shown-

Set F-
A → C
AC → D
E → AD
E → H

Set G-
A → CD
E → AH

Which of the following holds true?

(A) G ⊇ F (B) F ⊇ G (C) F = G (D) All of the above

Sol. **Determining whether F covers G-**

Step-01:

- $(A)^+$ = { A , C , D } // closure of left side of A → CD using set G
- $(E)^+$ = { A , C , D , E , H } // closure of left side of E → AH using set G

Step-02:

- $(A)^+$ = { A , C , D } // closure of left side of A → CD using set F
- $(E)^+$ = { A , C , D , E , H } // closure of left side of E → AH using set F

Step-03: Comparing the results of Step-01 and Step-02, we find-

- Functional dependencies of set F can determine all the attributes which have been determined by the functional dependencies of set G.
- Thus, we conclude F covers G i.e. F ⊇ G.

Determining whether G covers F-

Step-01:

- $(A)^+$ = { A , C , D } // closure of left side of A → C using set F
- $(AC)^+$ = { A , C , D } // closure of left side of AC → D using set F
- $(E)^+$ = { A , C , D , E , H } // closure of left side of E → AD and E → H using set F

Step-02:

- $(A)^+$ = { A , C , D } // closure of left side of A → C using set G
- $(AC)^+$ = { A , C , D } // closure of left side of AC → D using set G
- $(E)^+$ = { A , C , D , E , H } // closure of left side of E → AD and E → H using set G

Step-03: Comparing the results of Step-01 and Step-02, we find-

- Functional dependencies of set G can determine all the attributes which have been determined by the functional dependencies of set F.
- Thus, we conclude G covers F i.e. G ⊇ F.

Determining whether both F and G cover each other-

- From Step-01, we conclude F covers G.
- From Step-02, we conclude G covers F.
- Thus, we conclude both F and G cover each other i.e. F = G.

Thus, Option (D) is correct.

Characteristics-

- Canonical cover is free from all the extraneous functional dependencies.
- The closure of canonical cover is same as that of the given set of functional dependencies.
- Canonical cover is not unique and may be more than one for a given set of functional dependencies.

Need-

- Working with the set containing extraneous functional dependencies increases the computation time.
- Therefore, the given set is reduced by eliminating the useless functional dependencies.
- This reduces the computation time and working with the irreducible set becomes easier.

Steps To Find Canonical Cover-

Step-01: Write the given set of functional dependencies in such a way that each functional dependency contains exactly one attribute on its right side.

Example- The functional dependency $X \rightarrow YZ$ will be written as-

$X \rightarrow Y$

$X \rightarrow Z$

Step-02: Consider each functional dependency one by one from the set obtained in Step-01. Determine whether it is essential or non-essential.

To determine whether a functional dependency is essential or not, compute the closure of its left side-

- Once by considering that the particular functional dependency is present in the set
- Once by considering that the particular functional dependency is not present in the set

Then following two cases are possible-

Case-01: Results Come Out to be Same-

If results come out to be same,

- It means that the presence or absence of that functional dependency does not create any difference.
- Thus, it is non-essential.
- Eliminate that functional dependency from the set.

NOTE- Eliminate the non-essential functional dependency from the set as soon as it is discovered.

Do not consider it while checking the essentiality of other functional dependencies.

Case-01: Results Come Out to be Different-

If results come out to be different,

- It means that the presence or absence of that functional dependency creates a difference

- Thus, it is essential.
- Do not eliminate that functional dependency from the set.
- Mark that functional dependency as essential.

Step-03:

- Consider the newly obtained set of functional dependencies after performing Step-02.
- Check if there is any functional dependency that contains more than one attribute on its left side.

Then following two cases are possible-

Case-01: No-

- There exists no functional dependency containing more than one attribute on its left side.
- In this case, the set obtained in Step-02 is the canonical cover.

Case-01: Yes-

- There exists at least one functional dependency containing more than one attribute on its left side.
- In this case, consider all such functional dependencies one by one.
- Check if their left side can be reduced.

Use the following steps to perform a check-

- Consider a functional dependency.
- Compute the closure of all the possible subsets of the left side of that functional dependency.
- If any of the subsets produce the same closure result as produced by the entire left side, then replace the left side with that subset.

After this step is complete, the set obtained is the canonical cover.

PRACTICE PROBLEM BASED ON FINDING CANONICAL COVER-

1. The following functional dependencies hold true for the relational scheme R (W, X, Y, Z) –

$X \rightarrow W$

$WZ \rightarrow XY$

$Y \rightarrow WXZ$

Write the irreducible equivalent for this set of functional dependencies.

Sol. **Step-01:**

Write all the functional dependencies such that each contains exactly one attribute on its right side-

X → W
WZ → X
WZ → Y
Y → W
Y → X
Y → Z

Step-02: Check the essentiality of each functional dependency one by one.

For X → W:

- Considering X → W, $(X)^+ = \{X, W\}$
- Ignoring X → W, $(X)^+ = \{X\}$

Now,

- Clearly, the two results are different.
- Thus, we conclude that X → W is essential and can not be eliminated.

For WZ → X:

- Considering WZ → X, $(WZ)^+ = \{W, X, Y, Z\}$
- Ignoring WZ → X, $(WZ)^+ = \{W, X, Y, Z\}$

Now,

- Clearly, the two results are same.
- Thus, we conclude that WZ → X is non-essential and can be eliminated.

Eliminating WZ → X, our set of functional dependencies reduces to-

X → W
WZ → Y
Y → W
Y → X
Y → Z

Now, we will consider this reduced set in further checks.

For WZ → Y:

- Considering WZ → Y, $(WZ)^+ = \{W, X, Y, Z\}$
- Ignoring WZ → Y, $(WZ)^+ = \{W, Z\}$

Now,

- Clearly, the two results are different.
- Thus, we conclude that WZ → Y is essential and can not be eliminated.

For Y → W:

- Considering Y → W, $(Y)^+$ = {W, X, Y, Z}
- Ignoring Y → W, $(Y)^+$ = {W, X, Y, Z}

Now,

- Clearly, the two results are same.
- Thus, we conclude that Y → W is non-essential and can be eliminated.

Eliminating Y → W, our set of functional dependencies reduces to-

X → W
WZ → Y
Y → X
Y → Z

For Y → X:

- Considering Y → X, $(Y)^+$ = {W, X, Y, Z}
- Ignoring Y → X, $(Y)^+$ = {Y, Z}

Now,

- Clearly, the two results are different.
- Thus, we conclude that Y → X is essential and cannot be eliminated.

For Y → Z:

- Considering Y → Z, $(Y)^+$ = {W, X, Y, Z}
- Ignoring Y → Z, $(Y)^+$ = {W, X, Y}

Now,

- Clearly, the two results are different.
- Thus, we conclude that Y → Z is essential and cannot be eliminated.

From here, our essential functional dependencies are-

X → W
WZ → Y
Y → X
Y → Z

Step-03:

- Consider the functional dependencies having more than one attribute on their left side.
- Check if their left side can be reduced.

In our set,

- Only WZ → Y contains more than one attribute on its left side.
- Considering WZ → Y, $(WZ)^+$ = { W , X , Y , Z }

Now,

- Consider all the possible subsets of WZ.
- Check if the closure result of any subset matches to the closure result of WZ.

$(W)^+ = \{W\}$

$(Z)^+ = \{Z\}$

Clearly,

- None of the subsets have the same closure result same as that of the entire left side.
- Thus, we conclude that we cannot write WZ → Y as W → Y or Z → Y.
- Thus, set of functional dependencies obtained in step-02 is the canonical cover.

Finally, the canonical cover is-

X → W

WZ → Y

Y → X

Y → Z

Canonical Cover

Decomposition of a Relation- The process of breaking up or dividing a single relation into two or more sub relations is called as decomposition of a relation.

Properties of Decomposition- The following two properties must be followed when decomposing a given relation-

1. Lossless decomposition- Lossless decomposition ensures-

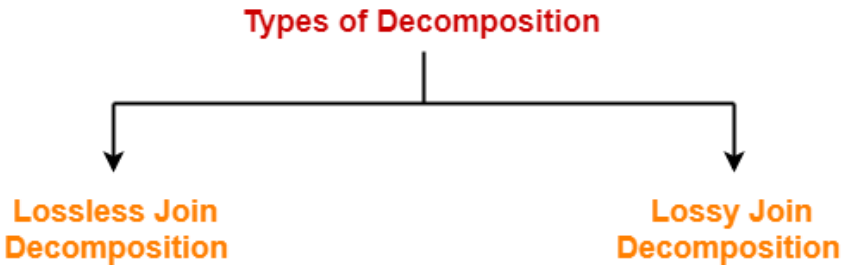
- No information is lost from the original relation during decomposition.
- When the sub relations are joined back, the same relation is obtained that was decomposed.

Every decomposition must always be lossless.

2. Dependency Preservation- Dependency preservation ensures-

- None of the functional dependencies that holds on the original relation are lost.
- The sub relations still hold or satisfy the functional dependencies of the original relation.

Types of Decomposition- Decomposition of a relation can be completed in the following two ways-



1. Lossless Join Decomposition-

- Consider there is a relation R which is decomposed into sub relations R_1, R_2, …. , R_n.
- This decomposition is called lossless join decomposition when the join of the sub relations results in the same relation R that was decomposed.
- For lossless join decomposition, we always have-

$$R_1 \bowtie R_2 \bowtie R_3 \ldots\ldots \bowtie R_n = R$$

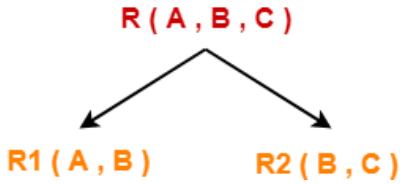
where ⋈ is a natural join operator

Example- Consider the following relation R(A, B, C)-

A	B	C
1	2	1
2	5	3
3	3	3

R(A, B, C)

Consider this relation is decomposed into two sub relations R_1(A, B) and R_2(B, C)-



The two sub relations are-

A	B
1	2
2	5

3	3

R_1(A , B)

B	C
2	1
5	3
3	3

R_2(B , C)

Now, let us check whether this decomposition is lossless or not.

For lossless decomposition, we must have-

$$R_1 \bowtie R_2 = R$$

Now, if we perform the natural join (⋈) of the sub relations R_1 and R_2, we get-

A	B	C
1	2	1
2	5	3
3	3	3

This relation is same as the original relation R.

Thus, we conclude that the above decomposition is lossless join decomposition.

NOTE-

- Lossless join decomposition is also known as **non-additive join decomposition.**
- This is because the resultant relation after joining the sub relations is same as the decomposed relation.
- No extraneous tuples appear after joining of the sub-relations.

2. Lossy Join Decomposition-

- Consider there is a relation R which is decomposed into sub relations R_1, R_2, …. , R_n.
- This decomposition is called lossy join decomposition when the join of the sub relations does not result in the same relation R that was decomposed.
- The natural join of the sub relations is always found to have some extraneous tuples.
- For lossy join decomposition, we always have-

  $R_1 \bowtie R_2 \bowtie R_3 \ldots\ldots \bowtie R_n \supset R$

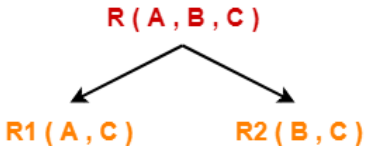
  where $\bowtie$ is a natural join operator

Example- Consider the following relation R(A, B, C)-

A	B	C
1	2	1
2	5	3
3	3	3

R(A, B, C)

Consider this relation is decomposed into two sub relations as R_1(A, C) and R_2(B, C)-



The two sub relations are-

A	C
1	1
2	3
3	3

R_1(A, B)

B	C
2	1
5	3
3	3

R_2(B, C)

Now, let us check whether this decomposition is lossy or not.

For lossy decomposition, we must have-

$\mathbf{R_1 \bowtie R_2 \supset R}$

Now, if we perform the natural join ($\bowtie$) of the sub relations R_1 and R_2 we get-

A	B	C
1	2	1
2	5	3
2	3	3
3	5	3
3	3	3

This relation is not same as the original relation R and contains some extraneous tuples.

Clearly, $R_1 \bowtie R_2 \supset R$.

Thus, we conclude that the above decomposition is lossy join decomposition.

NOTE-

- Lossy join decomposition is also known as **careless decomposition.**
- This is because extraneous tuples get introduced in the natural join of the sub-relations.
- Extraneous tuples make the identification of the original tuples difficult.

Determining Whether Decomposition Is Lossless Or Lossy- Consider a relation R is decomposed into two sub relations R_1 and R_2.

Then,

- If all the following conditions satisfy, then the decomposition is lossless.
- If any of these conditions fail, then the decomposition is lossy.

Condition-01: Union of both the sub relations must contain all the attributes that are present in the original relation R.

Thus, $\mathbf{R_1 \cup R_2 = R}$

Condition-02: Intersection of both the sub relations must not be null.

In other words, there must be some common attribute which is present in both the sub relations.

Thus, $\mathbf{R1 \cap R2 \neq \emptyset}$

Condition-03: Intersection of both the sub relations must be a super key of either R_1 or R_2 or both.

Thus, **$R_1 \cap R_2$ = Super key of R_1 or R_2**

PRACTICE PROBLEMS BASED ON DETERMINING WHETHER DECOMPOSITION IS LOSSLESS OR LOSSY-

1. Consider a relation schema R (A, B, C, D) with the functional dependencies A → B and C → D. Determine whether the decomposition of R into R_1 (A, B) and R_2 (C, D) is lossless or lossy.

Sol. To determine whether the decomposition is lossless or lossy,

- We will check all the conditions one by one.
- If any of the conditions fail, then the decomposition is lossy otherwise lossless.

Condition-01: According to condition-01, union of both the sub relations must contain all the attributes of relation R.

So, we have-

R_1 (A, B) ∪ R_2 (C, D)

= R (A, B, C, D)

Clearly, union of the sub relations contain all the attributes of relation R.

Thus, condition-01 satisfies.

Condition-02: According to condition-02, intersection of both the sub relations must not be null.

So, we have-

R_1 (A, B) ∩ R_2 (C, D)

= Φ

Clearly, intersection of the sub relations is null.

So, condition-02 fails.

Thus, we conclude that the decomposition is lossy.

2. Consider a relation schema R (A, B, C, D) with the following functional dependencies-

A → B

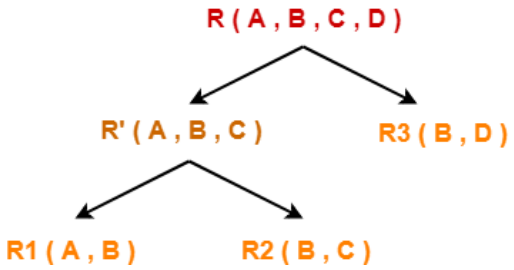
B → C

C → D

D → B

Determine whether the decomposition of R into R_1 (A, B), R_2 (B, C) and R_3 (B, D) is lossless or lossy.

Sol. Consider the original relation R was decomposed into the given sub relations as shown-



Decomposition of R(A, B, C, D) into R'(A, B, C) and R_3(B, D)- To determine whether the decomposition is lossless or lossy,

- We will check all the conditions one by one.
- If any of the conditions fail, then the decomposition is lossy otherwise lossless.

Condition-01: According to condition-01, union of both the sub relations must contain all the attributes of relation R.

So, we have-

R' (A, B, C) ∪ R_3 (B, D)

= R (A, B, C, D)

Clearly, union of the sub relations contain all the attributes of relation R.

Thus, condition-01 satisfies.

Condition-02: According to condition-02, intersection of both the sub relations must not be null.

So, we have-

R' (A, B, C) ∩ R_3 (B, D)

= B

Clearly, intersection of the sub relations is not null.

Thus, condition-02 satisfies.

Condition-03: According to condition-03, intersection of both the sub relations must be the super key of one of the two sub relations or both.

So, we have-

R' (A, B, C) ∩ R_3 (B, D)

= B

Now, the closure of attribute B is-

B^+ = {B, C, D}

Now, we see-

- Attribute 'B' cannot determine attribute 'A' of sub relation R'.
- Thus, it is not a super key of the sub relation R'.
- Attribute 'B' can determine all the attributes of sub relation R_3.
- Thus, it is a super key of the sub relation R_3.

Clearly, intersection of the sub relations is a super key of one of the sub relations.
So, condition-03 satisfies.
Thus, we conclude that the decomposition is lossless.

Decomposition of R'(A, B, C) into R_1(A, B) and R_2(B, C)-

To determine whether the decomposition is lossless or lossy,

- We will check all the conditions one by one.
- If any of the conditions fail, then the decomposition is lossy otherwise lossless.

Condition-01: According to condition-01, union of both the sub relations must contain all the attributes of relation R'.
So, we have-

$R_1 (A , B) \cup R_2 (B , C)$
$= R' (A , B , C)$

Clearly, union of the sub relations contain all the attributes of relation R'.
Thus, condition-01 satisfies.

Condition-02: According to condition-02, intersection of both the sub relations must not be null.
So, we have-

$R_1 (A , B) \cap R_2 (B , C)$
$= B$

Clearly, intersection of the sub relations is not null.
Thus, condition-02 satisfies.

Condition-03: According to condition-03, intersection of both the sub relations must be the super key of one of the two sub relations or both.
So, we have-

$R_1 (A , B) \cap R_2 (B , C)$
$= B$

Now, the closure of attribute B is-

$B^+ = \{ B , C , D \}$

Now, we see-

- Attribute 'B' can not determine attribute 'A' of sub relation R_1.\
- Thus, it is not a super key of the sub relation R_1.
- Attribute 'B' can determine all the attributes of sub relation R_2.
- Thus, it is a super key of the sub relation R_2.

Clearly, intersection of the sub relations is a super key of one of the sub relations.
So, condition-03 satisfies.

Thus, we conclude that the decomposition is lossless.

Conclusion- Overall decomposition of relation R into sub relations R_1, R_2 and R_3 is lossless.

Normalization in DBMS-

In DBMS, database normalization is a process of making the database consistent by-

- Reducing the redundancies
- Ensuring the integrity of data through lossless decomposition

Normalization is done through normal forms.

Normal Forms- The standard normal forms used are-

1. First Normal Form (1NF)
2. Second Normal Form (2NF)
3. Third Normal Form (3NF)
4. Boyce-Codd Normal Form (BCNF)

There exists several other normal forms even after BCNF but generally we normalize till BCNF only.

First Normal Form- A given relation is called in First Normal Form (1NF) if each cell of the table contains only an atomic value.

OR

A given relation is called in First Normal Form (1NF) if the attribute of every tuple is either single valued or a null value.

Example- The following relation is not in 1NF-

Student_id	Name	Subjects
100	Akshay	Computer Networks, Designing
101	Aman	Database Management System
102	Anjali	Automata, Compiler Design

Relation is not in 1NF

However,

- This relation can be brought into 1NF.
- This can be done by rewriting the relation such that each cell of the table contains only one value.

Student_id	Name	Subjects
100	Akshay	Computer Networks
100	Akshay	Designing
101	Aman	Database Management System
102	Anjali	Automata
102	Anjali	Compiler Design

Relation is in 1NF

This relation is in First Normal Form (1NF).

NOTE-

- By default, every relation is in 1NF.
- This is because formal definition of a relation states that value of all the attributes must be atomic.

Second Normal Form- A given relation is called in Second Normal Form (2NF) if and only if-

1. Relation already exists in 1NF.
2. No partial dependency exists in the relation.

Example- Consider a relation- R (V, W, X, Y, Z) with functional dependencies-

$VW \rightarrow XY$

$Y \rightarrow V$

$WX \rightarrow YZ$

The possible candidate keys for this relation are-

VW, WX, WY

From here,

- Prime attributes = {V, W, X, Y}
- Non-prime attributes = {Z}

Now, if we observe the given dependencies-

- There is no partial dependency.
- This is because there exists no dependency where incomplete candidate key determines any non-prime attribute.

Thus, we conclude that the given relation is in 2NF.

Third Normal Form- A given relation is called in Third Normal Form (3NF) if and only if-

1. Relation already exists in 2NF.
2. No transitive dependency exists for non-prime attributes.

OR

A relation is called in Third Normal Form (3NF) if and only if-

Any one condition holds for each non-trivial functional dependency A → B

1. A is a super key
2. B is a prime attribute

Example- Consider a relation- R (A , B , C , D , E) with functional dependencies-

A → BC

CD → E

B → D

E → A

The possible candidate keys for this relation are-

A, E, CD, BC

From here,

- Prime attributes = {A, B, C, D, E}
- There are no non-prime attributes

Now,

- It is clear that there are no non-prime attributes in the relation.
- In other words, all the attributes of relation are prime attributes.
- Thus, all the attributes on RHS of each functional dependency are prime attributes.

Thus, we conclude that the given relation is in 3NF.

Boyce-Codd Normal Form- A given relation is called in BCNF if and only if-

1. Relation already exists in 3NF.
2. For each non-trivial functional dependency A → B, A is a super key of the relation.

Example- Consider a relation- R (A, B, C) with the functional dependencies-

A → B

B → C

C → A

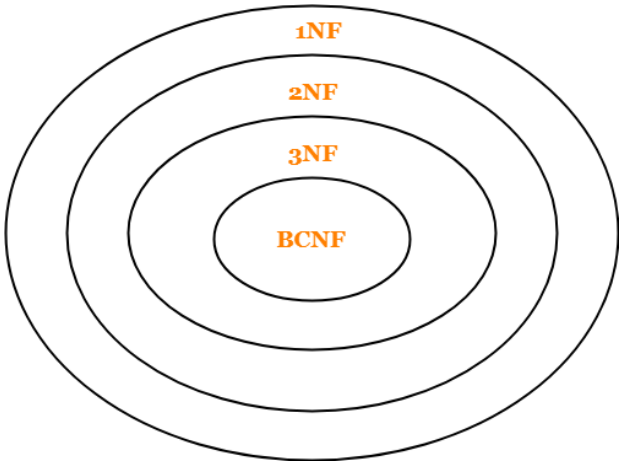
The possible candidate keys for this relation are-

Now, we can observe that RHS of each given functional dependency is a candidate key.

Thus, we conclude that the given relation is in BCNF.

Point-01: Remember the following diagram which implies-

- A relation in BCNF will surely be in all other normal forms.
- A relation in 3NF will surely be in 2NF and 1NF.
- A relation in 2NF will surely be in 1NF.



Point-02: The above diagram also implies-

- BCNF is stricter than 3NF.
- 3NF is stricter than 2NF.
- 2NF is stricter than 1NF.

Point-03: While determining the normal form of any given relation,

- Start checking from BCNF.
- This is because if it is found to be in BCNF, then it will surely be in all other normal forms.
- If the relation is not in BCNF, then start moving towards the outer circles and check for other normal forms in the order they appear.

Point-04: In a relational database, a relation is always in First Normal Form (1NF) at least.

Point-05: Singleton keys are those that consist of only a single attribute.

- If all the candidate keys of a relation are singleton candidate keys, then it will always be in 2NF at least.
- This is because there will be no chances of existing any partial dependency.
- The candidate keys will either fully appear or fully disappear from the dependencies.
- Thus, an incomplete candidate key will never determine a non-prime attribute.

Point-06: If all the attributes of a relation are prime attributes, then it will always be in 2NF at least.

- This is because there will be no chances of existing any partial dependency.

- Since there are no non-prime attributes, there will be no **Functional Dependency** which determines a non-prime attribute.

Point-07: If all the attributes of a relation are prime attributes, then it will always be in 3NF at least.

- This is because there will be no chances of existing any transitive dependency for non-prime attributes.

Point-08: Third Normal Form (3NF) is considered adequate for normal relational database design.

Point-09: Every binary relation (a relation with only two attributes) is always in BCNF.

Point-10: BCNF is free from redundancies arising out of functional dependencies (zero redundancy).

Point-11: A relation with only trivial functional dependencies is always in BCNF.

- In other words, a relation with no non-trivial functional dependencies is always in BCNF.

Point-12: BCNF decomposition is always lossless but not always dependency preserving.

Point-13: Sometimes, going for BCNF may not preserve functional dependencies.

- So, go for BCNF only if the lost functional dependencies are not required else normalize till 3NF only.

Point-14: There exist many more normal forms even after BCNF like 4NF and more.

- But in the real world database systems, it is generally not required to go beyond BCNF.

Point-15: Lossy decomposition is not allowed in 2NF, 3NF and BCNF.

- So, if the decomposition of a relation has been done in such a way that it is lossy, then the decomposition will never be in 2NF, 3NF and BCNF.

Point-16: Unlike BCNF, Lossless and dependency preserving decomposition into 3NF and 2NF is always possible.

Point-17: A prime attribute can be transitively dependent on a key in a 3NF relation.

- A prime attribute can not be transitively dependent on a key in a BCNF relation.

Point-18: If a relation consists of only singleton candidate keys and it is in 3NF, then it must also be in BCNF.

Point-19: If a relation consists of only one candidate key and it is in 3NF, then the relation must also be in BCNF.

Transaction in DBMS-

"Transaction is a set of operations which are all logically related."

OR

"Transaction is a single logical unit of work formed by a set of operations."

Operations in Transaction- The main operations in a transaction are-

1. Read Operation
2. Write Operation

1. Read Operation-

- Read operation reads the data from the database and then stores it in the buffer in main memory.
- For example- **Read(A)** instruction will read the value of A from the database and will store it in the buffer in main memory.

2. Write Operation-

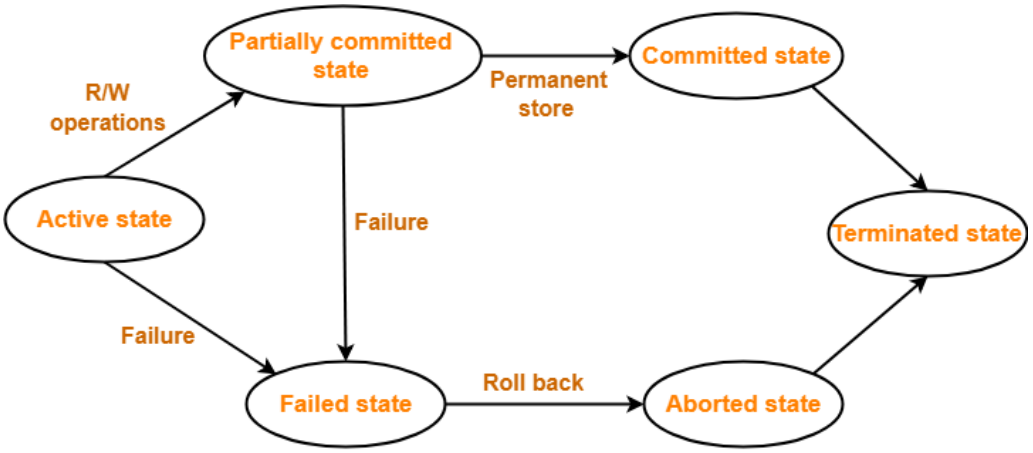
- Write operation writes the updated data value back to the database from the buffer.
- For example- **Write(A)** will write the updated value of A from the buffer to the database.

Transaction States- A transaction goes through many different states throughout its life cycle.

These states are called as **transaction states**.

Transaction states are as follows-

1. Active state
2. Partially committed state
3. Committed state
4. Failed state
5. Aborted state
6. Terminated state



Transaction States in DBMS

1. Active State-

- This is the first state in the life cycle of a transaction.
- transaction is called in an **active state** as long as its instructions are getting executed.
- All the changes made by the transaction now are stored in the buffer in main memory.

2. Partially Committed State-

- After the last instruction of transaction has executed, it enters into a **partially committed state**.
- After entering this state, the transaction is considered to be partially committed.
- It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.

3. Committed State-

- After all the changes made by the transaction have been successfully stored into the database, it enters into a **committed state**.
- Now, the transaction is considered to be fully committed.

4. Failed State-

- When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a **failed state**.

5. Aborted State-

- After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
- To undo the changes made by the transaction, it becomes necessary to roll back the transaction.

- After the transaction has rolled back completely, it enters into an **aborted state**.

6. Terminated State-

- This is the last state in the life cycle of a transaction.
- After entering the committed state or aborted state, the transaction finally enters into a **terminated state** where its life cycle finally comes to an end.

ACID Properties-

- It is important to ensure that the database remains consistent before and after the transaction.
- To ensure the consistency of database, certain properties are followed by all the transactions occurring in the system.
- These properties are called as **ACID Properties** of a transaction.

A = Atomicity
C = Consistency
I = Isolation
D = Durability

1. Atomicity-

- This property ensures that either the transaction occurs completely or it does not occur at all.
- In other words, it ensures that no transaction occurs partially.
- That is why, it is also referred to as "**All or nothing rule**".
- It is the responsibility of Transaction Control Manager to ensure atomicity of the transactions.

2. Consistency-

- This property ensures that integrity constraints are maintained.
- In other words, it ensures that the database remains consistent before and after the transaction.
- It is the responsibility of DBMS and application programmer to ensure consistency of the database.

3. Isolation-

- This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
- During execution, each transaction feels as if it is getting executed alone in the system. example - upi user

- A transaction does not realize that there are other transactions as well getting executed parallelly.
- Changes made by a transaction becomes visible to other transactions only after they are written in the memory.
- The resultant state of the system after executing all the transactions is same as the state that would be achieved if the transactions were executed serially one after the other.
- It is the responsibility of concurrency control manager to ensure isolation for all the transactions.

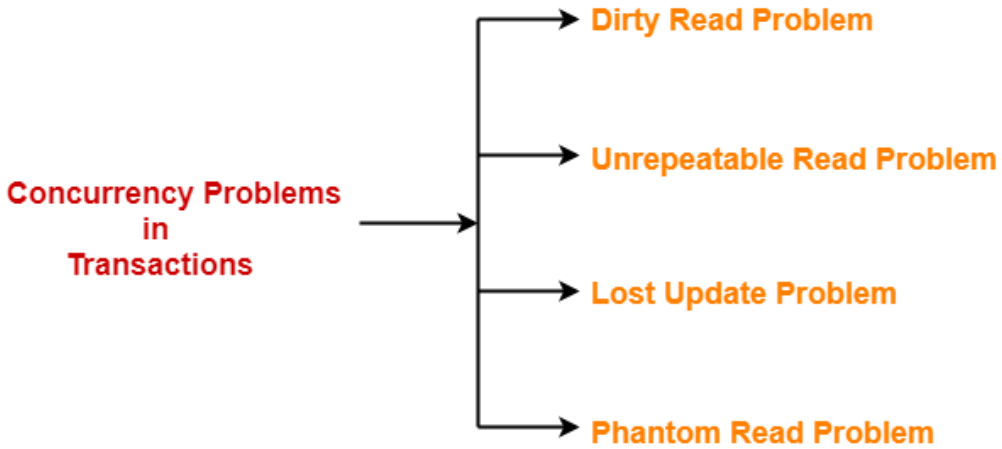
4. Durability-

- This property ensures that all the changes made by a transaction after its successful execution are written successfully to the disk./ Data Base.
- It also ensures that these changes exist permanently and are never lost even if there occurs a failure of any kind.
- It is the responsibility of recovery manager to ensure durability in the database.

Concurrency Problems in DBMS-

- When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems.
- Such problems are called as **concurrency problems**.

The concurrency problems are-



1. Dirty Read Problem- Reading the data written by an uncommitted transaction is called as dirty read.

This read is called as dirty read because-

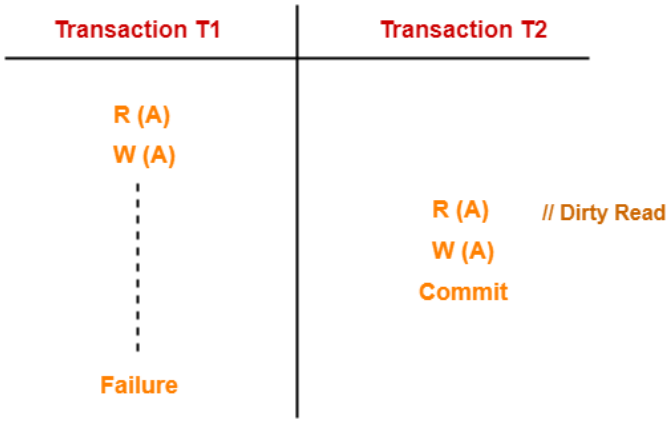
- There is always a chance that the uncommitted transaction might roll back later.
- Thus, uncommitted transaction might make other transactions read a value that does not even exist.

- This leads to inconsistency of the database.

NOTE-

- Dirty read does not lead to inconsistency always.
- It becomes problematic only when the uncommitted transaction fails and roll backs later due to some reason.

Example-



Here,

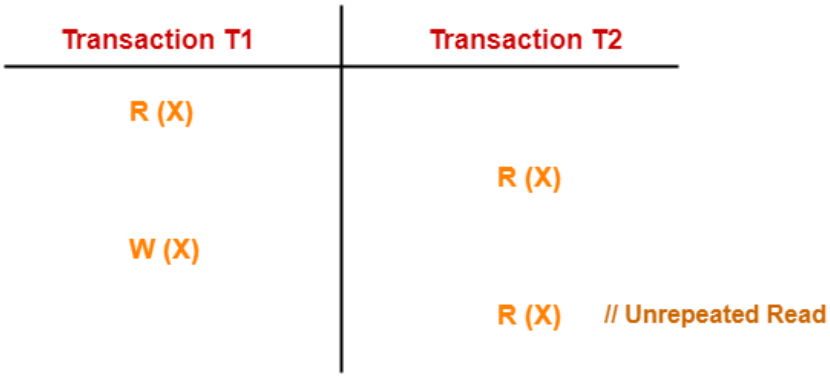
1. T1 reads the value of A.
2. T1 updates the value of A in the buffer.
3. T2 reads the value of A from the buffer.
4. T2 writes the updated the value of A.
5. T2 commits.
6. T1 fails in later stages and rolls back.

In this example,

- T2 reads the dirty value of A written by the uncommitted transaction T1.
- T1 fails in later stages and roll backs.
- Thus, the value that T2 read now stands to be incorrect.
- Therefore, database becomes inconsistent.

2. Unrepeatable Read Problem- This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.

Example-



Here,

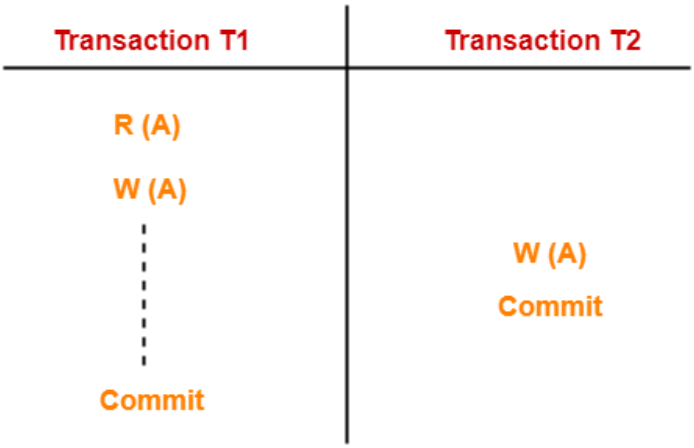
1. T1 reads the value of X (= 10 say).
2. T2 reads the value of X (= 10).
3. T1 updates the value of X (from 10 to 15 say) in the buffer.
4. T2 again reads the value of X (but = 15).

In this example,

- T2 gets to read a different value of X in its second reading.
- T2 wonders how the value of X got changed because according to it, it is running in isolation.

3. Lost Update Problem- This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

Example-



Here,

1. T1 reads the value of A (= 10 say).
2. T2 updates the value to A (= 15 say) in the buffer.
3. T2 does blind write A = 25 (write without read) in the buffer.
4. T2 commits.
5. When T1 commits, it writes A = 25 in the database.

In this example,

- T1 writes the over written value of X in the database.
- Thus, update from T1 gets lost.

4. Phantom Read Problem- This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later, it finds that the variable does not exist.

Example-

Transaction T1	Transaction T2
R (X)	
	R (X)
Delete (X)	
	Read (X)

Here,

1. T1 reads X.
2. T2 reads X.
3. T1 deletes X.
4. T2 tries reading X but does not find it.

In this example,

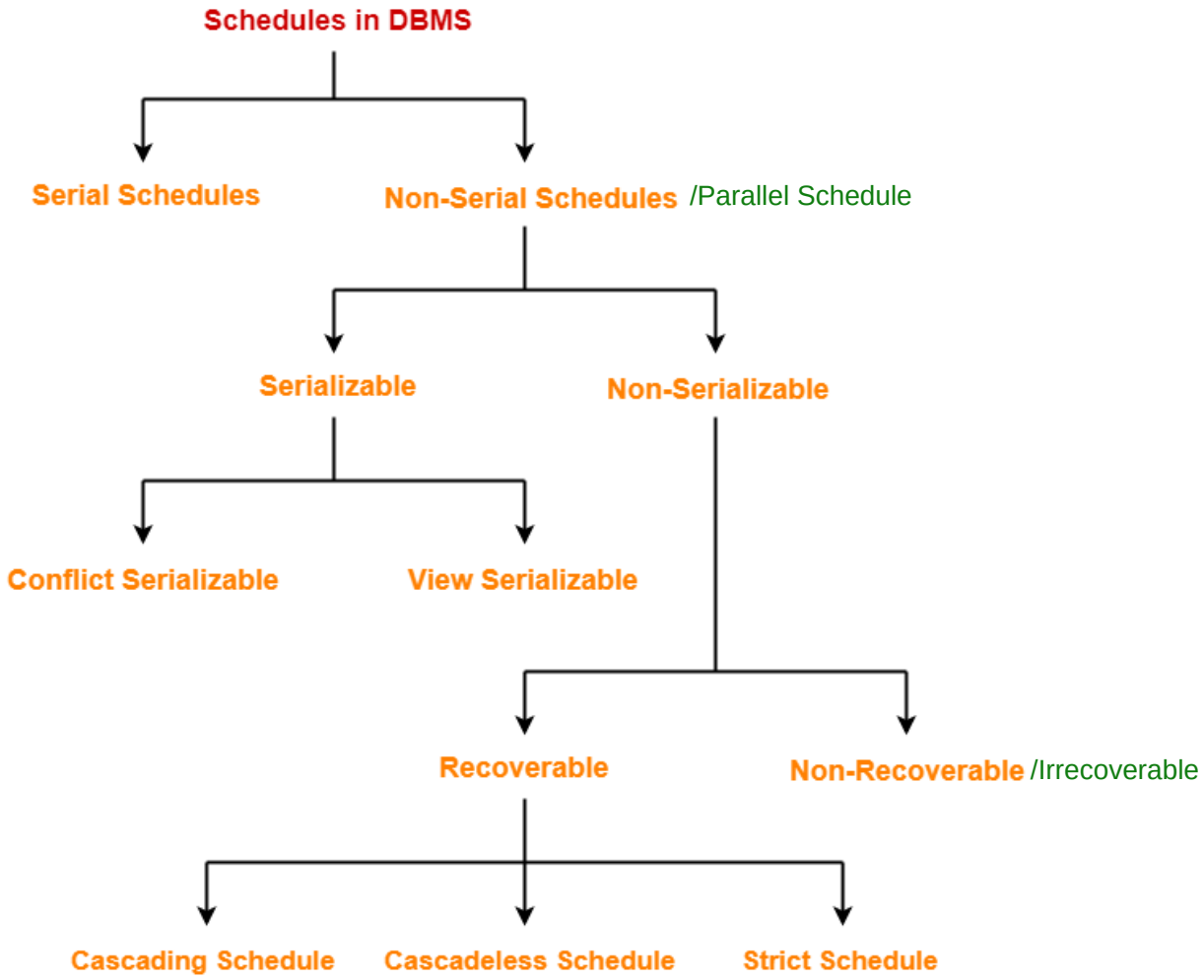
- T2 finds that there does not exist any variable X when it tries reading X again.
- T2 wonders who deleted the variable X because according to it, it is running in isolation.

Avoiding Concurrency Problems-

- To ensure consistency of the database, it is very important to prevent the occurrence of above problems.
- **Concurrency Control Protocols** help to prevent the occurrence of above problems and maintain the consistency of the database.

Types of Schedules-

In DBMS, schedules may be classified as-



Serial Schedules- In serial schedules,

- All the transactions execute serially one after the other.
- When one transaction executes, no other transaction is allowed to execute.

Characteristics- Serial schedules are always-

- Consistent
- Recoverable
- Cascadeless
- Strict

Example-01:

Transaction T1	Transaction T2
R (A)	
W (A)	
R (B)	
W (B)	
Commit	
	R (A)
	W (B)
	Commit

In this schedule,

- There are two transactions T1 and T2 executing serially one after the other.
- Transaction T1 executes first.
- After T1 completes its execution, transaction T2 executes.
- So, this schedule is an example of a **Serial Schedule**.

Example-02:

Transaction T1	Transaction T2
	R (A)
	W (B)
	Commit
R (A)	
W (A)	
R (B)	
W (B)	
Commit	

In this schedule,

- There are two transactions T1 and T2 executing serially one after the other.
- Transaction T2 executes first.
- After T2 completes its execution, transaction T1 executes.
- So, this schedule is an example of a **Serial Schedule**.

Non-Serial Schedules- In non-serial schedules,

- Multiple transactions execute concurrently.
- Operations of all the transactions are inter leaved or mixed with each other.

Characteristics- Non-serial schedules are **NOT** always-

- Consistent
- Recoverable
- Cascadeless
- Strict

Example-01:

Transaction T1	Transaction T2
R (A)	
W (B)	
	R (A)
R (B)	
W (B)	
Commit	
	R (B)
	Commit

In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.
- So, this schedule is an example of a **Non-Serial Schedule**.

Example-02:

Transaction T1	Transaction T2
	R (A)
R (A)	
W (B)	
	R (B)
	Commit
R (B)	
W (B)	
Commit	

In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.
- So, this schedule is an example of a **Non-Serial Schedule**.

Finding Number Of Schedules-

Consider there are n number of transactions T1, T2, T3 …. , Tn with N1, N2, N3 …. , Nn number of operations respectively.

Total Number of Schedules- Total number of possible schedules (serial + non-serial) is given by-

$$\frac{N1 + N2 + N3 + ... + Nn}{N1 ! \times N2 ! \times N3 ! \times .... \times Nn !}$$

Total Number of Serial Schedules- Total number of serial schedules

= Number of different ways of arranging n transactions

= n!

Total Number of Non-Serial Schedules- Total number of non-serial schedules

= Total number of schedules – Total number of serial schedules

PRACTICE PROBLEM BASED ON FINDING NUMBER OF SCHEDULES-

1. Consider there are three transactions with 2, 3, 4 operations respectively, find-

1. How many total number of schedules are possible?
2. How many total number of serial schedules are possible?
3. How many total number of non-serial schedules are possible?

Sol. **Total Number of Schedules-**

Using the above formula, we have-

$$\text{Total number of schedules} = \frac{(2 + 3 + 4) !}{2! \times 3! \times 4!}$$

$$= 1260$$

Total Number of Serial Schedules- Total number of serial schedules

= Number of different ways of arranging 3 transactions

= 3!

= 6

Total Number of Non-Serial Schedules- Total number of non-serial schedules

= Total number of schedules – Total number of serial schedules

= 1260 – 6

= 1254

Serializability in DBMS-

- Some non-serial schedules may lead to inconsistency of the database.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

Serializable Schedules- If a given non-serial schedule of 'n' transactions is equivalent to some serial schedule of 'n' transactions, then it is called as a serializable schedule**.**

Characteristics- Serializable schedules behave exactly same as serial schedules.

Thus, serializable schedules are always-

- Consistent
- **Recoverable**
- **Casacadeless**
- Strict

Serial Schedules Vs Serializable Schedules-

Serial Schedules	Serializable Schedules
No concurrency is allowed. Thus, all the transactions necessarily execute serially one after the other.	Concurrency is allowed. Thus, multiple transactions can execute concurrently.
Serial schedules lead to less resource utilization and CPU throughput.	Serializable schedules improve both resource utilization and CPU throughput.
Serial Schedules are less efficient as compared to serializable schedules. (due to above reason)	Serializable Schedules are always better than serial schedules. (due to above reason)

Types of Serializability- Serializability is mainly of two types-

1. Conflict Serializability
2. View Serializability

Conflict Serializability- If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a **conflict serializable schedule**.

Conflicting Operations- Two operations are called as **conflicting operations** if all the following conditions hold true for them-

- Both the operations belong to different transactions
- Both the operations are on the same data item
- At least one of the two operations is a write operation

Example- Consider the following schedule-

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

In this schedule,

- W1 (A) and R2 (A) are called as conflicting operations.
- This is because all the above conditions hold true for them.

Checking Whether a Schedule is Conflict Serializable Or Not- Follow the following steps to check whether a given non-serial schedule is conflict serializable or not-

Step-01: Find and list all the conflicting operations.

Step-02: Start creating a precedence graph by drawing one node for each transaction.

Step-03: Draw an edge for each conflict pair such that if X_i (V) and Y_j (V) forms a conflict pair then draw an edge from T_i to T_j.
This ensures that T_i gets executed before T_j.

Step-04: Check if there is any cycle formed in the graph.
If there is no cycle found, then the schedule is conflict serializable otherwise not.

NOTE-

- By performing the **Topological Sort** of the **Directed Acyclic Graph** so obtained, the corresponding serial schedule(s) can be found.
- Such schedules can be more than 1.

View Serializability- If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.

View Equivalent Schedules- Consider two schedules S1 and S2 each consisting of two transactions T1 and T2.

Schedules S1 and S2 are called view equivalent if the following three conditions hold true for them-

Condition-01: For each data item X, if transaction T_i reads X from the database initially in schedule S1, then in schedule S2 also, T_i must perform the initial read of X from the database.

Condition-02: If transaction T_i reads a data item that has been updated by the transaction T_j in schedule S1, then in schedule S2 also, transaction T_i must read the same data item that has been updated by the transaction T_j.

Condition-03: For each data item X, if X has been updated at last by transaction T_i in schedule S1, then in schedule S2 also, X must be updated at last by transaction T_i.

Checking Whether a Schedule is View Serializable Or Not-

Method-01: Check whether the given schedule is conflict serializable or not.

- If the given schedule is conflict serializable, then it is surely view serializable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be view serializable. Go and check using other methods.

Method-02: Check if there exists any blind write operation.

(Writing without reading is called as a blind write).

- If there does not exist any blind write, then the schedule is surely not view serializable. Stop and report your answer.
- If there exists any blind write, then the schedule may or may not be view serializable. Go and check using other methods.

Method-03: In this method, try finding a view equivalent serial schedule.

- By using the above three conditions, write all the dependencies.
- Then, draw a graph using those dependencies.
- If there exists no cycle in the graph, then the schedule is view serializable otherwise not.

PRACTICE PROBLEMS BASED ON VIEW SERIALIZABILITY-

1. Check whether the given schedule S is view serializable or not-

T1	T2	T3	T4
R (A)			
	R (A)		
		R (A)	
			R (A)
W (B)			
	W (B)		
		W (B)	
			W (B)

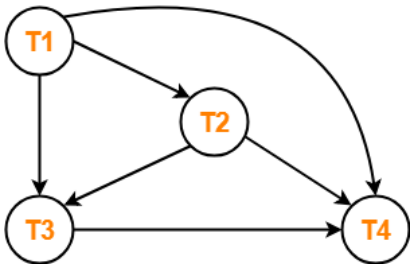
Sol. We know, if a schedule is conflict serializable, then it is surely view serializable.
So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

Step-01: List all the conflicting operations and determine the dependency between the transactions-

$W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
$W_1(B)$, $W_3(B)$ $(T_1 \rightarrow T_3)$
$W_1(B)$, $W_4(B)$ $(T_1 \rightarrow T_4)$
$W_2(B)$, $W_3(B)$ $(T_2 \rightarrow T_3)$
$W_2(B)$, $W_4(B)$ $(T_2 \rightarrow T_4)$
$W_3(B)$, $W_4(B)$ $(T_3 \rightarrow T_4)$

Step-02: Draw the precedence graph-



Clearly, there exists no cycle in the precedence graph.
Therefore, the given schedule S is conflict serializable.
Thus, we conclude that the given schedule is also view serializable.

2. Check whether the given schedule S is view serializable or not-

T1	T2	T3
R (A)		
	R (A)	
		W (A)
W (A)		

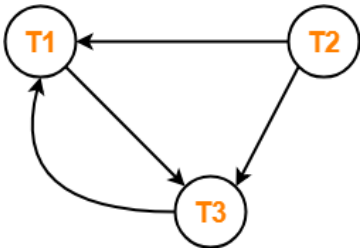
Sol. We know, if a schedule is conflict serializable, then it is surely view serializable.
So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

Step-01: List all the conflicting operations and determine the dependency between the transactions-

$R_1(A)$, $W_3(A)$ $(T_1 \rightarrow T_3)$
$R_2(A)$, $W_3(A)$ $(T_2 \rightarrow T_3)$
$R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
$W_3(A)$, $W_1(A)$ $(T_3 \rightarrow T_1)$

Step-02: Draw the precedence graph-



Clearly, there exists a cycle in the precedence graph.
Therefore, the given schedule S is not conflict serializable.

Now,
Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
To check whether S is view serializable or not, let us use another method.
Let us check for blind writes.

Checking for Blind Writes- There exists a blind write $W_3(A)$ in the given schedule S.
Therefore, the given schedule S may or may not be view serializable.

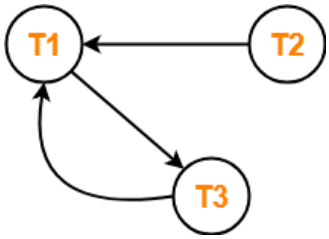
Now,
To check whether S is view serializable or not, let us use another method.

Let us derive the dependencies and then draw a dependency graph.

Drawing a Dependency Graph-

- T1 firstly reads A and T3 firstly updates A.
- So, T1 must execute before T3.
- Thus, we get the dependency **T1 → T3**.
- Final updation on A is made by the transaction T1.
- So, T1 must execute after all other transactions.
- Thus, we get the dependency **(T2, T3) → T1**.
- There exists no write-read sequence.

Now, let us draw a dependency graph using these dependencies-



- Clearly, there exists a cycle in the dependency graph.
- Thus, we conclude that the given schedule S is not view serializable.

3. Check whether the given schedule S is view serializable or not-

T1	T2
R (A)	
A = A + 10	
	R (A)
	A = A + 10
W (A)	
	W (A)
R (B)	
B = B + 20	
	R (B)
	B = B x 1.1
W (B)	
	W (B)

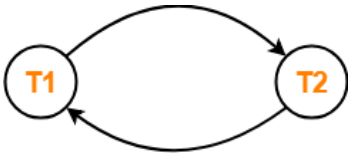
Sol. We know, if a schedule is conflict serializable, then it is surely view serializable.
So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

Step-01: List all the conflicting operations and determine the dependency between the transactions-

$R_1(A)$, $W_2(A)$ $(T_1 \rightarrow T_2)$
$R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
$W_1(A)$, $W_2(A)$ $(T_1 \rightarrow T_2)$
$R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
$R_2(B)$, $W_1(B)$ $(T_2 \rightarrow T_1)$

Step-02: Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
- To check whether S is view serializable or not, let us use another method.
- Let us check for blind writes.

Checking for Blind Writes-

- There exists no blind write in the given schedule S.
- Therefore, it is surely not view serializable.

Alternatively,

- You could directly declare that the given schedule S is not view serializable.
- This is because there exists no blind write in the schedule.
- You need not check for conflict serializability.

4. Check whether the given schedule S is view serializable or not. If yes, then give the serial schedule.

S : $R_1(A)$, $W_2(A)$, $R_3(A)$, $W_1(A)$, $W_3(A)$

Sol. For simplicity and better understanding, we can represent the given schedule pictorially as-

T1	T2	T3
R (A)		
	W (A)	
		R (A)
W (A)		
		W (A)

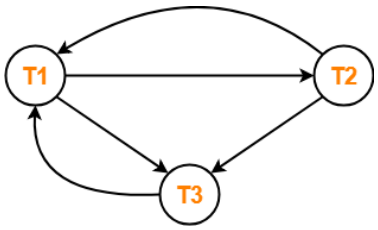
- We know, if a schedule is conflict serializable, then it is surely view serializable.
- So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

Step-01: List all the conflicting operations and determine the dependency between the transactions-

$R_1(A)$, $W_2(A)$ $(T_1 \rightarrow T_2)$
$R_1(A)$, $W_3(A)$ $(T_1 \rightarrow T_3)$
$W_2(A)$, $R_3(A)$ $(T_2 \rightarrow T_3)$
$W_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
$W_2(A)$, $W_3(A)$ $(T_2 \rightarrow T_3)$
$R_3(A)$, $W_1(A)$ $(T_3 \rightarrow T_1)$
$W_1(A)$, $W_3(A)$ $(T_1 \rightarrow T_3)$

Step-02: Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
- To check whether S is view serializable or not, let us use another method.
- Let us check for blind writes.

Checking for Blind Writes-

- There exists a blind write $W_2(A)$ in the given schedule S.
- Therefore, the given schedule S may or may not be view serializable.

Now,

- To check whether S is view serializable or not, let us use another method.
- Let us derive the dependencies and then draw a dependency graph.

Drawing a Dependency Graph-

- T1 firstly reads A and T2 firstly updates A.
- So, T1 must execute before T2.
- Thus, we get the dependency **T1 → T2**.
- Final updation on A is made by the transaction T3.
- So, T3 must execute after all other transactions.
- Thus, we get the dependency **(T1, T2) → T3**.
- From write-read sequence, we get the dependency **T2 → T3**

Now, let us draw a dependency graph using these dependencies-

Non-Serializable Schedules-

- A non-serial schedule which is not serializable is called as a non-serializable schedule.
- A non-serializable schedule is not guaranteed to produce the the same effect as produced by some serial schedule on any consistent database.

Characteristics- Non-serializable schedules-

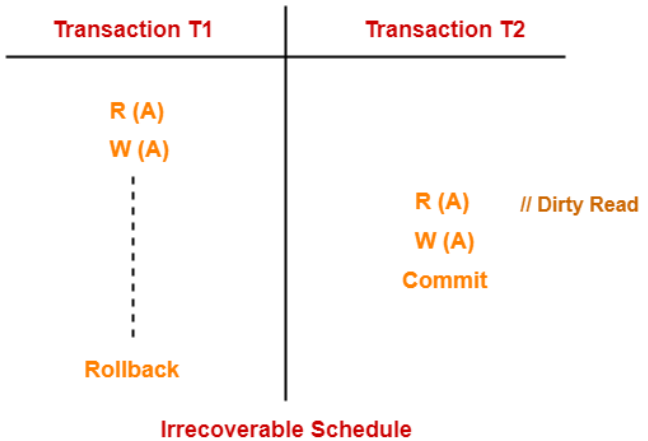
- may or may not be consistent
- may or may not be recoverable

Irrecoverable Schedules- If in a schedule,

- A transaction performs a dirty read operation from an uncommitted transaction
- And commits before the transaction from which it has read the value

then such a schedule is known as an **Irrecoverable Schedule**.

Example- Consider the following schedule-



Irrecoverable Schedule

Here,

- T2 performs a dirty read operation.
- T2 commits before T1.
- T1 fails later and roll backs.
- The value that T2 read now stands to be incorrect.
- T2 can not recover since it has already committed.

Recoverable Schedules- If in a schedule,

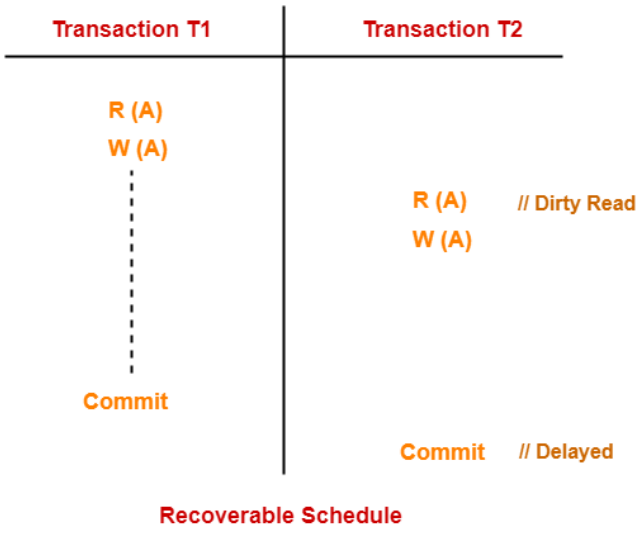
- A transaction performs a dirty read operation from an uncommitted transaction
- And its commit operation is delayed till the uncommitted transaction either commits or roll backs

then such a schedule is known as a **Recoverable Schedule**.

Here,

- The commit operation of the transaction that performs the dirty read is delayed.
- This ensures that it still has a chance to recover if the uncommitted transaction fails later.

Example- Consider the following schedule-



Recoverable Schedule

Here,

- T2 performs a dirty read operation.
- The commit operation of T2 is delayed till T1 commits or roll backs.
- T1 commits later.
- T2 is now allowed to commit.
- In case, T1 would have failed, T2 has a chance to recover by rolling back.

Checking Whether a Schedule is Recoverable or Irrecoverable-

Method-01: Check whether the given schedule is conflict serializable or not.

- If the given schedule is conflict serializable, then it is surely recoverable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be recoverable. Go and check using other methods.

Method-02: Check if there exists any dirty read operation.

(Reading from an uncommitted transaction is called as a dirty read)

- If there does not exist any dirty read operation, then the schedule is surely recoverable. Stop and report your answer.
- If there exists any dirty read operation, then the schedule may or may not be recoverable.

If there exists a dirty read operation, then follow the following cases-

Case-01: If the commit operation of the transaction performing the dirty read occurs before the commit or abort operation of the transaction which updated the value, then the schedule is irrecoverable.

Case-02: If the commit operation of the transaction performing the dirty read is delayed till the commit or abort operation of the transaction which updated the value, then the schedule is recoverable.

Recoverable Schedules-

If in a schedule,

- A transaction performs a dirty read operation from an uncommitted transaction
- And its commit operation is delayed till the uncommitted transaction either commits or roll backs

then such a schedule is called as a **Recoverable Schedule**.

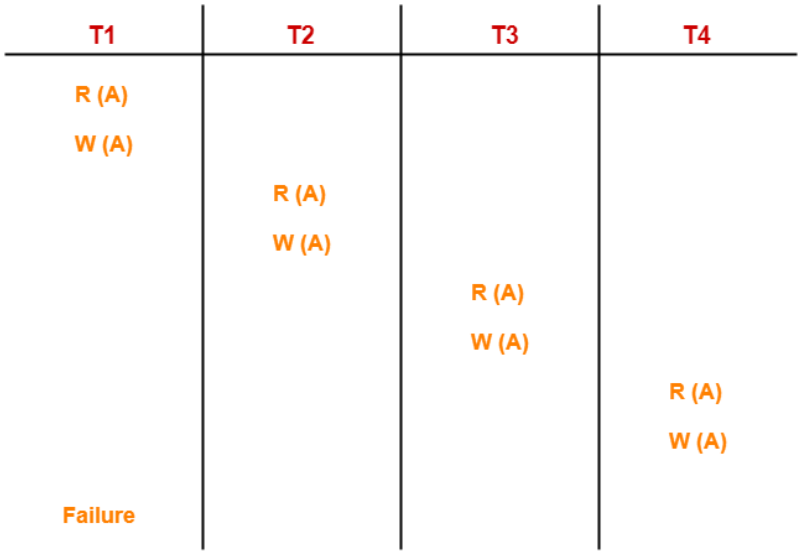
Types of Recoverable Schedules- A recoverable schedule may be any one of these kinds-

1. Cascading Schedule
2. Cascadeless Schedule
3. Strict Schedule

Cascading Schedule-

- If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a **Cascading Schedule** or **Cascading Rollback** or **Cascading Abort**.
- It simply leads to the wastage of CPU time.

Example-



Cascading Recoverable Schedule

Here,

- Transaction T2 depends on transaction T1.
- Transaction T3 depends on transaction T2.
- Transaction T4 depends on transaction T3.

In this schedule,

- The failure of transaction T1 causes the transaction T2 to rollback.
- The rollback of transaction T2 causes the transaction T3 to rollback.
- The rollback of transaction T3 causes the transaction T4 to rollback.

Such a rollback is called as a **Cascading Rollback**.

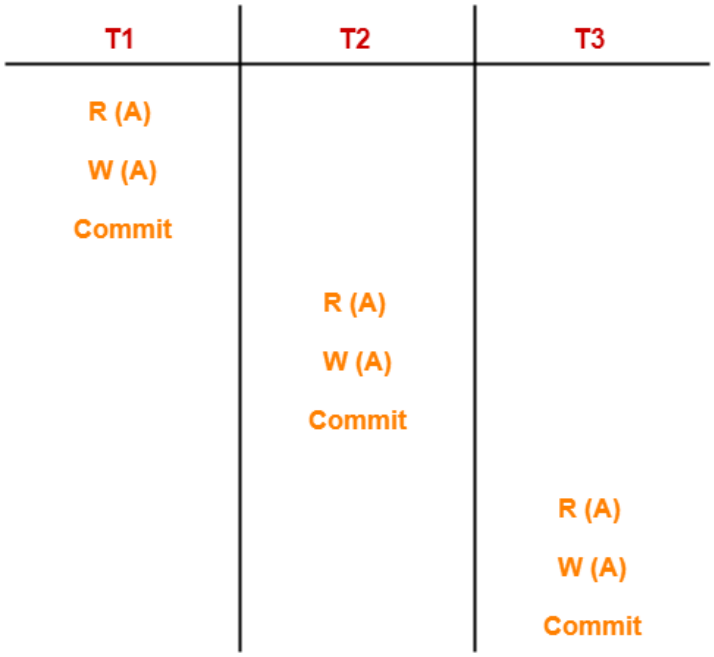
NOTE- If the transactions T2, T3 and T4 would have committed before the failure of transaction T1, then the schedule would have been irrecoverable.

Cascadeless Schedule- If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a **Cascadeless Schedule**.

In other words,

- Cascadeless schedule allows only committed read operations.
- Therefore, it avoids cascading roll back and thus saves CPU time.

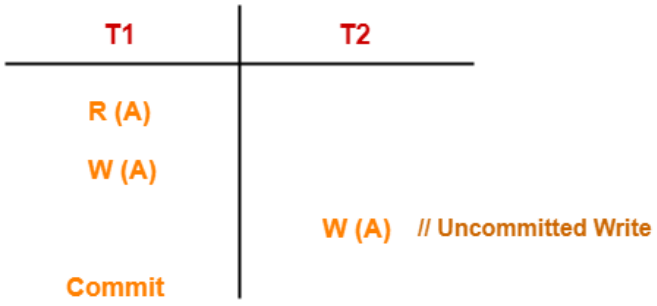
Example-



Cascadeless Schedule

NOTE- Cascadeless schedule allows only committed read operations.
However, it allows uncommitted write operations.

Example-



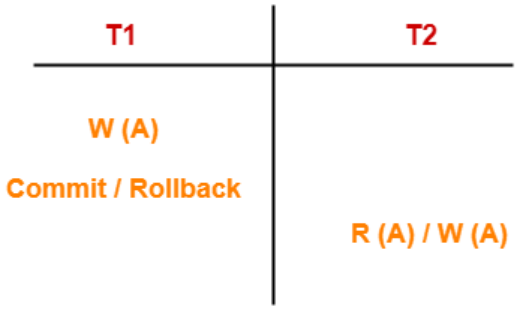
Cascadeless Schedule

Strict Schedule- If in a schedule, a transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a **Strict Schedule**.

In other words,

- Strict schedule allows only committed read and write operations.
- Clearly, strict schedule implements more restrictions than cascadeless schedule.

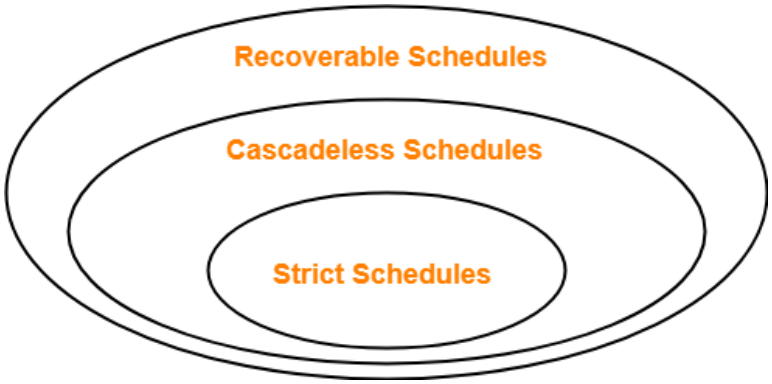
Example-



Strict Schedule

Remember-

- Strict schedules are more strict than cascadeless schedules.
- All strict schedules are cascadeless schedules.
- All cascadeless schedules are not strict schedules.



Equivalence of Schedules-

In DBMS, schedules may have the following three different kinds of equivalence relations among them-

1. Result Equivalence
2. Conflict Equivalence
3. View Equivalence

1. Result Equivalent Schedules-

- If any two schedules generate the same result after their execution, then they are called as result equivalent schedules.
- This equivalence relation is considered of least significance.
- This is because some schedules might produce same results for some set of values and different results for some other set of values.

2. Conflict Equivalent Schedules- If any two schedules satisfy the following two conditions, then they are called as conflict equivalent schedules-

1. The set of transactions present in both the schedules is same.

2. The order of pairs of conflicting operations of both the schedules is same.

PRACTICE PROBLEMS BASED ON EQUIVALENCE OF SCHEDULES-

1. Are the following three schedules result equivalent?

Schedule S1		Schedule S2		Schedule S3	
T1	T2	T1	T2	T1	T2
R (X)		R (X)			R (X)
X = X + 5		X = X + 5			X = X x 3
W (X)		W (X)			W (X)
R (Y)			R (X)	R (X)	
Y = Y + 5			X = X x 3	X = X + 5	
W (Y)			W (X)	W (X)	
	R (X)	R (Y)		R (Y)	
	X = X x 3	Y = Y + 5		Y = Y + 5	
	W (X)	W (Y)		W (Y)	

Sol. To check whether the given schedules are result equivalent or not,

- We will consider some arbitrary values of X and Y.
- Then, we will compare the results produced by each schedule.
- Those schedules which produce the same results will be result equivalent.

Let X = 2 and Y = 5.

On substituting these values, the results produced by each schedule are-

Results by Schedule S1- X = 21 and Y = 10

Results by Schedule S2- X = 21 and Y = 10

Results by Schedule S3- X = 11 and Y = 10

- Clearly, the results produced by schedules S1 and S2 are same.
- Thus, we conclude that S1 and S2 are result equivalent schedules.

2. Are the following two schedules conflict equivalent?

T1	T2
R (A)	
W (A)	
	R (A)
	W (A)
R (B)	
W (B)	

Schedule S1

T1	T2
R (A)	
W (A)	
R (B)	
W (B)	
	R (A)
	W (A)

Schedule S2

Sol. To check whether the given schedules are conflict equivalent or not,

- We will write their order of pairs of conflicting operations.
- Then, we will compare the order of both the schedules.
- If both the schedules are found to have the same order, then they will be conflict equivalent.

For schedule S1- The required order is-

$R_1(A)$, $W_2(A)$

$W_1(A)$, $R_2(A)$

$W_1(A)$, $W_2(A)$

For schedule S2- The required order is-

$R_1(A)$, $W_2(A)$

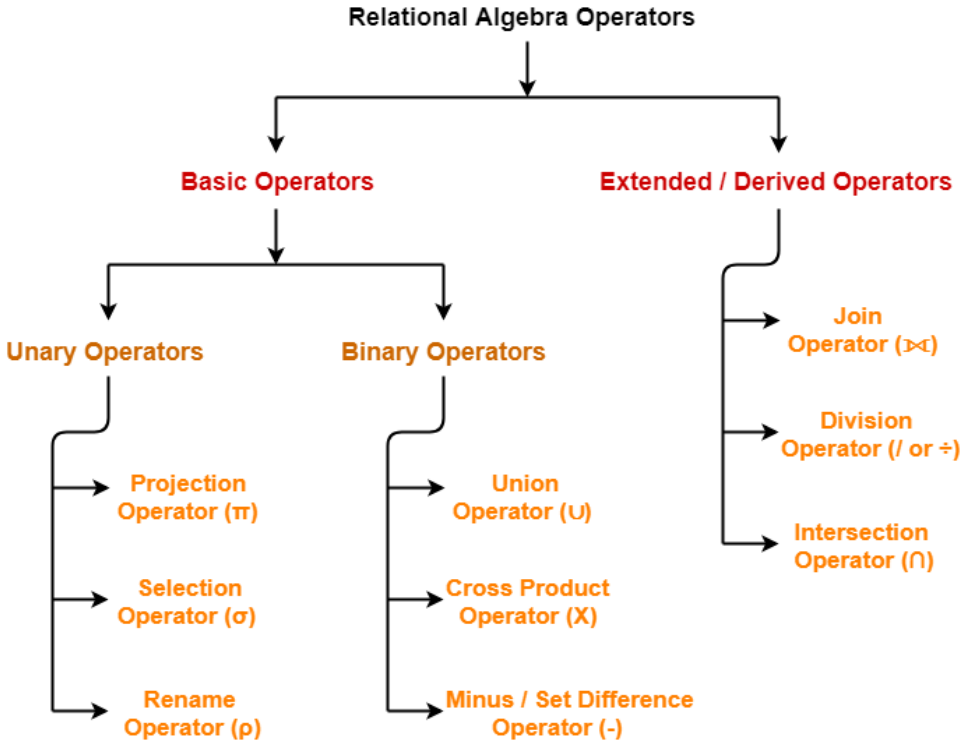
$W_1(A)$, $R_2(A)$

$W_1(A)$, $W_2(A)$

- Clearly, both the given schedules have the same order.
- Thus, we conclude that S1 and S2 are conflict equivalent schedules.

Relational Algebra- Relational Algebra is a procedural query language which takes a relation as an input and generates a relation as an output.

Relational Algebra Operators- The operators in relational algebra may be classified as-



Characteristics- Following are the important characteristics of relational operators-

- Relational Operators always work on one or more relational tables.
- Relational Operators always produce another relational table.
- The table produced by a relational operator has all the properties of a relational model.

Selection Operator-

- Selection Operator (σ) is a unary operator in relational algebra that performs a selection operation.
- It selects those rows or tuples from the relation that satisfies the selection condition.

Syntax- $\sigma_{<selection\_condition>}(R)$

Examples-

- Select tuples from a relation "Books" where subject is "database"

  $\sigma_{subject = "database"}$ (Books)
- Select tuples from a relation "Books" where subject is "database" and price is "450"

  $\sigma_{subject = "database" \wedge price = "450"}$ (Books)
- Select tuples from a relation "Books" where subject is "database" and price is "450" or have a publication year after 2010

  $\sigma_{subject = "database" \wedge price = "450" \vee year >"2010"}$ (Books)

Important Points-

Point-01: We may use logical operators like ∧ , ∨ , ! and relational operators like =, ≠, >, <, <=, >= with the selection condition.

Point-02: Selection operator only selects the required tuples according to the selection condition.

It does not display the selected tuples.

To display the selected tuples, projection operator is used.

Point-03: Selection operator always selects the entire tuple. It can not select a section or part of a tuple.

Point-04: Selection operator is commutative in nature i.e.

$$\sigma_{A \wedge B}(R) = \sigma_{B \wedge A}(R)$$

OR

$$\sigma_{B}(\sigma_{A}(R)) = \sigma_{A}(\sigma_{B}(R))$$

Point-05: Degree of the relation from a selection operation is same as degree of the input relation.

Point-06: The number of rows returned by a selection operation is obviously less than or equal to the number of rows in the original table.

Thus,

- Minimum Cardinality = 0
- Maximum Cardinality = |R|

Projection Operator-

- Projection Operator (π) is a unary operator in relational algebra that performs a projection operation.
- It displays the columns of a relation or table based on the specified attributes.

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

Example- Consider the following Student relation-

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Student

Then, we have-

Result for Query $\pi_{Name, Age}$(Student)-

Name	Age

Ashish	19
Rahul	20
Naina	20
Sameer	21

Result for Query $\pi_{ID\ ,\ Name}$(Student)-

ID	Name
100	Ashish
200	Rahul
300	Naina
400	Sameer

Important Points-

Point-01: The degree of output relation (number of columns present) is equal to the number of attributes mentioned in the attribute list.

Point-02: Projection operator automatically removes all the duplicates while projecting the output relation.

So, cardinality of the original relation and output relation may or may not be same.

If there are no duplicates in the original relation, then the cardinality will remain same otherwise it will surely reduce.

Point-03: If attribute list is a super key on relation R, then we will always get the same number of tuples in the output relation.

This is because then there will be no duplicates to filter.

Point-04: Projection operator does not obey commutative property i.e.

$$\pi_{<list2>}(\pi_{<list1>}(R)) \neq \pi_{<list1>}(\pi_{<list2>}(R))$$

Point-05: Following expressions are equivalent because both finally projects columns of list-1

$$\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$$

Point-06: Selection Operator performs horizontal partitioning of the relation.

Projection operator performs vertical partitioning of the relation.

Point-07: There is only one difference between projection operator of relational algebra and SELECT operation of SQL.

Projection operator does not allow duplicates while SELECT operation allows duplicates.

To avoid duplicates in SQL, we use "distinct" keyword and write SELECT distinct.

Thus, projection operator of relational algebra is equivalent to SELECT operation of SQL.

Set Theory Operators-

Following operators are called as set theory operators-

1. Union Operator (∪)
2. Intersection Operator (∩)
3. Difference Operator (-)

1. Union Operator (∪)- Let R and S be two relations.

Then-

- R ∪ S is the set of all tuples belonging to either R or S or both.
- In R ∪ S, duplicates are automatically removed.
- Union operation is both commutative and associative.

Example- Consider the following two relations R and S-

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation R

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation S

Then, R ∪ S is-

ID	Name	Subject
100	Ankit	English

200	Pooja	Maths
300	Komal	Science
400	Kajol	French

Relation R ∪ S

2. Intersection Operator (∩)- Let R and S be two relations.

Then-

- R ∩ S is the set of all tuples belonging to both R and S.
- In R ∩ S, duplicates are automatically removed.
- Intersection operation is both commutative and associative.

Example- Consider the following two relations R and S-

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation R

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation S

Then, R ∩ S is-

ID	Name	Subject
100	Ankit	English

Relation R ∩ S

3. Difference Operator (–)- Let R and S be two relations.

Then-

- R – S is the set of all tuples belonging to R and not to S.
- In R – S, duplicates are automatically removed.
- Difference operation is associative but not commutative.

Example- Consider the following two relations R and S-

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation R

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation S

Then, R – S is-

ID	Name	Subject
200	Pooja	Maths
300	Komal	Science

Relation R – S