E-R DIAGRAM SYLLABUS

- Introduction and basics ER diagram
- Entity, Entity set
- Attributes and types
- Relationship definitions, name, degree, cardinalities
- Strong and weak entity set
- Conversion of ER diagram into relational model
- ER diagram traps

E-R DIAGRAM/MODEL

- Introduced in 1976 by Dr Peter Chen, a non-technical design method works on conceptual level based on the perception of the real world.
- It consists of collections of basic objects, called entities and of relationships among these entities and attributes which defines their properties.
- E-R data model was developed to facilitate database design by allowing specification of an enterprise schema that represents ***the overall logical structure of a database***.
- E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema.
- It is free from ambiguities and provides a standard and logical way of visualizing the data.
- As basically it is a diagrammatical representation easy to understand even by a non-technical user.

Q An Entity - relationship diagram is a tool to represent: **(NET-JUNE-2005)**

(A) Data model
(B) Process model
(C) Event model
(D) Customer model

Ans: a

E R MODEL BASIC

- **ENTITY-** An entity is a thing or an object in the real world that is distinguishable from other object based on the values of the attributes it possesses.
- An entity may be **concrete**, such as a person or a book, or it may be **abstract**, such as a course, a course offering, or a flight reservation.
  Types of Entity-
  - *Tangible - Entities which physically exist in real world. E.g. - Car, Pen, locker*
  - *Intangible - Entities which exist logically. E.g. – Account, video.*

Name	FName	City	Age	Salary
Smith	John	3	35	$280
Doe	Jane	1	28	$325
Brown	Scott	3	41	$265
Howard	Shemp	4	48	$359
Taylor	Tom	2	22	$250

- In ER diagram we cannot represent an entity, as entity is an instant not schema, and ER diagram is designed to understand schema
- In a relational model entity is represented by a row or a tuple or a record in a table.

ENTIY SET- Collection of same type of entities that share the same properties or attributes.

- In an ER diagram an entity set is represented by a rectangle
- In a relational model it is represented by a separate table

Name	FName	City	Age	Salary
Smith	John	3	35	$280
Doe	Jane	1	28	$325
Brown	Scott	3	41	$265
Howard	Shemp	4	48	$359
Taylor	Tom	2	22	$250

Q An entity instance is a single occurrence of an _____. **(NET-JULY-2010)**
(A) entity type
(B) relationship type
(C) entity and relationship type
(D) None of these
Ans: a

Q An entity has: (**NET-DEC-2008)**
(i) a set of properties
(ii) a set of properties and values for all the properties
(iii) a set of properties and the values for some set of properties may non-uniquely identify an entity
(iv) a set of properties and the values for some set of properties may uniquely identify an entity
Which of the above are valid?
(A) (i) only **(B)** (ii) only **(C)** (iii) only **(D)** (iv) only
Ans: d

Q The E-R model is expressed in terms of: **(NET-DEC-2004)**
(i) Entities
(ii) The relationship among entities
(iii) The attributes of the entities
Then
(A) (i) and (iii)
(B) (i), (ii) and (iii)
(C) (ii) and (iii)
(D) None of the above
Ans: b

ATTRIBUTES

- Attributes are the units defines and describe properties and characteristics of entities.
- Attributes are the descriptive properties possessed by each member of an entity set. for each attribute there is a set of permitted values called domain.
- In an ER diagram attributes are represented by ellipse or oval connected to rectangle.
- While in a relational model they are represented by independent column. e.g. Instructor (ID, name, salary, dept_name)

Name	FName	City	Age	Salary
Smith	John	3	35	$280
Doe	Jane	1	28	$325
Brown	Scott	3	41	$265
Howard	Shemp	4	48	$359
Taylor	Tom	2	22	$250

Q In a relational schema, each tuple is divided in fields called: **(NET-DEC-2005)**

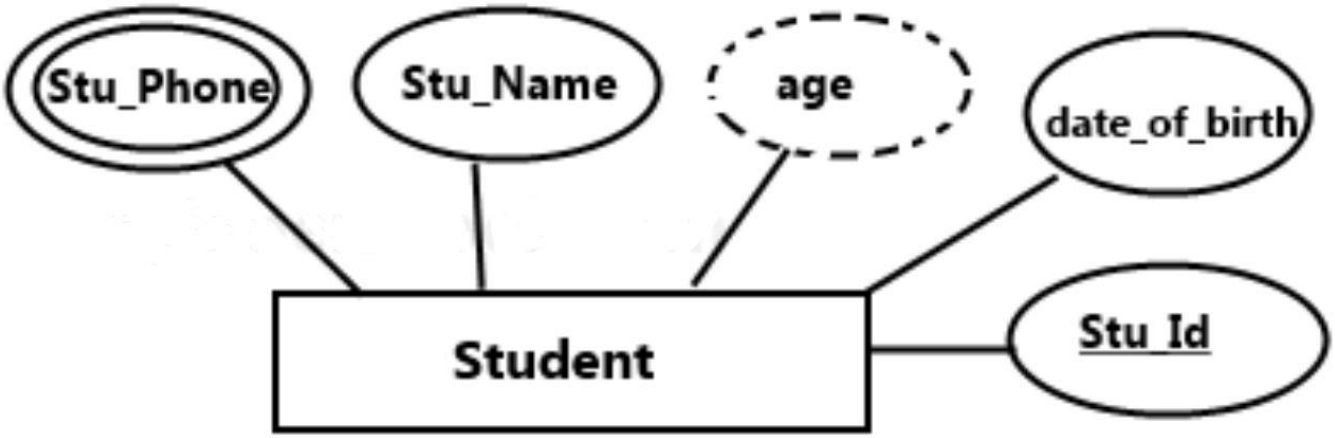
(A) Relations **(B)** Domains

(C) Queries **(D)** All the above

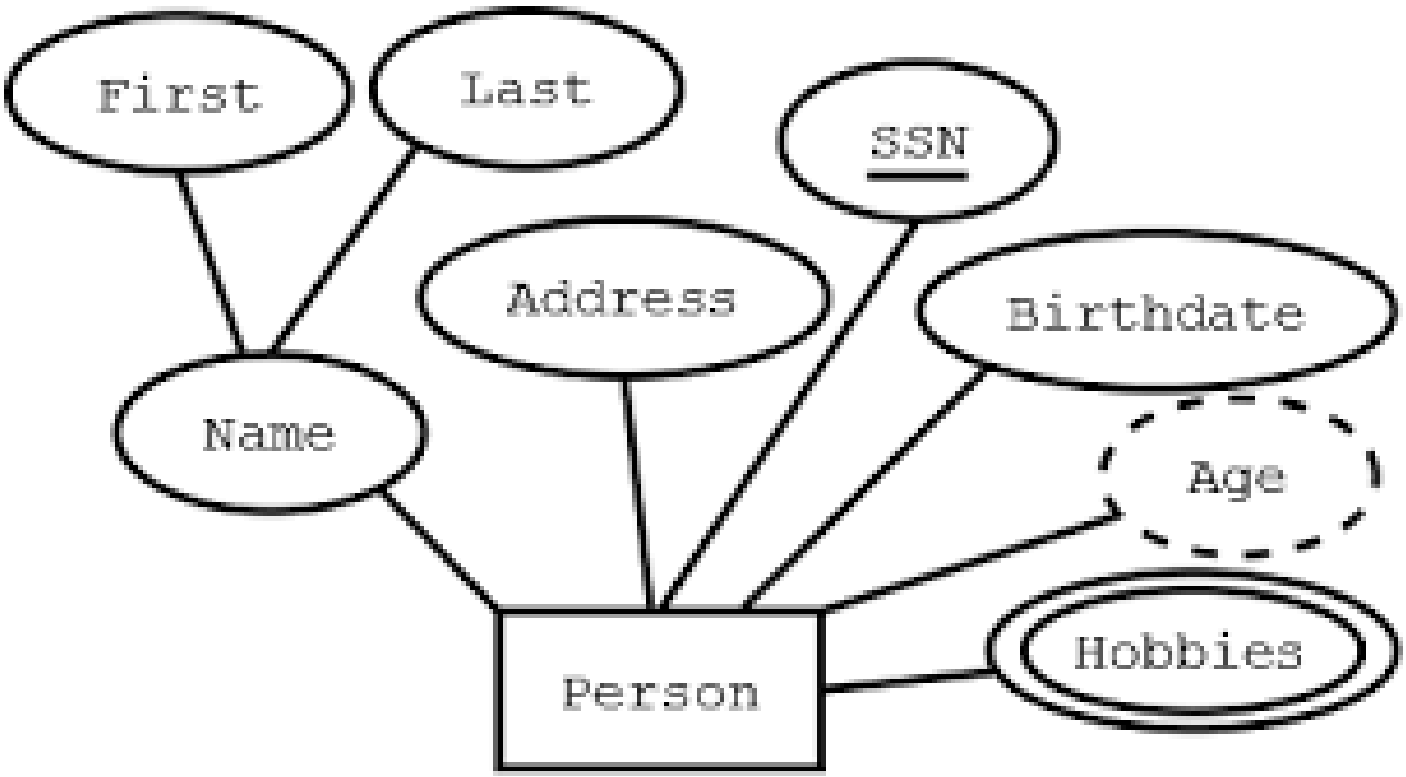
Ans: b

Types of Attributes

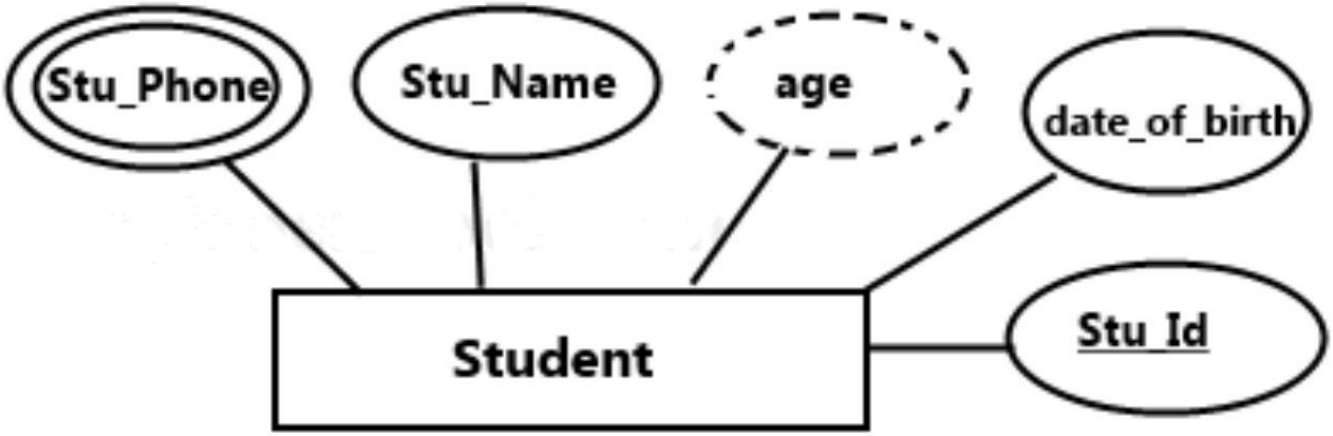
- **Single valued**- Attributes having single value at any instance of time for an entity. E.g. – Aadhar no, dob.
- **Multivalued** - Attributes which can have more than one value for an entity at same time. E.g. - Phone no, email, address.
    - A multivalued attribute is represented by a double ellipse in an ER diagram and by an independent table in a relational model.
- Separate table for each multivalued attribute, by taking mva and pk of main table as fk in new table



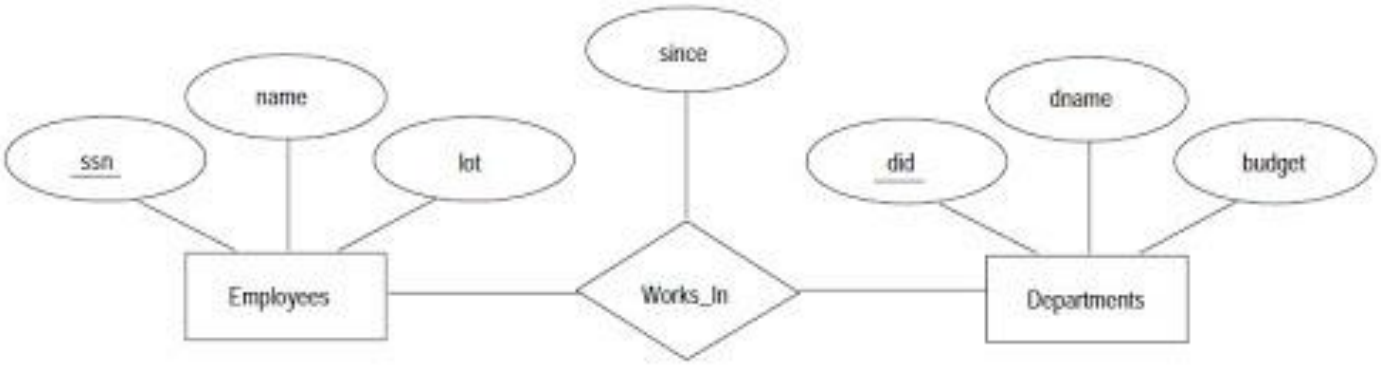
- **Simple** - Attributes which cannot be divided further into sub parts. E.g. Age
- **Composite** - Attributes which can be further divided into sub parts, as simple attributes. A composite attribute is represented by an ellipse connected to an ellipse and in a relational model by a separate column.



- **Stored** - Main attributes whose value is permanently stored in database. E.g. date_of_birth
- **Derived** -The value of these types of attributes can be derived from values of other Attributes. E.g. - Age attribute can be derived from date_of_birth and Date attribute.



- **Descriptive attribute** - Attribute of relationship is called descriptive attribute.
- An attribute takes a null value when an entity does not have a value for it. The null value may indicate “not applicable”— that is, that the value does not exist for the entity.



- Null can also designate that an attribute value is **unknown**. An unknown value may be either missing (the value does exist, but we do not have that information) or not known (we do not know whether or not the value actually exists).

Relationship / Association

- Is an association between two or more entities of same or different entity set.
- In ER diagram we cannot represent individual relationship as it is an instance or data.
- Note: - normally people use word relationship for relationship type so don't get confused.

Q The E-R model is expressed in term of **(NET-DEC-2009)**
I. Entities
II. The relationship among entities.
III. The attributes of the entities.
IV. Functional relationship.
(A) I, II **(B)** I, II, IV **(C)** II, II, IV **(D)** I, II, III
Ans: d

Q A schema describes: **(NET-DEC-2005)**
(A) data elements **(B)** records and files
(C) record relationship **(D)** all of the above
Ans: d

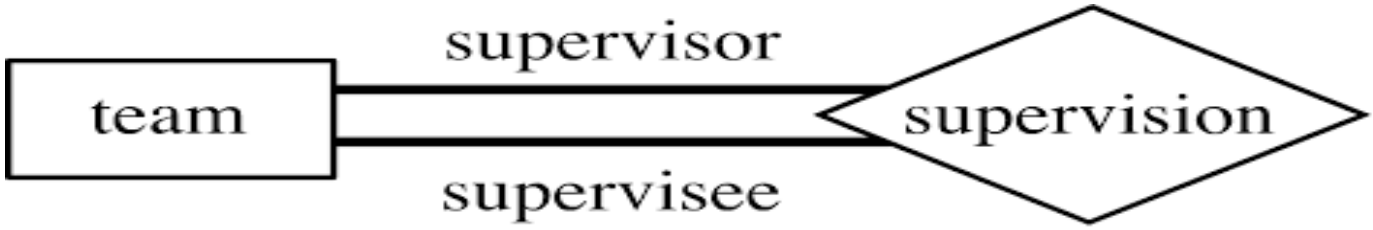
- In an ER diagram it is represented by a diamond, while in relational model sometimes through foreign key and other time by a separate table.

- Every relationship type has three components. Name, Degree, Structural constraints (cardinalities ratios, participation)

- **NAME**- every relation must have a unique name.

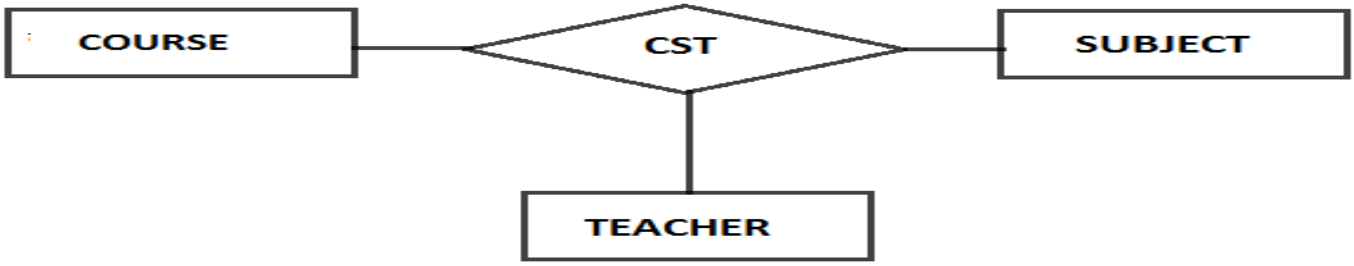
- **Degree of a relationship/relationship set**: - Means number of entities set(relations/tables) associated(participate) in the relationship set. Most of the relationship sets in a data base system are binary. Occasionally however relationship sets involve more than two entity sets. Logically, we can associate any number of entities set in a relationship called N-ary Relationship.

- **Unary Relationship** - One single entity set participate in a Relationship, means two entities of the same entity set are related to each other. These are also called as self - referential Relationship set. E.g.- A member in a team maybe supervisor of another member in team.

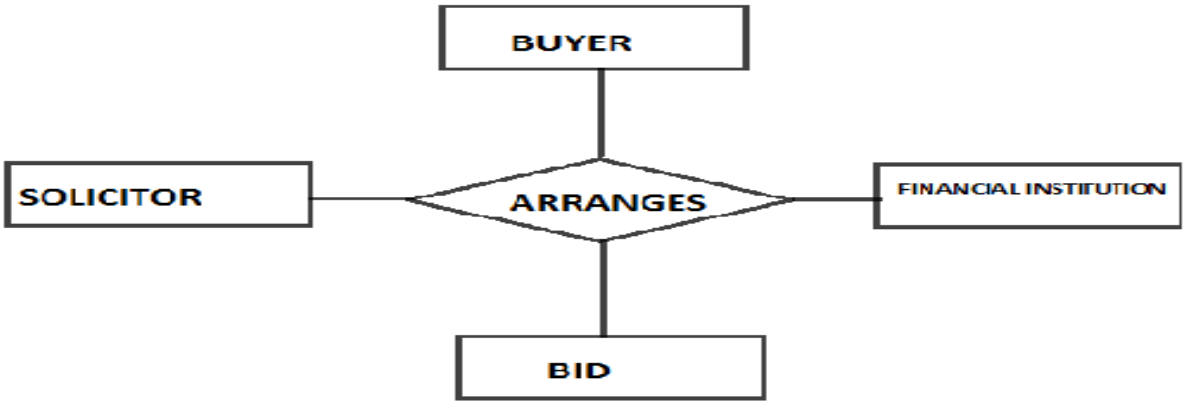


- **Binary Relationship** - Two entity sets participate in a Relationship. It is most common Relationship.

- **<u>Ternary Relationship</u>** - When three entities participate in a Relationship. E.g. The University might need to record which teachers taught which subjects in which courses.



- **<u>Quaternary Relationship</u>** - When four entities participate in a Relationship.



- **<u>N-ary relationship</u>** – where n number of entity set are associated
- But the most common relationships in ER models are ***Binary***.

- **Conversion of relationship having degree more than 2**
- Separate table is required take pk of every table and declare their combination as pk of new table.

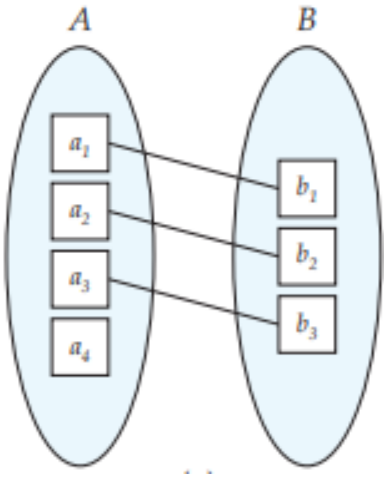
Structural constraints (Cardinalities Ratios, Participation)

- An E-R enterprise schema may define certain constraints to which the contents of a database must conform.

MAPPING CARDINALITIES / CARNINALITY RATIOS

- Express the number of entities to which another entity can be associated via a relationship set. Four possible categories are-
  - One to One (1:1) Relationship.
  - One to Many (1: M) Relationship.
  - Many to One (M: 1) Relationship.
  - Many to Many (M: N) Relationship.

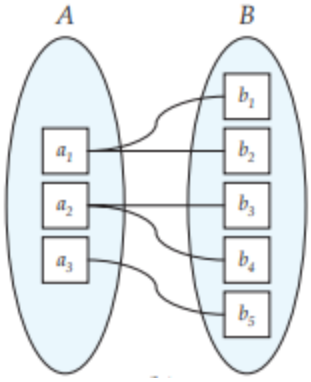
- **<u>One to One (1:1) Relationship-</u>** An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.



E.g.- The directed line from relationship set advisor to both entities set indicates that 'an instructor may advise at most one student, and a student may have at most one advisor'.

- **Conversion of 1-1 relationship(binary)**
- No separate table is required, take pk of one side as pk on other side, priority must be given to the side having total participation

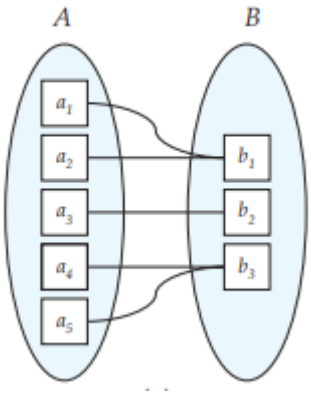
- **<u>One to Many (1: M) Relationship -</u>** An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.



<u>E.g.-</u> This indicates that an instructor may advise many students, but a student may have at most one advisor.

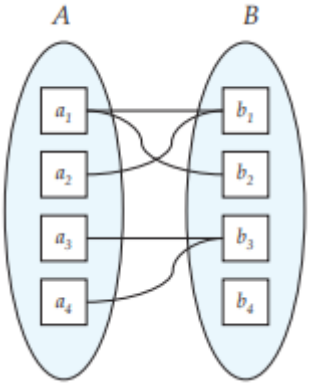
- **Conversion of 1-n or n-1 relationship (binary)**
- No separate table is required, modify n side by taking pk of 1 side a foreign key on n side

- **<u>Many to One (M: 1) Relationship-</u>** An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.



E.g.- This indicates that student may have many instructors but an instructor can advise at most one student.

- ***Many to Many(M:N) Relationship***- An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.



E.g.- This indicates a student may have many advisors and an instructor may advise many students.

- In a Relationship there are two methods to represent about cardinalities either we write the numbers or an edge.

- **Conversion of n-n relationship (binary)**
- Separate table is required take pk of both table and declare their combination as a pk of new table

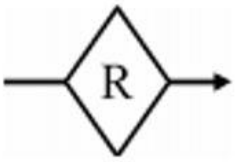
Q Match the following:

(a) 

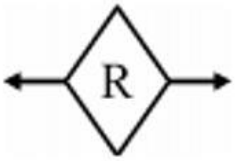
(i) One to one relationship

(b) 

(ii) Relationship

(c) 

(iii) Many to many relationship

(d) 

(iv) Many to one relationship

Codes:

	a	b	c	d
a	iii	iv	ii	i
b	iv	iii	ii	i
c	ii	iii	iv	i
d	iii	iv	i	ii

Participation constraints

- Participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.
- These constraints specify the minimum and maximum number of relationship instances that each entity must/can participate in.
- ***Max cardinality*** – it defines the maximum no of times an entity occurrence participating in a relationship.
- ***Min cardinality*** **-** it defines the minimum no of times an entity occurrence participating in a relationship.
- **PARTICIPATION CONSTRAINTS-** it defines participations of entities of an entity type in a relationship.
    - **Partial participation**
    - **Total Participation**

PARTIAL PARTICIPATION **(min cardinality zero)** - In Partial participation only some entities of entity set participate in Relationship set, that is there exists at least one entity which do not participate in a relation.

TOTAL PARTICIPATION **(min cardinality at least one) -** In total participation every entity of an entity set participates in at least one relationship in Relationship set.

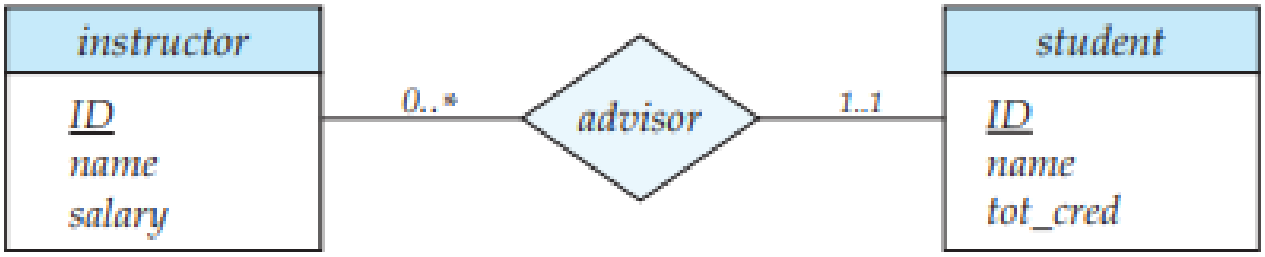
- Min max cardinalities can be represented either by single line double line or by ()min max() more information writing

Double lines indicate total participation of an entity in a relationship set.

Q What should be the condition for total participation of the entity in a relation?

a) Maximum cardinality should be one **b)** Minimum cardinality should be zero

c) Minimum cardinality should be one **d)** None of these



- A line may have an associated minimum and maximum cardinality, shown in the form l..h, where l is the minimum and h the maximum cardinality.
- The line between advisor and student has a cardinality constraint of 1..1, meaning the minimum and the maximum cardinality are both 1. That is, each student must have exactly one advisor.
- The limit 0..∗ on the line between advisor and instructor indicates that an instructor can have zero or more students. Thus, the relationship advisor is one-to-many from instructor to student, and further the participation of student in advisor is total, implying that a student must have an advisor.

Q In an Entity-Relationship (ER) model, suppose R is a many-to-one relationship from entity set E1 to entity set E2. Assume that E1 and E2 participate totally in R and that the cardinality of E1 is greater than the cardinality of E2. **(GATE-2018) (1 Marks)**
Which one of the following is true about R?
(a) Every entity in E1 is associated with exactly one entity in E2
(b) Some entity in E1 is associated with more than one entity in E2
(c) Every entity in E2 is associated with exactly one entity in E1.
(d) Every entity in E2 is associated with at most one entity in E1
Ans: a

Q An ER model of a database consists of entity types A and B. These are connected by a relationship R which does not have its own attribute. Under which one of the following conditions, can the relational table for R be merged with that of A? (GATE-2017) (1 Marks)
(a) Relationship R is one-to-many and the participation of A in R is total
(b) Relationship R is one-to-many and the participation of A in R is partial
(c) Relationship R is many-to-one and the participation of A in R is total
(d) Relationship R is one-to-many and the participation of A in R is partial
Ans: c

Q Consider an Entity-Relationship (ER) model in which entity sets E1 and E2 are connected by an m: n relationship R12, E1 and E3 are connected by a 1: n (1 on the side of E1 and n on the side of E3) relationship R13. E1 has two single-valued attributes a11 and a12 of which a11 is the key attribute. E2 has two single-valued attributes a21 and a22 is the key attribute. E3 has two single valued attributes a31 and a32 of which a31 is the key attribute. The relationships do not have any attributes. If a relational model is derived from the above

ER model, then the minimum number of relations that would be generated if all the relations are in 3NF is _____. **(GATE-2015) (2 Marks)**

a) 2 **b)** 3 **c)** 4 **d)** 5

Answer: 4

Q Let M and N be two entities in an E-R diagram with simple single value attributes. R1 and R2 are two relationship between M and N, where as
R1 is one-to-many and R2 is many-to-many.
The minimum number of tables required to represent M, N, R1 and R2 in the relational model are _____. **(NET-JAN-2017)**

(1) 4 **(2)** 6 **(3)** 7 **(4)** 3

Ans: 4

Q Given the basic ER and relational models, which of the following is INCORRECT? **(GATE-2012) (1 Marks)**

(A) An attribute of an entity can have more than one value
(B) An attribute of an entity can be composite
(C) In a row of a relational table, an attribute can have more than one value
(D) In a row of a relational table, an attribute can have exactly one value or a NULL value

Ans: c

Q Let E1 and E2 be two entities in an E/R diagram with simple single-valued attributes. R1 and R2 are two relationships between E1 and E2, where R1 is one-to-many and R2 is many-to-many. R1 and R2 do not have any attributes of their own. What is the minimum number of tables required to represent this situation in the relational model? **(GATE-2005) (2 Marks)**

a) 2 **b)** 3 **c)** 4 **d)** 5

Ans: b

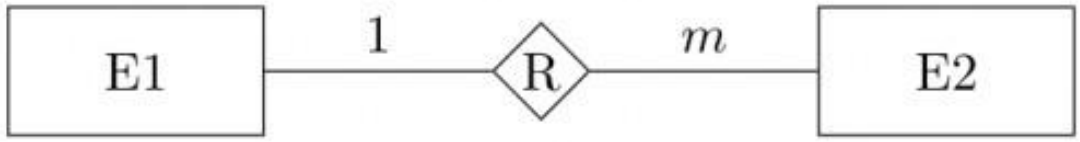
Q Consider the entities 'hotel room', and 'person' with a many to many relationship 'lodging' as shown below:

If we wish to store information about the rent payment to be made by person (s) occupying different hotel rooms, then this information should appear as an attribute of (GATE-2005) (1 Marks)

(A) Person **(B)** Hotel Room **(C)** Lodging **(D)** None of these

Ans: c

Q Consider the following entity relationship diagram (ERD), where two entities E1 and E2 have a relation R of cardinality 1 : m.



The attributes of E1 are A11, A12 and A13 where A11 is the key attribute. The attributes of E2 are A21, A22 and A23 where A21 is the key attribute and A23 is a multi-valued attribute. Relation R does not have any attribute. A relational database containing minimum number of tables with each table satisfying the requirements of the third normal form (3NF) is designed from the above ERD. The number of tables in the database is (GATE-2004)(2 Marks)

(A) 2 **(B)** 3 **(C)** 5 **(D)** 4

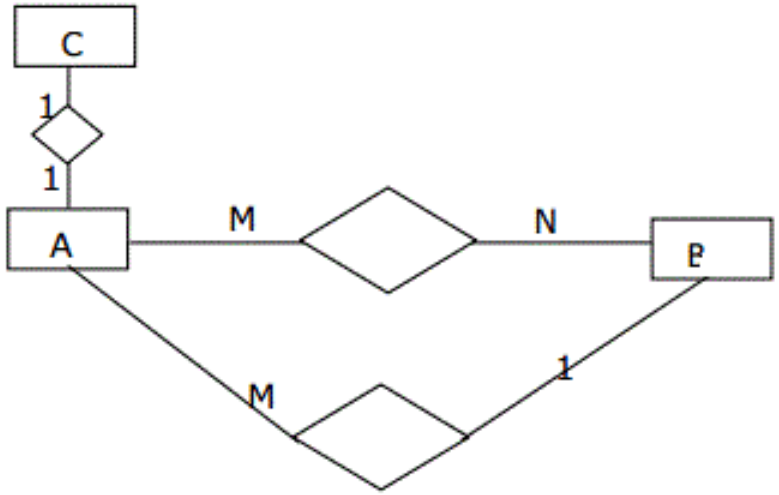
Ans: b

Q Let E1 and E2 be two entities in E-R diagram with simple single valued attributes. R1and R2 are two relationships between E1 and E2 where R1 is one - many and R2 is many - many. R1 and R2 do not have any attribute of their own. How many minimum numbers of tables are required to represent this situation in the Relational Model? **(NET-DEC-2015)**

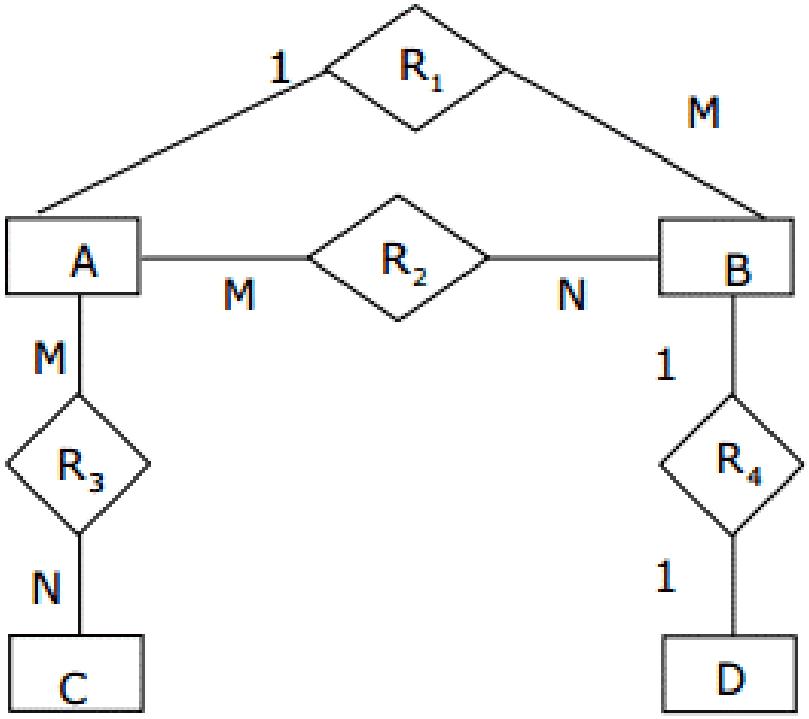
(1) 4 **(2)** 3 **(3)** 2 **(4)** 1

Ans. 2

Q The minimum number of tables required to convert the following ER diagram to relation model is _____

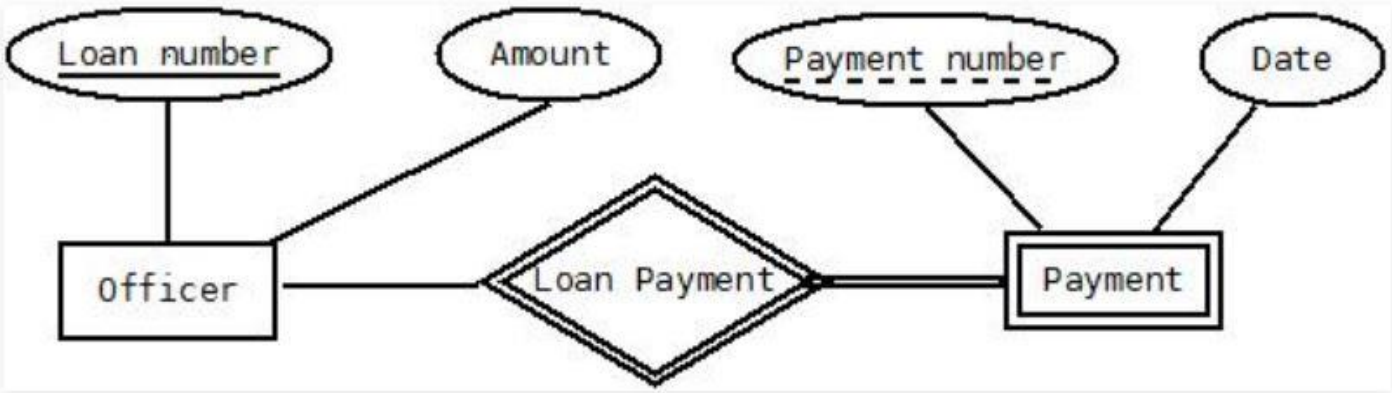


Q The minimum number of tables required to convert the following ER diagram to Relational model is _____



STRONG AND WEAK ENTITY SET

- A key for an entity is a set of attributes that suffice to distinguish entities from each other. The concepts of super key, candidate key, and primary key are applicable to entity sets just as they are applicable to relation schemas.
- An entity set is called strong entity set, if it has a primary key, all the tuples in the set are distinguishable by that key. Convert every strong entity set into an independent table
-
- An entity set that does not process sufficient attributes to form a primary key is called a weak entity set. It contains discriminator attributes (partial key) which contain partial information about the entity set, but it is not sufficient enough to identify each tuple uniquely. Represented by double rectangle
- The discriminator of a weak entity set is also called the ***partial key*** of the entity set.
    - The discriminator of a weak entity is underlined with a dashed, rather than a solid, line.



- For a weak entity set to be meaningful and converted into strong entity set, it must be associated with another entity set called the **identifying or owner entity set** i.e. weak entity set is said to be **existence dependent** on the identity set.
- The identifying entity set is said to own weak entity set that it identifies. the primary key of weak entity set will be the union of primary key and discriminator attributes.
- The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship (double diamonds)**.
- The identifying relationship is many to one from the weak entity set to identifying entity set, and the participation of the weak entity set in relationship is always total.

- The identifying relationship set should not have any ***descriptive*** attributes, since any such attributes can instead be associated with the weak entity set.
- A weak entity set may participate as owner in an identifying relationship with another weak entity set.
    - Convert every weak entity set into a table

REASONS TO HAVE WEAK ENTITY SET

- Weak entities reflect the logical structure of an entity being dependent on another.
- Weak entity can be deleted automatically when their strong entity is deleted.
- Without weak entity set it will lead to duplication and consequent possible inconsistencies.

Q For a weak entity set to be meaningful, it must be associated with another entity set in combination with some of their attribute values, is called as: **(NET-DEC-2015)**

(1) Neighbor Set
(2) Strong Entity Set
(3) Owner Entity Set
(4) Weak Set

Ans. 3

Q Which of the following statements is FALSE about weak entity set? **(NET-DEC-2015)**

a) Weak entities can be deleted automatically when their strong entity is deleted.

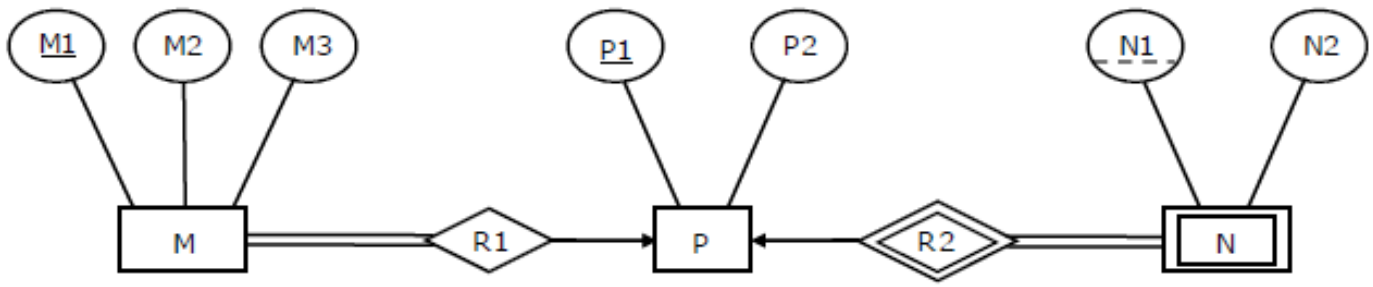
b) Weak entity set avoids the data duplication and consequent possible inconsistencies caused by duplicating the key of the strong entity.

c) A weak entity set has no primary keys unless attributes of the strong entity set on which it depends are included

d) Tuples in a weak entity set are not partitioned according to their relationship with tuples in a strong entity set.

Ans. D

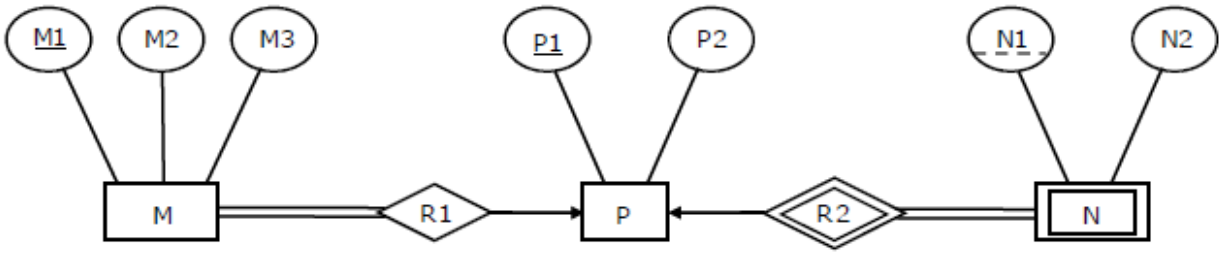
Consider the following ER diagram



The minimum number of tables needed to represent M, N, P, R1, R2 is

Ans: 3
(GATE-2008) (2 Marks)

Consider the following ER diagram



Which of the following is a correct attribute set for one of the tables for the correct answer to the above question?

a) {M1, M2, M3, P1}
b) {M1, P1, N1, N2}
c) {M1, P1, N1}
d) {M1, P1}
Ans: a

Q The entity type on which the ..... type depends is called the identifying owner. **(NET-DEC-2004)**
(A) Strong entity
(B) Relationship
(C) Weak entity
(D) E – R
Ans: c

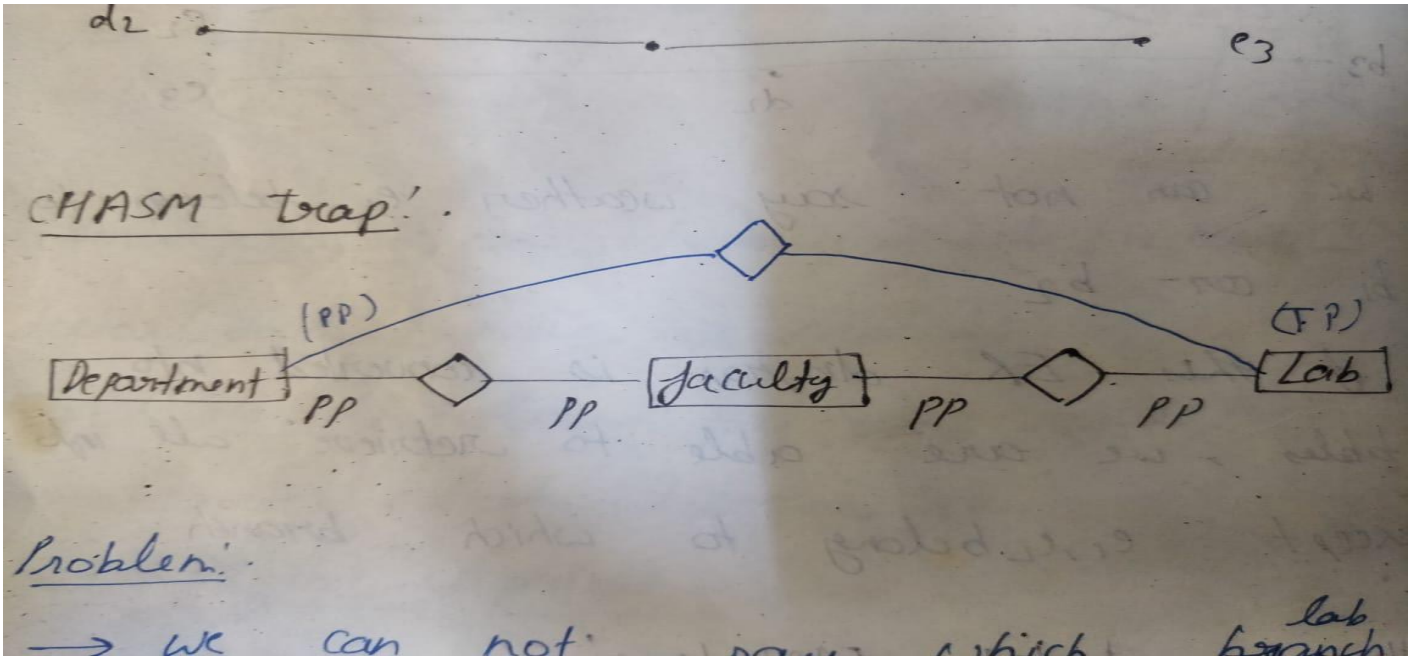
Q Map the following statements with true (T)/false (F)p-
S_1: Participation of the weak entity set in identifying relationship must be total.
S_2: Multi valued attributes in E- R diagram require separate tables when converted into relational model
a) F T **b)** T F **c)** F F **d)** T T

Trapes

- It is possible that even after all efforts there remain some problem with ER diagram. These problems are called trapes. There are two types of trapes in ER diagram.
- **FAN TRAP -** If two or more 1 to M relationships are emerging out from single entity. Then there will be a FAN trap.

- A single site contains many departments and employs many staff. However, which staff work in a particular department
- The fan trap is resolved by restructuring the original ER model to represent the correct association.

- **CHASM TRAP** - If two directly related entities are connected through another(third) entity with partial participation then there is a chasm trap. So, logic suggests the existence of a relationship between entity sets, but the relationship does not exist between certain entity occurrences in ER diagram.
- A model suggests the existence of a relationship between entity types, but the pathway does not exist between certain entity occurrences. Following is the example in different notations.



- **How to eliminate -** Create direct relationship between these 2 entities.

ADVANTAGES OF E-R DIGRAM-

1) Constructs used in the ER diagram can easily be transformed into relational tables.
2) It is simple and easy to understand with minimum training.

DISADVANTAGE OF E-R DIGRAM-

1) Loss of information content
2) Limited constraints representation
3) It is overly complex for small projects

RELATIONAL DATABASE MANAGEMENT SYSTEM

- A relational database management system (RDBMS) is a database engine/system based on the relational model specified by Edgar F. Codd--the father of modern relational database design--in 1970.
- Most modern commercial and open-source database applications are relational in nature. The most important relational database features include an ability to use tables for data storage while maintaining and enforcing certain data relationships.

Student

NAME	ID	CITY	COUNTRY	HOBBY
NISHA	1c	AGRA	INDIA	PLAYING
NIKITA	2	DELHI	INDIA	DANCING
AJAY	3	AGRA	INDIA	CHESS
ARPIT	4	PATNA	INDIA	READING

- **Tuple** - Each row of a Relation/Table is called Tuple.
- **Arity/Degree** - No. of columns/attributes of a Relation. E.g. - Arity is 5 in Table Student above.
- **Cardinality** - No of rows/tuples/record of a Relational instance. E.g. - Cardinality is 4 in table Student.
- **Domain (set of permissible value in particular column)** D is a set of atomic values. By **atomic** we mean that each value in the domain is indivisible as far as the formal relational model is concerned.
- A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.
    - **E.g.** Names: The set of character strings that represent names of persons.

Properties of Relational tables

- Each row is unique
- Each column has a unique name
- The sequence of rows is insignificant
- The sequence of columns is insignificant.
- Values are atomic
- Column values are of the same kind
- No two tables can have the same name in a relational schema.
- For all relations, the domains of all attribute should be unique. It means elements of all attribute in particular domain is indivisible (cannot be divided). And all attributes should have same property of a domain.

Goals for relational database

- Avoiding redundancies which resulting update, insertion and deletion anomalies by decomposing schemes as necessary.
- Ensure that all decompositions are lossless-join.
- Try all decompositions are dependency preserving.

Update Anomalies- Anomalies that cause redundant work to be done during insertion into and Modification of a relation and that may cause accidental loss of information during a deletion from a relation.

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

Insertion anomalies: An independent piece of information cannot be recorded into a relation unless an irrelevant information must be inserted together at the same time.

Modification anomalies: The update of a piece of information must occur at multiple locations.

Deletion Anomalies: The deletion of a piece of information unintentionally removes other information.

Roll no	name	Age	Br_code
1	A	19	101
2	B	18	101
3	C	20	101
4	D	20	102

Br_code	Br_name	Br_hod_name
101	Cs	Abc
102	Ec	Pqr

Purpose of Normalization

- Normalization may be simply defined as refinement process. Which includes creating tables and establishing relationships between those tables according to rules designed both to protect data and make the database more flexible by eliminating Redundancy
- Without normalization data base system may be inaccurate, slow and inefficient and they might not produce the data we expect.

FUNCTIONAL DEPENDENCY

- A formal tool for analysis of relational schemas.
- In a Relation R, if 'X' ⊑ R AND Y ⊑ R, then attribute or a Set of attribute 'X' Functionally derives an attribute or set of attributes 'Y',
    - iff each 'X' value is associated with precisely one 'Y' value.
    - For all pairs of tuples t_1 and t_2 in R such that
        - If $T_1[X] = T_2[X]$
        - Then, $T_1[Y] = T_2[Y]$
    - X- Determinant (Determines Y value).
    - Y- Dependent (Dependent on X).
    - If k --> R, the K is a super key of R

- **Note:** A functional dependency is a property of the relation schema R, not of a particular legal relation state/instance r of R.

- **Trivial Functional dependency** - If Y is a subset of X, then the functional dependency X -> Y will always hold.

Q Consider the relation X(P, Q, R, S, T, U) with the following set of functional dependencies F = {

{P, R} → {S,T},

{P, S, U} → {Q, R}

}

Which of the following is the trivial functional dependency in F+ is closure of F? **(GATE-2016) (2 Marks)**

(a) {P, R} → {S, T}
(b) {P, R} → {R, T}
(c) {P, S} → {S}
(d) {P, S, U} → {Q}

Ans: c

Q Which of the following functional dependencies are satisfied by the instance? **(GATE CS 2000)**

X	Y	Z
1	4	2
1	5	3
2	6	3
3	2	2

(A) XY -> Z and Z -> Y
(B) YZ -> X and Y -> Z
(C) YZ -> X and X -> Z
(D) XZ -> Y and Y -> X
Answer: (B)

Q Consider the following relation instance

A	B	C
1	2	4
3	5	4
3	7	2
1	4	2

Which of the following dependencies are satisfied by the above relation instance?
a) A->B, BC->A **b)** C->B, CA->B **c)** B->C, AB->C **d)** A->C, BC->A

Q Which of the following dependencies are satisfied by the relation instance?

A	B	C
1	1	4
1	2	4
2	1	3
2	2	3
2	4	3

A>B
B>C
B>A
C>B
C>A
A>C

Q Which of the following dependencies are satisfied by the relation instance?

X	Y	Z
1	4	3
1	5	3
4	6	3
3	2	2

XZ>X
XY>Z
Z>Y
Y>Z
XZ>Y

Q Consider the following relation instance, which of the following dependency doesn't hold

A	B	C
1	2	3
4	2	3
5	3	3

A) A > b **B)** BC > A **C)** B > C **D)** AC > B

Q Which of the following dependency doesn't hold good?

A	B	C	D	E
a	2	3	4	5
2	a	3	4	5
a	2	3	6	5
a	2	3	6	6

A) A>BC **B)** DE>C **C)** C>DE **D)** BC>A

Ans c

Q From the following instance of a relation scheme R (A, B, C), we can conclude that **(CS-2002)**

A	B	C
1	1	1
1	1	0
2	3	2
2	3	2

(A) A functionally determines B and B functionally determines C
(B) A functionally determines B and B does not functionally determine C
(C) B does not functionally determine C
(D) A does not functionally determine B and B does not functionally determine C
Answer: (B)

Q From the following instance of the relation schema R (A, B, C) we can conclude that

A	B	C
a1	b1	c1
a2	b1	c1
a3	b3	c3
a3	b3	c4
a5	b5	c5

a) A functionally determines B, B functionally determines C
b) B functionally determines C, C functionally determines A
c) A functionally determines B, but B doesn't functionally determine C

d) None of these

ATTRIBUTES CLOSURE/CLOSURE ON ATTRIBUTE SET/ CLOSURE SET OF ATTRIBUTES

- Attribute closure of an attribute set F, can be defined as set of attributes which can be functionally determined from F.
- DENOTED BY F^+

ARMSTRONG'S AXIOMS

- An **axiom** or **postulate** is a statement that is taken to be true, to serve as a premise or starting point for further reasoning and arguments.
- **Armstrong's axioms** are a set of axioms (or, more precisely, inference rules) used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong in his 1974 paper.
- The axioms are sound in generating only functional dependencies in the closure of a set of functional dependencies (denoted as F^+) when applied to that set (denoted as F).
- **Why Armstrong axioms refers to the Sound and Complete?**
    - By sound, we mean that given a set of functional dependencies F specified on a relation schema R, any dependency that we can infer from F by using the primary rules of Armstrong axioms holds in every relation state r of R that satisfies the dependencies in F.
    - By complete, we mean that using primary rules of Armstrong axioms repeatedly to infer dependencies until no more dependencies can be inferred results in the complete set of all possible dependencies that can be inferred from F.

Armstrong Axioms -

- **Reflexivity**: If Y is a subset of X, then $X \rightarrow Y$
- **Augmentation**: If $X \rightarrow Y$, then $XZ \rightarrow YZ$
- **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

From these rules, we can derive these secondary rules-

- **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- **Decomposition**: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- **Pseudo transitivity**: If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$
- **Composition**: If $X \rightarrow Y$ and $Z \rightarrow W$, then $XZ \rightarrow YW$

Q Armstrong (1974) proposed systematic approach to derive functional dependencies. Match the following w.r.t. Functional dependencies: **(NET-DEC-2013)**

List – I	List – II

a. Decomposition rule	**i.** If X → Y and Z → W then {X, Z} → {Y, W}
b. Union rule	**ii**. If X → Y and {Y, W} →Z then {X, W} → Z
c. Composition rule	**iii.** If X → Y and X→ Z then X → {Y, Z}
d. Pseudo transitivity rule	**iv.** If X → {Y, Z} then X → Y and X → Z

Codes:

	a	**b**	**c**	**d**
(A)	iii	ii	iv	i
(B)	i	iii	iv	ii
(C)	ii	i	iii	iv
(D)	iv	iii	i	ii

Ans: d

Q Decomposition rules is

a) XZ → YZ, X → Y
b) X → Y, Y → Z | X → YZ
c) X → YZ | X → Y, X → Z
d) X → Y, WY → Z | WX → Z

Ans: c

Q In a Relation R (A, B, C, D), with set of functional dependencies as-

{

A → B
B → C
AB → D

}

A^+ = {A, B, C, D}.

Q Consider the relation X (P, Q, R, S, T, U) with the following set of functional dependencies

F = {
{P, R} → {S, T},
{P, S, U} → {Q, R}
}

Which of the following is the trivial functional dependency in F^+ is closure of F?

a) {P,R} → {S,T}
b) {P,R} → {R,T}
c) {P,S} → {S}
d) {P,S,U} → {Q}

Q R(ABCDEFG)

A>B
BC>DE
AEG>G
$(AC)^+$ =?

Q R(ABCDE)

A>BC
CD>E
B>D
E>A
$(B)^+$ =

Q R(ABCDEF)
AB>C
BC>AD
D>E
CF>B
$(AB)^+$ =

Q R(ABCDEFG)
A>BC
CD>E
E>C
D>AEH
ABH>BD
DH>BC
$(BCD)^+$ =

Q Consider the following functional dependencies over the relation R (ABCDEF)
A → B
C → DE
AC→F
What is the closure of (AC)?
a) ACF **b)** ACFDE **c)** ACEFDB **d)** ACFD

Q Let R = ABCDE is a relational scheme with functional dependency set F = {A → B, B → C, AC → D}.
The attribute closures of A and E are **(NET-DEC-2014)**
(A) ABCD, ϕ **(B)** ABCD, E **(C)** Φ, ϕ **(D)** ABC, E
Ans: b

Q In a Relation R (A, B, C, D) Given, F= {A->B, B->C, C->D} *To check whether, A->C is valid or not?*

APPLICATION OF ATTRIBUTE CLOSURE

Equivalence of Two FD sets-

- Two FD sets F_1 and F2 are equivalent if
  - $F_1^+ = F_2^+$
  - $F_1 \sqsubseteq F_2$ and $F_2 \sqsubseteq F_1$

Q Consider the following set of fd R(ACDEH)

F:	G:
A>C	A>CD
AC>D	E>AH
E>AD	
E>H	

Q R(VWXYZ)

F:	G:
V>W	V>W
VW>X	V>X
Y>VX	Y>V
Y>Z	Y>Z

Q Consider the following set of fd R(ABCDE)

F:	G:
B>CD	B>CDE
AD>E	A>BC
B>A	AD>E

Q consider the following relation r(PQRS)

F:	G:
P>Q	P>QR
Q>R	R>S
R>S	

Q consider the following relation r(ABCD)

F:	G:
A>B	A> BC

B>C	B>A
C>A	C>A

Q consider the following relation r(VWXYZ)

F:	**G:**
W>X	W>XY
WX>Y	Z>WX
Z>WY	
Z>V	

To find the MINIMAL COVER /CANONICAL COVER/IRREDUCIBLE SET

Minimal cover- It means to eliminate any kind of redundancy from a FD set.

- A canonical cover of a set of functional dependencies F, is a simplified set of functional dependencies that has the same closure as the original set F.
- There may be any following type of redundancy in the set of functional dependencies: -
    - Complete production may be Redundant.
    - One or more than one attributes may be redundant on right hand side of a production.
    - One or more than one attributes may be redundant on Left hand side of a production.

Procedure to find MINIMAL COVER-

- Use decomposition rule wherever applicable so that RHS of a production/FD contains only single attribute.
- For every production find the closure value of LHS of production keeping the production in a set, and next time ignoring the production to be in a set. If both closures set matches, it means the production is redundant.
- Remove extraneous attribute on LHS of a production by finding the closure for every possible subset, if in any case the closure is same it means remaining attributes are redundant.

Q R(ABCD)
A>B
C>B
D>ABC
AC>D

Q R(VWXYZ)
V>W
VW>X
Y>VX
Y>Z

Q Which of the following FD set can be reduced further?
a) A->B **b)** B->C **c)** A->B **d)** None of these

C->B
D->AC
AC->D

C->B
A->B

C->B
AB ->D

Q Find the minimal cover for the FD set {AC → BD, A→C, B → C, D → C}

a) {A → B, A → C, B → C, A → D}

b) {A →B, A → D, A → C, B →D}

c) {A → B, A → D, A → C, B → C}

d) {A → B, A → D, D →C, B → C}

Q R(WXYZ)

X>W

WZ>XY

Y>WXZ

Q The following functional dependencies hold true for the relational schema {V, W, X, Y, Z}:

GATE (2017 SET 1)

V → W

VW → X

Y → VX

Y → Z

Which of the following is irreducible equivalent for this set of functional dependencies?

(a)	(b)	(c)	(d)
V→W	V→W	V→W	V→W
V→X	W→X	V→X	W→X
Y→V	Y→V	Y→V	Y→V
Y→Z	Y→Z	Y→X	Y→X
		Y→Z	Y→Z

Ans: a

To Find a candidate key of a Relation

- We must have to specify how tuples with in a given relation are distinguish from other tuples, so the value of one or more attributes of a tuple must be such that they can uniquely identify the tuple. So, a key field is a column value in a table that is used to uniquely identify tuple in a relation.
- Various Keys used in database System are as follows-

Super key

- Set of attributes using which we can identify each tuple uniquely, all the remaining attributes is called Super key.
- Let X be a set of attributes in a Relation R, if X^+ determines all attributes of R then X is said to be Super key of R.
- Every table will have a least one super key.
- Biggest Super Key possible in a Relation is a Set comprising all attributes of a Relation.
- A relation of '**n'** attributes with every attribute being a super key, then there are 2^n -1

E.g. For a FD set in a Relation (A, B, C, D)-

{

1. $A \rightarrow B$
2. $B \rightarrow C$
3. $AB \rightarrow D$

}

Q The maximum number of super keys for the relation schema R(E, F, G, H) with E as the key is (Gate-2014) (1 Marks)

a) 5 **b)** 6 **c)** 7 **d)** 8

Ans: d

Q A super key for an entity consists of: **(NET-JUNE-2008)**

(A) one attribute only **(B)** at least two attributes

(C) at most two attributes **(D)** one or more attributes

Ans: d

Candidate key

- A super key whose no proper subset is a super key is called Candidate key, also called as **MINIMAL SUPER KEY.**
- There should be at least one candidate key with **Not Null** constraint.
- Prime attribute- Attributes that are member of candidate Keys are called Prime attributes

Q (NET-JUNE-2019)

In relational database management, which of the following is/are property/properties of candidate key?

P : Uniqueness

Q : Irreducibility

1. P only
2. Q only
3. Both P and Q
4. Neither P nor Q

Primary key

- One of the candidate keys is selected by database administrator as a Primary Key.
- Primary Key attribute are not allowed to have Null values.
- Candidate key which are not chosen as primary key is alternate key.

Q A primary key for an entity is: **(NET-DEC-2007)**

(A) a candidate key **(B)** any attribute

(C) a unique attribute **(D)** a super key

Q Which of the following is NOT a superkey in a relational schema with attributes V, W, X, Y, Z and primary key VY? **(GATE – 2016) (2 Marks)**

(a) VXYZ **(b)** VWXZ **(c)** VWXY **(d)** VWXYZ

Ans: b

Q Which one is correct w.r.t. RDBMS? **(NET-JAN-2017)**
(1) primary key ⊆ super key ⊆ candidate key
(2) primary key ⊆ candidate key ⊆ super key
(3) super key ⊆ candidate key ⊆ primary key
(4) super key ⊆ primary key ⊆ candidate key
Ans: b

Q Consider the following database table having A, B, C and D as its four attributes and four possible candidate keys (I, II, III and IV) for this table: **(NET-JULY-2016)**

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_3	c_3	d_1
a_1	b_2	c_1	d_2

I: {B} **II:** {B, C} **III:** {A, D} **IV:** {C, D}
If different symbols stand for different values in the table (e.g., d1 is definitely not equal to d2), then which of the above could not be the candidate key for the database table?
(1) I and III only **(2)** III and IV only **(3)** II only **(4)** I only
Ans. 3

Foreign Keys

- A foreign key is a column or group of columns in a relational database table that refers the primary key of the same table or some other table to represent relationship.
- The ***concept of referential integrity*** is derived from foreign key theory.

Which of the following key constraints is required for functioning of foreign key in the context of relational databases?

1. Unique key
2. Primary key
3. Candidate key
4. Check key

Q Let R1(a, b, c) and R2(x, y, z) be two relations in which a is the foreign key of R1 that refers to the primary key of R2. Consider following four options. **(NET-JULY-2018)**

(a) Insert into R1 **(b)** Insert into R2
(c) Delete from R1 **(d)** Delete from R2

Which of the following is correct about the referential integrity constraint with respect to above?

(1) Operations (a) and (b) will cause violation.
(2) Operations (b) and (c) will cause violation.
(3) Operations (c) and (d) will cause violation.
(4) Operations (d) and (a) will cause violation.

Ans: 4

Q A many-to-one relationship exists between entity sets r1 and r2. How will it be represented using functional dependencies if Pk(r) denotes the primary key attribute of relation r ? **(NET-JULY-2018)**

(1) Pk(r1) → Pk(r2) **(2)** Pk(r2) → Pk(r1)
(3) Pk(r2) → Pk(r1) and Pk(r1) → Pk(r2) **(4)** Pk(r2) → Pk(r1) or Pk(r1) → Pk(r2)

Ans: a

Q Referential integrity is directly relating to **(NET-DEC-2012)**

(A) Relation key **(B)** Foreign key
(C) Primary key **(D)** Candidate key

Ans: b

Q The following table has two attributes A and C where AA is the primary key and CC is the foreign key referencing AA with on-delete cascade.

A	C
2	4
3	4
4	3
5	2
7	2
9	5
6	4

The set of all tuples that must be additionally deleted to preserve referential integrity when the tuple (2,4) is deleted is: **(GATE- 2005) (2 Marks)**

a) (3,4) and (6,4)
b) (5,2) and (7,2)
c) (5,2),(7,2) and (9,5)
d) (3,4),(4,3) and (6,4)

Ans: c

Q Drop Table cannot be used to drop a Table referenced by _____ constraint. **(NET-JUNE-2015)**

(a) Primary key **(b)**Sub key **(c)**Super key **(d)**Foreign key
(1)(a) **(2)**(a), (b) and (c **(3)**(d) **(4)**(a) and (d)

Ans.3

Q Consider the following table consisting of two attributes A and B, where 'B' is the foreign key referring the candidate key 'A' with on – delete cascade option

A	B
8	9
4	6
7	6
3	2
6	5
5	1
1	2
2	3

When we delete the tuple (3 2), we need to delete few tuples additionally in order to preserve the referential integrity. The number of tuples that are remaining in the table when we delete (3 2) and additional tuples if necessary is _____

Q Consider the following tables T1 and T2.

T1

P	Q
2	2
3	8
7	3
5	8
6	9
8	5
9	8

T2

R	S
2	2
8	3
3	2
9	7
5	7
7	2

In table T1, **P** is the primary key and **Q** is the foreign key referencing **R** in table T2 with on-delete cascade and on-update cascade. In table T2, **R** is the primary key and **S** is the foreign key referencing **P** in table T1 with on-delete set NULL and on-update cascade. In order to delete record ⟨3,8⟩ from table T1, the number of additional records that need to be deleted from table T1 is _____. **(GATE- 2017) (1 Marks)**

As Q refers to R so, deleting 8 from Q won't be an issue, however S refers P. But as the relationship given is on delete set NULL, 3 will be deleted from T1 and the entry in T2 having 3 in column S will be set to NULL. So, no more deletions. Answer is **0**.

Q A recursive foreign key is a: **(NET-JUNE-2007)**

(A) references a relation
(B) references a table
(C) references its own relation
(D) references a foreign key

Ans: c

Composite key – composite key is a key composed of more than one column sometimes it is also known as concatenated key.

Secondary key – secondary key is a key used to speed up the search and retrieval contrary to primary key, a secondary key does not necessary contain unique values.

Q Find Candidate key in each of the follwing schema

Q R(ABCD)(AD, BD, CD)
A>B
B>C
C>A

Q R(ABCD)(AB, BD)
AB>CD
D>A

Q R(ABCDEF)(BF)
AB>C
C>D
B>AE
Q R(ABC)(AB, BC)
AB>C
C>A

Q R(ABCDEFGHIJ)(AB)
AB>C
A>DE
B>F
F>GH
D>IJ

Q R(ABCDEFGHIJ)(ABD)
AB>C
AD>GH
BD>EF
A>I
H>J

Q R(ABCDE)(CE)
CE>D
D>B
C>A

Q R(ABCDEFGH)(AE)
A>BC
ABE>CDGH
C>GD
D>G
E>F

……

Q R(ABCDE)(ACD, BCD, CDE)
A>B
BC>E
DE>A

Q R(ABCD)(AB, AD, BC, CD)
AB>CD
C>A
D>B

Q R(ABCDE)(ACD, BCD, CDE)
A>B
BC>E
DE>A

Q R(ABCDE)(AB, BC, BD)
AB>CD
D>A
BC>DE

Q R(ABCDE)(BC, CD)
BC>ADE
D>B

……

Q R(ABCDEF)(ABD, BCD)
AB>C
DC>AE
E>F

Q R(ABCDEF)(C, D, AB, BE, BF)
AB>C
C>D

D>BE
E>F
F>A

Q R(WXYZ)(Y, XW, XZ)
Z>W
Y>XZ
XW>Y

Q R(VWXYZ)(WYZ)

Z>Y

Y>Z

X>YV

VW>X

Q R(ABCDEF)(ABC, ACD)
ABC>D
ABD>E
CD>F
CDF>B
BF>D

Q R(ABCDE)
A>BC
CD>E
B>D
E>A

Q R(ABCDEF)
A>BCDEF
BC>ADEF
DEF>ABC

Q (NET-DEC-2018)

Consider a relation schema R = (A, B, C, D, E, F) on which the following functional dependencies hold :

$A \rightarrow B$

$B, C \rightarrow D$

$E \rightarrow C$

$D \rightarrow A$

What are the candidate keys of R ?

a) AE and BE
b) AE, BE and DE
c) AEF, BEF and BCF
d) AEF, BEF and DEF

Q Match the following with respect to RDBMS: **(NET-NOV-2017)**

(a) Entity integrity	**(i)** enforces some specific business rule that do not fall into entity or domain
(b) Domain integrity	**(ii)** Rows can't be deleted which are used by other records
(c) Referential integrity	**(iii)** enforces valid entries for a column
(d) User defined integrity	**(iv)** No duplicate rows in a table ''

Code:

	a	**b**	**c**	**d**
1	iii	iv	i	ii
2	iv	iii	ii	i
3	iv	ii	iii	i
4	ii	iii	iv	i

Ans: 2

Q Let pk(R) denotes primary key of relation R. A many-to-one relationship that exists between two relations R1 and R2 can be expressed as follows: **(NET-JAN-2017)**

(1) pk(R2) → pk(R1)
(2) pk(R1) → pk(R2)
(3) pk(R2) → R1 ∩ R2
(4) pk(R1) → R1 ∩ R2

Ans: 2

Q A relation R = {A, B, C, D, E, F, G} is given with following set of functional dependencies: F = {AD → E, BE → F, B → C, AF → G} Which of the following is a candidate key? **(NET-DEC-2015)**

(1) A **(2)** AB **(3)** ABC **(4)** ABD

Ans. 4

[AD]+ = ADE. [BE]+ = BCEF. [B]+ = BC. [AF]+ = AFG. Nothing drives all the attributes, but if We add B in first key i.e. [ADB] then it will give all the attribute [ADB]+ = ABCDEFG

Q Find the candidate key(s) of the relation R(UVWXYZ) with FD set F = {UV →W, XW → Y, U→ XZ, Y → U} **(NET-DEC-2015)**

a) XW **b)** UV, YV and WXV **c)** YUV, XV and WV **d)** None of these

Answer: (B)

Q Let R = {A, B, C, D, E, F} be a relation schema with the following dependencies

C → F, E → A, EC → D, A → B

Which of the following is a key for R? **(NET-DEC-2014)**

(A) CD **(B)** EC **(C)** AE **(D)** AC

Answer: (B)

Q Identify the minimal key for relational scheme R(A, B, C, D, E) with functional dependencies F = {A → B, B → C, AC → D} **(NET-DEC-2014)**

(A) A **(B)** AE **(C)** BE **(D)** CE

Answer: (B)

Q Consider the relation scheme R = (E, F, G, H, I, J, K, L, M, N) and the set of functional dependencies {{E,F} → {G}, {F} → {I,J}, {E,H} → {K,L}, {K} → {M}, {L} → {N}} on R. What is the key for R? **GATE (2014 SET 1)**

(a) {E, F} **(b)** {E, F, H} **(c)** {E, F, H, K, L} **(d)** {E}

Ans: b

Q Given the STUDENTS relation as shown below.

StudentID	*StudentName*	*StudentEmail*	*StudentAge*	*CPI*
2345	Shankar	shankar@math	X	9.4
1287	Swati	swati@ee	19	9.5
7853	Shankar	shankar@cse	19	9.4
9876	Swati	swati@mech	18	9.3
8765	Ganesh	ganesh@civil	19	8.7

For (StudentName, Student Age) to be the key for this instance, the value X should not be equal to_____ **(GATE- 2014) (1 Marks)**

Ans 19

Q Relation R has eight attributes ABCDEFGH. Fields of R contain only atomic values. F = {CH -> G, A -> BC, B -> CFH, E -> A, F -> EG} is a set of functional dependencies (FDs) so that F+ is exactly the set of FDs that hold for R. How many candidate keys does the relation R have? **(GATE- 2013) (2 Marks)**

(A) 3 **(B)** 4 **(C)** 5 **(D)** 6

Answer: (B)

Q Consider a relation scheme R = (A, B, C, D, E, H) on which the following functional dependencies hold: {A–>B, BC–>D, E–>C, D–>A}. What are the candidate keys of R? **(GATE- 2005) (1 Marks)**

(A) AE, BE **(B)** AE, BE, DE **(C)** AEH, BEH, BCH **(D)** AEH, BEH, DEH

Answer: (D)

Q Consider a relational table with a single record for each registered student with the following attributes.

1. *Registration_Num*: Unique registration number of each registered student
2. *UID*: Unique identity number, unique at the national level for each citizen
3. *BankAccount_Num*: Unique account number at the bank. A student can have multiple accounts or join accounts. This attribute stores the primary account number.
4. *Name*: Name of the student
5. *Hostel_Room*: Room number of the hostel

Which one of the following option is **INCORRECT**? **(GATE- 2011) (1 Marks)**

A	BankAccount_Num is candidate key
B	Registration_Num can be a primary key
C	UID is candidate key if all students are from the same country
D	If S is a superkey such that S∩UID is NULL then SUUID is also a superkey

Ans: a

Q A relation R = {A, B, C, D, E, F} is given with following set of functional dependencies:
F = {A→B, AD→C, B→F, A→E}
Which of the following is candidate key? **(NET-JUNE-2006)**

(A) A **(B)** AC **(C)** AD **(D)** None of these

Ans: d

Q Given a relation R(A,B,C,D,E,F) and set of functional dependency (FD)
F = {A →BC, C → E, E → F, F → AB,}. How many candidate keys does the relation R have?

a) 1 **b)** 3 **c)** 4 **d)** 5

Q Match the following: **(NET-JUNE-2013)**

a. Foreign keys	**i.** Domain constraint
b. Private key	**ii.** Referential integrity

c. Event control action model	**iii.** Encryption
d. Data security	**iv.** Trigger

Codes:

	a	**b**	**c**	**d**
a)	iii	ii	i	iv
b)	ii	i	iv	iii
c)	iii	iv	i	ii
d)	i	ii	iii	iv

Ans: b

Q Match the following: **(NET-DEC-2009)**

List-I	List-II
(1) Determinants	(a) No attribute can be added
(2) Candidate key	(b) Uniquely identified a row
(3) Non-redundancy	(c) A constraint between two attributes
(4) Functional dependency	(d) Group of attributes on the left-hand side of arrow of function dependency.

(A) 1 – d, 2 – b, 3 – a, 4 – c
(B) 2 – d, 3 – a, 1 – b, 4 – c
(C) 4 – a, 3 – b, 2 – c, 1 – d
(D) 3 – a, 4 – b, 1 – c, 2 – d
Ans: 1

Q _____ constraints ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation. **(NET-SEP-2013)**
(A) Logical Integrity
(B) Referential Integrity
(C) Domain Integrity
(D) Data Integrity
Ans: b

Q The student marks should not be greater than 100. This is **(NET-DEC-2013)**
(A) Integrity constraint
(B) Referential constraint
(C) Over-defined constraint
(D) Feasible constraint
Ans: a

Q In RDBMS, the constraint that no key attribute (column) may be NULL is referred to as: **(NET-JULY-2016)**
(1) Referential integrity
(2) Multi-valued dependency
(3) Entity Integrity
(4) Functional dependency
Ans. 3

Normalization

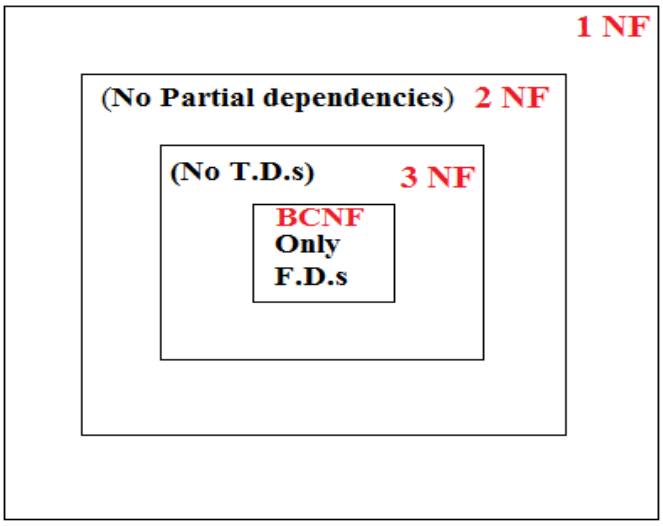
- As one paragraph contains a single idea similarly one table must contain an information about single idea, otherwise we have to repeat one info for other.

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

Roll no	name	Age	Br_code
1	A	19	101
2	B	18	101
3	C	20	101
4	D	20	102

Br_code	Br_name	Br_hod_name
101	Cs	Abc
102	Ec	Pqr

- **Normalization of data** (Decomposition of Relation) can be considered a process of analyzing the given relation schema to achieve the desirable properties of minimizing redundancy using Decomposition.
- The tool we use for normalization is functional dependencies and candidate keys.
- Functional dependency can be used only to normalize up to BCNF.
- A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be **normalized** to any desired degree.
- 1NF>>>2NF>>3NF>>BCNF



Hierarchy of Normal forms

Q Relational database schema normalization is NOT for: (**NET-AUG-2016)**
(1) reducing the number of joins required to satisfy a query.
(2) eliminating uncontrolled redundancy of data stored in the database.
(3) eliminating number of anomalies that could otherwise occur with inserts and deletes.
(4) ensuring that functional dependencies are enforced.
Ans: a

Q. match the following database terms to their function: **(NET-DEC-2015)**

	List - I		**List - II**
(a)	Normalization	(i)	Enforces match of primary key to foreign key
(b)	Data Dictionary	(ii)	Reduces data redundancy in a database
(c)	Referential Integrity	(iii)	Defines view(s) of the database for particular user(s)
(d)	External Schema	(iv)	Contains metadata describing database structure

Codes :

	(a)	**(b)**	**(c)**	**(d)**
(1)	(iv)	(iii)	(i)	(ii)
(2)	(ii)	(iv)	(i)	(iii)
(3)	(ii)	(iv)	(iii)	(i)
(4)	(iv)	(iii)	(ii)	(i)

Ans. 2

Q Decomposition help in eliminating some of the problems of bad design **(NET-JUNE-2011)**
(A) Redundancy
(B) Inconsistencies
(C) Anomalies
(D) All of the above
Ans: d

FIRST NORMAL FORM

- A Relation table is said to be in first normal form iff each attribute in each cell have single value(atomic). Means a Relation should not contain any multivalued or composite attributes.
- Other implications of first normal form
    - Every row should be unique, that is no two rows should have the same values of all the attributes.
    - There must be a primary key.
    - Every column should have a unique name
    - Order of row and column is irrelevant

Customer			
Customer ID	**First Name**	**Surname**	**Telephone Number**
123	Pooja	Singh	555-861-2025, 192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	John	Doe	555-808-9633

- This table is not in first normal form, as column telephone number, contains multiple value in a single cell.

Solution

- An apparent solution is to introduce more columns:

Customer				
Customer ID	**First Name**	**Surname**	**Telephone Number1**	**Telephone Number2**
123	Pooja	Singh	555-861-2025	192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53	182-929-2929
789	John	Doe	555-808-9633	

- An arbitrary and hence meaningless ordering has been introduced: why is 555-861-2025 put into the Telephone Number1 column rather than the Telephone Number2 column?
- There's no reason why customers could not have more than two telephone numbers, so how many Telephone Number *N* columns should there be?
- It is not possible to search for a telephone number without searching an arbitrary number of columns.
- Adding an extra telephone number may require the table to be reorganized by the addition of a new column rather than just having a new row (tuple) added.

Designs that comply with 1NF

- To bring the model into the first normal form, we split the strings we used to hold our telephone number information into "atomic" (i.e. indivisible) entities: single phone numbers. And we ensure no row contains more than one phone number.

Customer			
Customer ID	**First Name**	**Surname**	**Telephone Number**
123	Pooja	Singh	555-861-2025
123	Pooja	Singh	192-122-1111
456	San	Zhang	182-929-2929
456	San	Zhang	(555) 403-1659 Ext. 53
789	John	Doe	555-808-9633

- Note that the "ID" is no longer unique in this solution with duplicated customers. To uniquely identify a row, we need to use a combination of (ID, Telephone Number). The value of the combination is unique although each column separately contains repeated values. Being able to uniquely identify a row (tuple) is a requirement of 1NF.

An alternative design uses two tables:

Customer Name		
Customer ID	**First Name**	**Surname**
123	Pooja	Singh
456	San	Zhang
789	John	Doe

Customer ID	Telephone Number
123	555-861-2025
123	192-122-1111
456	(555) 403-1659 Ext. 53
456	182-929-2929
789	555-808-9633

- Using **Customer ID** as key, a *one-to-many* relationship exists between the name and the number tables. A row in the "parent" table, **Customer Name**, can be associated with many telephone numbers rows in the "child" table, **Customer Telephone Number**, but each telephone number belongs to one, and only one customer. It is worth noting that this design meets the additional requirements for second and third normal form.

Q Relations produced from E - R Model will always be in _____. **(NET-JULY-2018)**

(1) 1 NF **(2)** 2 NF **(3)** 3 NF **(4)** 4 NF

Ans (1)

- **Prime attribute: -** A attribute is said to be prime if it is part of any of the candidate key
- **Non-Prime attribute: -** A attribute is said to be non-prime if it is not part of any of the candidate key

E.g. R(ABCD)

AB>CD

Here candidate key is AB so, A and B are prime attribute, C and D are non-prime attributes.

- ***PARTIAL DEPENDENCY*-** When a non –prime attribute is dependent only on a part (Proper subset) of candidate key then it is called partial dependency. (PRIME > NON-PRIME)

- ***TOTAL DEPENDENCY*-** When a non –prime attribute is dependent on the entire candidate key then it is called total dependency.

Q R(ABCD) AB>D, A>C

SECOND NORMAL FORM

Relation R is in 2NF if,

- R should be in 1 NF.
- R should not contain any Partial dependency. (that is every non-key column should be fully dependent upon candidate key)S

Example to see advantage of redundancy

Q R(A, B, C) B>C

A	B	C
a	1	X
b	2	Y
a	3	Z
C	3	Z
D	3	Z
E	3	Z

A	B
A	1
B	2
A	3
C	3
D	3
E	3

B	C
1	X
2	Y
3	z

Even if one some subset become null then also, we can find the desired attribute

Q Consider the following relation R(ABCDEF) with the FD set F = (A → B, C → D, E → F).
Find the 2 NF decomposition for R.
a) (ABC) (CD) (EF) (ABCD)
b) (AB)(CD)(EF)
c) (AB) (CDE)(EF) (ACE)
d) (AB)(CD)(EF)(ACE)

- Every table with two attributes will always be in second normal form.

Q If a relation is in 2NF then: **(NET-JUNE-2008)**
(A) every candidate key is a primary key
(B) every non-prime attribute is fully functionally dependent on each relation key
(C) every attribute is functionally independent
(D) every relational key is a primary key
Ans: B

TRANSITIVE DEPENDENCY

A functional dependency from non-Prime attribute to non-Prime attribute is called transitive
E.g.- R(A, B, C, D) with A as a candidate key
A->B
B->C [transitive dependency]
C->D [transitive dependency]

THIRD NORMAL FORM

Let R be the relational schema, it is said to be in 3 NF

- R should be in 2NF
- It must not contain any transitive dependency

THIRD NORMAL FORM DIRECT DEFINATION-

A relational schema R is said to be 3 NF if every functional *dependency in R from α-->β, either α is super key or β is the prime attribute*

A	B	C
A	1	P
B	2	Q
C	2	Q
D	2	Q
E	3	R
F	3	R
G	4	S

A	B
A	1
B	2
C	2
D	2
E	3
F	3
G	4

B	C
1	P
2	Q
3	R
4	S

Q Which normal form is considered as adequate for usual database design? **(NET-JUNE-2013)**
(A) 2NF **(B)** 3NF **(C)** 4NF **(D)** 5NF
Ans: b

Q Third normal form is based on the concept of _____. **(NET-DEC-2012)**
(A) Closure Dependency **(B)** Transitive Dependency
(C) Normal Dependency **(D)** Functional Dependency
Ans: b

Q If a relation is in 2NF and 3NF forms then: **(NET-DEC-2007)**
(A) no non-prime attribute is functionally dependent on other non-prime attributes
(B) no non-prime attribute is functionally dependent on prime attributes
(C) all attributes are functionally independent
(D) prime attribute is functionally independent of all non-prime attributes
Ans: a

BCNF (BOYCE CODD NORMAL FORM)

A relational schema R is said to be BCNF if every functional *dependency in R from α-->β*

- *α must be a super key*

E.g.- R (A, B, C, D)

{

AB->C [No violation of 2NF,3NF, BCNF]

C->D [No violation of 2NF,3NF, BCNF]

D->A [violation of BCNF, D not a candidate/super key]

} *Candidate key= {AB}, {DB}, {CB}*

R(A, B, C) AB>C, C>B

A	B	C
A	B	B
B	B	C
B	A	D
A	A	E
C	C	B
D	C	B
E	C	B
F	C	B

A	B
A	B
B	B
B	A
A	A
C	C
D	C
e	C
f	c

C	B
B	B
C	B
D	A
E	A

Q (NET-JULY-2019)

In relational databases, if relation R is in BCNF, then which of the following is true about relation R?

1. R is in 4NF
2. R is not in 1NF
3. R is in 2NF and not in 3NF
4. R is in 2NF and 3NF

Ans: 4

Some important note points on Normalization:

- If a relation R does not contain any non- trivial dependency, then R Is in BCNF.
- A Relation with two attributes is always in BCNF.
- A relation schema R consist of only simple candidate key then, R is always in 2NF but may or may not be in 3NF or BCNF.
- A Relation schema R consist of only prime attributes then R is always in 3NF, but may or may not be in BCNF.
- A relation schema R in 3NF and with only simple candidate keys, then R surely in BCNF.

Q R(ABCDEF) (A, BC, DEF) (BCNF) →
A>BCDEF
BC>ADEF
DEF>ABC

……

Q R(ABC)(AB, BC)(3 NF)
AB>C
C>A

Q R(ABCD)(AD, BD, CD)(3 NF)
A>B
B>C
C>A

Q R(ABCD)(AB, BD)(3 NF)
AB>CD
D>A

Q R(ABCDE)(ACD, BCD, CDE)(3 NF)
A>B
BC>E
DE>A

Q R(ABCD)(AB, AD, BC, CD)(3 NF)
AB>CD
C>A
D>B

Q R(ABCDE)(AB, BC, BD)(3 NF)
AB>CD
D>A
BC>DE

Q R(ABCDE)(BC, CD)(3 NF)
BC>ADE
D>B

Q R(ABCDEF)(C, D, AB, BE, BF)(3 NF)
AB>C
C>D
D>BE
E>F
F>A

Q R(WXYZ)(Y, XW, XZ)(3 NF)
Z>W
Y>XZ
XW>Y

Q R(ABCDE)(A, E, BC, CD)(3 NF)
A>BC
CD>E
B>D
E>A

Q (ABCDE)(ACD, BCD, CDE)(3 NF)
A>B
BC>E
DE>A

……..

Q R(ABCDE)(AE)(2 NF)
A>B
B>E
C>D

……

Q R(ABCDE)(ac)(1NF)
A>B

B>E

C>D

Q R(ABCDE)(ab)(1NF)

AB>C

B>D

D>E

Q R(ABCDE)(AB)(1NF)

AB>C

B>D

D>E

Q R(ABCD)(AB)(1 NF)

AB>C

B>D

Q R(ABCDEF)(BF)(1 NF)

AB>C

C>D

B>AE

Q R(ABCDEFGHIJ)(AB)(1 NF)

AB>C

A>DE

B>F

F>GH

D>IJ

Q R(ABCDEFGHIJ)(ABD) (1NF)

AB>C

AD>GH

BD>EF

A>I

H>J

Q R(ABCDE)(CE)(1 NF)
CE>D
D>B
C>A

Q R(ABCDEFGH)(AE)(1 NF)
A>BC
ABE>CDGH
C>GD
D>G
E>F

Q R(ABCDEF)(ABD, BCD)(1 NF)
AB>C
DC>AE
E>F

Q R(VWXYZ)(VW, XW)(1NF)
Z>Y
Y>Z
X>YV
VW>X

Q R(ABCDEF)(ABC, ACD)(1 NF)
ABC>D
ABD>E
CD>F
CDF>B
BF>D

Q R(ABCDE)(ABD)(1 NF)
BD>E
A>C

Q Consider the following four relational schemas. For each schema, all non-trivial functional dependencies are listed. The underlined attributes are the respective primary keys. (GATE-

2020) (2 Marks)

Schema I: Registration (rollno, courses)
Field ‘courses’ is a set-valued attribute containing the set of courses a student has registered for.
Non-trivial functional dependency
rollno → courses

Schema II: Registration (rollno, coursid, email)
Non-trivial functional dependencies:
rollno, courseid → email
email → rollno

Schema III: Registration (rollno, courseid, marks, grade)
Non-trivial functional dependencies:
rollno, courseid, → marks, grade
marks → grade

Schema IV: Registration (rollno, courseid, credit)
Non-trivial functional dependencies:
rollno, courseid → credit
courseid → credit

Which one of the relational schemas above is in 3NF but not in BCNF?
(a) Schema 1 **(b)** Schema 2 **(c)** Schema 3 **(d)** Schema 4
Ans: b

Q Which one of the following statements if FALSE? **(GATE- 2017) (1 Marks)**
a) Any relation with two attributes is in BCNF
b) A relation in which every key has only one attribute is in 2NF
c) A prime attribute can be transitively dependent on a key in a 3NF relation
d) A prime attribute can be transitively dependent on a key in a BCNF relation
ANSWER D

Q A database of research articles in a journal uses the following schema. **(GATE- 2016) (2 Marks)**

(VOLUME, NUMBER, STARTPAGE, ENDPAGE, TITLE, YEAR, PRICE)
The primary key is (VOLUME, NUMBER, STARTPAGE, ENDPAGE) and the following functional dependencies exist in the schema.

(VOLUME, NUMBER, STARTPAGE, ENDPAGE) → TITLE
(VOLUME, NUMBER) → YEAR
(VOLUME, NUMBER, STARTPAGE, ENDPAGE) → PRICE

The database is redesigned to use the following schemas.

(VOLUME, NUMBER, STARTPAGE, ENDPAGE, TITLE, PRICE)
(VOLUME, NUMBER, YEAR) Which is the weakest normal form that the new database satisfies, but the old one does not

(1) 1NF **(2)** 2NF **(3)** 3NF **(4)** BCNF

Ans: 2

Q If every non-key attribute is functionally dependent on the primary key, then the relation is in _____. **(NET-NOV-2017)**

(1) First normal form
(2) Second normal form
(3) Third normal form
(4) Fourth normal form

Ans: (2)

Q In RDBMS, different classes of relations are created using _____ technique to prevent modification anomalies. **(NET-NOV-2017)**

(1) Functional Dependencies
(2) Data integrity
(3) Referential integrity
(4) Normal Forms

Ans: 4

Q For a database relation R(a, b, c, d) where the domains of a, b, c and d include only atomic values, and only the following functional dependencies and those that can be inferred from them hold : a → c b → d The relation is in _____.**(NET-JULY-2017)**

(1) First normal form but not in second normal form
(2) Second normal form but not in third normal form
(3) Third normal form
(4) BCNF

Ans: a

Q For a database relation R(A, B, C, D) where the domains of A, B, C and D include only atomic values, only the following functional dependencies and those that can be inferred from them are : A → C B → D The relation R is in _____. **(NET-JAN-2017)**

(1) First normal form but not in second normal form.

(2) Both in first normal form as well as in second normal form.
(3) Second normal form but not in third normal form.
(4) Both in second normal form as well as in third normal form.
Ans: a

Q Consider a relation R (A, B, C, D, E, F, G, H), where each attribute is atomic, and following functional dependencies exist. **(NET-NOV-2017)**
CH → G
A → BC
B → CFH
E → A
F → EG
The relation R is _____ .
(1) in 1NF but not in 2NF
(2) in 2NF but not in 3NF
(3) in 3NF but not in BCNF
(4) in BCNF
Ans: 1

Q Which of the following statements is false? **(NET-DEC-2014)**
(A) Any relation with two attributes is in BCNF.
(B) A relation in which every key has only one attribute is in 2NF.
(C) A prime attribute can be transitively dependent on a key in 3NF relation.
(D) A prime attribute can be transitively dependent on a key in BCNF relation.
Ans: d

Q The best normal form of relation scheme R(A, B, C, D) along with the set of functional dependencies F = {AB → C, AB → D, C → A, D → B} is **(NET-DEC-2014)**
(A) Boyce-Codd Normal form
(B) Third Normal form
(C) Second Normal form
(D) First Normal form
Ans: b

Q Given the following two statements:
S1: Every table with two single-valued attributes is in 1NF, 2NF, 3NF and BCNF.
S2: AB->C, D->E, E->C is a minimal cover for the set of functional dependencies AB->C, D->E, AB->E, E->C.
Which one of the following is CORRECT? **(GATE- 2014) (1 Marks)**
A) S1 is TRUE and S2 is FALSE.
B) Both S1 and S2 are TRUE.
C) S1 is FALSE and S2 is TRUE.
D) Both S1 and S2 are FALSE.
Ans: a

Q Which of the following is TRUE? **(GATE- 2012) (1 Marks)**

a) Every relation in 3NF is also in BCNF

b) A relation R is in 3NF if every non-prime attribute of R is fully functionally dependent on every key of R

c) Every relation in BCNF is also in 3NF

d) No relation can be in both BCNF and 3NF

Ans: c

Q For a database relation R(a, b, c, d) where the domains of a, b, c, d include only the atomic values. The functional dependency a → c, b → d holds in the following relation **(NET-JUNE-2013)**

(A) In 1NF not in 2NF
(B) In 2NF not in 3NF
(C) In 3NF
(D) In 1NF

Ans: a

Q Which of the following is true? **(NET-DEC-2012)**

(A) A relation in BCNF is always in 3NF.
(B) A relation in 3NF is always in BCNF.
(C) BCNF and 3NF are same.
(D) A relation in BCNF is not in 3NF.

Ans: a

Q A function that has no partial functional dependencies is in ……. form. **(NET-DEC-2009)**

(A) 3 NF **(B)** 2 NF **(C)** 4 NF **(D)** BCNF

Ans: b

Q Which of the following is true? **(NET-JUNE-2008)**

(A) A relation in 3NF is always in BCNF
(B) A relation in BCNF is always in 3NF
(C) BCNF and 3NF are totally different
(D) A relation in BCNF is in 2NF but not in 3NF

Ans: b

Q Relation R with an associated set of functional dependencies, F is decomposed into BCNF. The redundancy (arising out of functional dependencies) in the resulting set relations is. **(GATE- 2002) (1 Marks)**

a) Zero

b) More than zero but less than that of an equivalent 3NF decomposition

c) Proportional to the size of F+
d) Indeterminate
Ans: a

Relation R has eight attributes ABCDEFGH. Fields of R contain only atomic values.

F = {CH→G, A→BC, B→CFH, E→A, F→EG} is a set of functional dependencies (FDs) so that F^+ is exactly the set of FDs that hold for R.

The relation R is
(1) in 1NF, but not in 2NF
(2) in 2NF, but not in 3NF
(3) in 3NF, but not in BCNF
(4) in BCNF
Ans: 1

Q Consider the relation schema R (A B C D) with following FD set
F = {A → BC, C → D}; The relation R is in _____
a) 1 NF **b)** 2 NF **c)** 3 NF **d)** BCNF
Ans b

Q Consider the relation R (ABCDE) with the FD set F = {A → CE, B → D, AE → D}. Identify the highest normal form satisfied by the relation R.
a) 1 NF **b)** 2 NF **c)** 3 NF **d)** BCNF

Q Consider the following relational schema:
Suppliers (Sid: integer, sname: string, city: string, street: string)
Parts (pid: integer, pname: string, color: string)
Catalog (sid: integer, pid: integer, cost: real)

Assume that, in the supplier's relation above, each supplier and each street within a city has a unique name, and (sname, city) forms a candidate key. No other functional dependencies are implied other than those implied by primary and candidate keys. Which one of the following is TRUE about the above schema? **(GATE- 2009) (1 Marks)**
a) The schema is in BCNF
b) The schema is in 3NF but not in BCNF
c) The schema is in 2NF but not in 3NF
d) The schema is not in 2NF
Ans: a

Q Consider the following relational schemes for a library database
Book (Title, Author, Catalog_no, Publisher, Year, Price)
Collection (Title, Author, Catalog_no)
with in the following functional dependencies:
I. Title Author --> Catalog_no
II. Catalog_no --> Title Author Publisher Year
III. Publisher Title Year --> Price
Assume {Author, Title} is the key for both schemes. Which of the following statements is true? **(GATE- 2008) (1 Marks)** **(NET-JUNE-2014)**
(A) Both Book and Collection are in BCNF
(B) Both Book and Collection are in 3NF only
(C) Book is in 2NF and Collection is in 3NF
(D) Both Book and Collection are in 2NF only
Answer: (C)

Q The relation scheme Student Performance (name, courseNo, rollNo, grade) has the following functional dependencies:
name, courseNo → grade
rollNo, courseNo → grade
name → rollNo
rollNo → name
The highest normal form of this relation scheme is **(GATE- 2004) (1 Marks)**
a) 2 NF **b) 3 NF** **c) BCNF** **d) 4 NF**
Ans: b

Q Consider the following functional dependencies in a database **(GATE- 2003) (1 Marks)**
Data_of_Birth → Age
Age → Eligibility
Name → Roll_number
Roll_number → Name
Course_number → Course_name
Course_number → Instructor
(Roll_number, Course_number) → Grade
The relation (Roll_number, Name, Date_of_birth, Age) is:
(A) In second normal form but not in third normal form
(B) In third normal form but not in BCNF
(C) In BCNF

(D) None of the above

Answer: (D)

Q R (W, X, Y, Z) F.D. {Z → W, Y → XZ, XW → Y} the normal form of R is _____

Q The Relation Vendor Order (V_no, V_ord_no, V_name, Qty_sup, unit_price) is in 2NF because **(NET-JUNE-2015)**

(1) Non_key attribute V_name is dependent on V_no which is part of composite key

(2) Non_key attribute V_name is dependent on Qty_sup

(3) Key attribute Qty_sup is dependent on primary_key unit price

(4) Key attribute V_ord_no is dependent on primary_key unit price

Ans. 1

"The Relation Vendor Order (V_no, V_ord_no, V_name, Qty_sup, unit_price) is in 2NF because: Non_key attribute V_name is dependent on V_no which is part of composite key."

Q Given the following two statements:

S1: Every table with two single-valued attributes is in 1NF, 2NF, 3NF and BCNF.

S2: AB->C, D->E, E->C is a minimal cover for the set of functional dependencies AB->C, D->E, AB->E, E->C.

Which one of the following is CORRECT? (GATE – 2014) (2 Marks)

(a) S1 is TRUE and S2 is FALSE

(b) Both S1 and S2 are TRUE

(c) S1 is FALSE and S2 is TRUE

(d) Both S1 and S2 are FALSE

Ans: a

Decomposition

2 NF DECOMPOSTION

Q R(ABCD)
AB>D, B>C

Q R(ABCDE)
A>B, B>E, C>D

Q R(ABCDEFGHIJ)(ABD) (1NF)
AB>C, AD>GH, BD>EF, A>I, H>J

3NF DECOMPOSITON

Q R(ABCDE)

A>B, B>E, C>D

Q R(ABCDEFGHIJ)

AB>C, A>DE, B>F, F>GH, D>IJ

Q R(ABCDE)

AB>C, B>D, D>E

BCNF DECOMPOSITION

Q R(ABCD) AB (2 NF) ABC BCD
AB>C, BC>D

Q R(ABCD) AB, BD (3 NF) AFTER DECOMPOSITION AD, BCD
AB>CD, D>A

Q R(ABCD) AB, BD (3 NF) AFTER DECOMPOSITION AD, BCD
AB>CD, D>A

Q R(ABCDE) (A, E, BC, CD) (3 NF) AFTER DECOMPOSITION BD, ABC, AEA>BC, CD>E, B>D, E>A

Q R(ABCD) (A, B) (2NF) AFTER DECOMPOSITION ABC, CD
A>B, B>A, B>C, C>D

Q R(ABDLPT) (AB) (1 NF) AFTER DECOMPOSITION TL, BPT, AB, AD
B>PT, T>L, A>D

Q R(ABCDE) (AB, BC, DB, EB) (3NF) AFTER BCNF DECOMPOSITION BC, EA, DE, CD
AB>C, C>D, D>E, E>A

Q Consider the Relation R (A, B, C, D) with the following functional dependencies A → B, C → D, B → C. The BCNF decomposition of R is

a) {(A, B), (C, D), (B, C)}
b) {(A, B), (C, D), (A, C)}
c) {(B, C), (A, D), (A, B)}
d) all of the above

Q If the relation R (ABCDE) with FD set {AB - > CDE, A -> C, C->D} is converted into BCNF then the number of foreign keys exist in resulting relations is _____

Q Consider the schema
R = {S, T, U, V} and the dependencies
S → T, T → U, U → V and V → S
If R = (R1 and R2) be a decomposition such that R1 ∩ R2= ϕ, then the decomposition is
(NET-JUNE-2014)

(A) not in 2NF
(B) in 2NF but not in 3NF
(C) in 3NF but not in 2NF
(D) in both 2NF and 3NF

Ans: d

FOURTH NORMAL FORM (4NF)

Multivalued Dependency-

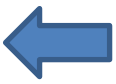
- Multivalued dependencies are a *consequence of first normal form (1NF*) which disallows an attribute in a tuple to have a *set of values (Multiple values)*.
- Denoted by, A ->-> B, Means, for every value of A, there may exist more than one value of B.
- If there is functional dependency from A-> B, then there will also a multivalued functional dependency from A->->B.
- A trivial multivalued dependency X→→Y is one where either *Y* is a subset of *X*, or *X* and *Y* together form the whole set of attributes of the relation.
- *E.g.* let the constraint specified by MVD in relation EMP as

  Ename ->-> Pname

  Ename ->-> Dname

EMP

Ename	***Pname***	***Dname***
Smith	*X*	*John*
Smith	*Y*	*Anna*
Smith	*X*	*Anna*
Smith	*Y*	*John*



Redundancy due to two independent multivalued dependencies in same Relation.

EMP_PROJECTS

Ename	***Pname***
Smith	X
Smith	Y

EMP_DEPENDENT

Ename	***Dname***
Smith	john
Smith	anna

NOTE: **The above EMP schema is in BCNF as no functional dependency holds on EMP, but still redundancy due to MVD.**

***Hence 4NF is stricter than BCNF*.**

Consider the following example:

Pizza Delivery Permutations		
Restaurant	**Pizza Variety**	**Delivery Area**
A1 Pizza	Thick Crust	Springfield
A1 Pizza	Thick Crust	Shelbyville
A1 Pizza	Thick Crust	Capital City
A1 Pizza	Stuffed Crust	Springfield
A1 Pizza	Stuffed Crust	Shelbyville
A1 Pizza	Stuffed Crust	Capital City
Elite Pizza	Thin Crust	Capital City
Elite Pizza	Stuffed Crust	Capital City
Vincenzo's Pizza	Thick Crust	Springfield
Vincenzo's Pizza	Thick Crust	Shelbyville
Vincenzo's Pizza	Thin Crust	Springfield
Vincenzo's Pizza	Thin Crust	Shelbyville

- Each row indicates that a given restaurant can deliver a given variety. The table has no non-key attributes because its only key is {Restaurant, Pizza Variety, Delivery Area}. Therefore, it meets all normal forms up to BCNF.
- If we assume, however, that pizza varieties offered by a restaurant are not affected by delivery area (i.e. a restaurant offers all pizza varieties it makes to all areas it supplies), then it does not meet 4NF. The problem is that the table features two non-trivial multivalued dependencies on the {Restaurant} attribute (which is not a super key). The dependencies are:
    - {Restaurant} →→ {Pizza Variety}
    - {Restaurant} →→ {Delivery Area}
- These non-trivial multivalued dependencies on a non-superkey reflect the fact that the varieties of pizza a restaurant offers are independent from the areas to which the restaurant delivers. This state of affairs leads to redundancy in the table:
- for example, we are told three times that A1 Pizza offers Stuffed Crust, and if A1 Pizza starts producing Cheese Crust pizzas then we will need to add multiple rows, one for each of A1 Pizza's delivery areas.

Varieties By Restaurant	
Restaurant	Pizza Variety
A1 Pizza	Thick Crust
A1 Pizza	Stuffed Crust
Elite Pizza	Thin Crust
Elite Pizza	Stuffed Crust
Vincenzo's Pizza	Thick Crust
Vincenzo's Pizza	Thin Crust

Delivery Areas By Restaurant	
Restaurant	Delivery Area
A1 Pizza	Springfield
A1 Pizza	Shelbyville
A1 Pizza	Capital City
Elite Pizza	Capital City
Vincenzo's Pizza	Springfield
Vincenzo's Pizza	Shelbyville

- If we have two or more multivalued *independent* attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attributes involved. This constraint is specified by a multivalued dependency.

4NF- A relation is in 4NF iff,

- It is in BCNF
- There must not exist any non-trivial multivalued dependency.
- Each MVD is decomposed in separate table, where it becomes trivial MVD.

- A 1992 paper by Margaret S. Wu notes that the teaching of database normalization typically stops short of 4NF, perhaps because of a belief that tables violating 4NF (but meeting all lower normal forms) are rarely encountered in business applications. This belief may not be accurate, however. Wu reports that in a study of forty organizational databases, over 20% contained one or more tables that violated 4NF while meeting all lower normal forms.

Q Multi-valued dependency among attribute is checked at which level? **(NET-JUNE-2005)**
(A) 2 NF **(B)** 3 NF **(C)** 4 NF **(D)** 5 NF

Q Which of the following is false? **(NET-DEC-2014)**
(A) Every binary relation is never be in BCNF.
(B) Every BCNF relation is in 3NF.
(C) 1 NF, 2 NF, 3 NF and BCNF are based on functional dependencies.
(D) Multivalued Dependency (MVD) is a special case of Join Dependency (JD).
Ans: a

Q Match the following: **(NET-DEC-2005)**

(i) 5 NF	**(a)** Transitive dependencies eliminated
(ii) 2 NF	**(b)** Multivalued attribute removed
(iii) 3 NF	**(c)** Contains no partial functional dependencies
(iv) 4 NF	**(d)** Contains no join dependency

(A) i-a, ii-c, iii-b, iv-d
(B) i-d, ii-c, iii-a, iv-b
(C) i-d, ii-c, iii-b, iv-a
(D) i-a, ii-b, iii-c, iv-d

Ans: b

Lossy/Lossless-Dependency Preserving Decomposition

- Because of a normalization a table is Decomposed into two or more tables, but during this decomposition we must ensure satisfaction of some properties out of which the most important is lossless join property/decomposition.
- if we decompose a table r into two tables r_1 and r_2 because of normalization then at some later stage if we want to join(combine) (natural join) these tables r_1 and r_2, then we must get back the original table r, without any extra or less tuple. But some information may be lost during retrieval of original relation or table. For e.g.

R (A, B, C)

A	B	C
1	a	p
2	b	q
3	a	r

R_1 (A, B)

A	B
1	a
2	b
3	a

R_2 (B, C)

B	C
a	p
b	q
a	r

R (A, B, C)

A	B	C
1	a	p
1	a	r
2	b	q
3	a	p
3	a	r

- Decomposition is lossy if R1 ⋈ R2 ⊃ R
- Decomposition is lossy if R ⊃ R1 ⋈ R2

- Decomposition is lossless if $R_1 \bowtie R_2 = R$ "The decomposition of relation R into R1 and R2 is **lossless** when the join of R1 and R2 yield the same relation as in R." which guarantees that the spurious (extra or less) tuple generation problem does not occur with respect to the relation schemas created after decomposition.
- ***This property is extremely critical and must be achieved at any cost.***

A	B	C	D	E
A	122	1	W	A
E	236	4	X	B
A	199	1	Y	C
B	213	2	Z	D

How to check for lossless join decomposition using FD set, following conditions must hold:

- Union of Attributes of R_1 and R_2 must be equal to attribute of R. Each attribute of R must be either in R_1 or in R_2. Att(R1) U Att(R2) = Att(R)
- Intersection of Attributes of R1 and R2 must not be NULL. Att(R1) ∩ Att(R2) ≠ Φ
- Common attribute must be a key for at least one relation (R1 or R2)
- Att(R1) ∩ Att(R2) -> Att(R1) or Att(R1) ∩ Att(R2) -> Att(R2)
- If X ∩ Y forms a superkey of either X or Y, the decomposition of R is a lossless decomposition.

E.g. Detailed example Explaination:-

Consider the following relationship : **R (A,B,C,D)**

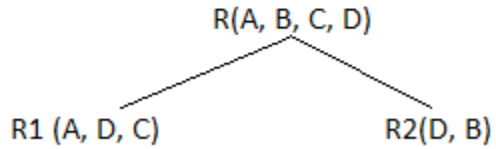
and following dependencies :

A -> BCD
BC -> AD
D -> B

Above relationship is already in 3rd NF. Keys are **A** and **BC.**

Hence, in the functional dependency, **A -> BCD**, A is the super key.
in second relation, **BC -> AD**, BC is also a key.
but in, **D -> B**, D is not a key.

Hence we can break our relationship R into two relationships **R1** and **R2.**



Breaking, table into two tables, one with A, D and C while the other with D and B.

5 NF

A Relational table R is said to be in 5th normal form if
a) it is in 4 NF
B) it cannot be further non-loss decomposed

Dependency Preserving Decomposition

Let relation R be decomposed into Relations $R_1, R_2, R_3 \ldots\ldots\ldots\ldots R_N$ with their respective functional Dependencies set as $F_1, F_2, F_3 \ldots\ldots\ldots\ldots F_N$, then the Decomposition is Dependency Preserving iff-

$$\{F_1 \cup F_2 \cup F_3 \cup F_4 \ldots\ldots\ldots \cup F_N\}^+ = F^+$$

Dependency preservation property, although desirable, is sometimes sacrificed.

Q R (A, B, C)
A--> B, B→C, C-→A
R_1(A, B) AND R_2(B, C)
LOSSLESS AND FD PREVERSING

Q R (A, B, C, D)
AB→CD, D→A
R_1(A, D), R_2(B, C, D)
LOSSLESS AND NOT FD PREVERSING

Q R (A, B, C, D)
A--> B, B→C, C-→D, D-→A
R1(A, B), R2(B, C) AND R3(C, D)
LOSSLESS AND FD PREVERSING

Q R(ABCDEG)(NF)R1(ABC) R2(ABDE) R3(EG)(LS)(DP)
AB>C
AC>B
AD>E
B>D
BC>A
E>G

Q R(ABCDE)(NF) R1(AB) R2(BC) R3(ABCD) R4(EG)(! LS)()
A>BC
C>DE
D>E

E.g.1 R (A, B, C, D)
F={AB->C, C->A, C->D}
Given decomposition as R1(A, C, D) and R2 (B, C)
Solution-
$R_1 \cup R_2 = R$ [True]
$R_1 \cap R_2 = C = \phi$
$C^+ = \{A, C, D\}$ So C -> R_1
Hence Given decomposition is LOSSLESS.

E.g. 2 R (A, B, C, D)
F={B->C, D->A}
Decomposition as –
R_1(B, C) and R_2(A, D)
Solution-
1.) $R_1 \cup R_2 = R$ [True]
2.) $R_1 \cap R_2 = \phi$ [not satisfied]
Hence LOSSY Decomposition

Q Consider a scheme R (A B C D E)
F → D
D → F
C → A D

AB → C
is decomposed into R, (ABC)& R(CDE)is the decomposition lossy

Q Identify the lossy decomposition R (ABCD)F.D [A → B, B → C D]
a) (ABC) (BD) b) (AB) (BC)(CD) c) (AB) (CD) d) None

Q Identify the lossy decomposition on the relation R(ABCD) with functional dependencies
A →B
B→ C
C → D
a) (ABC) (BD) b) (AB) (BC) (CD) c) (AB)(CD) d) None of these

Q Consider the relation R (ABCD) with the FD set F = {A → BC, B → CD, C→ AD} which is decomposed into set of tables D = {(AB), (BC), CD}. Which of the following is true about the decomposition D?
a) It is lossless and dependency preserving
b) It is lossy but dependency preserving
c) It is lossless but dependencies are nor preserved
d) It is neither lossless nor dependency preserving

Q R(A,B,C,D) is a relation. Which of the following does not have a lossless join, dependency preserving BCNF decomposition? **(Gate - 2001) (2 Marks)**
(A) A->B, B->CD **(B)** A->B, B->C, C->D
(C) AB->C, C->AD **(D)** A ->BCD
Answer: (C)

Q Consider a schema R (A, B, C, D) and functional dependencies A->B and C->D
Then the decomposition of R into R1(AB) and R2(CD) is **(GATE-2001) (2 Marks)**
(A) dependency preserving and lossless join
(B) lossless join but not dependency preserving
(C) dependency preserving but not lossless join
(D) not dependency preserving and not lossless join
Answer: (C)

Q Let the set of functional dependencies F = {QR → S, R → P, S → Q} hold on a relation schema X = (PQRS). X is not in BCNF. Suppose X is decomposed into two schemas Y and Z, where Y = (PR) and Z = (QRS).
Consider the two statements given below.

I. Both Y and Z are in BCNF

II. Decomposition of X into Y and Z is dependency preserving and lossless

Which of the above statements is/are correct? **(GATE- 2019) (1 Marks)**

(a) I only **(b)** Neither I nor II

(c) II only **(d)** Both I and II

Ans: c

Q Consider a schema R(A, B, C, D) and functional dependencies A → B and C → D. Then the decomposition R1(A, B) and R2(C, D) is **(NET-JUNE-2012) (Gate - 2001) (2 Marks)**

(A) Dependency preserving but not lossless join

(B) Dependency preserving and lossless join

(C) Lossless Join but not dependency preserving

(D) Lossless Join

Ans: c

Q Consider a schema R(A, B, C, D) and following functional dependencies.

A → B

B → C

C → D

D → B

Then decomposition of R into R1(A, B), R2(B, C) and R3(B, D) is _____. **(NET-NOV-2017)**

(1) Dependency preserving and lossless join.

(2) Lossless join but not dependency preserving.

(3) Dependency preserving but not lossless join.

(4) Not dependency preserving and not lossless join.

Ans: a

Q Consider a schema R(MNPQ) and functional dependencies M → N, P → Q. Then the decomposition of R into R1(MN) and R2(PQ) is _____. **(NET-JAN-2017)**

(1) Dependency preserving but not lossless join

(2) Dependency preserving and lossless join

(3) Lossless join but not dependency preserving

(4) Neither dependency preserving nor lossless join.

Ans: 1

Q Which of the following statements is TRUE? **(NET-JULY-2016)**

D1: The decomposition of the schema R(A, B, C) into R1(A, B) and R2 (A, C) is always lossless.

D2: The decomposition of the schema R(A, B, C, D, E) having AD → B, C → DE, B → AE and AE → C, into R1 (A, B, D) and R2 (A, C, D, E) is lossless.

(1) Both D1 and D2 **(2)** Neither D1 nor D2

(3) Only D1 **(4)** Only D2

Ans. 4

Only D2 is True because AD is key and present in both the tables. D1 is not always true because FD's not given and if we take B->A and C->A then it is lossy decomposition because no common attributes contain key from one of the tables.

Q Consider the table R with attributes A, B and C. The functional dependencies that hold on R are : A → B, C → AB. Which of the following statements is/are True? **(NET-AUG-2016)**

I. The decomposition of R into R1(C, A) and R2(A, B) is lossless.

II. The decomposition of R into R1(A, B) and R2(B, C) is lossy.

(1) Only I **(2)** Only II **(3)** Both I and II **(4)** Neither I nor II

Ans: c

Q

If a relation with a Schema R is decomposed into two relations R_1 and R_2 such that $(R_1 \cup R_2) = R_1$ then which one of the following is to be satisfied for a lossless joint decomposition (→ indicates functional dependency)

(A) $(R_1 \cap R_2) \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$

(B) $R_1 \cap R_2 \rightarrow R_1$

(C) $R_1 \cap R_2 \rightarrow R_2$

(D) $R_1 \cap R_2 \rightarrow R_1$ and $R_1 \cap R_2 \rightarrow R_2$

Q (NET-DEC-2010)

Match the following :

I.	2 NF	(a)	transitive dependencies eliminated
II.	3 NF	(b)	multivalued attribute removed
III.	4 NF	(c)	contain no partial functional dependencies
IV.	5 NF	(d)	contains no join dependency

Codes :

	I	II	III	IV
(A)	(a)	(c)	(b)	(d)
(B)	(d)	(a)	(b)	(c)
(C)	(c)	(d)	(a)	(b)
(D)	(d)	(b)	(a)	(c)

Q The dependency preservation decomposition is a property to decompose database schema D, in which each functional dependency X → Y specified in F, **(NET-DEC-2010)**
(A) appeared directly in one of the relation schemas Ri in the decomposed D.
(B) could be inferred from dependencies that appear in some Ri.
(C) both (A) and (B)
(D) None of these
Ans c

Q The relation schemas R1 and R2 form a Lossless join decomposition of R if and only if: **(NET-JUNE-2015)**
(a) R1 ∩ R2 ↠ (R1 - R2)
(b) R1 → R2
(c) R1 ∩ R2 ↠ (R2 - R1)
(d) (R2 → R1) ∩ R2
(1) (a) and (b) happens
(2) (a) and (d) happens
(3) (a) and (c) happens
(4) (b) and (c) happens
Ans. 3

Q Relation R is decomposed using a set of functional dependencies, F and relation S is decomposed using another set of functional dependencies G. One decomposition is definitely BCNF, the other is definitely 3NF, but it is not known which is which. To make a guaranteed identification, which one of the following tests should be used on the

decompositions? (Assume that the closures of F and G are available).(Gate-2002) (2 Marks)
(A) Dependency-preservation **(B)** Lossless-join
(C) BCNF definition **(D)** 3NF definition
Answer: (B)

Q Suppose R is a relation schema and F is a set of functional dependencies on R. Further, suppose R_1 and R_2 form a decomposition of R. Then the decomposition is a lossless join decomposition of R provided that: **(NET-DEC-2008)**
(A) $R_1 \cap R_2 \rightarrow R_1$ is in F^+
(B) $R_1 \cap R_2 \rightarrow R_2$ is in F^+
(C) both $R_1 \cap R_2 \rightarrow R_1$ and $R_1 \cap R_2 \rightarrow R_2$ functional dependencies are in F^+
(D) at least one from $R_1 \cap R_2 \rightarrow R_1$ and $R_1 \cap R_2 \rightarrow R_2$ is in F^+
Ans: d

Q Select the 'False' statement from the following statements about Normal Forms: **(NET-JUNE-2015)**
(1) Lossless preserving decomposition into 3NF is always possible
(2) Lossless preserving decomposition into BCNF is always possible
(3) Any Relation with two attributes is in BCNF
(4) BCNF is stronger than 3NF
Ans. 2

Q Which one of the following statements about normal forms is FALSE? (GATE-2005) (2 Marks)
(A) BCNF is stricter than 3NF
(B) Lossless, dependency-preserving decomposition into 3NF is always possible
(C) Lossless, dependency-preserving decomposition into BCNF is always possible
(D) Any relation with two attributes is in BCNF
Answer: (C)

Q Choose the correct statements.
I: 3 NF decomposition is always lossless join and dependency preserving.
II: 3 NF decomposition is always lossless join but may or may not be dependency preserving
III: BCNF decomposition always lossless join and dependency preserving.
IV: BCNF decomposition is always lossless join but may or may not be dependency preserving.
a) Both I and III b) Both II and IV c) Both I and IV d) Both II and III

Q Consider the given relation R (ABCDEFG) with attributes as follows: -

A→ Key Attribute
B and C → Atomic Attributes
D and E → Multi – valued Attributes
F and G → Composite Attributes
Which of the following is the correct 1 NF decomposition for the above relation?
a) R1 (ABC), R2(AD), R3 (AE), R4(AF), R5 (AG)
b) R1 (AB), R2(BC), R3 (DE), R4(FG)
c) R1 (ABC), R2 (ADE), R3 (AFG)
d) R1 (ABC), R2 (DEFG)

Database Design goals	1NF	2NF	3NF	BCNF
0% Redundancy	No	NO	No	Yes[due to FD's] NO[due to MVD's]
Lossless Decomposition	YES [Always]	YES [Always]	YES [Always]	YES [Always]
Dependency Preservation.	YES [Always]	YES [Always]	YES [Always]	May not be Always.

Indexing

- Reason for indexing - For a large file when it contains a large number of records which will eventually acquire large number of blocks, then its access will become slow. In todays events we want a high-speed database.
- Theoretically relational database is derived from set theory, and in a set the order of elements in a set is irrelevant, so does in relations(tables).
- But in practice implementation we have to specify the order.
- A number of properties, like search, insertion and deletion will depend on the order in which elements are stored in the tables.

File organization/ organization of records in a file

1) Ordered file organization: - All the records in the file are ordered on some search key field. Here binary search is possible for e.g. dictionary.

- Because of binary search, searching is efficient
- Maintenance (insertion & deletion) is costly, as it requires re organization of entire file.
- Notes that we will get binary search only if we are using that key for searching on which indexing is done, otherwise it will behave as unsorted file

employee_id	first_name	last_name	hire_date	salary
100	Steven	King	1987-06-17	24000.00
101	Neena	Kochhar	1989-09-21	17000.00
102	Lex	De Haan	1993-01-13	17000.00
103	Alexander	Hunold	1990-01-03	9000.00
104	Bruce	Ernst	1991-05-21	6000.00
105	David	Austin	1997-06-25	4800.00
106	Valli	Pataballa	1998-02-05	4800.00

2) Unordered file organization: - All the records are inserted usually in the end of the file so not ordered according to any field, Because of this only linear search is possible.

- Because of linear search, searching is slow
- Maintenance (insertion & deletion) is easy, as it does not require re organization of entire file.

Q Suppose we have ordered file with records stored r = 30,000 on a disk with Block Size B = 1024 B. File records are of fixed size and are unspanned with record length R = 100 B. Suppose that ordering key field of file is 9 B long and a block pointer is 6 B long, Implement primary indexing?

Important Points about Indexing

- Additional auxiliary access structure is called indexes, a data technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.
- Index typically provides secondary access path, which provide alternative way to access the records without affecting the physical placement of records in the main file.
- Index file is always ordered, irrespective of weather main file is ordered or unordered. So that we can take the advantage of binary search.

Secondary Index

Search key	Address
Azad	0x2300
Priya	0x2100
Ram	0x1F00
Tom	0x2200
Vishal	0X2000

Data File

Address	RegNo	Name	Program	Age
0x1F00	14MS01	Ram	MS CS	21
0x2000	14MS10	Vishal	MS CS	22
0x2100	14MS29	Priya	MS CS	21
0x2200	14MS30	Tom	MS CS	21
0X2300	14MT59	Azad	MTech	23

- Index file always contains two columns one the attribute on which search will be done and other the block or record pointer.
- The size of index file is way smaller than that of the main file.
- As the size of the records is very smaller compare to the main file, as index file record contain only two columns key (attribute in which searching is done) and block pointer (base address of the block of main file which contains the record holding the key), while main file contains all the columns.
- Normally apart from secondary indexing (a type of indexing), the number of records in index file <= the number records in the main file.

- One index file is designed according to an attribute, means more than one index file can be designed for a main file.
- Indexing gives the advantage of faster time, but space taken by index file will be an overhead.
- Number of access required to search the correct block of main file is $\log_2$(number of blocks in index file) + 1
- Index can be created on any field of relation (primary key, non-key)

Q Data which improves the performance and accessibility of the database are called: **(NET-DEC-2015)**

(1) Indexes
(2) User Data
(3) Application Metadata
(4) Data Dictionary

Ans. 1

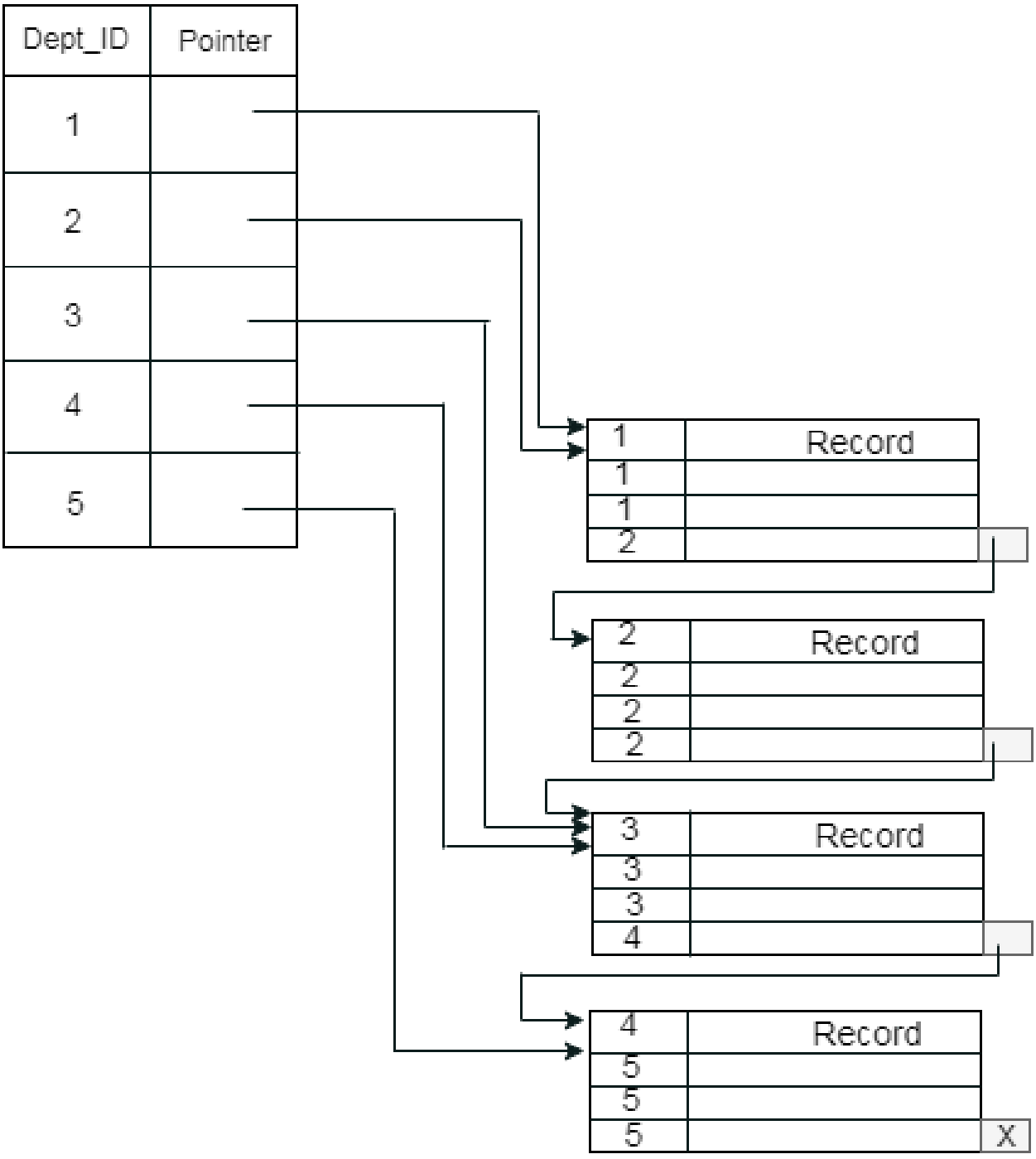
Indexing can be classified on number of criteria's one of them could be –

- Dense Index
- Sparse Index

***DENSE INDEX**: -*

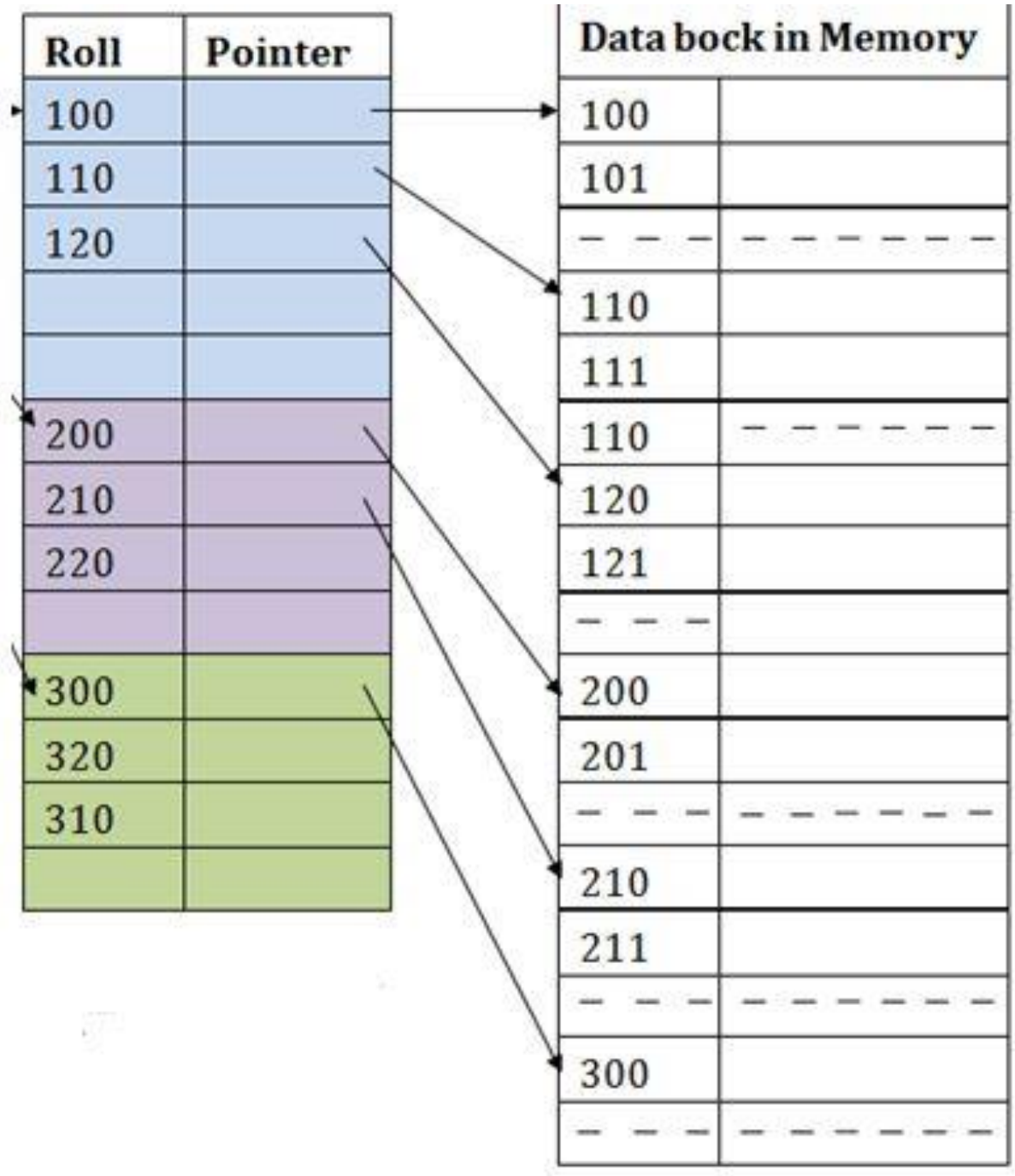
- In dense index, there is an entry in the index file for every search key value in the main file. This makes searching faster but requires more space to store index records itself.

- Note that it is not for every record, it is for every search key value. Sometime number of records in the main file >= number of search keys in the main file, for example if search key is repeated.



SPARSE INDEX:

- If an index entry is created only for some records of the main file, then it is called sparse index.
- No. of index entries in the index file < No. of records in the main file.

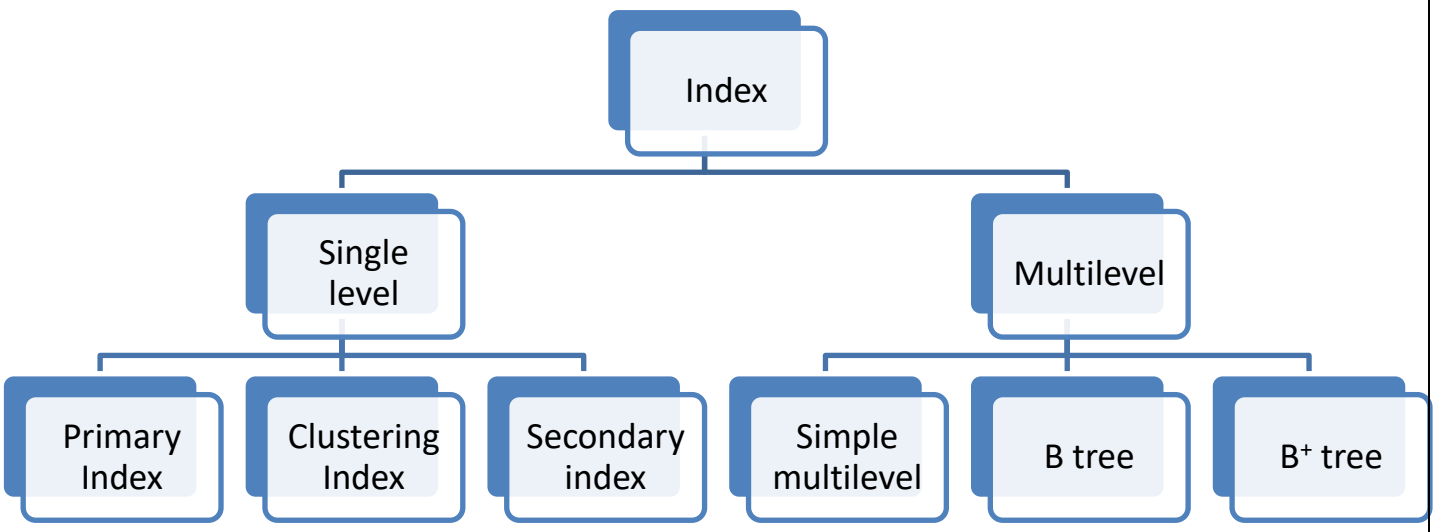


- Note: - dense and sparse are not complementary to each other, sometimes it is possible that a record is both dense and sparse.

Basic term used in Indexing

- BLOCKING FACTOR = No. of Records per block= ⌊block size/record size⌋
- No of blocks required by file = ⌈no of records / blocking factor⌉
- If file is unordered then no of block assesses required to reach correct block which contain the desired record is O(n), where n is the number of blocks.
- if file is unordered then no of block assesses required to reach correct block which contain the desired record is $O(\log_2 n)$, where n is the number of blocks.

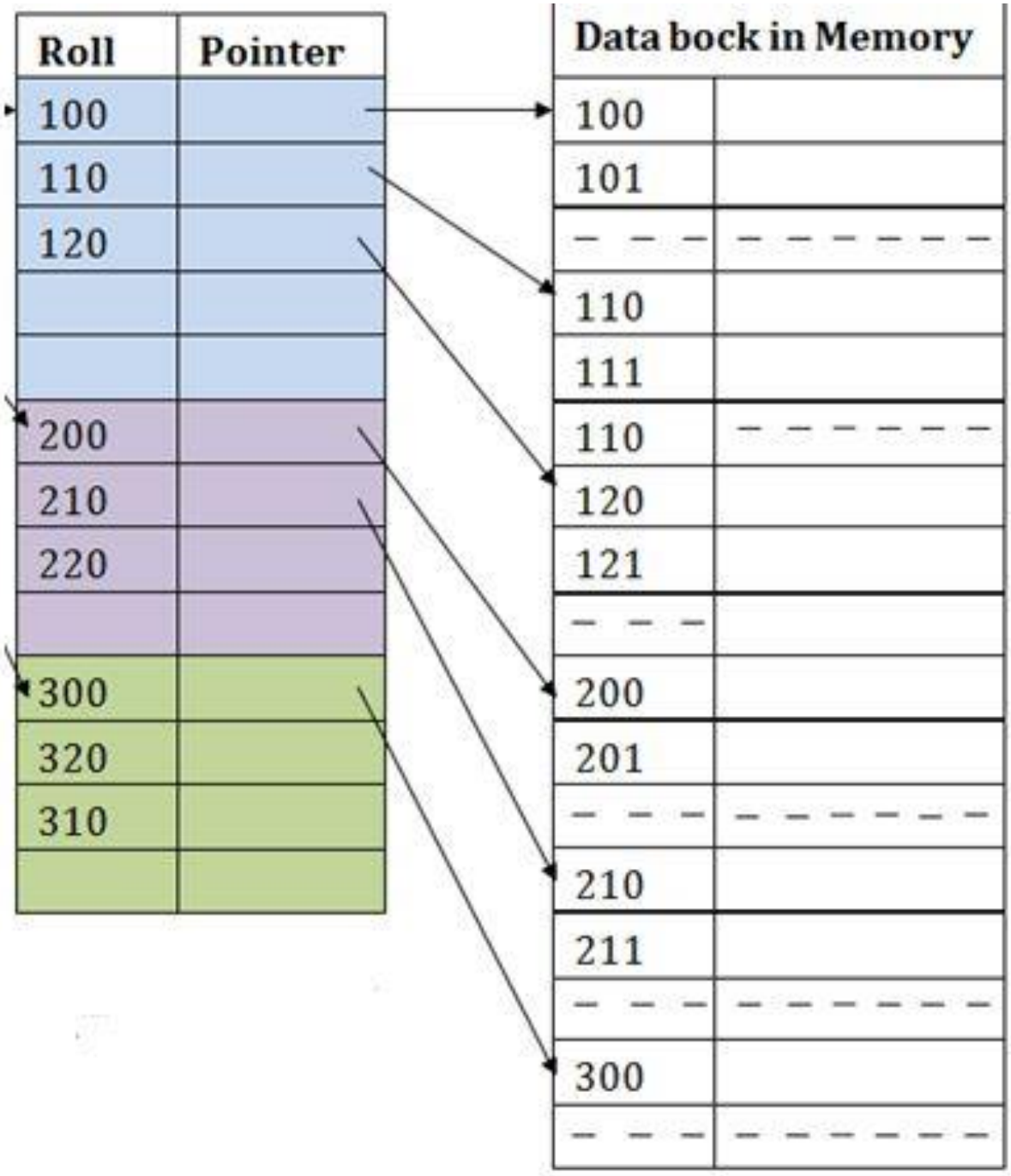
TYPES OF INDEXING



- Single level index means we create index file for the main file, and then stop the process.
- Multiple level index means, we further index the index file and keep repeating the process until we get one block.

PRIMARY INDEXING

- Main file is always sorted according to primary key.
- Indexing is done on Primary Key, therefore called as primary indexing
- Index file have two columns, first primary key and second anchor pointer (base address of block)
- It is an example of Sparse Indexing.
- Here first record (anchor record) of every block gets an entry in the index file
- No. of entries in the index file = No of blocks acquired by the main file.



CLUSTERED INDEXING

- Main file will be ordered on some non-key attributes
- No of entries in the index file = no of unique values of the attribute on which indexing is done.
- It is the example of Sparse as well as dense indexing

Q An index is clustered, if **(GATE-2013) (1 Marks)**
(1) it is on a set of fields that form a candidate key
(2) it is on a set of fields that include the primary key
(3) the data records of the file are organized in the same order as the data entries of the index
(4) the data records of the file are organized not in the same order as the data entries of the index
Ans: 3

Q A clustering index is defined on the fields which are of type **(GATE-2008) (1 Marks)**
1) non-key and ordering
2) non-key and non-ordering
3) key and ordering
4) key and non-ordering
ANSWER A

Q A clustering index is created when_____. **(NET-DEC-2014)**
(A) primary key is declared and ordered
(B) no key ordered
(C) foreign key ordered
(D) there is no key and no order
Ans: c

Q (NET-DEC-2018)
A clustering index is defined on the fields which are of type

Options :

91394342537. **non-key and ordering**

91394342538. **non-key and non-ordering**

91394342539. **key and ordering**

91394342540. **key and non-ordering**

SECONDARY INDEXING

- Most common scenarios, suppose that we already have a primary indexing on primary key, but there is frequent query on some other attributes, so we may decide to have one more index file with some other attribute.
- Main file is ordered according to the attribute on which indexing is done(unordered).
- Secondary indexing can be done on key or non-key attribute.
- No of entries in the index file is same as the number of entries in the index file.
- It is an example of dense indexing.

Q Suppose we have ordered file with records stored r = 30,000 on a disk with Block Size B = 1024 B. File records are of fixed size and are unspanned with record length R = 100 B. Suppose that ordering key field of file is 9 B long and a block pointer is 6 B long, Implement Secondary indexing?

Q Consider a file of 16384 records. Each record is 32 bytes long and its key field is of size 6 bytes. The file is ordered on a non-key field, and the file organization is unpanned. The file is stored in a file system with block size 1024 bytes, and the size of a block pointer is 10 bytes. If the secondary index is built on the key field of the file, and a multi-level index scheme is used to store the secondary index, the number of first-level and second-level blocks in the multi-level index are respectively **(GATE-2008) (1 Marks)**

1) 8 and 0 **2)** 128 and 6 **3)** 256 and 4 **4)** 512 and 5

ANSWER C

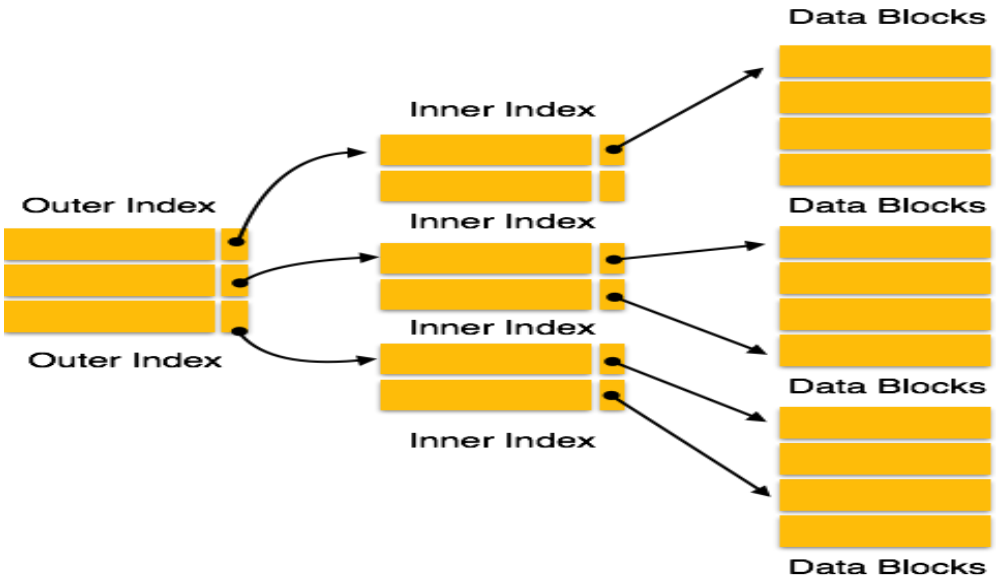
Q A file is organized so that the ordering of data records is the same as or close to the ordering of data entries in some index. Then that index is called **(GATE-2015) (1 Marks)**

(A) Dense **(B)** Sparse **(C)** Clustered **(D)** Unclustered

Answer: (C)

MULTILEVEL INDEXING

- Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block-0, which can easily be accommodated anywhere in the main memory.



Reason to have B tree and B+ tree

- After studying indexing in detail now we understand that an index file is always sorted in nature and will be searched frequently, and sometimes index files can be so large that even we want to index the index file (Multilevel index), therefore we must search best data structure to meet our requirements.
- There are number of options in data structure like array, stack, link list, graph, table etc. but we want a data structure which support frequent insertion deletion but at the same time also provide speed search and give us the advantage of having a sorted data.
- If we look at the data structures option then tree seems to be the most appropriate but every kind of tree in the available option have some problems either simple tree or binary search tree or AVL tree, so we end up on designing new data structure called B-tree which are kind of specially designed for sorted stored index files in databases.
- In general, with multilevel indexing, we require dynamic structure, b and b^+ tree is generalized implementation of multilevel indexing, which are dynamic in nature, that is increasing and decreasing number of records. In the first level index file can be easily supported by other level index.
- B tree and B^+ tree also provides efficient search time, as the height of the structure is very less and they are also perfectly balanced.

B tree

- A B-tree of order m if non-empty is an m-way search tree in which.
  - The root has at least two child nodes and at most m child nodes.
  - The internal nodes except the root have at least celling(m/2) child nodes and at most m child nodes.
  - The number of keys in each internal node is one less than the number of child nodes and these keys partition the subtrees of the nodes in a manner similar to that of m-way search tree.
  - All leaf nodes are on the same level.

Root

Rules	**MAX**	**MIN**
CHILD	m	0
DATA	m-1	1

Internal except Root

Rules	***MAX***	***MIN***
CHILD	*m*	[m/2]
DATA	*m-1*	[m/2] - 1

Leaf

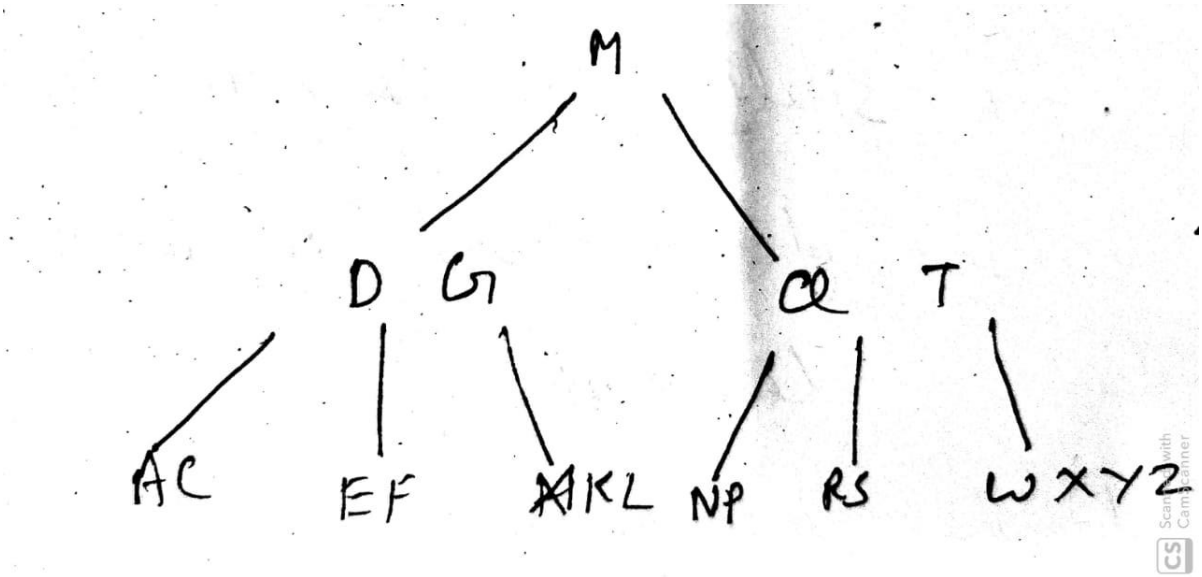
Rules	***MAX***	***MIN***
CHILD	*0*	*0*
DATA	*m-1*	[m/2] - 1

Q Consider the following elements 5, 10, 12, 13, 14, 1, 2, 3, 4 insert them into an empty b-tree of order = 3.

Q Consider the following elements 5, 10, 12, 13, 14, 1, 2, 4, 20, 18, 19, 17, 16, 15, 25, 23, 24, 22, 11, 30, 31, 28, 29 insert them into an empty b-tree of order = 3.

- A B-tree starts with a single root node (which is also a leaf node) at level 0 (zero). Once the root node is full with p – 1 search key values and we attempt to insert another entry in the tree, the root node splits into two nodes at level 1. Only the middle value is kept in the root node, and the rest of the values are split evenly between the other two nodes. When a non-roof node is full and a new entry is inserted into it, that node is split into two nodes at the same level, and the middle entry is moved to the parent node along with two pointers to the new split nodes. If the parent node is full, it is also split. Splitting can propagate all the way to the root node, creating a new level if the root is split.

Q Consider the Following B-tree of order m=6, delete the following nodes H, T, R, E, A, C, S in sequence?



Deletion in B-TREE-

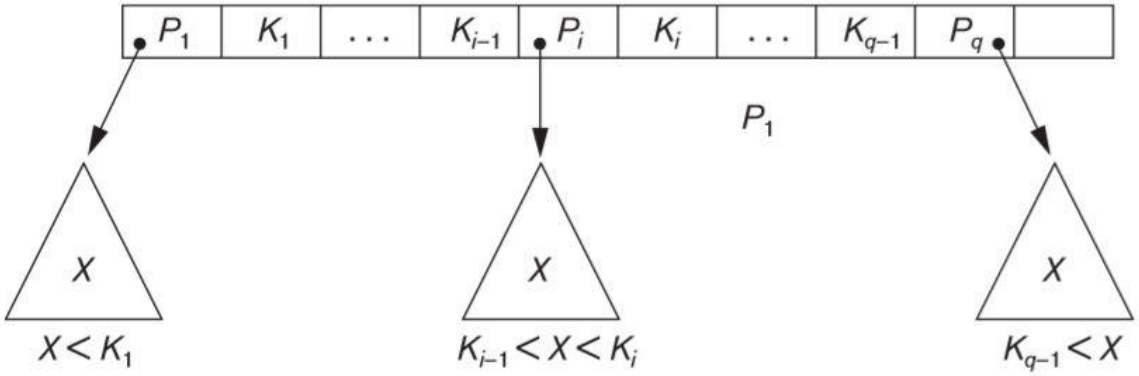
- If the deletion is from the leaf node and leaf node is satisfying the minimal condition even after the deletion, then delete the value directly.
- If deletion from leaf node renders leaf node in minimal condition, then first search the extra key in left sibling and then in the right sibling. Largest value from left sibling or smallest value from right sibling is pushed into the root node and corresponding value can be fetched from parent node to leaf node.
- If the deletion is to be from internal node, then first we check for the extra key in the left and then in the right child. If we find one, we fetch the value in the required node. And delete the key.

Deletion

- If deletion of a value causes a node to be less than half full, it is combined with its neighboring nodes, and this can also propagate all the way to the root. Hence, deletion can reduce the number of tree levels. It has been shown by analysis and simulation that, after numerous random insertions and deletions on a B-tree, the nodes are approximately 69 percent full when the number of values in the tree stabilizes. This is also true of B^+ trees. If this happens, node splitting and combining will occur only rarely, so insertion and deletion become quite efficient. If the number of values grows, the tree will expand without a problem—although splitting of nodes may occur, so some insertions will take more time.

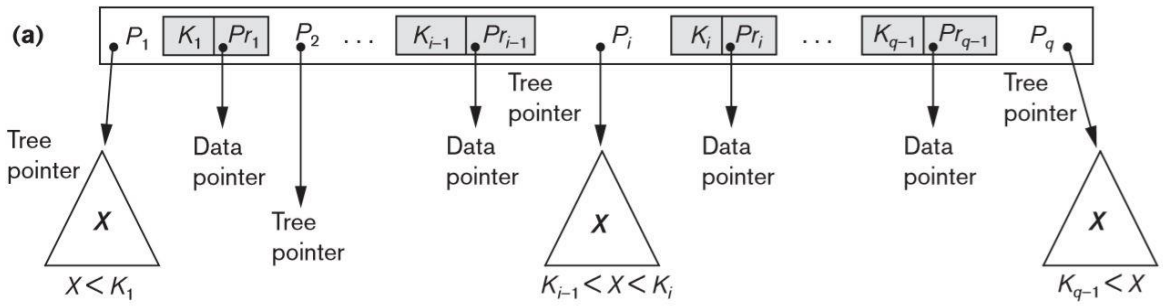
Analysis

- B- TREE- In computer science, a B-tree is a self-balancing tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time.
- A **search tree of order** *p* is a tree such that each node contains *at most p* -1 search values and *p* pointers in the order $<P_1, K_1, P_2, K_2, ..., P_{q-1}, K_{q-1}, P_q>$, where $q <= p$. Each P_i is a pointer to a child node (or a NULL pointer), and each K_i is a search value from some ordered set of values. All search values are assumed to be unique. Two constraints must hold at all times on the search tree:
  - Within each node, $K_1 < K_2 < ... < K_{q-1}$.
  - For all values X in the subtree pointed at by P_i, we have $K_{i-1} < X < K_i$ for $1 < i < q$; $X < K_i$ for $i = 1$; and $K_{i-1} < X$ for $i = q$.



- We can use a search tree as a mechanism to search for records stored in a disk file. The values in the tree can be the values of one of the fields of the file, called the **search field** (which is the same as the index field if a multilevel index guides the search). Each key value in the tree is associated with a pointer to the record in the data file having that value.
- To guarantee that nodes are evenly distributed, so that the depth of the tree is minimized for the given set of keys and that the tree does not get skewed with some nodes being at very deep levels.
- To make the search speed uniform, so that the average time to find any random key is roughly the same
- While minimizing the number of levels in the tree is one goal, another implicit goal is to make sure that the index tree does not need too much restructuring as records are inserted into and deleted from the main file. Thus, we want the nodes to be as full as possible and do not want any nodes to be empty if there are too many deletions. Record deletion may leave some nodes in the tree nearly empty, thus wasting storage space and increasing the number of levels. The B-tree addresses both of these problems by specifying additional constraints on the search tree.

- The B-tree has additional constraints that ensure that the tree is always balanced and that the space wasted by deletion, if any, never becomes excessive. The algorithms for insertion and deletion, though, become more complex in order to maintain these constraints. Nonetheless, most insertions and deletions are simple processes; they become complicated only under special circumstances—namely, whenever we attempt an insertion into a node that is already full or a deletion from a node that makes it less than half full. More formally, a **B-tree of order *p***, when used as an access structure on a *key field* to search for records in a data file, can be defined as follows:
- Each internal node in the B-tree is of the form
    - $<P_1, <K_1, Pr_1>, P_2, <K_2, Pr_2>, ..., <K_{q-1}, Pr_{q-1}>, P_q>$ where $q <= p$. Each P_i is a **tree pointer**—a pointer to another node in the B- tree. Each Pr_i is a **data pointer**—a pointer to the record whose search key field value is equal to K_i (or to the data file block containing that record).
    - Within each node, $K_1 < K_2 < ... < K_{q-1}$.
    - For all search key field values X in the subtree pointed at by P_i (the ith sub- tree), we have: $K_{i-1} < X < K_i$ for $1 < i < q$; $X < K_i$ for $i = 1$; and $K_{i-1} < X$ for $i = q$.
    - Each node has at most p tree pointers.
    - Each node, except the root and leaf nodes, has at least $\lceil (p/2) \rceil$ tree pointers. The root node has at least two tree pointers unless it is the only node in the tree.
    - A node with q tree pointers, $q <= p$, has $q - 1$ search key field values (and hence has $q - 1$ data pointers).
    - All leaf nodes are at the same level. Leaf nodes have the same structure as internal nodes except that all of their *tree pointers* P_i are NULL.



Conclusion: -

- Very less internal fragmentation, memory utilization is very good.
- Less number of nodes(blocks) are used and height is also optimized, so access will be very fast.
- Difficulty of traversing the key sequentially. Means B-TREE do not hold good for range-based queries of database.

Q A B-Tree used as an index for a large database table has four levels including the root node. If a new key is inserted in this index, then the maximum number of nodes that could be newly created in the process are: **(Gate-2005) (1 Marks)**

(A) 5 **(B)** 4 **(C)** 3 **(D)** 2

Answer: (A)

B^+ Tree

Q Consider the following elements 5, 10, 12, 13, 14, 1, 2, 3, 4 insert them into an empty b^+ tree of order = 3.

Q Consider the following elements 5, 10, 12, 13, 14, 1, 2, 4, 20, 18, 19, 17, 16, 15, 25, 23, 24, 22, 11, 30, 31, 28, 29 insert them into an empty b^+ tree of order = 5

Insertion in B^+ Tree-

- Start from root node and proceed towards leaf using the logic of binary search tree. Value is inserted in the leaf.
- If overflow condition occurs pick the median and push it into the parent node. Also copy the median or key inserted in parent node to the left or right child node.
- Repeat this procedure until tree is maintained.

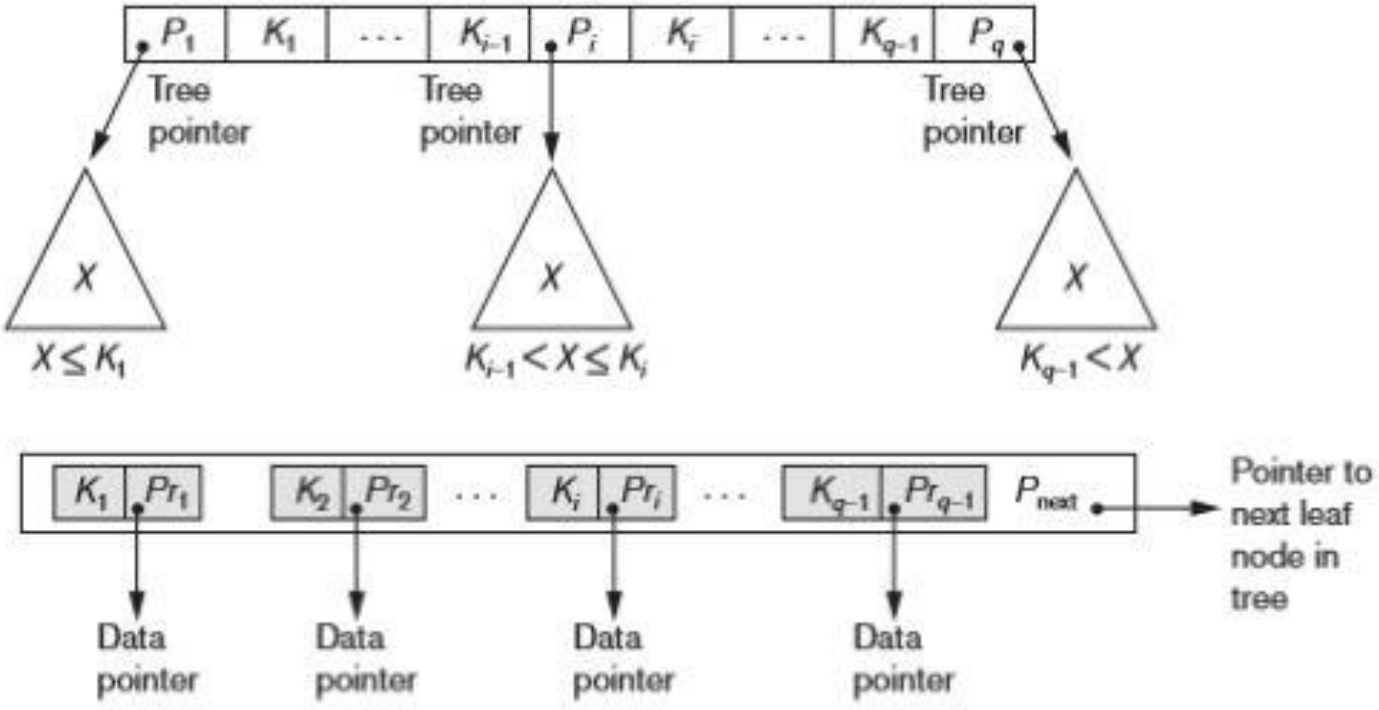
Q Consider the following elements 5, 8, 1, 7, 3, 12, 9, 6 insert them into an empty b^+ tree of order = 3. and then delete following nodes in sequence 9, 8, 12?

Deletion in B^+ Tree-

- if B^+ tree entries are deleted at the leaf nodes, then the target entry is searched and deleted.
- If it is an internal node, delete and replace with the entry from the left position.
- After deletion, underflow is tested,
- If underflow occurs, distribute the entries from the nodes left to it.
- If distribution is not possible from left, then
- Distribute from the nodes right to it.
- If distribution is not possible from left or from right, then Merge the node with left and right to it

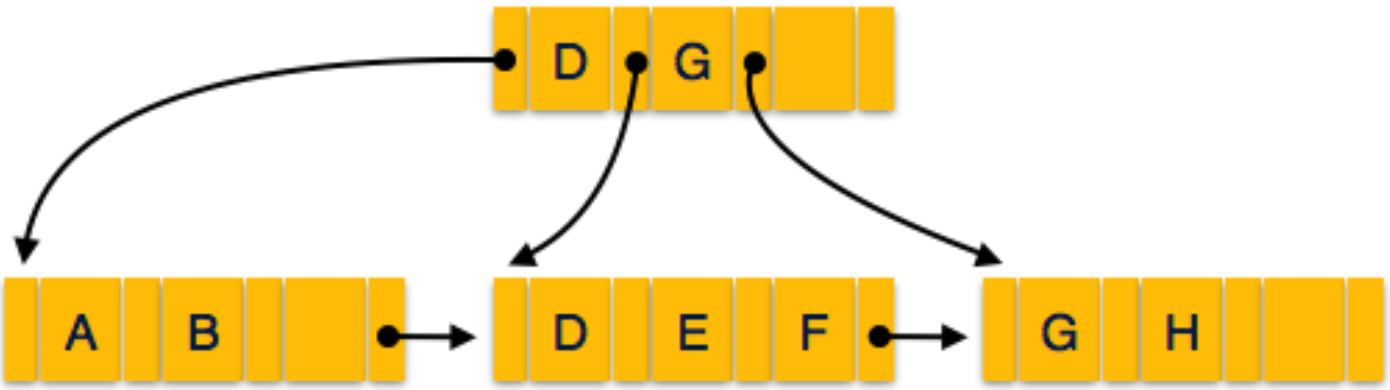
Analysis

- Most implementations of a dynamic multilevel index use a variation of the B-tree data structure called a **B^+-tree**. In a B-tree, every value of the search field appears once at some level in the tree, along with a data pointer.
- In a B^+-tree, data pointers are stored *only at the leaf nodes* of the tree; hence, the structure of leaf nodes differs from the structure of internal nodes.
- The leaf nodes have an entry for *every* value of the search field, along with a data pointer to the record (or to the block that contains this record) if the search field is a key field.
- The leaf nodes of the B^+-tree are usually linked to provide ordered access on the search field to the records.
- Each internal node is of the form $<P_1, K_1, P_2, K_2, ..., P_{q-1}, K_{q-1}, P_q>$
- Within each internal node, $K_1 < K_2 < ... < K_{q-1}$.
- For all search field values X in the subtree pointed at by P_i, we have $K_{i-1} < X <= K_i$ for $1 < i < q$; $X <= K_i$ for $i = 1$; and $K_{i-1} < X$ for $i = q$.
- Each internal node has at most p tree pointers.
- Each internal node, except the root, has at least $\lceil (p/2) \rceil$ tree pointers. The root node has at least two tree pointers if it is an internal node.
- An internal node with q pointers, $q <= p$, has $q - 1$ search field values.
- The structure of the *leaf nodes* of a B^+-tree of order p is as follows:
- Each leaf node is of the form $<<K_1, Pr_1>, <K_2, Pr_2>, ..., <K_{q-1}, Pr_{q-1}>, P_{next}>$ where $q <= p$, each Pr_i is a data pointer, and P_{next} points to the next *leaf node* of the B^+-tree.
- Within each leaf node, $K_1 <= K_2 ... , K_{q-1}, q <= p$.
- Each Pr_i is a **data pointer** that points to the record whose search field value is K_i or to a file block containing the record (or to a block of record pointers that point to records whose search field value is K_i if the search field is not a key).
- Each leaf node has at least $\lceil (p/2) \rceil$ values.
- All leaf nodes are at the same level.



B^+ is the modified B-tree, where we perform two major modifications in order to support sequential search.

- Every non-leaf data will have a copy(left-right) in the leaf node.
- Every leaf node will have a pointer which will support to the node on its right.



The following key values are inserted into a B^+-tree in which order of the internal nodes is 3, and that of the leaf nodes is 2, in the sequence given below. The order of internal nodes is the maximum number of tree pointers in each node, and the order of leaf nodes is the maximum number of data items that can be stored in it. The B^+-tree is initially empty.

10, 3, 6, 8, 4, 2, 1

The maximum number of times leaf nodes would get split up as a result of these insertions is

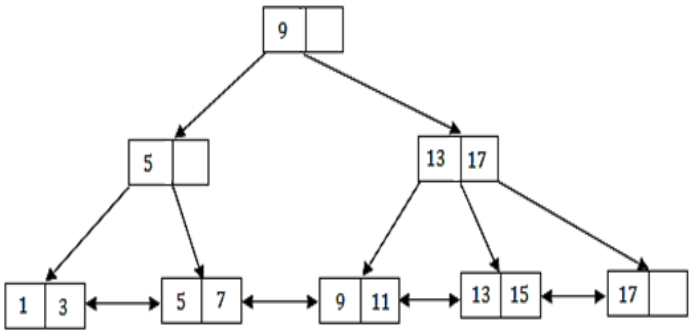
a) 2 **b)** 3 **c)** 4 **d)** 5

(GATE-2009) (2 Marks)

ANSWER d

Q (GATE-2015) (1 Marks)

With reference to the B^+ tree index of order 1 shown below, the minimum number of nodes (including the Root node) that must be fetched in order to satisfy the following query: "Get all records with a search key greater than or equal to 7 and less than 15" is _____.



Ans: 4

Q Consider a B^+-tree in which the maximum number of keys in a node is 5. What is the minimum number of keys in any non-root node? **(GATE-2010) (1 Marks)**

Answer 2

Q B^+ Trees are considered **BALANCED** because **(GATE-2016) (1 Marks)**

a) the lengths of the paths from the root to all leaf nodes are all equal

b) the lengths of the paths from the root to all leaf nodes differ from each other by at most 1

c) the number of children of any two non-leaf sibling nodes differ by at most 1

d) the number of records in any two leaf nodes differ by at most 1

Ans: a

Q Which of the following is a key factor for preferring B^+-trees to binary search trees for indexing database relations? **(Gate-2005) (1 Marks)**

a) Database relations have a large number of records

b) Database relations are sorted on the primary key

c) B^+-trees require less memory than binary search trees

d) Data transfer form disks is in blocks

Ans: d

Q B+ trees are preferred to binary trees in databases because **(Gate-2000) (1 Marks)**
(A) Disk capacities are greater than memory capacities
(B) Disk access is much slower than memory access
(C) Disk data transfer rates are much less than memory data transfer rates
(D) Disks are more reliable than memory
Answer: (B)

Q Which of the following is correct? **(Gate-1999) (1 Marks**)

a) B-trees are for storing data on disk and B+ trees are for main memory.

b) Range queries are faster on B+ trees.

c) B-trees are for primary indexes and B++ trees are for secondary indexes.

d) The height of a B+ tree is independent of the number of records.

Ans: b

Q Which one of the following statements is NOT correct about the B^+ tree data structure used for creating an index of a relational database table? **(GATE-2019) (1 Marks)**
(1) Each leaf node has a pointer to the next leaf node
(2) Non-leaf nodes have pointers to data records

(3) B+ Tree is a height-balanced tree

(4) Key values in each node are kept in sorted order

Ans: b

Q In a B+ tree, if the search-key value is 8 bytes long, the block size is 512 bytes and the block pointer is 2 bytes, then the maximum order of the B+ tree is. **(GATE-2017) (2 Marks)**
Ans: 52

Q Consider B+ tree in which the search key is 12 bytes long, block size is 1024 bytes, record pointer is 10 bytes long and block pointer is 8 bytes long. The maximum number of keys that can be accommodated in each non-leaf node of the tree is (Gate-2015) (2 Marks)
Answer: 50

Q The order of a leaf node in a B^+- tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1K bytes, data record pointer is 7 bytes

long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node? **(GATE-2007) (1 Marks)**

a) 63 **b)** 64 **c)** 67 **d)** 68

ANSWER a

Q In a database file structure, the search key field is 9 bytes long, the block size is 512 bytes, a record pointer is 7 bytes and a block pointer is 6 bytes. The largest possible order of a non-leaf node in a B+ tree implementing this file structure is **(Gate-2006) (2 Marks)**

(A) 23 **(B)** 24 **(C)** 34 **(D)** 44

Answer: (C)

Q The order of an internal node in a B+ tree index is the maximum number of children it can have. Suppose that a child pointer takes 6 bytes, the search field value takes 14 bytes, and the block size is 512 bytes. What is the order of the internal node? **(Gate-2004) (2 Marks)**

(A) 24 **(B)** 25 **(C)** 26 **(D)** 27

Answer: (C)

Q A B+ -tree index is to be built on the Name attribute of the relation STUDENT. Assume that all student names are of length 8 bytes, disk block are size 512 bytes, and index pointers are of size 4 bytes. Given this scenario, what would be the best choice of the degree (i.e. the number of pointers per node) of the B+ -tree? **(Gate-2002) (2 Marks)**

(A) 16 **(B)** 42 **(C)** 43 **(D)** 44

Answer: (C)

Q Consider a table T in a relational database with a key field K. A B-tree of order p is used as an access structure on K, where p denotes the maximum number of tree pointers in a B-tree index node. Assume that K is 10 bytes long; disk block size is 512 bytes; each data pointer P_D is 8 bytes long and each block pointer P_B is 5 bytes long. In order for each B-tree node to fit in a single disk block, the maximum value of p is **(Gate-2004) (2 Marks)**

(A) 20 **(B)** 22 **(C)** 23 **(D)** 32

Answer: (C)

It is **23.**

$$(p-1)(\text{key\_ptr\_size} + \text{record\_ptr\_size}) + p.(\text{block\_ptr\_size}) \leq 512$$
$$\implies (p-1)(10+8) + p \times 5 \leq 512$$
$$\implies 23p \leq 530$$
$$\implies p \leq 23.04$$

So, maximum value of p possible will be 23.

Q Consider a join (relation algebra) between relations r(R) and s(S) using the nested loop method. There are 3 buffers each of size equal to disk block size, out of which one buffer is reserved for intermediate results. Assuming size(r(R)) < size(s(S)), the join will have fewer number of disk block accesses if **(GATE-2014) (2 Marks)**

(1) relation r(R) is in the outer loop

(2) relation s(S) is in the outer loop

(3) join selection factor between r(R) and s(S) is more than 0.5

(4) join selection factor between r(R) and s(S) is less than 0.5

Ans: a

Q Match the following: **(NET-DEC-2013)**

List – I	List – II
a. Secondary Index	**i.** Functional Dependency
b. Non-procedural Query Language	**ii.** B-Tree
c. Closure of set of Attributes	**iii.** Relational Algebraic Operation
d. Natural JOIN	**iv.** Domain Calculus

Codes:

	a	b	c	d
a)	i	ii	iv	iii
b)	ii	i	iv	iii
c)	i	iii	iv	ii
d)	ii	iv	i	iii

Ans: D

Q B + tree are preferred to binary tree in database because **(NET-DEC-2011)**

(A) Disk capacities are greater than memory capacities

(B) Disk access much slower than memory access

(C) Disk data transfer rates are much less than memory data transfer rate

(D) Disk are more reliable than memory

Ans: b

Q A B-tree of order 4 is built from scratch by 10 successive insertions. What is the maximum number of node splitting operations that may take place? **(GATE-2008) (1 Marks)**

Answer 5

Q In the indexed scheme of blocks to a file, the maximum possible size of the file depends on: **(NET-JUNE-2015)**

(1) The number of blocks used for index and the size of index

(2) Size of Blocks and size of Address

(3) Size of index

(4) Size of block

Ans. 1

TRANSACTION

- Why we study transaction?
  - According to general computation principle (operating system) we may have partially executed program, as the level of atomicity is instruction i.e. either an instruction is executed completely or not
  - But in DBMS view, user perform a logical work(operation) which is always atomic in nature i.e. either operation is execute or not executed, there is no concept like partial execution. For example, Transaction T_1 which transfer 100 units from account A to B

T_1
Read(A)
A = A-100
Write(A)
Read(B)
B = B+100
Write(B)

  - In this transaction if a failure occurs after Read(B) then the final statue of the system will be inconsistent as 100 units are debited from account A but not credited in account B, this will generate inconsistency.
  - Here for 'consistency' before (A + B) == after (A + B)"

What is Transaction

- To remove this partial execution problem, we increase the level of atomicity and bundle all the instruction of a logical operation into a unit called transaction.
- So formally 'A transaction is a Set of logically related instructions to perform a logical unit of work'.
- As here we are only concerned with DBMS so we well only two basic operation on database
  - **READ (X)** - Accessing the database item x from disk (where database stored data) to memory variable also name as X.
  - **WRITE (X)** - Writing the data item from memory variable X to disk.

Q A transaction can include following basic database access operations: **(NET-JUNE-2011)**

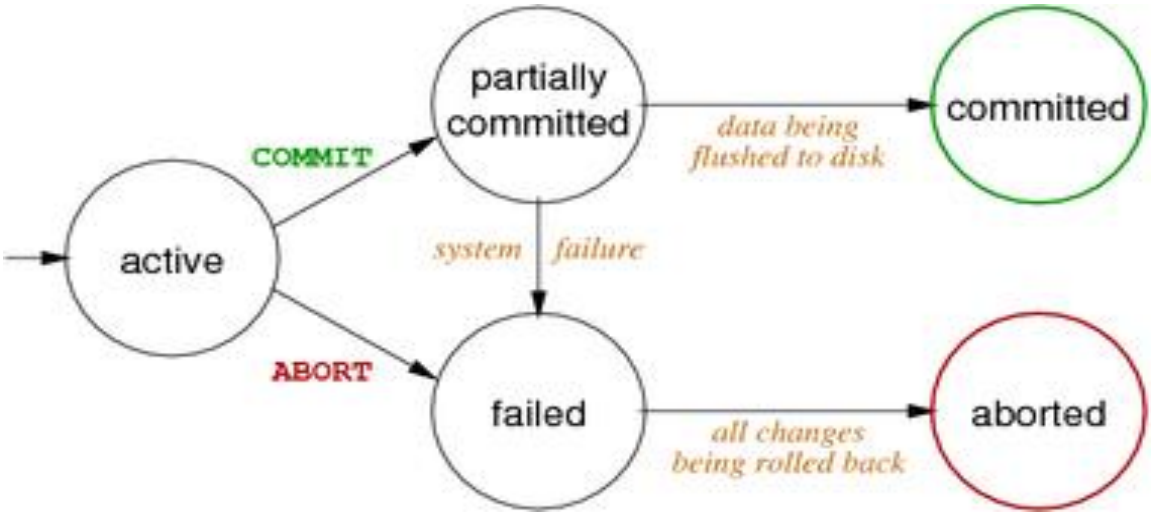
(A) Read_item(X)
(B) Write_item(X)
(C) Both (A) and (B)
(D) None of these

Ans: c

Desirable Properties of Transaction

- Now as the smallest unit which have atomicity in DBMS view is transaction, so if want that our data should be consistent then instead of concentrating on data base, we must concentrate on the transaction for our data to be consistent.
- Transactions should possess several properties, often called the **ACID** properties; to provide integrity and consistency of the data in the database. The following are the ACID properties:
    - **Atomicity -** A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all. It is the responsibility of ***recovery control manager / transaction control manager of DBMS*** to ensure atomicity
    - **Consistency** - A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another. The definition of consistency may change from one system to another. The preservation of consistency of database is the responsibility of programmers(users) or the DBMS modules that enforces integrity constraints.
    - **Isolation -** A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently. The isolation property of database is the responsibility of ***concurrency control manager of database***.
    - **Durability -** The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure. It is the responsibility of ***recovery control manager of DBMS.***

Transaction states



- **ACTIVE** - It is the initial state. Transaction remains in this state while it is executing operations.
- **PARTIALLY COMMITTED** - After the final statement of a transaction has been executed, the state of transaction is partially committed as it is still possible that it may have to be aborted (due to any failure) since the actual output may still be temporarily residing in main memory and not to disk.
- **FAILED** - After the discovery that the transaction can no longer proceed (because of hardware /logical errors). Such a transaction must be rolled back
- **ABORTED** - A transaction is said to be in aborted state when the when the transaction has been rolled back and the database has been restored to its state prior to the start of execution.
- **COMMITTED** - A transaction enters committed state after successful completion of a transaction and final updation in the database

Q if the transaction is in which of the state that we can guarantee that data base is in consistent state

a) aborted **b)** committed **c)** both aborted & committed **d)** none

Q Match:

Column I	Column II
1. Atomicity	A. Recovery Manager
2. Durability	B. Concurrency control manager
3. Isolation	C. Programmer
4. Consistency	

a) 1-a, 2–a, 3-b, 4–c **b)** 1-a, 2-b,3-b,4-c **c)** 1-a,2-a,3-b,4-b **d)** none of these

PROBLEMS DUE TO CONCURRENT EXECUTION OF TRANSACTION

Concurrent execution is necessary because-

- It leads to good database performance.
- Disk accesses are frequent and relatively slow.
- Overlapping I/O activity with CPU increases throughput and response time.

But interleaving of instructions between transactions may also lead to many problems that can lead to inconsistent database. Sometimes it is possible that even though individual transaction are satisfying the acid properties even though the final statues of the system will be inconsistent.

Lost update problem / Write - Write problem

- If there is any two write operation of different transaction on same data value, and between them there is no read operations, then the second write over writes the first write.

T_1	T_2
Read(A)	
Write(A)	
	Write(A)
	Commit
Commit	

Dirty read problem/ Read -Write problem

- In this problem, the transaction reads a data item updated by another uncommitted transaction, this transaction may in future be aborted or failed.
- The reading transactions end with incorrect results.

T_1	T_2
Read(A)	
Write(A)	
	Read(A)
	Commit
Abort	

Q The problem that occurs when one transaction updates a database item and then the transaction fails for some reason is _____. **(NET-JUNE-2012)**

(A) Temporary Select Problem
(B) Temporary Modify Problem
(C) Dirty Read Problem
(D) None

Ans: d

Unrepeatable read problem/phantom read problem

- When a transaction tries to read a value of a data item twice, and another transaction updates the data item in between, then the result of the two read operation of the first transaction will differ, this problem is called, Non-repeatable read problem

T_1	T_2
Read(A)	
	Read(A)
	Write(A)
Read(A)	

- Phantom read problem

T_1	T_2
Read(A)	
	Delete(A)
Read(A)	

Q The following schedule is suffering from

T_1	T_2
R(y)	
	R(x) R(y) y = x + y w(y)
R(y)	

a) Lost update problem
b) Unpredictable read problem
c) Both A and B
d) Neither A and B

Q Which of the following scenarios may lead to an irrecoverable error in a database system? **(GATE – 2008) (1 Marks)**
(A) A transaction writes a data item after it is read by an uncommitted transaction
(B) A transaction reads a data item after it is read by an uncommitted transaction
(C) A transaction reads a data item after it is written by a committed transaction
(D) A transaction reads a data item after it is written by an uncommitted transaction
Answer: (D)

Solution is Schedule

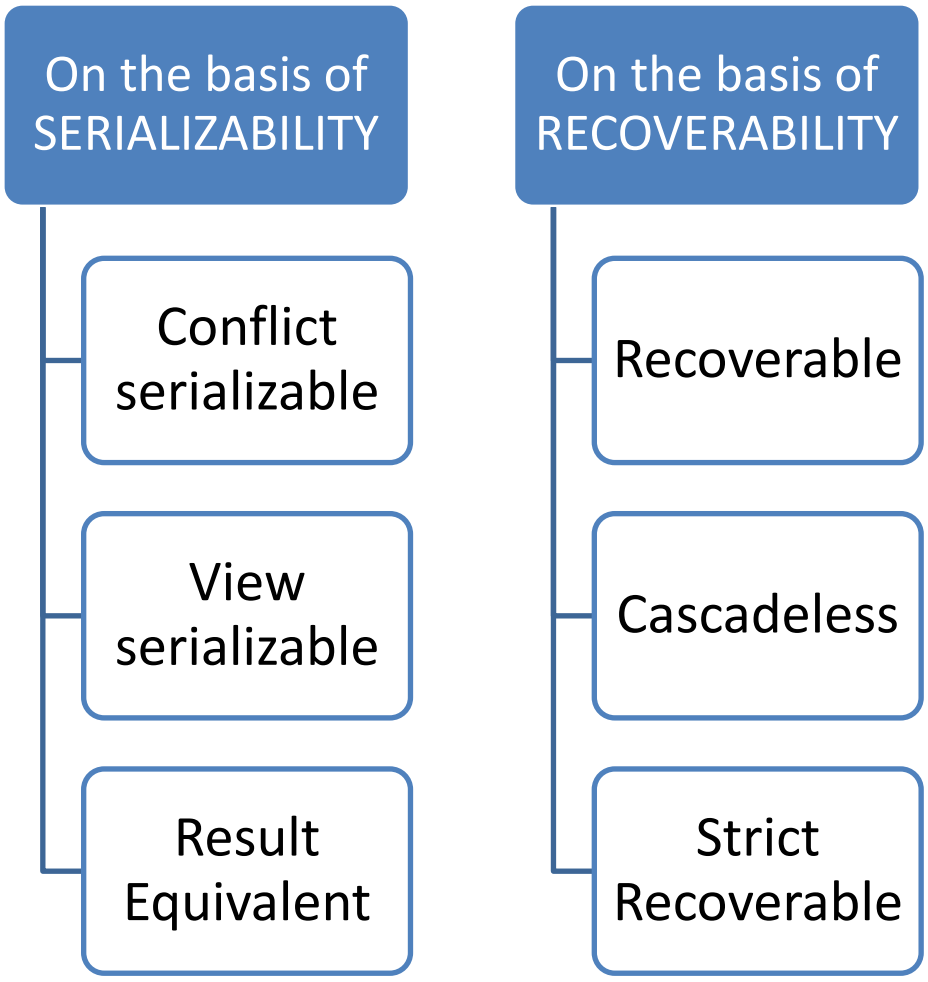
- When two or more transaction executed together or one after another then they can be bundled up into a higher unit of execution called schedule, A **schedule** of *n* transactions *T*1, *T*2, ..., *Tn* is an ordering of the operations of the transactions. Operations from different transactions can be interleaved in the schedule *S*.
- However, schedule for a set of transaction must contain all the instruction of those transaction, and for each transaction *Ti* that participates in the schedule *S*, the operations of *Ti* in *S* must appear in the same order in which they occur in *Ti*.
- Schedule can be of two types-
  - **Serial schedule** - A serial schedule consists of sequence of instruction belonging to different transactions, where instructions belonging to one single transaction appear together. Before complete execution of one transaction another transaction cannot be started.
  - For a set of n transactions, there exist **n!** different valid serial schedules. Every serial schedule lead database into consistent state. Throughput of system is less.

T_0	T_1
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

  - **Non-serial schedule** - A schedule in which sequence of instructions of a transaction appear in the same order as they appear in individual transaction but the instructions may be interleaved with the instructions of different transactions i.e. concurrent execution of transactions takes place.

T_2	T_3
read(B)	
	read(B)
	write(B)
read(A)	
	read(A)
	write(A)

- So the number of schedules for n different transaction $T_1, T_2, T_3, \text{---------} T_N$ where each transaction conations $n_1, n_2, n_3, \text{----------}, n_4$ respectively will be
- $\{(n_1, n_2, n_3, \text{----------}, n_n)!\} / (n_1!\ n_2!\ n_3!\ \text{----------} n_n!)$
- **Conclusion of schedules**
  - We do not have any method to proof that a schedule is consistent , but from the above discussion we understand that a serial schedule will always be consistent , so if somehow we proof that a non-serial schedule will also have same effects as of a serial schedule that we get a proof that , this particular non-serial schedule will also be consistent “find those schedules that are logically equal to serial schedules”.
  - For a concurrent schedule to result in consistent state, it should be equivalent to a serial schedule. i.e. it must be serializable.



SERIALIZABILITY

Conflicting instructions **-** Let I and J be two consecutive instructions belonging to two different transactions T_i and T_j in a schedule S, the possible I and J instruction can be as-

I= READ(Q), J=READ(Q) ->*Non-conflicting*
I= READ(Q), J=WRITE(Q) ->*Conflicting*
I= WRITE(Q), J=READ(Q) ->*Conflicting*
I= WRITE(Q), J=WRITE(Q) ->*Conflicting*

So, the instructions I and J are said to be conflicting, if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation.

- **Conflict equivalent** – if one schedule can be converted to another schedule by swapping of non- conflicting instruction then they are called conflict equivalent schedule.

T_1	T_2
R(A)	
A=A-50	
	R(B)
	B=B+50
R(B)	
B=B+50	
	R(A)
	A=A+10

T_1	T_2
	R(B)
	B=B+50
R(A)	
A=A-50	
R(B)	
B=B+50	
	R(A)
	A=A+10

CONFLICT SERIALIZABLE

- The schedules which are conflict equivalent to a serial schedule are called conflict serializable schedule. If a schedule S can be transformed into a schedule S' by a series of swaps of non- conflicting instructions, we say that S and S' are ***conflict equivalent.***
- A schedule S is ***conflict serializable***, if it is conflict equivalent to a serial schedule.

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Procedure for determining conflict serializability of a schedule

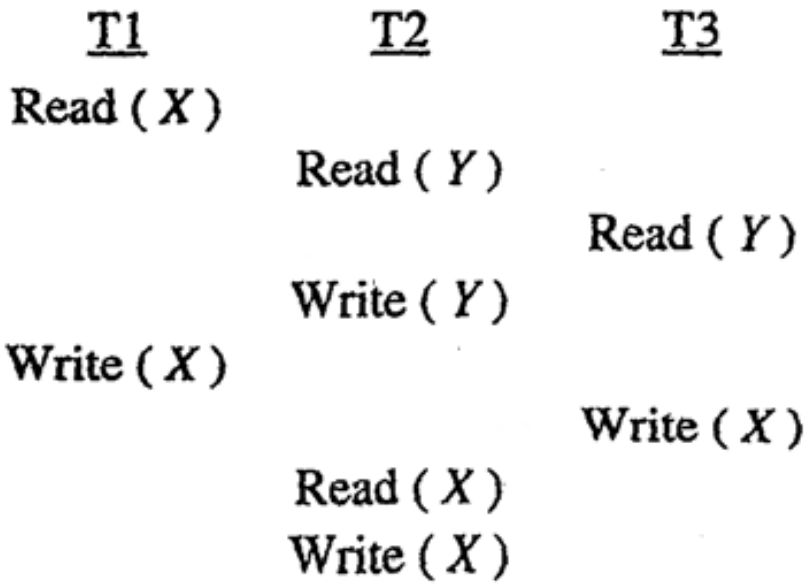
- It can be determined using PRECEDENCE GRAPH method:
- A precedence graph consists of a pair G (V, E)
  - V= set of vertices consisting of all the transactions participating in the schedule.
  - E= set of edges consists of all edges T_i -> T_j, for which one of the following conditions holds:
    - T_i executes write(Q) before T_j executes read(Q)
    - T_i executes read(Q) before T_j executes write(Q)
    - T_i executes write(Q) before T_j executes write(Q)

- If an edge T_i -> T_j exists in the precedence graph, then in any serial schedule S' equivalent to S, Ti must appear before T_j.

- **If the precedence graph for S has no cycle, then schedule S is conflict serializable, else it is not.** This cycle detection can be done by cycle detection algorithms, one of them based on depth first search takes **$O(n^2)$** time.

- The serialializibility order of transactions of equivalent serial schedule can be determined using **topological order** in a precedence graph.

Q Consider the following schedule for transactions T1, T2 and T3: **(GATE – 2010) (2 Marks)**



Which one of the schedules below is the correct serialization of the above?

(A) T1->>T3->>T2 **(B)** T2->>T1->>T3 **(C)** T2->>T3->>T1 **(D)** T3->>T1->>T2

Answer: (A)

Q Consider two transactions T_1 and T_2 which form schedules S_1, S_2, S_3 and S_4 as follows: -

T_1: R_1 [A], W_1 [A], W_1 [B] **T_2**: R_2 [A], R_2 [B], W_2 [B]

S_1: R_1[A], R_2 [A], R_2[B], W_1[A], W_2[B], W_1[B]

S_2: R_1[A], R_2 [A], R_2[B], W_1[A], W_1[B], W_2 [B]

S_3: R_2[A], R_1 [A], R_2[B], W_1[A], W_1[B], W_2[B]

S_4: R_1[A], W_2 [A], R_2[A], W_1[B], R_2[B], W_2[B]

Which of the above schedules is conflicts serializable? **(GATE - 2009) (2 Marks)**

a) Only S_1 **b)** Both S_1 and S_2 **c)** Both S_1 and S_4 **d)** Both S_3 and S_4

Q (GATE - 2009) (2 Marks)

Consider two transactions T_1 and T_2, and four schedules S_1, S_2, S_3, S_4 of T_1 and T_2 as given below:

T_1: R_1 [x]W_1 [x]W_1 [y]

T_2: R_2 [x]R_2 [y]W_2 [y]

S_1: R_1 [x]R_2 [x]R_2 [y]W_1 [x]W_1 [y]W_2 [y]

S_2: R_1 [x]R_2 [x]R_2 [y]W_1 [x]W_2 [y]W_1 [y]

S_3: R_1 [x]W_1 [x]R_2 [x]W_1 [y]R_2 [y]W_2 [y]

S_4: R_2 [x]R_2 [y]R_1 [x]W_1 [x]W_1 [y]W_2 [y]

Which of the above schedules are conflict-serializable?

a) S_1 and S_2 **b)** S_2 and S_3 **c)** S_3 only **d)** S_4 only

Ans: b

Q Consider the following four schedules due to three transactions (indicated by the subscript) using read and write on a data item x, denoted by r(x) and w(x) respectively. Which one of them is conflict serializable. **(GATE - 2014) (2 Marks)**

(a) $r_1(x); r_2(x); w_1(x); r_3(x); w_2(x)$

(b) $r_2(x); r_1(x); w_2(x); r_3(x); w_1(x)$

(c) $r_3(x); r_2(x); r_1(x); w_2(x); w_1(x)$

(d) $r_2(x); w_2(x); r_3(x); r_1(x); w_1(x)$

Ans: d

Q Consider the transactions T1, T2, and T3 and the schedules S1 and S2 given below. **(GATE - 2006) (2 Marks)**

T1: r1(X); r1(Z); w1(X); w1(Z)

T2: r2(Y); r2(Z); w2(Z)

T3: r3(Y); r3(X); w3(Y)

S1: r1(X); r3(Y); r3(X); r2(Y); r2(Z);w3(Y); w2(Z); r1(Z); w1(X); w1(Z)

S2: r1(X); r3(Y); r2(Y); r3(X); r1(Z);r2(Z); w3(Y); w1(X); w2(Z); w1(Z)

Which one of the following statements about the schedules is TRUE?

(A) Only S1 is conflict-serializable.

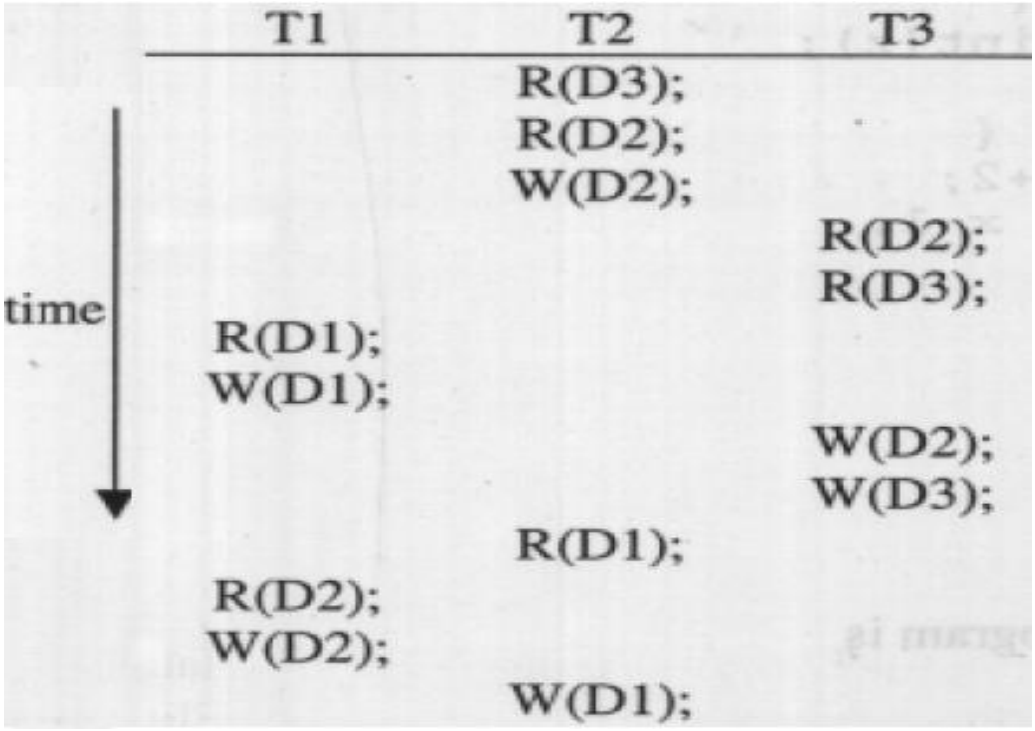
(B) Only S2 is conflict-serializable.

(C) Both S1 and S2 are conflict-serializable.

(D) Neither S1 nor S2 is conflict-serializable.

Answer: (A)

Q Consider three data items D1, D2 and D3 and the following execution schedule of transactions T1, T2 and T3. In the diagram, R(D) and W(D) denote the actions reading and writing the data item D respectively. **(GATE – 2003) (2 Marks)**



Which of the following statements is correct?
(A) The schedule is serializable as T2; T3; T1 **(B)** The schedule is serializable as T2; T1; T3
(C) The schedule is serializable as T3; T2; T1 **(D)** The schedule is not serializable
Answer: (D)

Q Consider following schedules involving two transactions : S 1 : r1(X); r1(Y); r2(X); r2(Y); w2(Y); w1(X) S 2 : r1(X); r2(X); r2(Y); w2(Y); r1(Y); w1(X) Which of the following statement is true ? **(NET-JAN-2017)**
(1) Both S1 and S2 are conflict serializable.
(2) S1 is conflict serializable and S2 is not conflict serializable.
(3) S1 is not conflict serializable and S2 is conflict serializable.
(4) Both S1 and S2 are not conflict serializable.
Ans: c

Q Consider the following four schedules due to three transactions (indicated by the subscript) using read and write on a data item X, denoted by r(X) and w(X) respectively. Which one of them is conflict serializable? **(NET-NOV-2017)**
S1: r1(X); r2(X); w1(X); r3(X); w2(X) **S2**: r2(X); r1(X); w2(X); r3(X); w1(X)
S3: r3(X); r2(X); r1(X); w2(X); w1(X) **S4**: r2(X); w2(X); r3(X); r1(X); w1(X)
(1) S1 **(2)** S2 **(3)** S3 **(4)** S4
Ans: d

Q Consider the following transactions with data items P and Q initialized to zero: **(GATE-2012) (2 Marks)**

```
T1: read (P);
    read (Q);
    if P = 0 then Q: = Q + 1;
    write (Q);

T2: read (Q);
    read (P);
    if Q = 0 then P: = P + 1;
    write (P);
```

Any non-serial interleaving of T1 and T2 for concurrent execution leads to
(A) A serializable schedule
(B) A schedule that is not conflict serializable

(C) A conflict serializable schedule
(D) A schedule for which a precedence graph cannot be drawn
Answer: (B)

Q Consider the following transaction involving two bank accounts x and y. **(GATE - 2015) (1 Marks)**

```
read(x);
x: = x – 50;
write(x);
read(y);
y: = y + 50;
write(y)
```

The constraint that the sum of the accounts x and y should remain constant is that of
(A) Atomicity **(B)** Consistency **(C)** Isolation **(D)** Durability
Answer: (B)

Q Consider the following schedule S of transactions T1, T2, T3, T4 **(GATE - 2014) (2 Marks)**

T1	T2	T3	T4
	Reads(X)		
		Writes(X) Commit	
Writes(X) Commit			
	Writes(Y) Reads(Z) Commit		
			Reads(X) Reads(Y) Commit

Which one of the following statements is CORRECT?
(A) S is conflict-serializable but not recoverable
(B) S is not conflict-serializable but is recoverable
(C) S is both conflict-serializable and recoverable

(D) S is neither conflict-serializable nor is it recoverable
Answer: (C)

Q Consider the following partial Schedule S involving two transactions T1 and T2. Only the read and the write operations have been shown. The read operation on data item P is denoted by read(P) and the write operation on data item P is denoted by write(P).

Time	Transaction-id	
	$T1$	$T2$
1	$read(A)$	
2	$write(A)$	
3		$read(C)$
4		$write(C)$
5		$read(B)$
6		$write(B)$
7		$read(A)$
8		$commit$
9	$read(B)$	

Suppose that the transaction T1 fails immediately after time instance 9. Which one of the following statements is correct? **(GATE-2015) (2 Marks)**
(A) T2 must be aborted and then both T1 and T2 must be re-started to ensure transaction atomicity
(B) Schedule S is non-recoverable and cannot ensure transaction atomicity
(C) Only T2 must be aborted and then re-started to ensure transaction atomicity
(D) Schedule S is recoverable and can ensure atomicity and nothing else needs to be done
Ans: b

Q Consider a simple checkpointing protocol and the following set of operations in the log.

(start, T4);
(write, T4, y, 2, 3);
(start, T1);

(commit, T4);
(write, T1, z, 5, 7);
(checkpoint);
(start, T2);
(write, T2, x, 1, 9);
(commit, T2);
(start, T3);
(write, T3, z, 7, 2);

If a crash happens now and the system tries to recover using both undo and redo operations, what are the contents of the undo list and the redo list **(GATE - 2015) (2 Marks)**

(A) Undo: T3, T1; Redo: T2
(B) Undo: T3, T1; Redo: T2, T4
(C) Undo: none; Redo: T2, T4, T3; T1
(D) Undo: T3, T1, T4; Redo: T2
Answer: (A)

VIEW SERIALIZABLE

- If a schedule is not conflict serializable, still it can be consistent, so let us study a weaker form of serializability called View serializability, and even if a schedule is view serializable still it can be consistent.
- If a schedule is conflict serializable then it will also be view serializable, so we must check view serializability only if a schedule is not conflict serializable.
- If a schedule is not conflict serializable then it must have at least one blind write to be eligible for view serializable. i.e. if a schedule is not conflict serializable and it does not contain any blind write then it can never be view serializable, but if not conflict serializable and have blind write then may or may not be view serializable.
- If a schedule is not conflict serializable and if there exist a blind write. First tabulate all serial schedules possible. Then check one by one whether given schedule is view equivalent to any of the serial schedule. If yes then schedule is view serializable otherwise not.
- Two schedules S and S' are view equivalent, if they satisfy following conditions –
    - For each data item Q, if the transaction T_i reads the initial value of Q in schedule S , then then the transaction T_i must, in schedule S' ,also read the initial value of Q.
    - If a transaction T_i in schedule S reads any data item Q, which is updated by transaction T_j, then a transaction T_i must in schedule S' also read data item Q updated by transaction T_j in schedule S'.
    - For each data item Q, the transaction (if any) that performs the final write(Q) operation in schedule S, then the same transaction must also perform final write(Q) in schedule S'.
- Complexity wise finding the schedule is view serializable or not is a **NP- complete** problem.

View Serializable

A schedule S is view serializable, if it is view equivalent to a serial schedule.

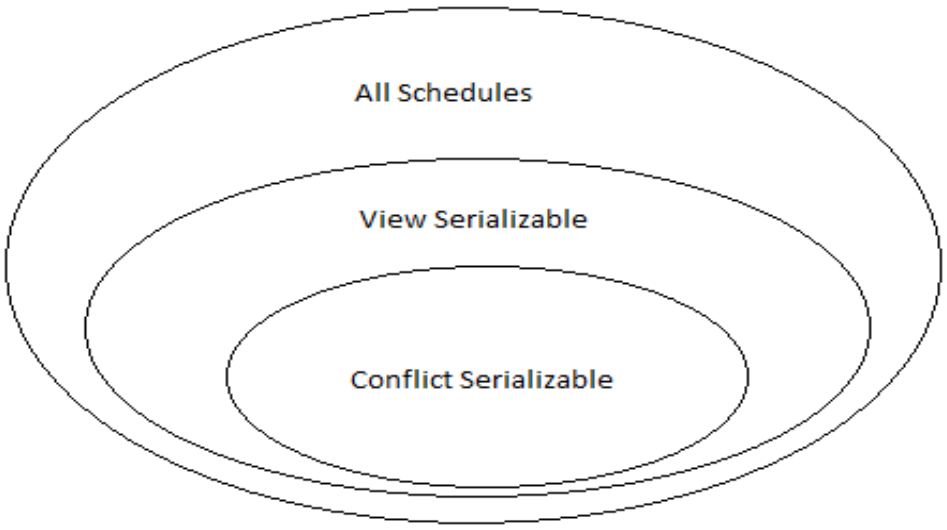
E.g.

Schedule A		
T3	T4	T6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Serial schedule <T3,T4,T6>		
T3	T4	T6
read(Q)		
write(Q)		
	write(Q)	
		write(Q)

In above example schedule A is view equivalent to serial schedule <T3 T4 T6>,
But the schedule A is not conflict equivalent to any serial schedule.
Hence the schedule A is VIEW SERIALIZABLE, but not CONFLICT SERIALIZABLE.

- ***BLIND WRITES***- In the above example, transaction T4 and T6 perform write operation on data item Q without accessing (reading the data item), such updation without knowing/accessing previous value of data item, are called Blind updation or **BLIND WRITE.**
- Every view serializable that is not conflict serializable has a **BLIND WRITE**



Q Which of the following statements (s) is/are TRUE?
S_1: All view serializable schedules are also conflict serializable.
S_2: All conflict serializable schedules are also view serializable.
S_3: If a schedule is not conflict serializable then it is not view serializable
S_4: If a schedule is not view serializable then it is not conflict serializable.
a) S_1 and S_2 only **b)** S_2 and S_3 only **c)** S_2 and S_4 only **d)** S_1 and S_3 only

Q Consider the following schedule 'S' with three transactions.
S: $R_1(B)$; $R_3(C)$; $R_1(A)$; $W_2(A)$; $W_1(A)$, $W_2(B)$; $W_3(A)$; $W_1(B)$; $W_3(B)$, $W_3(C)$
Which of the following is TRUE with respect to the above schedule?
a) It is conflict serializable with sequence [T_1, T_2 T_3]
b) It is conflict serializable with sequence [T_2, T_1 T_3]
c) It is view serializable but not conflict serializable
d) It is neither conflict serializable nor view serializable

Q The following schedule S is having 4 transactions and is executed concurrently. The order of their operations is given below.
S: $R_1(x)$; $R_2(y)$; $W_2(x)$; $W_3(z)$; $R_4(z)$; $R_3(x)$; $W_3(y)$; $W_1(x)$; $W_2(y)$ $W_3(x)$; $R_4(x)$; $W_4(y)$; commit 1; commit 2; commit 3; commit 4;
The schedule is
a) Conflict serializable with sequence [T_1 T_2 T_3 T_4]
b) Conflict serializable with sequence [T_2 T_1 T_3 T_4]
c) View serializable but not conflict serializable
d) Neither conflict serializable nor view serializable

RECOVERABILITY

RECOVERABLE SCHEDULE-

A schedule in which for each pair of transaction T_i and T_j, such that if T_j reads a data item previously written by T_i, then the commit or abort of T_i must appear before T_j. Such a schedule is called Recoverable schedule.

E.g. –

Sa: r1(X); r2(X); w1(X); r1(Y); w2(X); c2; w1(Y); c1; is ***recoverable***

Sc: r1(X); w1(X); r2(X); r1(Y); w2(X); c2; a1; ***is not recoverable***

NON- RECOVERABLE SCHEDULE-

A schedule in which for each pair of transaction T_i and T_j, such that if T_j reads a data item previously written by T_i, then the commit or abort operation of T_i appears before T_j. Such a schedule is called Non- Recoverable schedule.

CASCADING ROLLBACK

- It is a phenomenon, in which a single transaction failure leads to a series of transaction rollbacks, is called cascading rollback.
- Even if the schedule is recoverable the, the commit of transaction may lead lot of transaction to rollback.
- Cascading rollback is undesirable, since it leads to undoing of a significant amount of work.
- Uncommitted reads are not allowed in cascade less schedule.
- ***E.g.***

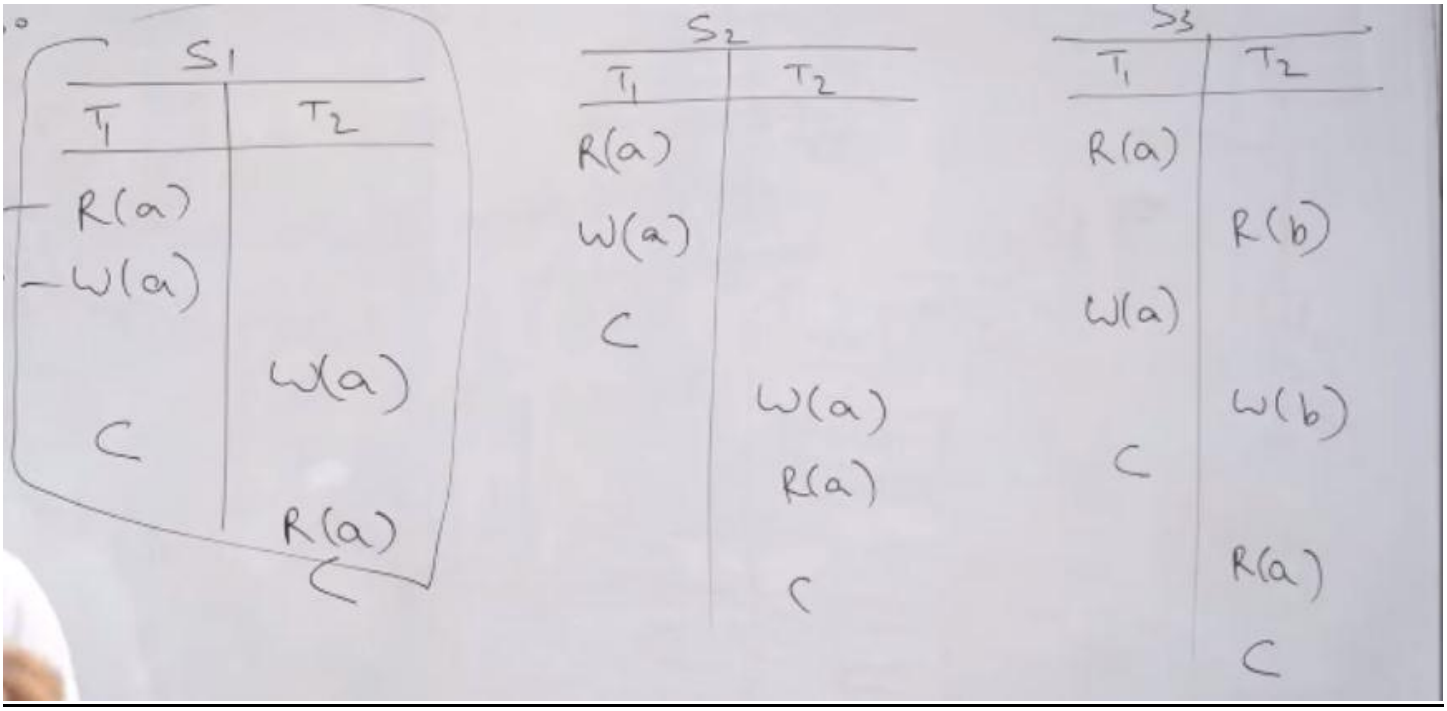
T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

In above schedule T11 reads a data item written by T10 and T12 reads a data item written by T11. So aborting of T10 may lead T11 and so T12 to roll back.

CASCADELESS SCHEDULE-

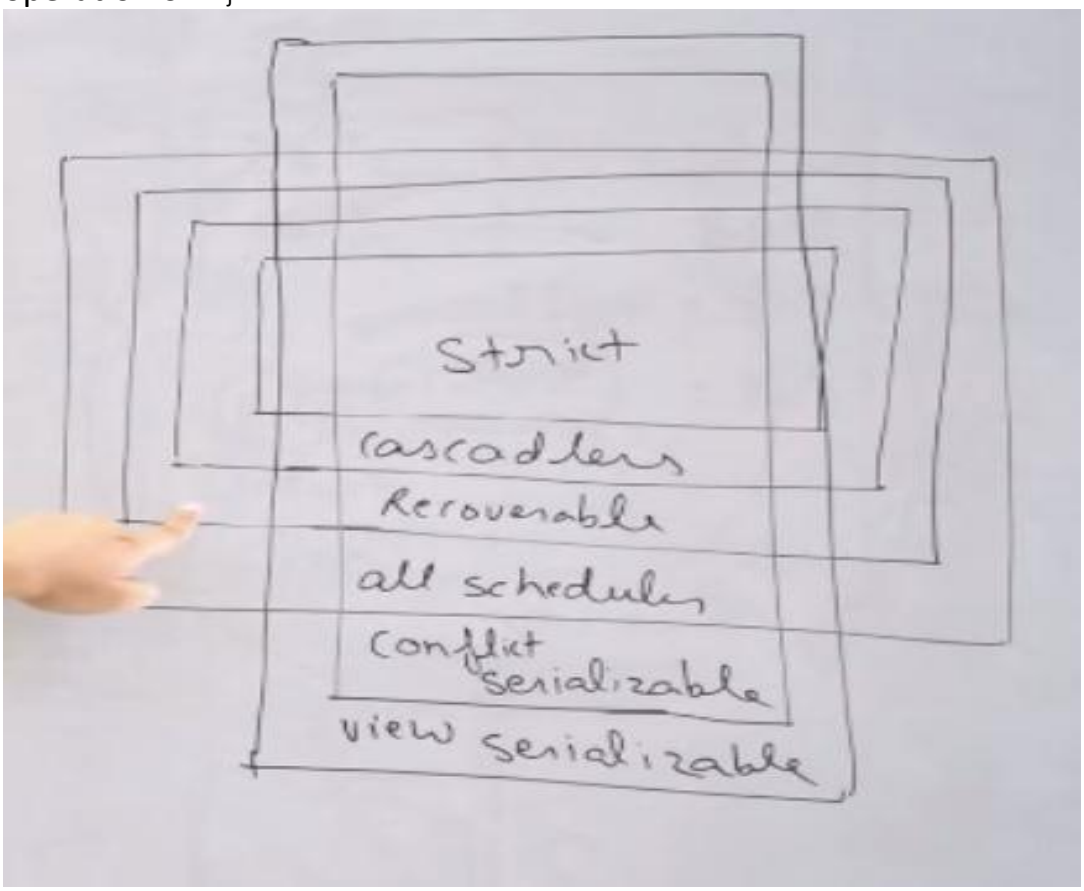
- To avoid cascading rollback, cascade less schedule are used.
- A schedule in which for each pair of transactions T_i and T_j, such that if T_j reads a data item previously written by T_i then the commit or abort of T_i must appear before read operation of T_j. Such a schedule is called cascade less schedule.

STRICT



SCHEDULE

A schedule in which for each pair of transactions T_i and T_j, such that if T_j reads a data item previously written by T_i then the commit or abort of T_i must appear before read and write operation of T_j.



Q Consider the following schedules:

S1: $R_1(x)\ W_1(x)R_1(y)R_2(x)W_2(x)C_2,\ C_1;$

S2: $R_2(x)\ W_2(x)R_1(y)R_1(x)W_2(x)C_2,\ C_1;$

Which of the following is true?

a) Both S1 and S2 are recoverable **b)** S1 is recoverable but S2 is not

c) S2 is recoverable but S1 is not **d)** Both schedule are not Recoverable

Q Consider the following schedule 'S'.

S: r1 (X); r2 (Z); r3 (X); r1 (Z); r2 (Y); r3 (Y); w1 (X); c1; w2 (Z); w3 (Y); w2 (Y); c3; c2; The schedule 'S' is

a) Recoverable **b)** Cascade less **c)** recoverable and cascade-less **d)** None

Q Consider the given schedule

R1 (X), R2 (Z), R1 (Z), R3 (X), R3 (Y), W1 (X), Commit1, W3 (Y), Commit3, R2 (Y), W2 (Z), W2 (Y), Commit2. The given schedule is

a) Recoverable only **b)** Cascade less only **c)** recoverable and cascade-less **d)** None

Q Which of the following statement is/are correct

a) Every view serializable schedule is conflict serializable

b) Every strict schedule is conflict serializable

c) Every conflict serializable schedule is cascade less

d) None of the above

Q A Schedule that is not conflict serializable and contains at least one blind write then the schedule is

a) Always view serializable **b)** Always non-serializable

c) May be serializable **d)** None of the above

Q Consider the following schedule

S; $R_2(x),\ w_2(x),\ R_3(y),\ R_1(x),\ R_1(y),\ w_1(x),\ w_3(y),\ R_3(x), R_1(y),\ C_3,\ C_2,\ C_1;$

The above schedule is

a) Recoverable but not cascade less **b)** Recoverable and cascade less but not strict

c) Recoverable, cascade less and also strict

d) Not recoverable

Q Two transactions T1 and T2 are given as:

T1: r1(X)w1(X)r1(Y)w1(Y)
T2: r2(Y)w2(Y)r2(Z)w2(Z)

where $r_i(V)$ denotes a read operation by transaction Ti on a variable V and $w_i(V)$ denotes a write operation by transaction Ti on a variable V. The total number of conflict serializable schedules that can be formed by T1 and T2 is _____**(GATE-2017) (2 Marks)**

Answer: 54

Q Suppose a database schedule S involves transactions T1, T2, ...,Tn. Consider the precedence graph of S with vertices representing the transactions and edges representing the conflicts. If S is serializable, which one of the following orderings of the vertices of the precedence graph is guaranteed to yield a serial schedule? **(NET-NOV-2017)**

(1) Topological order

(2) Depth - first order

(3) Breadth - first order

(4) Ascending order of transaction indices

Ans: a

System log /transaction log

- To be able to Recover from failures, recovery subsystem of database maintains transaction log to keep track of all operations of a transaction that effects database items, as well as other information that may be needed in recovery operation until the commit/abort point of a transaction.
- Log records can be used to trace out the transaction steps in the event of failure where the transaction need to be rolled back and all the operations of a transaction either need to be redone/ undone.
- A log file is written on the disk to avoid hardware/logical failures.
- A log record for a transaction has following syntax-

  - **[start_transaction, *T*] -** Indicates that transaction *T* has started execution.
  - **[write_item, *T*, *X*, *old_value*, *new_value*]** - Indicates that transaction *T* has changed the value of database item *X* from *old_value* to *new_value.*
  - **[read_item, *T*, *X*]**- Indicates that transaction *T* has read the value of database item *X*.
  - **[commit, *T*]**- Indicates that transaction *T* has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
  - **[Abort, *T*]**. - Indicates that transaction *T* has been aborted.
  - Following operation may be taken at time of failure-
  - **a.)Undo operation-**We can undo the effect of each WRITE operation of a transaction T by tracing backward through the log and resetting all items changed by the WRITE operation of T to their old values.
  - **b.)Redo operation-**We can also redo the effect of the WRITE operations of a transaction T by tracing forward through the log and setting all items changed by a WRITE operation of T to their new values.
  - **Commit Point of a Transaction-** A transaction T reaches its commit point when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database has been recorded in the log.
    Beyond the commit point, the transaction is said to be committed, and its effect is assumed to be permanently recorded in the database.
    The transaction then writes an entry [commit,T] into the log. No entry is made in the log after the commit entry, so a transaction cannot be aborted after it is committed.

Q Usage of Pre-emption and Transaction Rollback prevents _____. **(NET-SEP-2013)**

(A) Unauthorized usage of data file
(B) Deadlock situation
(C) Data manipulation
(D) File pre-emption

Ans: b

Q **(NET-DEC-2018)**

Consider the following sequence of two transactions on a bank account (A) with initial balance 20,000 that transfers 5,000 to another account (B) and then apply 10% interest.

(i) T1 start

(ii) T1 A old = 20,000 new 15,000

(iii) T1 B old = 12,000 new = 17,000

(iv) T1 commit

(v) T2 start

(vi) T2 A old = 15,000 new = 16,500

(vii) T2 commit

Suppose the database system crashes just before log record (vii) is written. When the system is restarted, which one statement is true of the recovery process ?

Options :

91394342529. We must redo log record (vi) to set A to 16,500.

91394342530. We must redo log record (vi) to set A to 16,500 and then redo log records (ii) and (iii).

91394342531. We need not redo log records (ii) and (iii) because transaction T1 has committed.

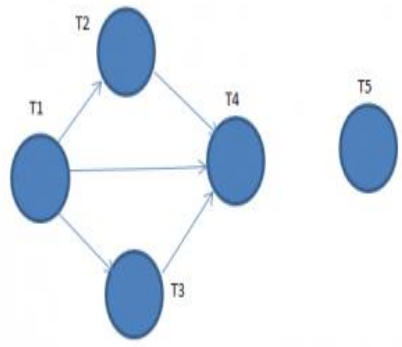
91394342532. We can apply redo and undo operations in arbitrary order because they are idempotent.

Topological order in a precedence graph

- Visit the vertex V in a graph G with in degree zero and delete it from the graph.
- Repeat the step 1 till the graph is empty.
- The order in which the vertex is deleted is the serializability order of the equivalent serial schedule. **The number of conflict equal schedules is equal to no. of topological orders possible in given acyclic precedence graph**

T1	T2	T3	T4	T5
	read(X)			
read(Y) read(Z)				
				read(V) read(W) write(W)
	read(Y) write(Y)			
		write(Z)		
read(U)				
			read(Y) write(X) read(Z) write(Z)	
read(V) write(U)				

Precedence Graph-



`

Equivalent serial schedules

T1->T2->T3->T4->T5
T1->T3->T2->T4->T5
T5->T1->T2->T3->T4
T5->T1->T3->T2->T4
T1->T5->T2->T3->T4
T1->T2->T5->T3->T4
T1->T2->T3->T5->T4
T1->T5->T3->T3->T4
T1->T3->T2->T5->T4

Q (GATE - 2007) (2 Marks)

Consider the following schedules involving two transactions. Which one of the following statements is **TRUE**?

S_1: $r_1(X); r_1(Y); r_2(X); r_2(Y); w_2(Y); w_1(X)$

S_2: $r_1(X); r_2(X); r_2(Y); w_2(Y); r_1(Y); w_1(X)$

a) Both S_1 and S_2 are conflict serializable
b) S_1 is conflict serializable and S_2 is not conflict serializable
c) S_1 is not conflict serializable and S_2 is conflict serializable
d) Both S_1 and S_2 are not conflict serializable.
Ans: C

Q Two transactions T_1 and T_2 are given as
T_1: $r_1(X)w_1(X)r_1(Y)w_1(Y)$
T_2: $r_2(Y)w_2(Y)r_2(Z)w_2(Z)$ where $r_i(V)$ denotes a read operation by transaction T_i on a variable V and $w_i(V)$ denotes a write operation by transaction T_i on a variable V. The total number of conflict serializable schedules that can be formed by T_1 and T_2 is _____ **(GATE-2017) (1 Marks)**
Ans: 54

Q NOT a part of the **ACID** properties of database transactions? **(GATE- 2016) (1 Marks)**
(a) Atomicity
(b) Consistency
(c) Isolation
(d) Deadlock-freedom
Ans: d

Q Consider the following database schedule with two transactions, T_1 and T_2.
$S = r_2(X); r_1(X); r_2(Y); w_1(X); r_1(Y); w_2(X); a_1; a_2$
where $r_i(Z)$ denotes a read operation by transaction T_i on a variable Z, $w_i(Z)$ denotes a write operation by T_i on a variable Z and a_i denotes an abort by transaction T_i.
Which one of the following statements about the above schedule is **TRUE**? **(GATE- 2016) (1 Marks)**
(a) S is non-recoverable
(b) S is recoverable, but has a cascading abort
(c) S does not have a cascading abort
(d) S is strict
Ans: c

Consider the following transaction involving two bank account x and y.

read (x) ; x : = x – 50; write (x) ; read (y); y : = y + 50 ; write (y)

The constraint that the sum of the accounts x and y should remain constant is that of **(GATE – 2015) (1 Marks)**

(a) Atomicity **(b)** Consistency **(c)** Isolation **(d)** Durability

Ans: b

Q Suppose a database schedule *S* involves transactions $T_1, ..., T_n$. Construct the precedence graph of *S* with vertices representing the transactions and edges representing the conflicts. If *S* is serializable, which one of the following orderings of the vertices of the precedence graph is guaranteed to yield a serial schedule? **(GATE- 2016) (1 Marks)**

(a) Topological order **(b)** Depth-first order

(c) Breadth-first order **(d)** Ascending order of transaction indices

Ans: a

Q Consider the following schedules involving two transactions.
S1: r1(X); r1(Y); r2(X); r2(Y); w2(Y); w1(X) S2: r1(X); r2(X); r2(Y); w2(Y); r1(Y); w1(X) Which one of the following statements is correct with respect to above? **(NET-JULY-2018)**

(1) Both S1 and S2 are conflict serializable.

(2) Both S1 and S2 are not conflict serializable.

(3) S1 is conflict serializable and S2 is not conflict serializable.

(4) S1 is not conflict serializable and S2 is conflict serializable

Ans: 3

Q In a certain database there are 2 transactions, one contains 5 instructions and another contains 3 instructions. Then number of concurrent schedules possible is _____.

CONCURRENCY CONTROL

- Now we understood that if there is a schedule how to check whether it will work correctly or not i.e. weather it will maintain the consistency of the data base or not. (conflict serializability, view serializability, recoverability and cascade less)

- Now we will understand those protocol which guarantee how to design those schedules which ensure conflict serializability or other properties.

- There are different approach or idea to ensure conflict serializability which is the most important property. So first we must understand what is the possibility of conflict between two instruction and if somehow, we manage than the generated schedule will always be conflict serializable

- If we remember, two instructions are conflicting if and only if three things happen simultaneously

    - Belong to different transactions
    - Must operate on same data value
    - At least one of them should be a write instruction \
- if we think sufficiently, there will be no way to change any of these three conditions. But actual problem is not that two instructions are trying to access same data base but, they are trying to do that at same time.
- So point if we somehow by any technique manage that two transaction do not access same data at same time then ensuring conflict serializability will be easy

- Now question is how to approach conflict serializability, there are two popular approaches to go forwards.

    - **Time stamping based method: -** where before entering the system, a specific order is decided among the transaction, so in case of a clash we can decide which one to allow and which to stop.
    - **Lock based method: -** where we ask a transaction to first lock a data item before using it. So that no different transaction can use a data at the same time, removing any possibility of conflict.
        - 2 phase locking
            - Basic 2pl
            - Conservative 2pl
            - Rigorous 2pl
            - Strict 2pl
        - Graph based protocol
    - **Validation based protocol –** Majority of transactions are read only transactions, the rate of conflicts among the transaction may be low, thus many of transaction, if executed without the supervision of a concurrency control scheme, would nerveless leave the system in a consistent state.

- **Goals of a Protocol**: - We desire the following properties from schedule generating protocols
    - Concurrency should be as high as possible, as this is our ultimate goal because of which we are making all the effort.
        - The time taken by a transaction should also be less.
    - Properties satisfied by the protocol
    - Easy to understand and implement
    - In the last after discussing all the protocols we will compare different protocol only on the bases of degree of concurrency they provide and along which how many prope
    - rties they ensure.

TIME STAMP ORDERING PROTOCOL

- Basic idea of time stamping is to decide the order between the transaction before they enter in the system using a stamp (time stamp), in case of any conflict during the execution order can be decided using the time stamp.
- Let's understand how this protocol works, here we have two idea of timestamping, one for the transaction, and other for the data item.
- Time stamp with transaction,
    - With each transaction t_i, in the system, we associate a unique fixed timestamp, denoted by $TS(t_i)$. this timestamp is assigned by database system to a transaction at time transaction enters into the system. If a transaction has been assigned a timestamp $TS(t_i)$ and a new transaction t_j , enters into the system with a timestamp $TS(t_j)$, then always $TS(t_i) < TS(t_j)$.
    - Two things are to be noted, first time stamp of a transaction remain fixed throughout the execution, second it is unique means no two transaction can have the same timestamp.
    - The reason why we called time stamp not stamp, because for stamping we use the value of the system clock as stamp, advantage is, it will always be unique as time never repeats and there is no requirement of refreshing and starting with fresh value.
    - The time stamp of the transaction also determines the serializability order. Thus if $TS(t_i) < TS(t_j)$, then the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction t_i appears before transaction t_j.

- Time stamp with data item
    - In order to assure such scheme, the protocol maintains for each data item Q two timestamp values:
    - **W-timestamp(Q)** is the largest time-stamp of any transaction that executed write(Q) successfully.
    - **R-timestamp(Q)** is the largest time-stamp of any transaction that executed read(Q) successfully.
    - These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.
- Suppose a transaction Ti request a ***read(Q)***
    - If **TS(Ti) < W-timestamp(Q),** then Ti needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and Ti is rolled back.
    - If **TS(Ti)≥ W-timestamp(Q),** then the read operation is executed, and R-timestamp(Q) is set to the maximum of R-timestamp(Q) and TS(T_i).
- Suppose that transaction Ti issues ***write(Q).***
    - If **TS(Ti) < R-timestamp(Q),** then the value of Q that Ti is producing was needed previously, and the system assumed that that value would never be produced. Hence, the write operation is rejected, and Ti is rolled back.
    - If **TS(Ti) < W-timestamp(Q),** then Ti is attempting to write an obsolete value of Q. Hence, this write operation is rejected, and Ti is rolled back.
    - If **TS(Ti) ≥ R-timestamp(Q),** then the write operation is executed, and W-timestamp(Q) is set to max(W-timestamp(Q), TS(Ti)).
    - If **TS(Ti) ≥ W-timestamp(Q),** then the write operation is executed, and W-timestamp(Q) is set to max(W-timestamp(Q), TS(Ti)).
- If a transaction T_i is rolled back by the concurrency control scheme as a result of either a read or write operation, the system assigns it's a new timestamp and restarts it.

Properties

Conflict serializability	View serializability	recoverability	Cascade less ness	Deadlock Independence
yes	yes	No	no	Yes

- Time stamp ordering protocol ensures conflict serializability. Because conflicting operations are processed in timestamp order, since all the arcs in the precedence graph are of the form thus, there will be no cycles in the precedence graph.
- As we know that view is liberal form conflict so view serializability also holds good
- As there is a possibility of dirty read, and no restriction on when to commit, so can be irrecoverable and may suffer from cascading rollback.
- At the time of request, here either we allow or we reject, so there is no idea of deadlock.
- If a schedule is not conflict serializable then it is not allowed by time stamp ordering scheme.
- But it is not necessary that all conflict serializable schedule generated by time stamping.
- Further modifications are possible if we want to ensure recoverability and cascade lessness, using different approaches
    - By performing all writes together at the end of the transaction, i.e. while writers are in progress, no transaction is permitted to access any of data items that have been written.
    - By using a limited form of locking, where by read of uncommitted items are postponed until the transaction that uploaded the item commit.
    - Recoverability alone can be ensured by tracking uncommitted writes and allowing a transaction t_i to commit only after the commit of any transaction that wrote a value that t_i read.

Conclusion: -

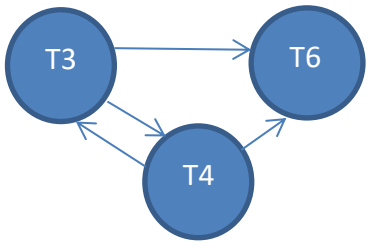
- It may cause starvation to occur, as if a sequence of conflicting short transactions causes repeated restarting of the long transaction, and then there is a possibility of starvation of long transaction.
- It is relatively slow as before executing every instruction we have to check conditions before. Time stamping protocol ensure that the schedule designed through this protocol will always be conflict serializable.
- This protocol can also be used for determining the serializability order (order in which transaction must execute) among the transaction in advance.

THOMAS WRITE RULE

- Thomas write is an improvement in time stamping protocol, which makes some modification and may generate those protocols that are even view serializable, because it allows greater potential concurrency.
- It is a Modified version of the timestamp-ordering protocol in which obsolete write operations may be ignored under certain circumstances.
- The protocol rules for read operations remain unchanged. while for write operation, there is slightly change in Thomas write rule than timestamp ordering protocol.
- **When Ti attempts to write data item Q, if TS(Ti) < W-timestamp(Q),** then Ti is attempting to write an obsolete value of {Q}. Rather than rolling back Ti as the timestamp ordering protocol would have done, this {write} operation can be ignored. Otherwise this protocol is the same as the timestamp ordering protocol.
- This modification is valid as the any transaction with **TS(Ti) < W-timestamp(Q),** the value written by this transaction will never be read by any other transaction performing Read(Q) ignoring such obsolete write operation is considerable.
- Thomas' Write Rule allows greater potential concurrency. Allows some view-serializable schedules that are not conflict serializable.
- ***E.g.-***

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Precedence Graph



- Above schedule is not allowed in timestamp ordering as the schedule is not conflict serializable (precedence graph consist of cycle) , while the schedule is allowed in Thomas write rule protocol because it ignores the write(Q) operation of T3,being it an obsolete updation to a value.And equivalent serial schedule is so T3->T4->T6.

Q In a database system, unique timestamps are assigned to each transaction using Lamport's logical clock. Let TS(T_1) and TS(T_2) be the timestamps of transactions T_1 and T_2 respectively. Besides, T_1 holds a lock on the resource R and T_2 has requested a conflicting lock on the same resource R. The following algorithm is used to prevent deadlocks in the database system assuming that a killed transaction is restarted with the same timestamp.

if TS(T_2)<TS(T_1) then
    T_1 is killed
else T_2 waits.

Assume any transaction that is not killed terminates eventually. Which of the following is TRUE about the database system that uses the above algorithm to prevent deadlocks?
(GATE-2017) (2 Marks)
(a) The database system is both deadlock-free and starvation-free
(b) The database system is deadlock-free, but not starvation-free
(c) The database system is starvation-free, but not deadlock-free
(d) The database system is neither deadlock-free nor starvation-free
Ans: a

Lock Based Protocols

- To ensure isolation is to require that data items be accessed in a mutually exclusive manner i.e. while one transaction is accessing a data item, no other transaction can modify that data item. Locking is the most fundamental approach to ensure this. Lock based protocols ensure this requirement. Idea is first obtaining a lock on the desired data item then if lock is granted then perform the operation and then unlock it.
- In general, we support two modes of lock because, to provide better concurrency.
- A data item can be locked in two modes-
- ***Shared mode***
    - If transaction T_i has obtained a shared-mode lock (denoted by S) on any data item Q, then T_i can read, but cannot write Q, any other transaction can also acquire a shared mode lock on the same data item(this is the reason we called this shared mode).
- **Exclusive mode**
    - If transaction T_i has obtained an exclusive-mode lock (denoted by X) on any data item Q, then T_i can both read and write Q, any other transaction cannot acquire either a shared or exclusive mode lock on the same data item. (this is the reason we called this exclusive mode)
- ***Lock –Compatibility Matrix***

Requested Lock \ Current State of lock of data items	Exclusive	Shared	Unlocked
Exclusive	N	N	Y
Shared	N	Y	Y
Unlock	Y	Y	-

- Conclusion shared is compatible only with shared while exclusive is not compatible either with shared or exclusive.
- To access a data item, transaction T_i must first lock that item, if the data item is already locked by another transaction in an incompatible mode, or some other transaction is already waiting in non-compatible mode, then concurrency control manager will not grant the lock until all incompatible locks held by other transactions have been released. The lock is then granted.
- ***Pitfalls of lock-based protocols-*** Lock based protocol *do not ensure serializability* as granting and releasing of lock do not follow any order and any transaction any time may go for lock and unlock. Here in the example below we can see, that even this transaction

in using locking but neither it is conflict serializable nor independent from deadlock. Even others problems also persist like non – recoverability or cascading rollbacks may occur.

S

T_1	T_2
LOCK-X(A)	
READ(A)	
WRITE(A)	
UNLOCK(A)	
	LOCK-S(B)
	READ(B)
	UNLOCK(B)
LOCK-X(B)	
READ(B)	
WRITE(B)	
UNLOCK(B)	
	LOCK-S(A)
	READ(A)
	UNLOCK(A)

- If we do not use locking, or if we unlock data items too soon after reading or writing them, we may get inconsistent states, as there exists a possibility of dirty read. On the other hand, if we do not unlock a data item before requesting a lock on another data item, deadlocks may occur and concurrency will be poor.
- We shall require that each transaction in the system follow a set of rules, called a **locking protocol**, indicating when a transaction may lock and unlock each of the data items for e.g. 2pl or graph based locking.
- Locking protocols restrict the number of possible schedules. The set of all such schedules is a proper subset of all possible serializable schedules.
- We say that a schedule S is **legal** under a given locking protocol if S is a possible schedule for a set of transactions that follows the rules of the locking protocol. We say that a locking protocol **ensures** conflict serializability if and only if all legal schedules are conflict serializable.
- When a transaction requests a lock on a data item in a particular mode, and no other transaction has a lock on the same data item in a conflicting mode, the lock can be granted. However, care must be taken to avoid the following scenario. Suppose a transaction T_2 has a shared-mode lock on a data item, and another transaction T_1 requests an exclusive-mode lock on the data item. Clearly, T_1 has to wait for T_2 to release the shared-mode lock. Meanwhile, a transaction T_3 may request a shared-mode lock on the same data item. The lock request is compatible with the lock granted to T_2, so T_3 may be granted the shared-mode lock. At this point T_2 may release the lock, but still T_1 has to wait for T_3 to finish. But again, there may be a new transaction T_4 that requests a shared-mode lock on the same data item, and is granted the lock before T_3 releases it.

In fact, it is possible that there is a sequence of transactions that each requests a shared-mode lock on the data item, and each transaction releases the lock a short while after it is granted, but T_1 never gets the exclusive-mode lock on the data item. The transaction T_1 may never make progress, and is said to be **starved**.

- We can avoid starvation of transactions by granting locks in the following manner: When a transaction T_i requests a lock on a data item Q in a particular mode M, the concurrency-control manager grants the lock provided that:
    - There is no other transaction holding a lock on Q in a mode that conflicts with M.
    - There is no other transaction that is waiting for a lock on Q and that made its lock request before T_i.
- a refinement of the basic two-phase locking protocol, in which **lock conversions** are allowed. We shall provide a mechanism for upgrading a shared lock to an exclusive lock, and downgrading an exclusive lock to a shared lock. We denote conversion from shared to exclusive modes by **upgrade**, and from exclusive to shared by **downgrade**. Lock conversion cannot be allowed arbitrarily. Rather, upgrading can take place in only the growing phase, whereas downgrading can take place in only the shrinking phase.

Two phase locking protocol(2PL)

- The protocol ensures that each transaction issue lock and unlock requests in two phases, note that each transaction will be 2 phased not schedule.
- Growing phase- A transaction may obtain locks, but not release any locks.
- Shrinking phase- A transaction may release locks, but may not obtain any new locks.
- Initially a transaction is in growing phase and acquires lock as needed and in between can perform operation reach to lock point and once a transaction releases a lock, it can issue no more lock requests i.e. it enters the shrinking phase.

.

T1	T2
LOCK-X(A)	
READ(A)	
WRITE(A)	
	LOCK-S(B)
	READ(B)
LOCK-X(B)	
READ(B)	
WRITE(B)	
	LOCK-S(A)
	READ(A)
	UNLOCK(B)
UNLOCK(A)	
UNLOCK(B)	
	UNLOCK(A)

Properties

2PL ensures conflict serializability, and the ordering of transaction over lock points is itself a serializability order of a schedule in 2PL.

If a schedule is allowed in 2PL protocol then definitely it is always conflict serializable. But it is not necessary that if a schedule is conflict serializable then it will be generated by 2pl. Equivalent serial schedule is based on the order of lock points.

View serializability is also guaranteed.

Does not ensure freedom from deadlock

May cause non-recoverability.

Cascading rollback may occur.

S2

T1	T2
LOCK-X(A)	
READ(A)	
WRITE(A)	
UNLOCK(A)	
	LOCK-S(A)
	READ(A)
	Commit
Commit	

Q Consider the following two statements about database transaction schedules:
I. Strict two-phase locking protocol generates conflict serializable schedules that are also recoverable.
II. Timestamp-ordering concurrency control protocol with Thomas Write Rule can generate view serializable schedules that are not conflict serializable.
Which of the above statements is/are TRUE? **(GATE-2019) (1 Marks)**

(a) Both I and II
(b) Neither I nor II
(c) II only
(d) I only

Ans: a

Q Which of the following concurrency control protocols ensure both conflict serializability and freedom from deadlock? **(GATE-2010) (2 Marks)**

I. 2-phase locking
II. Time-stamp ordering

(A) I only **(B)** II only **(C)** Both I and II **(D)** Neither I nor II

Answer: (B)

Q Which of the following concurrency protocol ensures both conflict serializability and freedom from deadlock? **(NET-JUNE-2015)**

(a) 2 - phase Locking
(b) Time stamp - ordering
(1) Both (a) and (b)
(2) (a) only
(3) (b) only
(4) Neither (a) nor (b)

Ans. 3

Q Which of the following statements is wrong? **(NET-JUNE-2007)**

(A) 2-phase Locking Protocols suffer from deadlocks
(B) Time-Stamp Protocols suffer from more aborts
(C) Time-Stamp Protocols suffer from cascading roll back where as 2-Phase locking Protocol do not
(D) None of these

Q Consider the following two-phase locking protocol. Suppose a transaction T accesses (for read or write operations), a certain set of objects$\{O_1,...,O_k\}$. This is done in the following manner:

Step1. T acquires exclusive locks to $O_1,...,O_k$ in increasing order of their addresses.
Step2. The required operations are performed.
Step3. All locks are released.

This protocol will (GATE- 2016) (1 Marks)

(a) guarantee serializability and deadlock-freedom
(b) guarantee neither serializability nor deadlock-freedom
(c) guarantee serializability but not deadlock-freedom
(d) guarantee deadlock-freedom but not serializability

Ans: a

Q Two phase protocol in a database management system is: **(NET-DEC-2006)**

(A) a concurrency mechanism that is not deadlock free
(B) a recovery protocol used for restoring a database after a crash
(C) Any update to the system log done in 2-phases
(D) not effective in Database

Ans: a

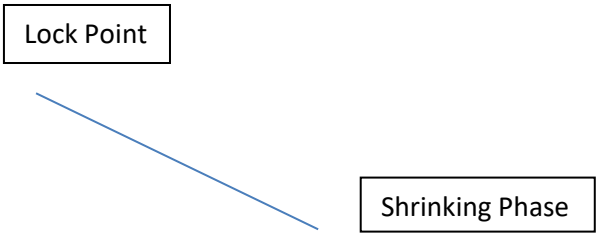
Q Which of the following is correct? **(NET-DEC-2014)**
I. Two phase locking is an optimistic protocol.
II. Two phase locking is pessimistic protocol
III. Time stamping is an optimistic protocol.
IV. Time stamping is pessimistic protocol.
(A) I and III **(B)** II and IV **(C)** I and IV **(D)** II and III
Ans: b

Q Which of the following concurrency protocol ensures both conflict serializability and freedom from deadlock: **(NET-JUNE-2014)**
I. 2-phase locking **II.** Time phase ordering
(A) Both I & II **(B)** II only **(C)** I only **(D)** Neither I nor II
Ans: b

Q Which of the following time stamp ordering protocol(s) allow(s) the following schedules?
T: $W_1(A)$; $W_2(A)$; $W_3(A)$; $R_2(A)$; $R_4(4)$;
Time stamps: $T_1 : 5$, $T_2 : 10$, $T_3 : 15$; $T_4 : 20$
a) Thomas write rule b) Basic time stamp

Variants of Two- Phase locking method

- Different variants of 2pl are used where we try ensure the properties like deadlock, recoverable, cascade less.
- Conservative 2pl
- The idea is there is no growing phase transaction start directly from lock point, i.e. transaction must first acquire all the required locks then only it can start execution. If all the locks are not available then transaction must release the acquired locks and must wait.
    - Shrinking phase will work as usual, and transaction can unlock any data item anytime.
    - we must have a knowledge in future to understand what is data required so that we can use it



- Properties
- Conflict serializable, view serializable, Independence from deadlock
    - Still have possibility of irrecoverable schedule and cascading rollbacks.

Q In conservative two-phase locking protocol, a transaction
a) Should release exclusive locks only after the commit operation
b) Should release all the locks only at beginning of the transaction
c) should acquire all the locks at beginning of the transaction
d) Should acquire all the exclusive locks at beginning transaction

RIGOROUS 2PL

- requires that all locks be held until the transaction commits.
- This protocol requires that locking be two phase and also all the locks taken be held by transaction until that transaction commit.
- Hence there is no shrinking phase in the system.

E.g.

T1	T2
$X(A)$	
$R(A)$	
$W(A)$	
$X(B)$	
$R(B)$	
$W(B)$	
Commit	
	$X(A)$
	$R(A)$
	$W(A)$
	$X(B)$
	$R(B)$
	$W(B)$
	Commit

- Properties
- Conflict serializable, view serializable, recoverable and cascade less
  - Still have possibility of deadlocks.

Q In a Rigorous 2 phase protocol

a) All shared locks held by the transaction are related after the transaction is committed

b) All exclusive locks held by the transaction are released after the transaction is committed

c) All locks held by the transaction are released after the transaction is committed

d) All locks held by the transaction are released before the transaction is committed

STRICT 2PL

- that all exclusive-mode locks taken by a transaction be held until that transaction commits. This requirement ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits, preventing any other transaction from reading the data.
- This protocol requires that locking be two phase and also that exclusive –mode locks taken by transaction be held until that transaction commits.
- So it is simplified form of rigorous 2pl

- ***E.g.***

Locking (Strict 2PL)

T1	T2
$S(A)$	
$R(A)$	
	$S(A)$
	$R(A)$
	$X(B)$
	$R(B)$
	$W(B)$
	Commit
$X(C)$	
$R(C)$	
$W(C)$	
Commit	

Schedule Following Strict 2PL with Interleaved Actions

- It ensures serializability (Equivalent serial schedule order based on the order of lock points.
- Ensures strict recoverability.
- Deadlock still possible in strict 2PL.
- In general, deadlocks are a necessary evil associated with locking, if we want to avoid inconsistent states. Deadlocks are definitely preferable to inconsistent states, since they can be handled by rolling back transactions, whereas inconsistent states may lead to real-world problems that cannot be handled by the database system.
- Strict two-phase locking and rigorous two-phase locking (with lock conversions) are used extensively in commercial database systems.

Q Which of the following is a false statement?
a) A schedule which is allowed under basic 2PL is always under strict 2PL
b) A schedule which is allowed under strict 2PL is always allowed under basic 2PL
c) A schedule which is allowed under basic time stamp protocol is always allowed under Thomas write rule
d) None of these

Q Which of the following statement is/are correct

a) Every conflict serializable schedule allowed under 2PL protocol is allowed by basic time stamping protocol.

b) Every schedule allowed under basic time stamping protocol is allowed by Thomas-write rule

c) Every schedule allowed under Thomas-write rule is allowed by basic time stamping protocol

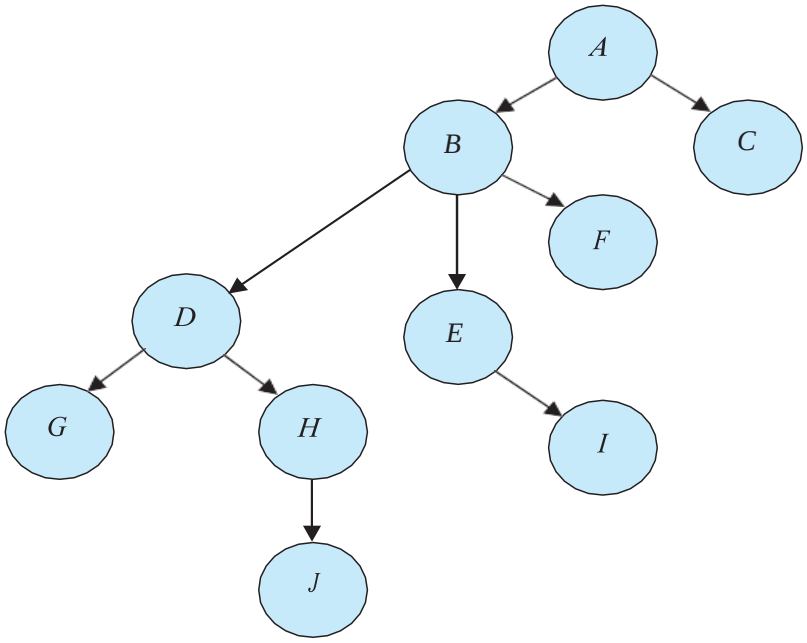
d) none

Graph based protocol

- if we wish to develop protocols that are not two phase, we need additional information on how each transaction will access the database.
- There are various models that can give us the additional information, each differing in the amount of information provided.
- The simplest model requires that we have prior knowledge about the order in which the database items will be accessed.
- Given such information, it is possible to construct locking protocols that are not two phases, but that, nevertheless, ensure conflict serializability.
- To acquire such prior knowledge, we impose a partial ordering → on the set **D** = $\{d_1, d_2, \ldots, d_h\}$ of all data items. If $d_i \rightarrow d_j$, then any transaction accessing both d_i and d_j must access d_i before accessing d_j.
- This partial ordering may be the result of either the logical or the physical organization of the data, or it may be imposed solely for the purpose of concurrency control.
- The partial ordering implies that the set **D** may now be viewed as a directed acyclic graph, called a **database graph**.
- Here for the sake of simplicity, we will follow two restriction
    - Will study graphs that are rooted trees.
    - Will restrict to employ only *exclusive* locks.

Tree Protocol

- In the tree protocol, the only lock instruction allowed is lock-X. Each transaction Ti can lock a data item at most once, and must observe the following rules:
- The first lock by Ti may be on any data item.
- Subsequently, a data item Q can be locked by Ti only if the parent of Q is currently locked by Ti.
- Data items may be unlocked at any time.



- A data item that has been locked and unlocked by Ti cannot subsequently be relocked by Ti.

T_1: lock-X(*B*); lock-X(*E*); lock-X(*D*); unlock(*B*); unlock(*E*); lock-X(*G*); unlock(*D*); unlock(*G*).
T_2: lock-X(*D*); lock-X(*H*); unlock(*D*); unlock(*H*).
T_3: lock-X(*B*); lock-X(*E*); unlock(*E*); unlock(*B*).
T_4: lock-X(*D*); lock-X(*H*); unlock(*D*); unlock(*H*).

Properties

- All schedules that are legal under the tree protocol are conflict serializable.
- tree protocol ensures freedom from deadlock.
- tree protocol does not ensure recoverability and cascadelessness.
- The tree-locking protocol has another advantage over the two-phase locking protocol in that unlocking may occur earlier. Earlier unlocking may lead to shorter waiting times, and to an increase in concurrency.
- A transaction may have to lock data items that it does not access. This additional locking results in increased locking overhead, the possibility of additional waiting time, and a potential decrease in concurrency.
- Without prior knowledge of what data items will need to be locked, transactions will have to lock the root of the tree, and that can reduce concurrency greatly.
- there may be conflict-serializable schedules that cannot be obtained through the tree protocol. Indeed, there are schedules possible under the two-phase locking protocol that are not possible under the tree protocol, and vice versa.
- To ensure recoverability and cascadelessness, the protocol can be modified to not permit release of exclusive locks until the end of the transaction. Holding exclusive locks until the end of the transaction reduces concurrency.

Deadlock Handling

- There are two principal methods for dealing with the deadlock problem.
- We can use a **deadlock prevention** protocol to ensure that the system will *never* enter a deadlock state, Prevention is commonly used if the probability that the system would enter a deadlock state is relatively high
- Alternatively, we can allow the system to enter a deadlock state, and then try to recover by using a **deadlock detection** and **deadlock recovery** scheme.
- both methods may result in transaction rollback more efficient. Note that a detection and recovery scheme require overhead that includes not only the run-time cost of maintaining the necessary information and of executing the detection algorithm, but also the potential losses inherent in recovery from a deadlock.

Deadlock Prevention

- One approach ensures that no hold & waits can occur it is the simplest scheme requires that each transaction locks all its data items before it begins execution, either all are locked in one step or none are locked e.g. conservative 2PL.
- Other approach ensures that no cyclic waits can occur by ordering the requests for locks, i.e. is to impose an ordering of all data items, and to require that a transaction lock data items only in a sequence consistent with the ordering e.g. tree protocol.
    - it is often hard to predict, before the transaction begins, what data items need to be locked;
    - data-item utilization may be very low, since many of the data items may be locked but unused for a long time.
- The other approach is closer to deadlock recovery, and performs transaction rollback instead of waiting for a lock, whenever the wait could potentially result in a deadlock i.e. to use preemption and transaction rollbacks.

- So when a transaction T_j requests a lock that transaction T_i holds, the lock granted to T_i may be **preempted** by rolling back of T_i, and granting of the lock to T_j. To control the preemption, we assign a unique timestamp, to each transaction when it begins. The system uses these timestamps only to decide whether a transaction should wait or roll back. Locking is still used for concurrency control. If a transaction is rolled back, it retains its *old* timestamp when restarted. Two different deadlock-prevention schemes using timestamps have been proposed:
    - The **wait–die** scheme is a non-preemptive technique. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp smaller than that of T_j (that is, T_i is older than T_j). Otherwise, T_i is rolled back (dies).
    - The **wound–wait** scheme is a preemptive technique. It is a counterpart to the wait–die scheme. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j (that is, T_i is younger than T_j). Otherwise, T_j is rolled back (T_j is *wounded* by T_i).
- The major problem with both of these schemes is that unnecessary rollbacks may occur. Another simple approach to deadlock prevention is based on **lock timeouts**. In this approach, a transaction that has requested a lock waits for at most a specified amount of time. If the lock has not been granted within that time, the transaction is said to time out, and it rolls itself back and restarts. If there was in fact a deadlock, one or more transactions involved in the deadlock will time out and roll back, allowing the others to proceed. This scheme falls somewhere between deadlock prevention, where a deadlock will never occur, and deadlock detection and recovery.
- The timeout scheme is particularly easy to implement, and works well if transactions are short and if long waits are likely to be due to deadlocks. However, in general it is hard to decide how long a transaction must wait before timing out. Too long a wait results in unnecessary delays once a deadlock has occurred. Too short a wait results in transaction rollback even when there is no deadlock, leading to wasted resources. Starvation is also

a possibility with this scheme. Hence, the timeout-based scheme has limited applicability.

Deadlock Detection and Recovery

- If a system does not employ some protocol that ensures deadlock freedom, then a detection and recovery scheme must be used. An algorithm that examines the state of the system is invoked periodically to determine whether a deadlock has occurred. If one has, then the system must attempt to recover from the deadlock. To do so, the system must:
- Maintain information about the current allocation of data items to transaction, as well as any outstanding data item requests.
- Provide an algorithm that uses this information to determine whether the system has entered a deadlock state.
- Recover from the deadlock when the detection algorithm determines that a deadlock exists.

- Deadlock Detection - Deadlocks can be described precisely in terms of a directed graph called a **wait- for graph**. This graph consists of a pair $G = (V, E)$, where V is a set of vertices and E is a set of edges. The set of vertices consists of all the transactions in the system. Each element in the set E of edges is an ordered pair $T_i \rightarrow T_j$. If $T_i \rightarrow T_j$ is in E, then there is a directed edge from transaction T_i to T_j, implying that transaction T_i is waiting for transaction T_j to release a data item that it needs.
- A deadlock exists in the system if and only if the wait-for graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked. To detect deadlocks, the system needs to maintain the wait-for graph, and periodically to invoke an algorithm that searches for a cycle in the graph.
- When should we invoke the detection algorithm? The answer depends on two factors:
- How often does a deadlock occur? - If deadlocks occur frequently, then the detection algorithm should be invoked more frequently. Data items allocated to deadlocked transactions will be unavailable to other transactions until the deadlock can be broken.
- How many transactions will be affected by the deadlock? - In addition, the number of cycles in the graph may also grow. In the worst case, we would invoke the detection algorithm every time a request for allocation could not be granted immediately.

Recovery from Deadlock

- When a detection algorithm determines that a deadlock exists, the system must **recover** from the deadlock. The most common solution is to roll back one or more transactions to break the deadlock. Three actions need to be taken:
- **Selection of a victim**. Given a set of deadlocked transactions, we must determine which transaction (or transactions) to roll back to break the deadlock. We should roll back those transactions that will incur the minimum cost. Unfortunately, the term *minimum cost* is not a precise one. Many factors may determine the cost of a rollback, including:
- How long the transaction has computed, and how much longer the transaction will compute before it completes its designated task.
- How many data items the transaction has used.
- How many more data items the transaction needs for it to complete.
- How many transactions will be involved in the rollback.
- **Rollback**. Once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be rolled back.
- The simplest solution is a **total rollback**: Abort the transaction and then restart it.
- However, it is more effective to roll back the transaction only as far as necessary to break the deadlock. Such **partial rollback** requires the system to maintain additional information about the state of all the running transactions. Specifically, the sequence of lock requests/grants and updates performed by the transaction needs to be recorded. The deadlock detection mechanism should decide which locks the selected transaction needs to release in order to break the deadlock. The selected transaction must be rolled back to the point where it obtained the first of these locks, undoing all actions it took after that point. The recovery mechanism must be capable of performing such partial rollbacks. Furthermore, the transactions must be capable of resuming execution after a partial rollback.
- **Starvation**. In a system where the selection of victims is based primarily on cost factors, it may happen that the same transaction is always picked as a victim. As a result, this transaction never completes its designated task, thus there is **starvation**. We must ensure that a transaction can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

Multiple Granularity

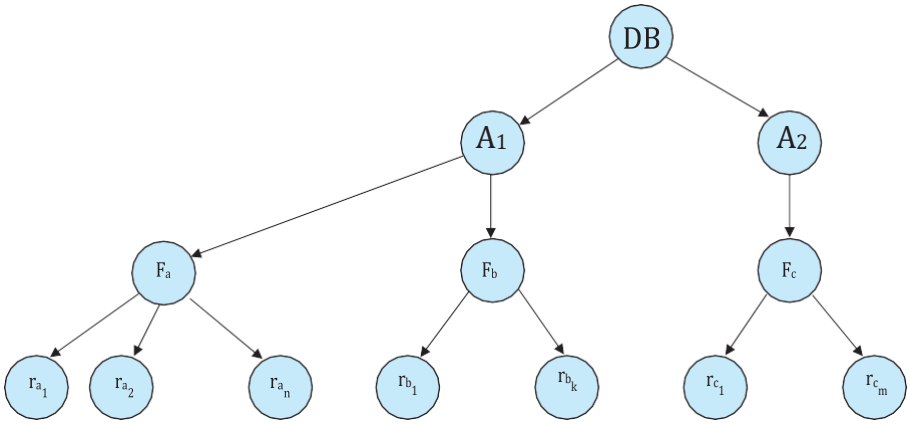
In the concurrency-control schemes described thus far, we have used each individual data item as the unit on which synchronization is performed.

There are circumstances, however, where it would be advantageous to group several data items, and to treat them as one individual synchronization unit. For example, if a transaction T_i needs to access the entire database, and a locking protocol is used, then T_i must lock each item in the database. Clearly, executing these locks is time-consuming. It would be better if T_i could issue a *single* lock request to lock the entire database. On the other hand, if transaction T_j needs to access only a few data items, it should not be required to lock the entire database, since otherwise concurrency is lost.

What is needed is a mechanism to allow the system to define multiple levels of **granularity**. This is done by allowing data items to be of various sizes and defining a hierarchy of data granularities, where the small granularities are nested within larger ones. Such a hierarchy can be represented graphically as a tree. A non-leaf node of the multiple-granularity tree represents the data associated with its descendants. In the tree protocol, each node is an independent data item.

As an illustration, consider the tree of Figure, which consists of four levels of nodes. The highest level represents the entire database. Below it are nodes of type *area*; the database consists of exactly these areas. Each area in turn has nodes of type *file* as its children. Each area contains exactly those files that are its child nodes. No file is in more than one area. Finally, each file has nodes of type *record*. As before, the file consists of exactly those records that are its child nodes, and no record can be present in more than one file.

Each node in the tree can be locked individually. As we did in the two-phase locking protocol, we shall use **shared** and **exclusive** lock modes. When a transaction locks a node, in either shared or exclusive mode, the transaction also has implicitly locked all the descendants of that node in the same lock mode. For example, if transaction T_i gets an **explicit lock** on file F_c of Figure, in exclusive mode, then it has an **implicit lock** in exclusive mode on all the records belonging to that file. It does not need to lock the individual records of F_c explicitly. explicitly.



Suppose now that transaction T_k wishes to lock the entire database. To do so, it simply must lock the root of the hierarchy. Note, however, that T_k should not succeed in locking the root node, since T_i is currently holding a lock on part of the tree (specifically, on file F_b). But how does the system determine if the root node can be locked? One possibility is for it to search the entire tree. This solution, however, defeats the whole purpose of the multiple-granularity locking scheme. A more efficient way to gain this knowledge is to introduce a new class of lock modes, called **intention lock modes**. If a node is locked in an intention mode, explicit locking is done at a lower level of the tree (that is, at a finer granularity). Intention locks are put on all the ancestors of a node before that node is locked explicitly. Thus, a transaction does not need to search the entire tree to determine whether it can lock a node successfully. A transaction wishing to lock a node—say, Q—must traverse a path in the tree from the root to Q. While traversing the tree, the transaction locks the various nodes in an intention mode.

There is an intention mode associated with shared mode, and there is one with exclusive mode. If a node is locked in **intention-shared (IS) mode**, explicit locking is being done at a lower level of the tree, but with only shared-mode locks. Similarly, if a node is locked in **intention-exclusive (IX) mode**, then explicit locking is being done at a lower level, with exclusive-mode or shared-mode locks. Finally, if a node is locked in **shared and intention-exclusive (SIX) mode**, the subtree rooted by that node is locked explicitly in shared mode, and that explicit locking is being done at a lower level with exclusive-mode locks.

	IS	IX	S	SIX	X
IS	true	true	true	true	false
IX	true	true	false	false	false
S	true	false	true	false	false
SIX	true	false	false	false	false
X	false	false	false	false	false

The **multiple-granularity locking protocol** uses these lock modes to ensure serializability. It requires that a transaction T_i that attempts to lock a node Q must follow these rules:

1. Transaction T_i must observe the lock-compatibility function of Figure 15.16.
2. Transaction T_i must lock the root of the tree first, and can lock it in any mode.
3. Transaction T_i can lock a node Q in S or IS mode only if T_i currently has the parent of Q locked in either IX or IS mode.
4. Transaction T_i can lock a node Q in X, SIX, or IX mode only if T_i currently has the parent of Q locked in either IX or SIX mode.

5. Transaction T_i can lock a node only if T_i has not previously unlocked any node (that is, T_i is two phase).
6. Transaction T_i can unlock a node Q only if T_i currently has none of the children of Q locked.

Observe that the multiple-granularity protocol requires that locks be acquired in *top-down* (root-to-leaf) order, whereas locks must be released in *bottom-up* (leaf- to-root) order. This protocol enhances concurrency and reduces lock overhead. It is particularly useful in applications that include a mix of:

- Short transactions that access only a few data items.
- Long transactions that produce reports from an entire file or set of files.

Validation-Based Protocols

In cases where a majority of transactions are read-only transactions, the rate of conflicts among transactions may be low. Thus, many of these transactions, if executed without the supervision of a concurrency-control scheme, would nevertheless leave the system in a consistent state.

A concurrency-control scheme imposes overhead of code execution and possible delay of transactions. It may be better to use an alternative scheme that imposes less overhead.

The **validation protocol** requires that each transaction T_i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. The phases are, in order:

Read phase. During this phase, the system executes transaction T_i. It reads the values of the various data items and stores them in variables local to Ti. It performs all write operations on temporary local variables, without updates of the actual database.

Validation phase. The validation test (described below) is applied to transaction T_i. This determines whether T_i is allowed to proceed to the write phase without causing a violation of serializability. If a transaction fails the validation test, the system aborts the transaction.

Write phase. If the validation test succeeds for transaction T_i, the temporary local variables that hold the results of any write operations performed by T_i are copied to the database.

To perform the validation test, we need to know when the various phases of transactions took place. We shall, therefore, associate three different timestamps with each transaction T_i:

1. **Start**(T_i), the time when T_i started its execution.
2. **Validation**(T_i), the time when T_i finished its read phase and started its validation phase.
3. **Finish**(T_i), the time when T_i finished its write phase.

We determine the serializability order by the timestamp-ordering technique, using the value of the timestamp Validation(T_i). Thus, the value TS(T_i) = Validation(T_i) and, if TS(T_j) < TS(T_k), then any produced schedule must be equivalent to a serial schedule in which transaction T_j appears before transaction T_k. The reason we have chosen Validation(T_i), rather than Start(T_i), as the timestamp of transaction T_i is that we can expect faster response time provided that conflict rates among transactions are indeed low.

The **validation test** for transaction T_i requires that, for all transactions T_k with TS(T_k) < TS(T_i), one of the following two conditions must hold:

1. Finish(T_k) < Start(T_i). Since T_k completes its execution before T_i started, the

serializability order is indeed maintained.

2. The set of data items written by T_k does not intersect with the set of data items read by T_i, and T_k completes its write phase before T_i starts its validation phase (Start(T_i) < Finish(T_k) < Validation(T_i)). This condition ensures that the writes of T_k and T_i do not overlap. Since the writes of T_k do not affect the read of T_i, and since T_i cannot affect the read of T_k, the serializability order is indeed maintained.

The validation scheme automatically guards against cascading rollbacks, since the actual writes take place only after the transaction issuing the write has committed. However, there is a possibility of starvation of long transactions, due to a sequence of conflicting short transactions that cause repeated restarts of the long transaction. To avoid starvation, conflicting transactions must be temporarily blocked, to enable the long transaction to finish.

This validation scheme is called the **optimistic concurrency-control** scheme since transactions execute optimistically, assuming they will be able to finish execution and validate at the end. In contrast, locking and timestamp ordering are pessimistic in that they force a wait or a rollback whenever a conflict is detected, even though there is a chance that the schedule may be conflict serializable.

When several transactions execute concurrently in the database, the consistency of data may no longer be preserved. It is necessary for the system to control the interaction among the concurrent transactions, and this control is achieved through one of a variety of mechanisms called *concurrency-control* schemes.

To ensure serializability, we can use various concurrency-control schemes. All these schemes either delay an operation or abort the transaction that issued the operation. The most common ones are locking protocols, timestamp- ordering schemes, validation techniques, and multiversion schemes.

- A locking protocol is a set of rules that state when a transaction may lock and unlock each of the data items in the database.
- The two-phase locking protocol allows a transaction to lock a new data item only if that transaction has not yet unlocked any data item. The protocol ensures serializability, but not deadlock freedom. In the absence of information concerning the manner in which data items are accessed, the two-phase locking protocol is both necessary and sufficient for ensuring serializability.
- The strict two-phase locking protocol permits release of exclusive locks only at the end of transaction, in order to ensure recoverability and cascadelessness

of the resulting schedules. The rigorous two-phase locking protocol releases all locks only at the end of the transaction.

- Graph-based locking protocols impose restrictions on the order in which items are accessed, and can thereby ensure serializability without requiring the use of two-phase locking, and can additionally ensure deadlock freedom.
- Various locking protocols do not guard against deadlocks. One way to pre- vent deadlock is to use an ordering of data items, and to request locks in a sequence consistent with the ordering.
- Another way to prevent deadlock is to use preemption and transaction roll- backs. To control the preemption, we assign a unique timestamp to each transaction. The system uses these timestamps to decide whether a transac- tion should wait or roll back. If a transaction is rolled back, it retains its old timestamp when restarted. The wound–wait scheme is a preemptive scheme.
- If deadlocks are not prevented, the system must deal with them by using a deadlock detection and recovery scheme. To do so, the system constructs a wait-for graph. A system is in a deadlock state if and only if the wait-for graph contains a cycle. When the deadlock detection algorithm determines that a deadlock exists, the system must recover from the deadlock. It does so by rolling back one or more transactions to break the deadlock.
- There are circumstances where it would be advantageous to group several data items, and to treat them as one aggregate data item for purposes of working, resulting in multiple levels of granularity. We allow data items of various sizes, and define a hierarchy of data items, where the small items are nested within larger ones. Such a hierarchy can be represented graphically as a tree. Locks are acquired in root-to-leaf order; they are released in leaf-to-root order. The protocol ensures

serializability, but not freedom from deadlock.

- A timestamp-ordering scheme ensures serializability by selecting an ordering in advance between every pair of transactions. A unique fixed timestamp is associated with each transaction in the system. The timestamps of the transactions determine the serializability order. Thus, if the timestamp of transaction T_i is smaller than the timestamp of transaction T_j, then the scheme ensures that the produced schedule is equivalent to a serial schedule in which transaction T_i appears before transaction T_j. It does so by rolling back a transaction whenever such an order is violated.
- A validation scheme is an appropriate concurrency-control method in cases where a majority of transactions are read-only transactions, and thus the rate of conflicts among these transactions is low. A unique fixed timestamp is associated with each transaction in the system. The serializability order is determined by the timestamp of the transaction. A transaction in this scheme is never delayed. It must, however, pass a validation test to complete. If it does not pass the validation test, the system rolls it back to its initial state.
- A multiversion concurrency-control scheme is based on the creation of a new version of a data item for each transaction that writes that item. When a read operation is issued, the system selects one of the versions to be read. The concurrency-control scheme ensures that the version to be read is selected in a manner that ensures serializability, by using timestamps. A read operation always succeeds.

  In multiversion timestamp ordering, a write operation may result in the rollback of the transaction.
  In multiversion two-phase locking, write operations may result in a lock wait or, possibly, in deadlock.

- Snapshot isolation is a multiversion concurrency-control protocol based on validation, which, unlike multiversion two-phase locking, does not require transactions to be declared as read-only or update. Snapshot isolation does not guarantee serializability, but is nevertheless supported by many database systems.
- A **delete** operation may be performed only if the transaction deleting the tuple has an exclusive lock on the tuple to be deleted. A transaction that inserts a new tuple into the database is given an exclusive lock on the tuple.
- Insertions can lead to the phantom phenomenon, in which an insertion log- ically conflicts with a query even though the two transactions may access no tuple in common. Such conflict cannot be detected if locking is done only on tuples accessed by the transactions. Locking is required on the data used to find the tuples in the relation. The index-locking technique solves this problem by requiring locks on certain index nodes. These locks ensure that all conflicting transactions conflict on a real data item, rather than on a phantom.
- Weak levels of consistency are used in some applications where consistency of query results is not critical, and using serializability would result in queries adversely affecting transaction processing. Degree-two consistency is one such weaker level of

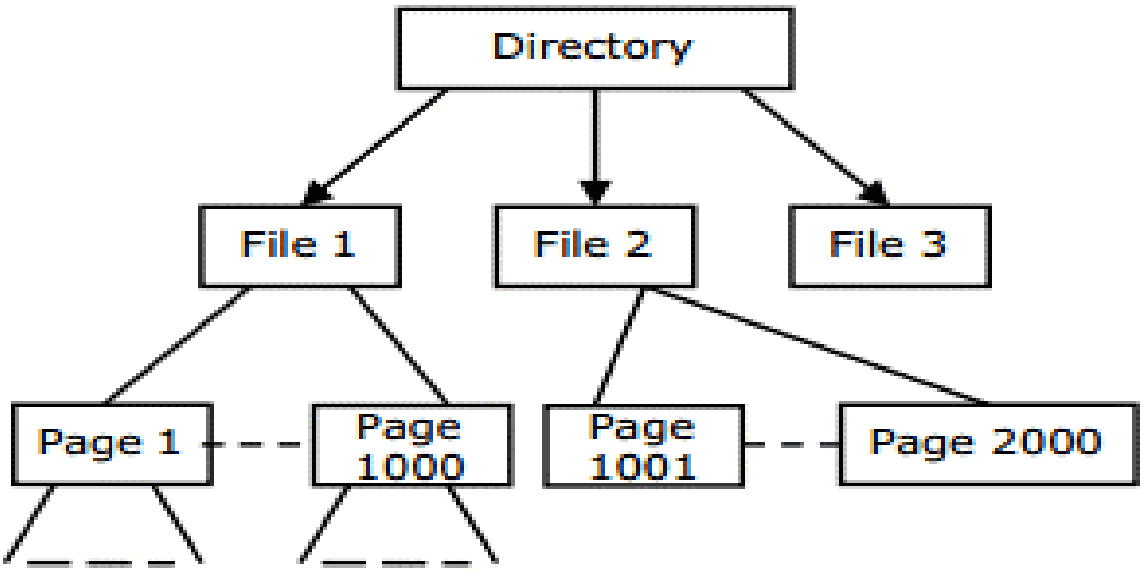
consistency; cursor stability is a special case of degree- two consistency, and is widely used.

- Concurrency control is a challenging task for transactions that span user interactions. Applications often implement a scheme based on validation of writes using version numbers stored in tuples; this scheme provides a weak level of serializability, and can be implemented at the application level without modifications to the database.
- Special concurrency-control techniques can be developed for special data structures. Often, special techniques are applied in B^+-trees to allow greater concurrency. These techniques allow non-serializable access to the B^+-tree, but they ensure that the B^+-tree structure is correct, and ensure that accesses to the database itself are serializable.

Q If the interference among the transactions is less, which of the following concurrency control techniques have less overloaded in the execution?

a) Locking techniques
b) Time stamping techniques
c) Validation technique
d) None

Q Consider a directory having 3 files, each file has 1000 pages and each page has 100 records



In multiple granularity locking protocol, if a transaction reads records from page 200 to page 700. What is the sequence of locks acquired?

a) IX on directory, S on file 1
b) IS on directory, S on file 1
c) IS on directory, X on file 1
d) IS on directory, S on file 2

Q _____ rules used to limit the volume of log information that has to be handled and

processed in the event of system failure involving the loss of volatile information. **(NET-DEC-2014)**

(A) Write-ahead log
(B) Check-pointing
(C) Log buffer
(D) Thomas

Ans: b

Q Immediate updates as a recovery protocol is preferable, when: **(NET-JUNE-2006)**

(A) Database reads more than writes
(B) Writes are more than reads
(C) It does not matter as it is good in both the situations
(D) There are only writes

Ans: b

Q In DBMS, deferred update means: **(NET-DEC-2006)**

(A) All the updates are done first but the entries are made in the log file later
(B) All the log files entries are made first but the actual updates are done later
(C) Every update is done first followed by a writing on the log file
(D) Changes in the views are deferred till a query asks for a view

Ans: b

Q Which of the following statement is true? **(NET-DEC-2009)**

I. 2-phase locking protocol suffer from dead lock.
II. Time stamp protocol suffer from more aborts.
III. A block hole in a DFD is a data store with only inbound flows.
IV. Multivalued dependency among attribute is checked at 3 NF level.
V. An entity-relationship diagram is a tool to represent event model.

(A) I, II, II **(B)** II, III, IV **(C)** III, IV, V **(D)** II, IV, V

Ans: a

Q Which of the following is an optimistic concurrency control method? **(NET-DEC-2010)**

(A) Validation based
(B) Time stamp ordering
(C) Lock-based
(D) None of these

Ans: a

Q The basic variants of time-stamp based method of concurrency control are **(NET-JUNE-2011)**

(A) Total time stamp-ordering
(B) Partial time stamp ordering
(C) Multiversion Time stamp ordering
(D) All of the above

Ans: d

Q Which of the following is the recovery management technique in DDBMS? **(NET-JUNE-2011)**

(A) 2PC (Two Phase Commit)
(B) Backup
(C) Immediate update
(D) All of the above

Ans: d

Q Match the following: **(NET-JUNE-2014)**

List – I	List – II
a. Timeout ordering protocol	i. Wait for graph
b. Deadlock prevention	ii. Roll back
c. Deadlock Detection	iii. Wait-die scheme
d. Deadlock recovery	iv. Thomas Write rule

Codes:

	a	**b**	**c**	**d**
(a)	iv	iii	i	ii
(b)	iii	ii	iv	i
(c)	ii	i	iv	iii
(d)	iii	i	iv	iii

Ans: a

Query Language

- After designing a data base, that is ER diagram followed by conversion in relational model followed by normalization and indexing, now next task is how to store, retrieve and modify data in the data base. Thought here we will be concentrating more on the retrieval part. Query languages are used for this purpose.

- **QUERY LANGUAGE**
  - A Languages using which user request some information from the database.
- **Procedural Query Language**
  - Here users instruct the system to performs a sequence of operations on the data base in order to compute the desired result.
  - Means user provides both what data to be retrieved and how data to be retrieved. e.g. Relational Algebra.
- **Non-Procedural Query Language**
  - In nonprocedural language, the user describes the desired information without giving a specific procedure for obtaining that information. What data to be retrieved e.g. Relational Calculus. **Tuple relational calculus, Domain relational calculus are** declarative query languages based on mathematical logic
- Relational Algebra (Procedural) and Relational Calculus (non-procedural) are mathematical system/ query languages which are used for query on relational model. RA and RC are not executed in any computer, they provide the fundamental mathematics on which SQL is based.
- SQL (structured query language) works on RDBMS, and it includes elements of both procedural or non-procedural query language.

Relational model	RDBMS
RA, RC	SQL
Algo	Code
Conceptual	Reality
Theoretical	Practical
Chess	Battle Field

RELATIONAL ALGEBRA

- RA like any other mathematical system provides a number of operators and use relations (tables) as operands and produce a new relation as their result.
- Every operator in the RA accepts (one or two) relation/table as input arguments and returns always a single relation instance as the result without a name.
- It also does not consider duplicity by default as it is based on set theory. Same query is written in RA and SQL the result may be different as SQL considers duplication.
- As it is pure mathematics no use of English keywords. Operators are represented using symbols.
- The relational algebra is a ***procedural query language***.
- The fundamental operations in the relational algebra are **select, project, union, set difference, Cartesian product, and Rename.**
- There are several other operations namely: **set intersection, natural join, and assignment.**
- The **select, project, and rename** operations are called **unary operations**, because they operate on one relation.
- **Union, Cartesian product and set difference** operate on pairs of relations and are, therefore, called **binary operations.**
- Relational algebra also provides the framework for query optimization.

- **Relational schema** - A **relation schema** R, denoted by R (A_1, A_2, ..., A_n), is made up of a relation name R and a list of attributes, A_1, A_2, ..., A_n. Each **attribute** Ai is the name of a role played by some domain D in the relation schema R. It is use to describe a Relation.
- E.g. Schema representation of Table **Student** is as –
    - **STUDENT (NAME, ID, CITY, COUNTRY, HOBBY).**
- **Relational Instance** - Relations with its data at particular instant of time.

Q If D_1, D_2, ... D_n are domains in a relational model, then the relation is a table, which is a subset of **(NET-JUNE-2013)**

(A) $D_1 + D_2 + ... + D_n$ **(B)** $D_1 \times D_2 \times ... \times D_n$

(C) $D_1 \cup D_2 \cup ... \cup D_n$ **(D)** $D_1 - D_2 - ... - D_n$

Ans: b

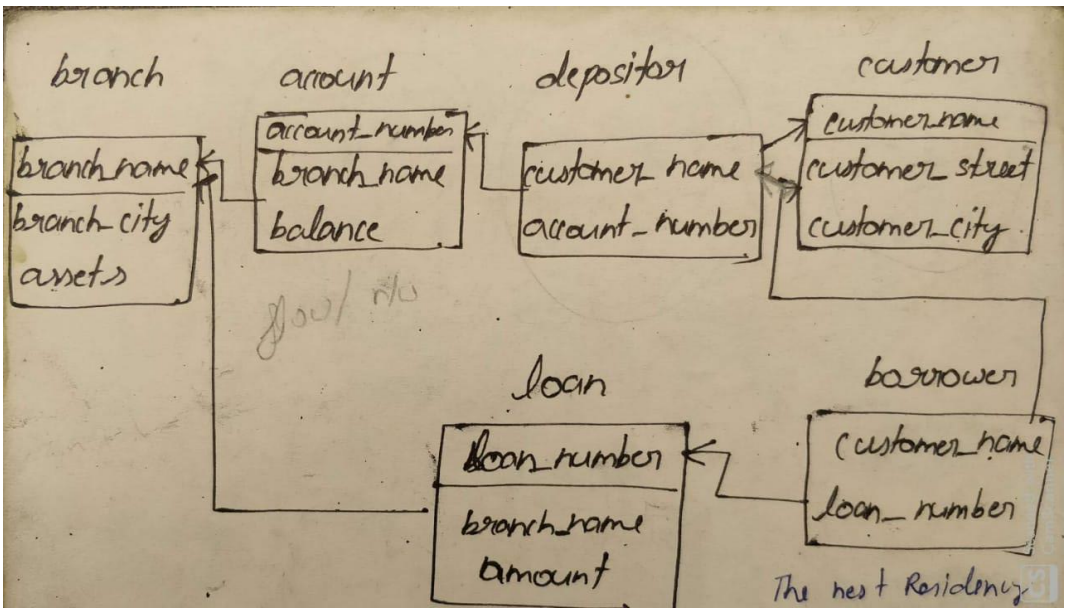
OPERATORS USED IN RELATIONAL ALGEBRA

- **BASIC / FUNDAMENTAL OPERATORS**

Name	Symbol
Select	(σ)
Project	(∏)
Union	(∪)
Set difference	(–)
Cross product	(X)
Rename	(ρ)

- **DERIVED OPERATORS**

Name	Symbol	DERIVED FROM
Join	(⋈)	(X)
Intersection	(∩)	(–) A∩B=A-(A-B)
Division	(÷)	(X,-, ∏)
Assignment	(=)	



The Project Operation (Vertical Selection)

- Main idea behind project operator is to select desired columns.
- The project operation is a unary operation that returns its argument relation, with certain attributes left out.
- Projection is denoted by the uppercase Greek letter pi (Π).
- **Π_{column_name} (table_name)**
- We list those attributes that we wish to appear in the result as a subscript to Π, argument relation follows in parentheses.
- Minimum number of columns selected can be 1, Maximum selected Columns can be n - 1.

- **Some points to remember**
  - Eliminates duplicate rows in a result relation by default.
  - $\prod_{A1, A2, An}$ (r), A_1, A_2 ,…., An refers to the set of attributes to be projected.
  - It is not commutative.

Q Write a RELATIONAL ALGEBRA query to find the name of all customer having bank account?

Q Write a RELATIONAL ALGEBRA query to find each loan number along with loan amount?

Q Write a RELATIONAL ALGEBRA query to find the name of all customer without duplication having bank account?

Q Write a RELATIONAL ALGEBRA query to find all the details of bank branches?

Q Suppose $R_1(A, B)$ and $R_2(C, D)$ are two relation schemas. Let r_1 and r_2 be the corresponding relation instances. B is a foreign key that refers to C in R_2. If data in r_1 and r_2 satisfy referential integrity constraints, which of the following is ALWAYS TRUE? **(Gate-2012) (2 Marks)**

a) $\prod B(r_1) - \prod C(r_2) = \emptyset$ **b)** $\prod C(r_2) - \prod B(r_1) = \emptyset$

c) $\prod B(r_1) = \prod C(r_2)$ **d)** $\prod B(r_1) - \prod C(r_2) \neq \emptyset$

Ans: a

Answer is **A.**

Referential integrity means, all the values in foreign key should be present in primary key.

$r2(c)$ is the super set of $r1(b)$

So, {subset - superset} is always empty set.

The Select Operation (Horizontal Selection)

- The select operation selects tuples that satisfy a given predicate/Condition p.
- Lowercase Greek letter sigma (σ) is used to denote selection.
- It is a unary operator.
- Eliminates only tuples/rows.
- $\sigma_{condition}$ (table_name)
- Predicate appears as a subscript to σ, the argument relation is in parentheses after the σ.
- Commutative in Nature, $\sigma_{p1}(\sigma_{p2}(r)) = \sigma_{p2}(\sigma_{p1}(r))$

Some points to remember

- We allow comparisons using =, ≠, <, >, ≤ and ≥ in the selection predicate.
- Using the connectives and (∧), or (∨), and not (¬), we can combine several predicates into a larger predicate.
- Minimum number of tuples selected can be 0, Maximum selected tuples can be all.
- Degree (Result relation) = degree (parent relation), where degree refers to no. of attributes.
- 0 <= cardinality (result relation) <= cardinality (parent relation), where cardinality refers to no. of tuples.

Q Write a RELATIONAL ALGEBRA query to find all account_no where balance is less the 1000?

Q Write a RELATIONAL ALGEBRA query to find branch name which is situated in Delhi and having assets less than 1,00,000?

Q Write a RELATIONAL ALGEBRA query to find branch name and account_no which has balance greater than equal to 1,000 but less than equal to 10,000?

Q Consider the following schemas: **(NET-DEC-2013)**
Branch = (Branch-name, Assets, Branch-city)
Customer = (Customer-name, Bank name, Customer-city)
Borrow = (Branch-name, loan number, customer account-number)
Deposit = (Branch-name, Account- number, Customer-name, Balance)
Using relational Algebra, the Query that finds customers who have balance more than 10,000 is _____

(A) $\pi_{customer\text{-}name}(\sigma_{balance > 10000}(Deposit))$
(B) $\sigma_{customer\text{-}name}(\sigma_{balance > 10000}(Deposit))$
(C) $\pi_{customer\text{-}name}(\sigma_{balance > 10000}(Borrow))$
(D) $\sigma_{customer\text{-}name}(\pi_{balance > 10000}(Borrow))$

Ans: a

Q Which of the following query transformations (i.e., replacing the l.h.s. expression by the r.h.s. expression) is incorrect? R_1 and R_2 are relations. C_1, C_2 are selection conditions and A_1, A_2 are attributes of R_1. **(Gate-1998) (2 Marks)**

a) $\sigma_{C1}(\sigma_{C2} R_1)) \rightarrow \sigma_{C2}(\sigma_{C1}(R_1))$

b) $\sigma_{C1}(\pi_{A1} R_1)) \rightarrow \pi_{A1}(\sigma_{C1}(R_1))$

c) $\sigma_{C1}(R_1 \cup R_2) \rightarrow \sigma_{C1}(R_1) \cup \sigma_{C1}(R_2)$

d) $\pi_{A1}(\sigma_{C1}(R_1)) \rightarrow \sigma_{C1}(\pi_{A1}(R_1))$

Answer: (D)

D) if the selection condition is on attribute A2, then we cannot replace it by RHS as there will not be any attribute A2 due to projection of A1 only.

Q What is the optimized version of the relation algebra expression $\pi_{A1}(\pi_{A2}(\sigma_{F1}(\sigma_{F2}(r))))$, where A_1, A_2 are sets of attributes in r with $A_1 \subset A_2$ and F_1, F_2 are Boolean expressions based on the attributes in r? **(Gate-2014) (2 Marks)**

a) $\pi_{A1}(\sigma_{(F1 \wedge F2)}(r))$ **b)** $\pi_{A1}(\sigma_{(F1 \vee F2)}(r))$ **c)** $\pi_{A2}(\sigma_{(F1 \wedge F2)}(r))$ **d)** $\pi_{A2}(\sigma_{(F1 \vee F2)}(r))$

Ans: a

The Union Operation

- It is a binary operation, denoted, as in set theory, by ∪.
- Written as, $Expression_1 \cup Expression_2$, $r \cup s = \{t \mid t \in r \text{ or } t \in s\}$
- For a union operation r ∪ s to be valid, we require that two conditions hold:
- The relations r and s must be of the same arity. That is, they must have the same number of attributes, the domains of the ith attribute of r and the ith attribute of s must be the same, for all i.
- Mainly used to fetch data from different relations.

- **Some points to remember**
  - Deg (R ∪ S) = Deg(R) = Deg(S)
  - Max (IRI, ISI) <= IRUSI <= (IRI+ISI)

Q Write a RELATIONAL ALGEBRA query to find all the customer name who have a loan or an account or both?

The Set-Difference Operation

- The set-difference operation, denoted by −, allows us to find tuples that are in one relation but are not in another. It is a binary operator.
- The expression r − s produces a relation containing those tuples in r but not in s.
- For a set-difference operation r − s to be valid, we require that the relations r and s be of the same arity, and that the domains of the ith attribute of r and the ith attribute of s be the same, for all i.
- 0 <= IR - SI <= IRI

Q Write a RELATIONAL ALGEBRA query to find all the customer name who have a loan but do not have an account?

The Cartesian-Product Operation

- The Cartesian-product operation, denoted by a cross (×), allows us to combine information from any two relations.
- It is a binary operator; we write the Cartesian product of relations R_1 and R_2 as $R_1 \times R_2$.
- Cartesian-product operation associates every tuple of R_1 with every tuple of R_2.
    - R_1 X R_2 = {rs | r ∈ R_1 and s ∈ R_2}, contains one tuple <r, s> (concatenation of tuples r and s) for each pair of tuples r ϵ R_1, s ϵ R_2.
- R_1 X R_2 returns a relational instance whose schema contains all the fields of R_1 (in order as they appear in R_1) and all fields of R_2 (in order as they appear in R_2).
- If R_1 has m tuples and R_2 has n tuples the result will be having = m*n tuples.
- Same attribute name may appear in both R_1 and R_2, we need to devise a naming schema to distinguish between these attributes.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

R_1 * R_2			
A	R_1.B	R_2.B	C
1	P	Q	X
1	P	R	Y
1	P	S	Z
2	Q	Q	X
2	Q	R	Y
2	Q	S	Z
3	R	Q	X
3	R	R	Y
3	R	S	Z

Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with account balance, who have an account in the bank?

- To solve this query we understand that customer who have an account are available in depositor and balance is available in account, so to answer this query each tuple of the table account must be matched with each tuple with depositor, and they have a common attribute account_no if there is a match then that tuple is valid otherwise redundant, must be eliminated with the conditions.

Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with account balance, who have an account in the bank?

Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with loan amount, who have a loan in the bank?

Q Write a RELATIONAL ALGEBRA query to find all loan_no along with amount and branch_name, which is situated in Delhi?

Q Write a RELATIONAL ALGEBRA query to find the name of the customer who have an account in the branch situated in Delhi and balance greater than 1000?

The Rename Operation

- The results of relational algebra are also relations but without any name.
- The rename operation allows us to rename the output relation. It is denoted with small Greek letter **rho** ρ. Where the result of expression **E** is saved with name of **x.**
- $\rho_{x(A1, A2, A3, A4,.....AN)}(E)$

Q Write a RELATIONAL ALGEBRA query to find the account_no along with balance with 8% interest as total amount, with table name as balance sheet?

- even if an attribute name can be derived from the base relations, we may want to change the attribute name in the result.
- One reason to rename a relation is to replace a long relation name with a shortened version that is more convenient to use elsewhere in the query.

Q Write a RELATIONAL ALGEBRA query to find the loan_no with maximum loan amount?

- Another reason to rename a relation is a case where we wish to compare tuples in the same relation. We then need to take the Cartesian product of a relation with itself and, without renaming, it becomes impossible to distinguish one tuple from the other.
- A and b are used to rename a relation is referred to as table alias, correlation variable or tuple variable.

Additional Relational-Algebra Operations

- If we restrict ourselves to just the fundamental operations, certain common queries are lengthy to express. Therefore, we use additional operations.
  - These additional operations do not add any power to the algebra.
  - They are used to simplify the queries.

The Set-Intersection Operation

- We will be using ∩ symbol to denote set intersection.
- $r \cap s = r - (r - s)$
- Set intersection is not a fundamental operation and does not add any power to the relational algebra.
- $r \cap s = \{t \mid t \in r \text{ and } t \in s\}$
- $0 <= I\ R \cap S\ I <= \min(IRI, ISI)$

Q Write a RELATIONAL ALGEBRA query to find all the customer name who have both a loan and an account?

The Natural-Join Operation

- The natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.
- The natural join of r and s, denoted by r ⋈ s
- The natural-join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas and finally removes duplicate attributes.
- The natural join of r and s is a relation on schema R ∪ S formally defined as follows:

$$r \bowtie s = \Pi_{R \cup S} \left(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} (r \times s) \right)$$

Some Points to Remember

- The natural join is **associative** in nature.
- That is if we have a natural join such as:
- $(instructor \bowtie teaches) \bowtie course = instructor \bowtie (teaches \bowtie course)$

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 \bowtie R_2$		
A	B	C
2	Q	X
3	R	Y

Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with account balance, who have an account in the bank?

Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with loan amount, who have a loan in the bank?

Q Write a RELATIONAL ALGEBRA query to find all loan_no along with amount and branch_name, which is situated in Delhi?

Q Write a RELATIONAL ALGEBRA query to find the name of the customer who have an account in the branch situated in Delhi and balance greater than 1000?

Q Let r be a relation instance with schema R = (A, B, C, D). We define $r_1 = \Pi_{A, B, C}(r)$ and $r_2 = \Pi_{A.D}(r)$. Let $s = r_1 * r_2$ where * denotes natural join. Given that the decomposition of r into r_1 and r_2 is lossy, which one of the following is TRUE? **(Gate-2005) (1 Marks)**

(A) $s \subset r$ **(B)** $r \cup s$ **(C)** $r \subset s$ **(D)** $r * s = s$

Answer is **C** $r \subset s$.

r

A	B	C	D
1	2	3	3
1	5	3	4

r1

A	B	C
1	2	3
1	5	3

r2

A	D
1	3
1	4

s = r1 * r2

A	B	C	D
1	**2**	**3**	**3**
1	2	3	4
1	5	3	4
1	**5**	**3**	**4**

All the rows of r are in s (marked bold). So, $r \subset s$.

And one more result $r * s = r$.

Q Let r and s be two relations over the relation schemes R and S respectively, and let A be an attribute in R. The relational algebra expression $\sigma_{A=a}(r \bowtie s)$ is always equal to **(Gate-2001) (1 Marks)**

a) $\sigma_{A=a}(r)$ **b)** r **c)** $\sigma_{A=a}(r) \bowtie s$ **d)** None of the above

Answer is C.

C is just the better form of query, more execution friendly because requires less memory while joining. query, given in question takes more time and memory while joining.

Q Consider the relations R(A, B) and S(B, C) and the following four relational algebra queries over R and S :

I. $\Pi_{A, B}(R \bowtie S)$

II. $R \bowtie \Pi_B(S)$

III. $R \cap (\Pi_A(R) \times \Pi_B(S))$

IV. $\Pi_{A, R.B}(R \times S)$

where R·B refers to the column B in table R. One can determine that: (**NET-JULY-2016**)

(1) I, III and IV are the same query. **(2)** II, III and IV are the same query.

(3) I, II and IV are the same query. **(4)** I, II and III are the same query

Ans. 4

I think answer is **4**

Let us take an example as R= {(10,1),(20,2)} and S={(2,30),(3,40)}

I) $R \rhd\lhd S = \{(20, 2,30)\}$. So $\pi_{A, B}\ R \rhd\lhd S = \{(20,2)\}$

II) $\pi_B(S) = \{(2),(3)\}$. So $R \rhd\lhd \pi_B(S) = \{(20,2)\}$

III) $\pi_A(R) \times \pi_B(S) = \{10,20\} \times \{2,3\} = \{(10,2),(10,3),(20,2),(20,3)\}$

$R \cap \pi_A(R) \times \pi_B(S) = \{(20,2)\}$

IV) $R \times S = \{(10,1,2,30),(10,1,3,40),(20,2,2,30),(20,2,3,40)\}$

$\pi_{A, R.B}\ R \times S = \{(10,2),(10,3),(20,2),(20,3)\}$

Q Let R and S be two relations with the following schema **(Gate-2008) (2 Marks)**

R (<u>P, Q</u>, R1, R2, R3) S (<u>P, Q</u>, S1, S2)

where {P, Q}is the key for both schemas. Which of the following queries are equivalent?

i) $\Pi_P(R \bowtie S)$ **ii)** $\Pi_P(R) \bowtie \Pi_P(S)$

iii) $\Pi_P(\Pi_{P, Q}(R) \cap \Pi_{P, Q}(S))$ **iv)** $\Pi_P(\Pi_{P, Q}(R) - (\Pi_{P, Q}(R) - \Pi_{P, Q}(S)))$

a) Only I and II **b)** Only I and III

c) Only I, II and III **d)** Only I, III and IV

Ans: d

(d) i, iii, iv

iv) is the expansion for natural join represented with other operators.

Why ii is not equivalent? Consider the following instances of R and S

$R : \{\langle \text{“1 ”}, \text{“}abc\text{ ”}, \text{“}p1\text{ ”}, \text{“}p2\text{ ”}, \text{“}p3\text{ ”}\rangle ,$
$\langle \text{“2 ”}, \text{“}xyz\text{ ”}, \text{“}p1\text{ ”}, \text{“}p2\text{ ”}, \text{“}p3\text{ ”}\rangle\}$

$S : \{\langle \text{“1 ”}, \text{“}abc\text{ ”}, \text{“}q1\text{ ”}, \text{“}q2\text{ ”}\rangle$
$\langle \text{“2 ”}, \text{“}def\text{ ”}, \text{“}q1\text{ ”}, \text{“}q2\text{ ”}\rangle\}$

Now, consider the given queries:

i. $R \bowtie S$ gives

$\{\langle \text{“1 ”}, \text{“}abc\text{ ”}, \text{“}p1\text{ ”}, \text{“}p2\text{ ”}, \text{“}p3\text{ ”}, \text{“}q1\text{ ”}, \text{“}q2\text{ ”}\rangle\}$

Projecting P gives $\{\langle \text{“1 ”}\rangle\}$

ii. $\pi_P (R) \bowtie \pi_P (S)$ gives

$\{\langle \text{“1 ”}\rangle \langle \text{“2 ”}\rangle\} \bowtie \{\langle \text{“1 ”}\rangle \langle \text{“2 ”}\rangle\}$

$= \{\langle \text{“1 ”}, \text{“2 ”}\rangle\}$

iii. $\Pi_P (\Pi_{P,Q} (R) \cap \Pi_{P,Q} (S))$ gives

$\{\langle \text{“1 ”}, \text{“}abc\text{ ”}\rangle , \langle \text{“2 ”}, \text{“}xyz\text{ ”}\rangle\} \cap \{\langle \text{“1 ”}, \text{“}abc\text{ ”}\rangle , \langle \text{“2 ”}, \text{“}def\text{ ”}\rangle\}$
$= \{\langle \text{“1 ”}, \text{“}abc\text{ ”}\rangle\}$

Projecting P gives $\{\langle \text{“1 ”}\rangle\}$

iv. $\Pi_P (\Pi_{P,Q} (R) - (\Pi_{P,Q} (R) - \Pi_{P,Q} (S)))$ gives

$\{\langle \text{“1 ”}, \text{“}abc\text{ ”}\rangle , \langle \text{“2 ”}, \text{“}xyz\text{ ”}\rangle\}$
$- (\{\langle \text{“1 ”}, \text{“}abc\text{ ”}\rangle , \langle \text{“2 ”}, \text{“}xyz\text{ ”}\rangle\} - \{\langle \text{“1 ”}, \text{“}abc\text{ ”}\rangle , \langle \text{“2 ”}, \text{“}def\text{ ”}\rangle\})$

$= \{\langle \text{“1 ”}, \text{“}abc\text{ ”}\rangle , \langle \text{“2 ”}, \text{“}xyz\text{ ”}\rangle\}$
$- \{\langle \text{“2 ”}, \text{“}xyz\text{ ”}\rangle\}$
$= \{\langle \text{“1 ”}, \text{“}abc\text{ ”}\rangle\}$

Projecting P gives $\{\langle \text{“1 ”}\rangle\}$

Q Consider the join of a relation R with a relation S. If K has *m tuples* and S has *n tuples*, then the maximum and minimum sizes of the join respectively are: **(Gate-1999) (1 Marks)**

(A) m+n and 0 **(B)** mn and 0 **(C)** m+n and m-n **(D)** mn and m+n

Answer: (B)

Answer is **B**.

mn
Case 1: if there is a common attribute between R and S, and every row of r matches with the each row of s- i.e., it means, the join attribute has the same value in all the rows of both r and s,
Case 2: If there is no common attribute between R and S.

0 There is a common attribute between R and S and nothing matches- the join attribute in r and s have no common value.

Q The following functional dependencies hold for relations R(A, B, C) and S(B, D, E).
B → A
A → C
The relation R contains 200 tuples and the relation S contains 100 tuples. What is the maximum number of tuples possible in the natural join R ⋈ S? (Gate-2010) (2 Marks)
a) 100 **b)** 200 **c)** 300 **d)** 2000
Ans: a

(A) 100.

Natural join will combine tuples with same value of the common rows(if there are two common rows then both vaues must be equal to get into the resultant set). So by this defn: we can get at the max only 100 common value.

Q (NET-DEC-2015)

Consider the following three tables R, S and T. In this question, all the join operations are natural joins ($\bowtie$). (π) is the projection operation of a relation :

R	
A	B
1	2
3	2
5	6
7	8
9	8

S	
B	C
6	2
2	4
8	1
8	3
2	5

T	
A	C
7	1
1	2
9	3
5	4
3	5

Possible answer tables for this question are also given as below :

A	B	C
1	2	4
1	2	5
3	2	4
3	2	5
5	6	2
7	8	1
7	8	3
9	8	1
9	8	3

(a)

A	B	C
1	2	2
3	2	5
5	6	4
7	8	1
9	8	3

(b)

A	B	C
1	6	2
3	2	5
5	2	4
7	8	1
9	8	3

(c)

A	B	C
3	2	5
7	8	1
9	8	3

(d)

What is the resulting table of $\pi_{A,B}(R \bowtie T) \bowtie \pi_{B,C}(S \bowtie T)$?

(1) (a) (2) (b) (3) (c) (4) (d)

Theta join / Conditional Join

- The theta join/ Conditional join operation is a variant of the natural-join operation that allows us to combine a selection and a Cartesian product into a single operation.
- The theta join operation $r \bowtie_\theta s$ is defined as follows: $r \bowtie_\theta s = \sigma_\theta(r \times s)$

Q Let R1 (A, B, C) and R2 (D, E) be two relation schema, where the primary keys are shown underlined, and let C be a foreign key in R1 referring to R2. Suppose there is no violation of the above referential integrity constraint in the corresponding relation instances r1 and r2. Which one of the following relational algebra expressions would necessarily produce an empty relation? **(Gate-2004) (1 Marks)**

a) $\Pi_D(r_2) - \Pi_C(r_1)$ **b)** $\Pi_C(r_1) - \Pi_D(r_2)$ **c)** $\Pi_D(r_1 \bowtie_{C \neq D} r_2)$ **d)** $\Pi_C(r_1 \bowtie_{C=D} r_2)$

Answer: (B)

C in R1 is a foreign key referring to the primary key D in R2. So, every element of C must come from some D element.

Q Consider the following relations A, B and C:

A

ID	Name	Age
12	Arun	60
15	Shreya	24
99	Rohit	11

B

ID	Name	Age
15	Shreya	24
25	Hari	40
98	Rohit	20
99	Rohit	11

C

ID	Phone	Area
10	2200	02
99	2100	01

How many tuples does the result of the following relational algebra expression contain? Assume that the schema of AUB is the same as that of A. **(Gate-2007) (2 Marks)**

$(A \cup B) \bowtie_{A.Id > 40 \vee C.Id < 15} C$

a) 7 **b)** 4 **c)** 5 **d)** 9

Q Suppose database table $T_1(P, R)$ currently has tuples {(10, 5), (15, 8), (25, 6)} and table $T_2(A, C)$ currently has {(10, 6), (25, 3), (10, 5)}. Consider the following three relational algebra queries RA_1, RA_2 and RA_3: **(NET-AUG-2016)**

RA_1: $T_1 \bowtie_{T1.P = T2.A} T_2$ where $\bowtie$ is natural join symbol

RA_2: $T_1 =\bowtie_{T1.P = T2.A} T_2$ where $=\bowtie$ is left outer join symbol

RA_3: $T_1 \bowtie_{T1.P = T2.A \text{ and } T1.R = T2.C} T_2$

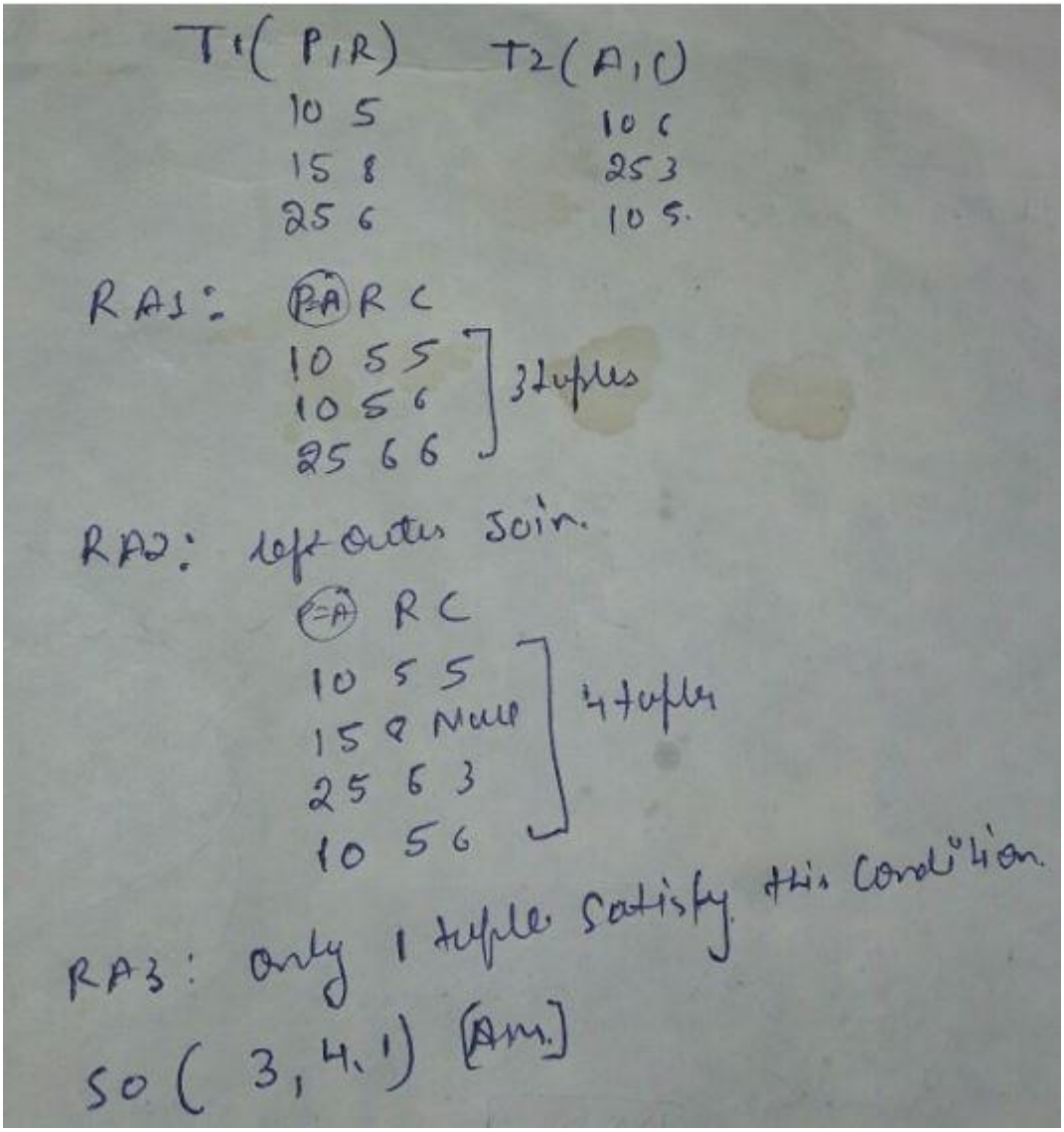
The number of tuples in the resulting table of RA_1, RA_2 and RA_3 are given by:

(1) 2, 4, 2 respectively **(2)** 2, 3, 2 respectively

(3) 3, 3, 1 respectively

(4) 3, 4, 1 respectively

Ans: 4



Outer join Operations

- The outer-join operation is an extension of the join operation to deal with missing information.
- The outer join operation works in a manner similar to the natural join operation, but preserves those tuples that would be lost in a join by creating tuples in the result containing null values.
- We can use the outer-join operation to avoid this loss of information.
- There are actually three forms of the operation: left outer join, denoted $\supset\!\bowtie$; right outer join, denoted $\bowtie\!\subset$; and full outer join, denoted $\supset\!\bowtie\!\subset$.

Left Outer Join

- The left outer join (⟕) takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 ⟕ R_2$		
A	B	C
1	P	NULL
2	Q	X
3	R	Y

Q Consider the relations r(A, B) and s(B, C), where s.B is a primary key and r.B is a foreign key referencing s.B. Consider the query

Q: $r ⋈ (\sigma_{B<5}(s))$

Let LOJ denote the natural left outer-join operation. Assume that r and s contain no null values. Which of the following is NOT equivalent to Q? **(Gate-2018) (2 Marks)**

a) $\sigma_{B<5}(r ⋈ s)$

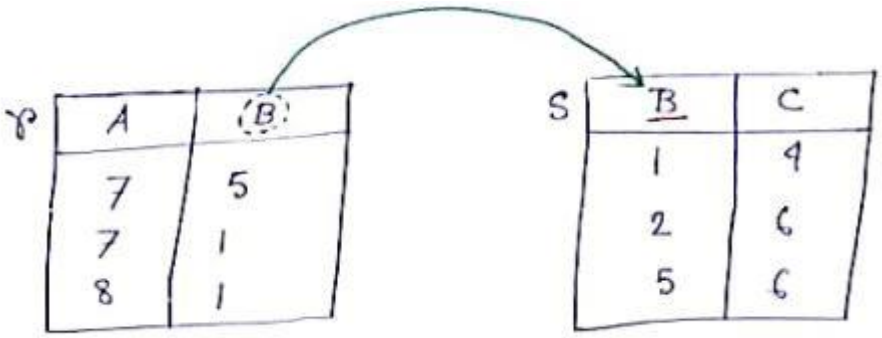
b) $\sigma_{B<5}(r$ LOJ $s)$

c) r LOJ $(\sigma_{B<5}(s))$

d) $\sigma_{B<5}(r)$ LOJ s

Option a, b, d will restrict all record with B<5 but option C will include record with $b >= 5$ also, so false.

C is answer.


R
A
B
7
5
7
1
8
1
S
B
C
1
4
2
6
5
6

Right Outer Join

- The right outer join (⟕) is symmetric with the left outer join: It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 ⟕ R_2$		
A	B	C
2	Q	X
3	R	Y
NULL	S	Z

Full Outer Join

- The full outer join(⟗) does both the left and right outer join operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 ⟗ R_2$		
A	B	C
1	P	NULL
2	Q	X
3	R	Y
NULL	S	Z

Q Consider two relations R1(A, B) with the tuples (1, 5), (3, 7) and R1(A, C) = (1, 7), (4, 9). Assume that R(A, B, C) is the full natural outer join of R1 and R2. Consider the following tuples of the form (A, B, C)

a = (1, 5, null),

b = (1, null, 7),

c = (3, null, 9),

d = (4, 7, null),

e = (1, 5, 7),

f = (3, 7, null),

g = (4, null, 9).

Which one of the following statements is correct? **(Gate-2015) (1 Marks)**

(A) R contains a, b, e, f, g but not c, d

(B) R contains a, b, c, d, e, f, g

(C) R contains e, f, g but not a, b

(D) R contains e but not f, g

Answer: (C)

Q Consider the following 2 tables

R_1

A	B	C
1	2	3
1	2	4
2	1	3
3	1	3

R_2

A	B	D
1	2	1
2	1	5
4	2	1
3	2	1

The number of rows where null entries are present in the table $R_1 \bowtie R_2$ $(R_1$ natural full outer join $R_2)$ is _____

Ans: 3

DIVISION

- Notation – Division is also a binary operator denoted like A ÷ B
- *A* ÷ *B* as the set of all *x* values (in the form of unary tuples) such that for *every y* value in (a tuple of) *B*, there is a tuple *(x, y)* in *A*.
- For each *x* value in (the first column of) *A*, consider the set of *y* values that appear in (the second field of) tuples of *A* with that *x* value. If this set contains (all *y* values in) *B*, the *x* value is in the result of *A* ÷ *B*.
- Division operator is derived using- ***projection, cartisan product, set difference as***

$$A \div B = \prod_x(A) - \prod_x((\prod_x(A \times B) - A))$$

Q Consider the given table R and S find the number of elements retrieved by the query

Table R

A	B
1	Dog
1	Cat
1	Cow
2	Cat
4	Cat
3	Dog
4	Dog
2	Dog
4	Cow

Table S

B
Cat
Dog

$\Pi_{A,B}(R) \div \Pi_B(S)$_____

Q Consider a database that has the relation schema CR (StudentName, CourseName). An instance of the schema CR is as given below.

StudentName	CourseName
SA	CA
SA	CB
SA	CC
SB	CB
SB	CC
SC	CA
SC	CB
SC	CC
SD	CA
SD	CB
SD	CC
SD	CD
SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

The following query is made on the database.

$T_1 \leftarrow \pi_{\text{CourseName}} (\sigma_{\text{StudentName=SA}}(CR))$

$T_2 \leftarrow CR \div T_1$

The number of rows in T2 is _____ . **(Gate-2017) (1 Marks)**

ANS) 4

T1 WILL GIVE :-

1. CA
2. CB
3. CC

T2 = CR ÷ T1 = All the tuples in **CR** which are matched with every tuple in T1 :

1. SA
2. SC
3. SD
4. SF

//**SB** IS NOT MATCHED WITH **CA**, **SE** IS NOT MATCHED WITH **CC**

Q Information about a collection of students is given by the relation *studInfo(studId, name, sex)*. The relation *enroll (studId, courseId)* gives which student has enrolled for (or taken) that course(s). Assume that every course is taken by at least one male and at least one female student. What does the following relational algebra expression represent?(Gate-2007) (2 Marks)

$\pi_{courceId}((\pi_{studId}(\sigma_{sex="female"}(studInfo)) \times \pi_{courseId}(enroll)) - enroll)$

(A) Courses in which all the female students are enrolled.
(B) Courses in which a proper subset of female students are enrolled.
(C) Courses in which only male students are enrolled.
(D) None of the above
Answer: (B)

STUDENTINFO

1	*A*	*M*
2	*A*	*F*
3	*A*	*F*

ENROLL

1	*C*1
1	*C*2
2	*C*1
2	*C*2
3	*C*2

Q Consider the relational schema given below, where eId of the relation dependent is a foreign key referring to empId of the relation employee. Assume that every employee has at least one associated dependent in the dependent relation. (Gate-2014) (2 Marks)
employee (empId, empName, empAge)
dependent(depId, eId, depName, depAge)

Consider the following relational algebra query:

$$\Pi_{empId}(employee) - \Pi_{empId}(employee \bowtie_{(empId=eID) \wedge (empAge \leq depAge)} dependent)$$

The above query evaluates to the set of *empIds* of employees whose age is greater than that of
(A) some dependent.
(B) all dependents.
(C) some of his/her dependents
(D) all of his/her dependents.
Answer: (D)
(D) all of his/her dependents. The inner query selects the employees whose age is less than or

equal to at least one of his dependents. So, subtracting from the set of employees, gives employees whose age is greater than all of his dependents.

Q Consider the relation Student (name, sex, marks), where the primary key is shown underlined, pertaining to students in a class that has at least one boy and one girl. What does the following relational algebra expression produce? (Note: ρ is the rename operator). **(Gate-2004) (2 Marks)**

$\pi_{name}\{\sigma_{sex=female}(Student)\} - \pi name(Student \bowtie_{(sex=female \wedge x=male \wedge marks \leq m)} \rho_{n,x,m}(Student))$

a) names of girl students with the highest marks
b) names of girl students with more marks than some boy student
c) names of girl students with marks not less than some boy student
d) names of girl students with more marks than all the boy students
ans: d

Name	Sex	Marks
S1	F	30
S2	F	10
S3	M	20

Name	Sex	Marks	
S1	M	100	> highest marks of M student
S2	F	50	> highest marks of F student
S3	M	40	
S4	F	30	

Answer is D.

(A) :-> This is simple select query query.

(B) :-> This is simple query we need to check X=Y in where clause.

(C) :-> Cycle < 3. Means cycle of length 1 & 2. Cycle of length 1 is easy., same as self loop. Cycle of length 2 is also not too hard to compute. Though it'll be little complex, will need to do like (X, Y) & (Y, X) both present & $X! = Y$,. We can do this with constant RA query.

(D) :-> This is most hard part. Here we need to find closure of vertices. This willl need kind of loop. If the graph is like skewed tree, our query must loop for $O(N)$ times. We can't do with constant length query here.

Answer :-> D

Q With respect to relational algebra, which of the following operations are included from mathematical set theory? **(NET-JUNE-2019)**

1) Join **2)** Intersection **3)** Cartesian Product **4)** Project

a) 1 and 4 **b)** 2 and 3 **c)** 3 and 4 **d)** 2 and 4

Ans: b

Q Given the relations **(Gate-2000) (2 Marks)**
employee (name, salary, dept-no), and
department (dept-no, dept-name, address),
Which of the following queries cannot be expressed using the basic relational algebra operations (σ, π, ×, ⋈, ∪, ∩, −)?

a) Department address of every employee

b) Employees whose name is the same as their department name

c) The sum of all employees' salaries

d) All employees of a given department

Ans: c

Possible solutions, relational algebra:

(a) Join relation using attribute dpart_no.

- $\Pi_{\text{address}}(\text{emp} \bowtie \text{depart})$ OR
- $\Pi_{\text{address}}(\sigma_{\text{emp.depart\_no.}=\text{depart.depart\_no.}}(\text{emp} \times \text{depart}))$

(b)

- $\Pi_{\text{name}}(\sigma_{\text{emp.depart\_no.}=\text{depart.depart\_no.} \wedge \text{emp.name}=\text{depart.depart\_name}}(\text{emp} \times \text{depart}))$ OR
- $\Pi_{\text{name}}(\text{emp} \bowtie_{\text{emp.name}=\text{depart.depart\_name}} \text{depart})$

(d) Let the given department number be x

- $\Pi_{\text{name}}(\sigma_{\text{emp.depart\_no.}=\text{depart.depart\_no.} \wedge \text{depart\_no.}=x}(\text{emp} \times \text{depart}))$ OR
- $\Pi_{\text{name}}(\text{emp} \bowtie_{\text{depart\_no.}=x} \text{depart})$

(c) We cannot generate relational algebra of aggregate functions using basic operations. We need extended operations here.

Option **(c)**.

Q Consider a selection of the form $\sigma_{A \leq 100}$ (r), where r is a relation with 1000 tuples. Assume that the attribute values for A among the tuples are uniformly distributed in the interval [0,500]. Which one of the following options is the best estimate of the number of tuples returned by the given selection query? **(Gate-2007) (2 Marks)**

a) 50 **b)** 100 **c)** 150 **d)** 200

$\sigma_{A \leq 100}(r)$
r has 1000 tuples

Values for A among the tuples are uniformly distributed in the interval [0, 500]. This can be split to 5 mutually exclusive (non-overlapping) and exhaustive (no other intervals) intervals of same width of 100 ($[0 - 100], [101 - 200], [201 - 300], [301 - 400], [401 - 500]$, 0 makes the first interval larger - this must be a typo in question) and we can assume all of them have same number of values due to Uniform distribution. So, number of tuples with A value in first interval should be

$$\frac{\text{Total no. of tuples}}{5} = 1000/5 = 200$$

Correct Answer: D

Q Consider a relational table r with sufficient number of records, having attributes A_1, A_2..., A_n and let $1 \leq p \leq n$. Two queries Q_1 and Q_2 are given below. **(Gate-2011) (2 Marks)**

Q_1: $\pi_{A1,...,Ap}(\sigma_{Ap=c}(r))$ where cc is a constant

Q_2: $\pi_{A1,...,Ap}(\sigma_{c1 \leq Ap \leq c2}(r))$ where c1 and c2 are constants.

The database can be configured to do ordered indexing on Ap or hashing on Ap. Which of the following statements is TRUE?

a) Ordered indexing will always outperform hashing for both queries

b) Hashing will always outperform ordered indexing for both queries

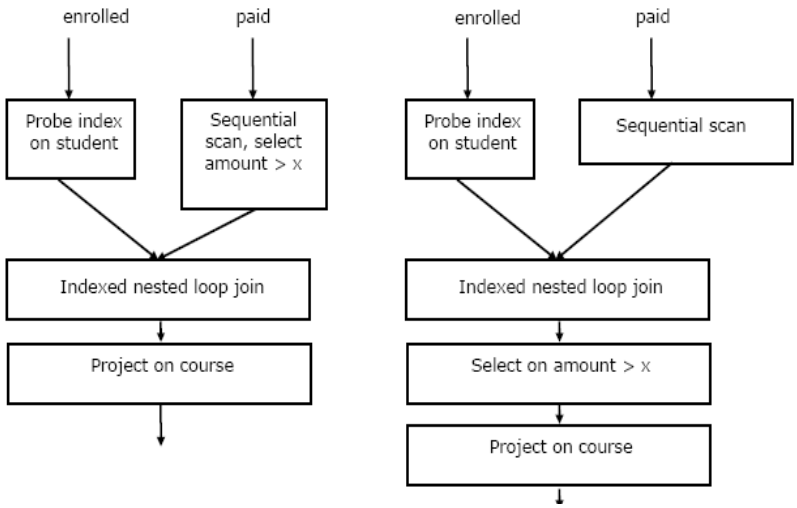
c) Hashing will outperform ordered indexing on Q_1, but not on Q_2

d) Hashing will outperform ordered indexing on Q_2, but not on Q_1

Ans: c

(C) Hashing works well on the 'equal' queries, while ordered indexing works well better on range queries too. For ex consider B+ Tree, once you have searched a key in B+ tree , you can find range of values via the block pointers pointing to another block of values on the leaf node level.

Q Consider the relation enrolled (student, course) in which (student, course) is the primary key, and the relation paid (student, amount), where student is the primary key. Assume no null values and no foreign keys or integrity constraints. Assume that amounts 6000, 7000, 8000, 9000 and 10000 were each paid by 20% of the students. Consider these query plans (Plan 1 on left, Plan 2 on right) to "list all courses taken by students who have paid more than x". **(Gate-2006) (2 Marks)**



A disk seek takes 4ms, disk data transfer bandwidth is 300 MB/s and checking a tuple to see if amount is greater than x takes 10 micro-seconds. Which of the following statements is correct?

(A) Plan 1 and Plan 2 will not output identical row sets for all databases.

(B) A course may be listed more than once in the output of Plan 1 for some databases

(C) For x = 5000, Plan 1 executes faster than Plan 2 for all databases.

(D) For x = 9000, Plan I executes slower than Plan 2 for all databases.

Answer: (C)

I think it should be **C)**

In all cases plan 1 is faster than plan 2 cause in plan 1 we are reducing the load by doing select amount $> x$ and then the loop

But, in case of plan 2 its in the nested loop so it need to check every time and will take more time to execute .

Q Consider a join (relation algebra) between relations r(R)and s(S) using the nested loop method. There are 3 buffers each of size equal to disk block size, out of which one buffer is reserved for intermediate results. Assuming size(r(R)) < size(s(S)), the join will have fewer number of disk block accesses if **(Gate-2014) (2 Marks)**

a) relation r(R) is in the outer loop.

b) relation s(S) is in the outer loop.

c) join selection factor between r(R) and s(S) is more than 0.5.

d) join selection factor between r(R) and s(S) is less than 0.5.

Ans: a

Q Suppose the adjacency relation of vertices in a graph is represented in a table Adj (X, Y). Which of the following queries cannot be expressed by a relational algebra expression of constant length?**(Gate-2001) (1 Marks)**

(A) List of all vertices adjacent to a given vertex

(B) List all vertices which have self-loops

(C) List all vertices which belong to cycles of less than three vertices

(D) List all vertices reachable from a given vertex

Answer is D.

(A) :-> This is simple select query query.

(B) :-> This is simple query we need to check X=Y in where clause.

(C) :-> Cycle < 3 . Means cycle of length 1 & 2. Cycle of length 1 is easy., same as self loop. Cycle of length 2 is also not too hard to compute. Though it'll be little complex, will need to do like (X, Y) & (Y, X) both present & $X! = Y$,. We can do this with constant RA query.

(D) :-> This is most hard part. Here we need to find closure of vertices. This willl need kind of loop. If the graph is like skewed tree, our query must loop for $O(N)$ times. We can't do with constant length query here.

Answer :-> D