DISCRETE MATHEMATICS STRUCTURE

IV C.S.

SET THEORY, RELATION AND FUNCTION 1

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 *Find the minimum number of students in a school to be sure that 5 of them are born in the same month.* *[R.T.U. 2019]*

Ans. Use pigeonhole principle, first find boxes and objects. Suppose that for each month, we have a box that contains persons who was born in that month. The number of boxes is 12 and Let the number of objects is 60. By the generalized pigeonhole principle, at least one of these boxes contains at least 60/12= 5 persons. So, there must be at least 5 persons who were born in the same month.

Q.2 *Find the generating function for the sequence* $\{1, 1, 0, 0, 1, 1, 1, \ldots \infty\}$ *[R.T.U. 2019]*

Ans. $1 + 1.x + 0.x^2 + 0.x^3 + 1.x^4 + 1.x^5 + 1.x^6 + \ldots + x^9$

$1 + x + x^4 (1 + x + x^2) + \ldots$

$1 + x + x^4 (1 - x)^{-1}$

$$1 + x + \frac{x^4}{1-x}$$

Q.3 *Find the domain of the following function :*

$$f(x) = \sqrt{\log\left(\frac{5x - x^2}{4}\right)}$$ *[R.T.U. 2019]*

Ans. The function f(x) will defined

$$\log\left(\frac{5x - x^2}{4}\right) \geq 0$$

$$\frac{5x - x^2}{4} \geq e^0$$

$$\frac{5x - x^2}{4} \geq 1$$

$5x - x^2 \geq 4$

$x^2 - 5x + 4 < 0$

$(x-1)(x-4) \leq 0$

$1 \leq x \leq 4$

Thus, domain of $f(x) = [1, 4]$

Q.4 *Prove that for any two sets A and B :* $A - (A \cap B) = A - B$ *[R.T.U. 2019]*

Ans. Let $x \in A - (A \cap B)$

$\Rightarrow x \in A$ or $x \notin (A \cap B)$

$\Rightarrow x \in A$ or $\{x \notin A$ or $x \notin B\}$

$\Rightarrow x \in A$ or $x \notin A$ or $x \in A$ or $x \notin B$

$\Rightarrow x \in \phi$ or $x \in A - B$

$\Rightarrow x \in A - B$

$A - (A \cap B) \leq A - B$ ...(1)

Let $x \in A - B$

$\Rightarrow x \in \phi$ or $x \in A - B$

$x + A$ or $x \notin A$ or $x \in A$ or $x \notin B$

$x \in A$ or $x \notin A \cap B$

$x \in A - (A \cap B)$

$A - B \leq A - (A \cap B)$ ...(2)

From equation (1) and (2)

$A - (A \cap B) = A - B$

Q.5 *Define the Cross partition of a set.* *[R.T.U. 2017]*

Ans. Cross Partition of a set : Cross Partition of a set is defined on two partitions of the set as shown below

Let $[A_1, A_2 \ldots A_m]$ and $[B_1, B_2 \ldots B_n]$ be partitions of S. Then, the collection of sets $[A_i \cap B_j,\ i = 1, 1 \ldots m,\ j = 1, 1 \ldots n]$ is called the cross partition.

Empty set is not included in the cross partition.

Example : Let set = {1, 2, 3..., 8, 9}

We will define cross partition on set with partitions.

$A_1 \quad A_2$

$P_1 = \{(1, 3, 5, 7, 9), (2, 4, 6, 8)\} \rightarrow A$

$P_2 = \{(1, 2, 3, 4), (5, 7), (6, 8, 9)\} \rightarrow B$

$B_1 \quad B_2 \quad B_3$

$A_1 \cap B_1 = \{1, 3\} \qquad A_2 \cap B_1 = \{2, 4\}$

$A_1 \cap B_2 = \{5, 7\} \qquad A_2 \cap B_2 = \{5\}$

$A_1 \cap B_3 = \{9\} \qquad A_2 \cap B_3 = \{6, 8\}$

Cross Partition is $\{\{1, 3\}, \{5, 7\}, \{9\}, \{2, 4\}, \{5\}, \{6, 8\}\}$

Q.6 *Define the Duality.* *[R.T.U. 2017]*

Ans. Duality : Suppose that A be a statement dealing with the equality of sets expression. Every expression may, possibly involve one or more occurence of sets and their compliments, void set ϕ and universal set u and only the set operations union ($\cup$) and intersection ($\cap$). The dual of A, represented by A^d, is derived from A by replacing.

1. Every occurence of ϕ and u by u and ϕ respectively in A.
2. Every occurence of u and $\cap$ by $\cap$ and $\cup$ respectively in A.

Let $A = (u \cap s) \cup (T \cap s) = S$

Then $A^d = (\phi \cup s) \cap (T \cup s) = S$

Q.7 *Define the Floor function or greatest integer function.* *[R.T.U. 2017]*

Ans. Floor Functions : The floor is mathematical functions which convert arbitrary real numbers to close integers.

The floor function of a real number x, denoted by $\lfloor x \rfloor$ or floor (x), is a function that returns the highest integer less than or equal to x. Formally, for all real numbers x,

$$\lfloor x \rfloor = \max\{n \in Z : n \leq x\}$$

For example,

Floor (2.9) = 2, floor (–2) = –2 and floor (–2.3) = –3.

For non-negative *x*, a more traditional name for floor(x) is the *integral part* or *integral* value of *x*. The function x–[x], also written as *x* mod 1 or {x}, is called the *fractional part of x*. Every fraction *x* can be written as a mixed number, the sum of an integer and a proper fraction. The floor function and fractional part functions extend this decomposition to all real values.

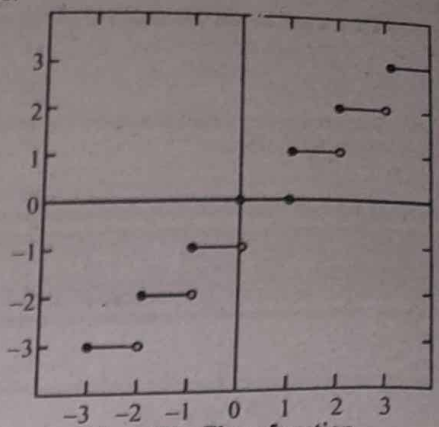


Fig. : The Floor function

Q.8 *Define the Bijection.* *[R.T.U. 2017]*

Ans. Bijection : A function that is both an injection and a surjection. In a bijection, each member of the range corresponds to an element of the domain i.e. mapped onto it and there is one-to-one correspondence between the members of the domain and the range.

Q.9 *Define countable and uncountable sets with example.* *[R.T.U. 2016]*

Ans. Uncountable Sets : An uncountable set is an infinite set that contains too many elements to be counted. The best known example of countable set is the set R of all real numbers.

Countable Sets : A countable set is either a finite set of natural numbers. The elements of a countable set can be counted. An example of countable set is as follows

$A = \{1,2,3,4\}$

Q.10 *Define mod functions and div functions with example.* *[R.T.U. 2016]*

Ans. Mod functions and Div functions : The mod function f(x, y) = x mod y denotes the remainder when an integer x is divided by a positive integer y.

Div function

The **div function** g(x, y) = *x* div *y* denotes the quotient when *x* is divided by *y*. Programming languages often provide two such built-in operators, **mod** and **div;** in C++, the mod operator is denoted by the percent symbol %, and the div operator by the forward slash '/'.

For example, 23 mod 5 = 3, 18 mod 6 = 0, 23 div 5 = 4, and 5 div 6 = 0.

The mod function can determine the day of the week in n days from a given day.

Q.11 ***Define the proof by contradiction with example.*** *[R.T.U. 2016, 2013]*

Ans. Proof by contradiction : If in same case we have that

$\sim p \rightarrow q$, is true

and also

$\sim p \rightarrow q$, is false ... (A)

But these are contradictory. As (A) and (B) are in contradiction one of them is false. ... (B)

Ex. : Proove by contradictory that if $x + y > 15$ then either $x > 10$ or $y > 5$

Sol. We assume the hypothesis $x + y > 15$. From here we must conclude that $x > 10$ or $y > 5$.

Assume to the contrary that

$x > 10$ or $y > 5$, is false

so $x \leq 10$ and $y \leq 5$

Adding both inequalities we get

$x + y \leq 10 + 5 = 15$

which contradicts the hypothesis

$x + y > 15$

From here we conclude that the assumption "$x \leq 10$ and $y \leq 5$" cannot be true,

So "$x > 10$" or "$y > 5$" must be true.

Q.12 ***Let $f: R \rightarrow R$ be a function defined as $f(X) = 3X + 5$ and $g: R \rightarrow R$ be another function defined as $g(X) = X+4$. Find $(gof)^{-1}$ and $f^{-1}og^{-1}$ and verify $(gof)^{-1} = f^{-1}og^{-1}$*** *[R.T.U. 2015]*

Ans. Given that,

$f(x) = 3x + 5$, and so, $x = (f - 5)/3$, i.e., $f^{-1} = (x - 5)/3$

$g(x) = x + 4$, and so, $x = (g - 4)$, i.e., $g^{-1} = x - 4$

Now,

$gof = g(f(x))$

$= g(3x + 5)$

$= (3x + 5) + 4$

$= 3x + 9$

Now, $x = ((gof) - 9)/3$

So, $(gof)^{-1} = (x - 9)/3$

Now,

$f^{-1}og^{-1} = f^{-1}(g^{-1}(x))$

$= f^{-1}(x - 4)$

$= ((x - 4) - 5)/3$

$= (x - 9)/3$

Hence, we see that $(gof)^{-1} = f^{-1}og^{-1}$

Q.13 ***Define the ceiling function with example.*** *[R.T.U. 2015]*

Ans. Ceiling Functions : The closely-related ceiling function, denoted by $\lceil x \rceil$ or ceil(x) or ceiling (x), is the function that returns the smallest integer nor less than x or formally,

$$\lceil x \rceil = \min \{n \in Z : x \leq n\}$$

For example, ceiling (2.3) = 3, ceiling (2) = 2 and ceiling (−2.3) = −2.

The names "floor" and "ceiling" and the corresponding notations were introduced by Kenneth E. Iverson in 1962.

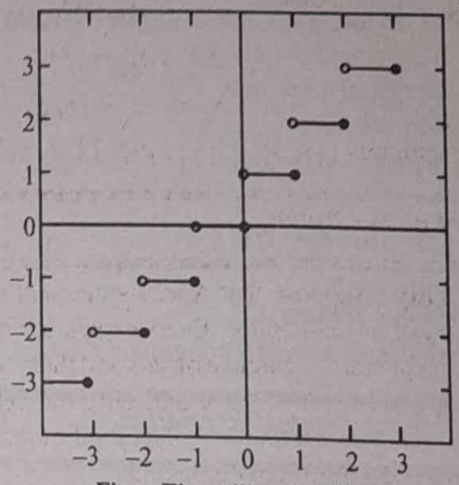


Fig. : The ceiling function

Q.14 ***Define the remainder function with example.*** *[R.T.U. 2015]*

Ans. Remainder Function : A remainder function (generally denoted by 'mod' or symbol '%') is one which given two numbers say, a,b returns the remainder when a is divided by b, i.e. "a mod b" is the remainder when a is divided by b. e.g.,

5 mod 2 = 1 (remainder when 5 is divided by 2)

11 mod 3 = 2 (remainder when 11 is divided by 3)

Q.15 ***Define the Reflexive relation.*** *[R.T.U. 2013]*

Ans. Reflexive Relation : A relation R on a non-void set A is known as reflexive relation if each member of A is R–related to itself, i.e. $x \, R \, x$ or $(x, x) \in R$, $\forall \, x \in A$.

A relation R on a set A is irreflexive if $(x,x) \notin R$, $\forall \, x \in A$.

Example 1. Let A be the set of all straight lines in a plane. The relation R in A defined by "x is parallel to y" is reflexive, since every straight line is parallel to itself.

Example 2. The relation "<" defined a set of real numbers is irreflexive because x is not less than x.

Example 3. If A be the set of men and R means "is husband of" then x R x is not true as man cannot be husband of himself. Thus, R is irreflexive.

Q.16 ***Define the Congruency relation.*** *[R.T.U. 2013]*

Ans. Congruency relation : An equivalence relation R on the semigroup (S, *) B called a congruence relation if

a R a' and b R b' imply (a * b) R (a' * b')

Example

Consider the semigroup (2, +) and the equivalence relation R on Z defined by

a R b if and only if a ≡ b (mod 2)

So

If a ≡ b (mod 2) and c ≡ d (mod 2)

then 2 divides a – b and 2 divjde c – d, so

a – b = 2m and c – d = 2n

where m and n are in z.

Adding, we have

(a – b) + (c – d) = 2m + 2n

or

(a + c) – (b + d) = 2(m + n)

So

a + c ≡ b + d (mod 2)

Hence the relation is a congrance relation.

Q.17 ***Define the Symmetric relation.*** *[R.T.U. 2013]*

Ans. Symmetric relation : A relation R on a non void set A is known as symmetric relation if x R $y \Rightarrow y$ R x, i.e., whenever $(x, y) \in$ R then $(y, x) \in$ R.

Example 1 : Let A be the set of all straight lines in a plane. The relation R defined by "a is perpendicular to b" is symmetric relation because.

$a \perp b \Rightarrow b \perp a, a, b \in A$

Example 2 : If A be the set of members of a family and R means "is the brother of" and x R y means that x is brother of y then x R y does not imply y R x, as y may be the sister of x. Hence R is not symmetric.

Q.18 ***Define the Asymmetric relation.*** *[R.T.U. 2013]*

Ans. Asymmetric relation : A relation R on a set A is said to be asymmetric if (a, b) $\in$ R $\Rightarrow$ (b, a) $\notin$ R, a, b $\in$ A.

Example : The relation $x < y$ is a symmetric as if $x < y$, then y is not less than x. The necessary and sufficient condition for a relation to be symmetric is $R = R^{-1}$

Q.19 ***Define the Transitive relation.*** *[R.T.U. 2013]*

Ans. Transitive relation : A relation on a set A is said to be transitive relation if for $x, y, z, \in A$, x R y and y R $z \Rightarrow x$ R z

i.e. whenever $(x, y) \in$ R and $(y, z) \in$ R then $(x, y) \in$ R.

Example 1 : The relation ">" defined on the set of natural numbers N is transitive because for $x, y, z \in$ N,

If $x > y$ and $y > z$ then $x > z$

i.e. x R y and y R $z \Rightarrow x$ R z.

Example 2 : The relation R of parallelism in the set of straight lines in a plane is a transitive relation because.

$x \parallel y$ and $y \parallel z \Rightarrow x \parallel z$.

i.e. x R y and y R $z \Rightarrow x$ R z, $x, y, z \in A$

The necessary and sufficient condition that a relation R be transitive is R operated on R $\subset$ R.

Q.20 ***Define the Anti symmetric relation.*** *[R.T.U. 2013]*

Ans. Anti symmetric relation : A relation R on a set A is said to be antisymmetric if whenever $x \neq$ then either $x \not R y$ or $y \not R x$.

Or

A relation R on a set A is said to be antisymmetric

If x R y and y R $x \Rightarrow x = y, x y \in A$.

Example 1 : Let N be the set of natural numbers and let R be the relation defined by "a divides b", $\forall$ a, b, $\in$ N. Then R is an antisymmetric relation as a divides b and b divides a $\Rightarrow$ a = b.

Example 2 : Let P be a family of sets, then the relation R on P defined by "A is a subset of B" is anti-symmetric because A R B and B RA $\Rightarrow$ A $\subset$ B and B $\subset$ A $\Rightarrow$ A = B.

The necessary and sufficient condition that R is antisymmetric is

$R \cap R^{-1} = \phi$.

Q.21 ***Show that in the power set P(A) = (Set of all subset of A), the relation of contained in defined as A1RA2 if A1 is a subset of A2, is a partial order relation.*** *[R.T.U. 2013]*

Ans. Let P = P(A) be the power set of the set A. Recall that P(A) is defined to be the set $\{S \mid S \subseteq A\}$, that is the set of all subsets of A. For example P({ 1,2, 3}) is the set

{θ, {1}, {2}, {3}, {2,3}, {1,3}, {1,2}, {1,2,3}}

Let $R = \{(X, Y) \in P \times P \mid X \subseteq Y\}$. That is R is a relation on P, and for X, Y $\in$ P, XRY if and only if X $\subseteq$ Y. Two elements X and Y of P are related by R if X is a subset of Y.

For any subset X of A note that XRX since $X \subseteq X$. Also θRX, and XRA, recall that $A \subseteq A$, so $A \in P$. For this relation it is true that if $X, Y \in P$ then $(X \cap Y)$ RX and $(X \cap Y)$ RY.

Also X R $(X \cup Y)$ and YR $(X \cup Y)$. That means for this relation, given X and Y there is always some U such that URX and URY and there is always some V such that XRV and YRV.

Q.22 ***Let A = {a, b, c, d, e} and B = {c, e, f, h, k, m} then prove if A and B are finite sets then $|A \cup B| = |A| + |B| - |A \cap B|$.*** *[R.T.U. 2011]*

Ans. $A = \{a, b, c, d, e\}$

$n(A) = |A| = 5$

$B = \{c, e, f, h, k, m\}$

$n(B) = |B| = 6$

Now, $A \cup B = \{a, b, c, d, e, f, h, k, m\}$

$\therefore$ $|A \cup B| = 9$

and $A \cap B = \{c, e\}$

$\therefore$ $|A \cap B| = 2$

Thus, we have

$|A \cup B| = |A| + |B| - |A \cap B|$

$9 = 5 + 6 - 2$

Hence Proved.

PART-B

Q.23 ***Let $f : R \to R$ and $g : R \to R$ where R is the set of real numbers. Find gof and fog where $f(x) = x^2 - 2$ and $g(x) = x + 4$. State whether these functions are injective, surjective or bijective.*** *[R.T.U. 2019]*

Ans. $fog = x^2 + 8x + 14$

$gof = x^2 + 2$

The minimum value taken up by $fog = x^2 + 8x + 14$ is 14. Similarly, the minimum value taken up by $gof = x^2 + 2$ is 2.

Hence the entire range is not mapped.

Minimum value for $fog = x^2 + 8x + 14$ is given for

$x = -4$ i.e. -2.

Q.24 ***Show that in the power set P(A) of all subsets of a set A = {a, b, c}, 'Set inclusion, $\subseteq$' is a partial order relation. Also draw the Hasse diagram for the POSET.*** *[R.T.U. 2019]*

Ans. $A = \{a, b, c\}$

Power set of A

$P(A) = \{\phi\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a, b, c\}$

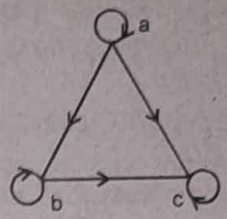
includes partial order relation

$\{<a, a>, <b, b>, <c, c>, <a, b>, <b, c>, <a, c>\}$

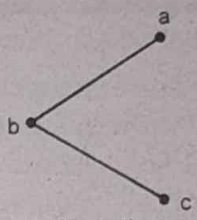
i.e. It satisfies (i) Reflexive (ii) Antisymmetric

(iii) Transitive properties

Hasse diagram of POSET



Digraph of partial ordered set



Hasse diagram

Q.25 ***Write the scope and objective of DMS in Computer Science?*** *[R.T.U. 2019]*

Ans. Scope of D.M.S. : Though discrete mathematics has found application in almost every conceivable area of study. It is integral part of science course. It provides the mathematical foundation for many computer courses viz algorithms, database management, automata, compiler theory, operating system, computer language, to name a few with wrong mathematical foundation, these computer science subject become easy to understand.

Objectives : The objectives of this course is to provide the fundamental and concepts of Discrete Mathematical Structures with application of computer science including mathematical logic, Boolean Algebra, and its applications, switching circuits and logic gates. Groups and Trees, Important computer theorem with constructive proofs, real life problems and graphs, theoretic algorithms, to help the students to understand the computational and algorithmic assects of sets, relations, functions and algebric structure in field of computer science and its application.

Q.26 *(a) Show that the set of odd positive integers is a countable set.*

(b) A survey is taken on method of commuter travel. Each respondent is asked to check BUS, TRAIN or AUTOMOBILE as a major method of travelling to work. More than one answer is permitted. The results reported were as follows :

(i) 30 people checked BUS
(ii) 35 people checked TRAIN
(iii) 100 people checked AUTOMOBILE
(iv) 15 people checked BUS and TRAIN
(v) 15 people checked BUS and AUTOMOBILE
(vi) 20 people checked TRAIN and AUTOMOBILE
(vii) 5 people checked all three methods
How many respondents completed their surveys? *[R.T.U. 2017]*

Ans.(a) Let T be the set of odd positive integers and N be the set of natural numbers :

Consider a function

$f : N \to T$

such that $f(n) = 2n - 1, n \in N$

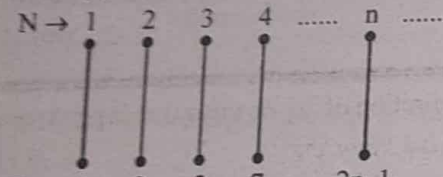


Fig. : Bisection between N and T

We need to show that f is bijection

One-One : Let $f(n_1) = f(n_2)$

$2n_1 - 1 = 2n_2 - 1 \quad 2n_1 = 2n_2$

$n_1 = n_2$

On-to : Let $t \in T$, then t is an odd positive integer

- $t + 1$ is on even positive integer.
- $\frac{t+1}{2}$ is a positive integer or $\frac{t+1}{2} \in N$.

Now for each $t \in T \; \exists \left(\frac{t+1}{2}\right) \in N$ such that

$$f\left(\frac{t+1}{2}\right) = 2\left(\frac{t+1}{2}\right) - 1 = (t+1) - 1 = t$$

So each element of T has its preimage in N. Thus f is onto.

Hence f is a bijection between N and T and then

$|N| = |T|$

Therefore, T is countable.

Ans.(b) Let A, B, C denote the set of respondent who checked Bus, Train and Automobile respectively.

So the number of respondent who checked Bus = $|A|$

The number of respondent who checked Train = $|B|$

The number of respondent who checked Automobile = $|C|$

The number of respondent who checked Bus and Train = $|A \cap B|$

The number of respondent who checked Train and Automobile = $|B \cap C|$

The number of respondent who checked Bus and Automobile = $|A \cap C|$

The number of respondent who checked all the three = $|A \cap B \cap C|$

In this question, it is given

$|A| = 30, |B| = 35, |C| = 100$

$|A \cap B| = 15, |A \cap C| = 15,$

$|B \cap C| = 20, |A \cap B \cap C| = 5$

It is required to find how many respondent completed the survey, i.e. $|A \cup B \cup C|$ is to be found out. According to the principle of inclusion and exclusion :

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|$$

$$= 30 + 35 + 100 - 15 - 15 - 20 + 5$$

$|A \cap B \cap C| = 120$ **Ans.**

Q.27 *If the set of integers I = {....., –3, –2, –1, 0, 1, 2, 3, ...} be partitioned by the equivalence relation aRb as $a \equiv b \pmod 3$. Obtain the set I/R.* *[R.T.U. 2017]*

Ans. Since difference of any two elements in

$P_1 = \{\ldots, -6, -3, 0, 3\ 6, 9, \ldots.\} = [0]$

is a multiple of –3, p_1 is an equivalence class in I.

Next,

$P_2 = \{\ldots.., -5, -2, 1, 4, 7, \ldots..\} = [1]$

is a set where difference of any two elements is a multiple of 3. So all these elements are in relation. So P_2 is another equivalence class of I (or z).

and in $P_3 = \{..., -4, -1, 2, 5, 8, ...\} = [2]$

The difference of any two elements is a multiple of 3. So all the elements of P_3 are in relation R. Hence P_3 is an equivalence class.

We have

$P_1 \cup P_2 \cup P_3 = I$

And $P_1 \cap P_2 = \phi, P_2 \cap P_3 = \phi, P_3 \cap P_1 = \phi$

So, P_1, P_2, P_3 are partitions of I induced by the relation R.

Hence the set $\frac{I}{R} = \{P_1, P_2, P_3\} = \{[0], [1], [2]\}$

Q.28 *Let A = Z the set of integers Relation R defined by A by aRb as 'a is congruent to b mod 2'. Show that R is an equivalence relations.* *[R.T.U. 2013, 12]*

OR

Define congruency relation in Modulo system. If A = Z (the set of integers), Relation R defined in A set by aRb as "a is congruent to b mod 2", then prove that R is an equivalence relation. *[R.T.U. 2017]*

OR

Let A = Z, the set of integers relation R define in A by aRb as "a congruent to b mod 2". Prove that R is an equivalence relation. *[R.T.U. 2014]*

Ans. Here R is $a \equiv b \pmod 2$ *i.e.* $a - b$ is divisible by 2 or $a - b$ is multiple of 2.

(i) Reflexive : Let $a \in A$, then

$a - a = 0 = 0 \times (2)$ a multiple of 2

$\Rightarrow$ aRa

$\therefore$ R is reflexive.

(ii) Symmetric: Let $a, b \in A$, then

$aRb \Rightarrow a \equiv b \pmod 2 \Rightarrow a - b$ is divisible by 2

$\Rightarrow$ $a - b = 2k$, $k \in Z$

$\Rightarrow$ $b - a = -2k = 2(-k)$, $-k \in Z$

$\Rightarrow$ $b \equiv a \pmod 2$

$\Rightarrow$ bRa

$\therefore$ R is symmetric.

(iii) Transitive: Let $a, b, c \in A$, then

$aRb \Rightarrow a - b = 2k_1$; $k_1 \in Z$ ....(i)

$bRc \Rightarrow b - c = 2k_2$; $k_2 \in Z$ ....(ii)

From eq. (i) + (ii)

$\Rightarrow$ $a - c = 2(k_1 + k_2)$; $k_1 + k_2 \in Z$

$\Rightarrow$ $a \equiv c \pmod 2$

$\Rightarrow$ aRc

$\therefore$ R is transitive.

Hence, it is an equivalence relation.

Q.29 *Define equivalence Relation. If R and S be two equivalence relations in a set A, then prove that R ∩ S is also an equivalence relation in A.* *[R.T.U. 2016]*

Ans. Equivalence relation : An equivalence relation is a binary relation which is reflexive relation, a symmetric relation and a transitive relation.

Now, to prove $R \cap S$ is an equivalence relation in A. Let $a \in A$ be an arbitary value. Since, R and S are equivalence relation on A, they are reflexive, so that $(a, a) \in R$ and $(a, a) \in S$. Hence, $(a, a) \in R \cap S$. Hence, $R \cap S$ is reflexive.

Now, Suppose $a, b \in A$ are such that $(a, b) \in R \cap S$. then, $(a, b) \in R$ and $(a, b) \in S$. Since R and S are symmetric, hence, $(b, a) \in R$ and $(b, a) \in S$. Hence, $(b, a) \in R \cap S$, so that $R \cap S$ is symmetric.

Suppose, $a, b, c \in A$ are such that $(a, b), (b, c) \in R \cap S$. Then, $(a, b), (b, c) \in R$ and $(a, b), (b, c) = S$. Since, R is transitive, $(a, b), (b, c) \in R$ implies that $(a, c) \in R$. Since, S is transitive, $(a, b), (b, c) \in S$ and hence, $(a, c) \in S$. Thus, $(a, c) \in R$ and $(a, c) \in S$, so that $(a, c) \in R \cap S$. Hence, $R \cap S$ is transitive.

Therefore, $R \cap S$ is an equivalence relation.

Q.30 *An equivalence relation R on a set A decomposes A into equivalence classes which are either distinct or completely overlapping and the set A is the union of such distinct equivalence classes.* *[R.T.U. 2016]*

OR

Prove that an equivalence relation R on a set A decomposes A into equivalence classes which are either distinct or completely overlapping and the set A is the union of such distinct equivalence classes. *[R.T.U. 2013, 09]*

Ans. By the definition of [a] we have that $[a] \subseteq A$. Hence $\bigcup_{a \in A} [a] \subseteq A$. We next show that $A \subseteq \bigcup_{a \in A} [a]$. Indeed let $a \in A$ since A is reflexive *i.e.* $a \in [a]$ and consequently $a \in \bigcup_{a \in A} [b]$. Hence $A \subseteq \bigcup_{a \in A} [b]$. It follows that $A = \bigcup_{a \in A} [a]$. This establishes (i) it remains to show that if $[a] \neq [b]$ then $[a] \cap [b] = \phi$ for $a, b \in A$. Suppose the contarary *i.e.*, suppose $[a] \cap [b] = \phi$. Then there is an element $c \in [a] \cap [b]$. This means that $c \in [a]$ and $c \in [b]$. Hence, aRc and bRc. Since R is symmetric and transitive then aRb. We will now prove that the conclusion aRb leads to [a] = [b].

The proof is by double inclusions. Let $x \in [a]$. Thus, xRa, since aRb and R is transitive then xRb which means that $x \in [b]$. Thus, $[a] \subseteq [b]$. Now, interchange the letters *a* and *b* to show that $[b] \subseteq [a]$.

Hence [a] = [b] which contradicts our assumption that $[a] \neq [b]$. Thus, A/R is a partition of A.

Let set [a] called the equivalence classes of A given by the relation R. The element a in [a] is called a representative of the equivalence class [a].

Q.31 *(i) Prove, for finite sets A and B;*

$n(A \cup B) = n(A) + n(B) - n(A \cap B)$

(ii) In a class of 50 students, 15 play Tennis, 20 play Cricket and 20 play Hockey, 3 play Tennis and Cricket, 6 play Cricket and Hockey, and 5 play Tennis and Hockey, 7 play no game at all. How many play Cricket, Tennis and Hockey? *[R.T.U. 2014]*

Ans.(i) We know that

$$(A - B) \cup (A \cap B) \cup (B - A) = A \cup B \quad ...(1)$$

and $A - B$, $A \cap B$ and $B - A$ are pair wise disjoint therefore,

$$n(A \cup B) = n(A - B) + n(A \cap B) + n(B - A) \quad ...(2)$$

Further $A = (A - B) \cup (A \cap B)$

and $(A - B) \cap (A \cap B) = \phi$

so $n(A) = n(A - B) + n(A \cap B)$ ...(3)

Similarly

$$n(B) = n(A \cap B) + n(B - A) \quad ... (4)$$

Adding (3) and (4), we have

$$n(A) + n(B) = \{n(A - B) + n(A \cap B) + n(B - A)\} + n(A \cap B)$$
$$= n(A \cup B) + n(A \cap B) \text{ [Using (2)]}$$

Thus

$$n(A \cup B) = n(A) + n(B) - n(A \cap B)$$

Ans.(ii) Let A = Student play Tennis

B = Student play Cricket

C = Student play Hockey

so $n(A) = 15$; $n(B) = 20$; $n(c) = 20$;

$N(A \cap B) = 3$; $n(B \cap C) = 6$; $n(A \cap C) = 5$

$\because$ 50 students in the class in which 7 play no game at all so

$$n(A \cup B \cup C) = 50 - 7 = 43$$

Now number of students those play Cricket, Hockey and Tennis is

$$N(A \cap B \cap C) = n(A \cup B \cup C) - n(A) - n(B) - n(C) + n(A \cap B) + n(B \cap C) + n(C \cap A)$$
$$= 43 - 15 - 20 - 20 + 3 + 6 + 5$$
$$= 57 - 55$$
$$= 2$$

Q.32 *(i) If $f: A \to B$ be one-one onto then the inverse map of f is unique. Prove it.*

(ii) Show that set of even positive integers is a countable set. *[R.T.U. 2014]*

Ans.(i) If possible, let g and h be inverses of f.

Then by the definition of inverse

$$g.f = f.g = I \quad ... (1)$$

and $h.f = f.h = I$ ... (2)

where I is an identity function. Now

$h = h.I = h.(f.g)$ (Using (1))

$= (h.f).g$ (Associativity Rule)

$= I.g$ (Using (2))

$= g$

So $h = g$

Hence the result.

Ans. (ii) Proof : Let's count the elements in the set of even positive integers. We do that by comparing the even numbers with the counting numbers:

1 2 3 4 5 6 7 8 9 10 11 . . . (counting numbers)

2 4 6 8 10 12 14 16 18 20 22 . . . (even numbers)

This correspondence goes on forever. And we can find no even number that does not match up with a counting number. We can only deduce that the two sets have the same number of elements. This may seem counter-intuitive, but it is straightforward. The set of even positive integers has the same number of elements as the set of counting numbers. They are both infinite sets of the same size. The set of all integers (positive and negative) matches up the same way. So does the set of all integers ending in 000. These are all **countable sets**, because they have the same number of elements as the counting numbers.

Q.33 *Compute the number of partitions of a set with four elements.* *[R.T.U. 2014]*

Ans. Here $n = 4$

Thus the number of partitions $= \sum_{r=1}^{4} S(4, r)$

$= S(4, 1) + S(4, 2) + S(4, 3) + S(4, 4)$ ... (1)

Now $S(4, 1) = 1 = S(4, 4)$ ... (2)

$S(4, 2) = S(3, 1) + 2S(3, 2)$ ... (3)

$S(4, 3) = S(3, 2) + 3S(3, 3)$ ... (4)

Since $S(3, 2) = S(2, 1) + 2S(2, 2)$ ... (5)

Now $S(2, 1) = S(2, 2) = 1$

$[\because S(n, 1) = 1 = S(n, n)$

Thus eq.(5) gives

$S(3, 2) = 1 + 2(1) = 3$

From (3)

$S(4, 2) = 1 + 2(3) = 7$

From (4)

$S(4, 3) = 3 + 3(1) = 6$

Hence the number of partition $= 1 + 7 + 6 + 1 = 15$

Q.34 *Out of 250 failed students, 128 fails in Maths, 87 in Physics and 134 in Aggregate, 31 failed in Maths and Physics, 54 failed in Aggregate and in Maths, 30 failed in Aggregate and in Physics. Find how many candidates failed.*

(i) All three subjects

(ii) In Maths not in Physics

(iii) In Aggregate but not in Maths

(iv) In Physics but not in Aggregate or Maths

(v) In the Aggregate or in Maths, but not in Physics.

[R.T.U. 2013]

Ans. Let M, P and A be the sets of students who failed in Maths, Physics and Aggregate, respectively.

Given

$|M| = 128$

$|P| = 87$

$|A| = 134$

$|M \cap P| = 31$

$|A \cap M| = 54$

$|A \cap P| = 30$

We know that

$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |C \cap A| + |A \cap B \cap C|$

(i) All three subjects

$|M \cup P \cup A| = |M| + |P| + |A| - |M \cap P| - |P \cap A| - |A \cap M| + |M \cap P \cap A|$

$250 = 128 + 87 + 134 - 31 - 30 - 54 + |M \cap P \cap A|$

$250 = 349 - 115 + |M \cap P \cap A|$

$250 - 234 = |M \cap P \cap A|$

$16 = |M \cap P \cap A|$

Candidates failed in all three subjects = 16.

(ii) In Maths not in Physics

After seeing the following fig.

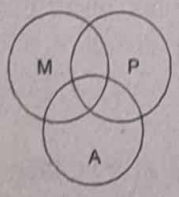


Fig.

Candidates failed in Maths not in Physics

$= 128 - 31 = 97$

(iii) In Aggregate but not in Maths

$= 134 - 54 = 80$

(iv) In Physics but not in Aggreegate or Maths

$= 87 - 30 - 31 + 16 = 42$

(v) In the aggregate or in Maths, but not in Physics.

$= 128 + 134 - 31 - 30 + 16 = 217$

Q.35 *(a) In the Survey of 60 people it was found that 25 read News week, 26 read Time and 26 read the magazine Fortune. Also 9 read both News weak and Fortune, 11 read News week and Time and 8 read both time and fortune. If 8 read none of the three magazines, determine the number of people who read exactly one magazine.*

[R.T.U 2012, Raj. Univ. 2007, 2001]

(b) Let $f : A \to B$ and $g : B \to C$ be two invertible then

(i) gof is invertible and

(ii) $(gof)^{-1} = f^{-1} \circ g^{-1}$ *[R.T.U. 2012]*

Ans.(a) Using the given data and the number of elements in various parts of the sets, we have the equation

$C = 8$ ....(1)

Those who read News week only = x

Those who read Fortune only = y

Those who read Time only = z

$(x + y + z) + (p + q + r) + \lambda = 60 - 8 = 52$ ....(2)

$x + (p + q) + \lambda = 25$ ....(3)

$y + (q + r) + \lambda = 26$ ....(4)

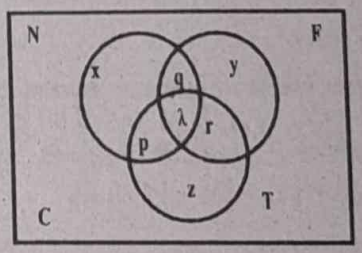
$z + (r + p) + \lambda = 26$ ....(5)

$q + \lambda = 9$ ....(6)

$p + \lambda = 11$ ....(7)

$r + \lambda = 8$ ....(8)

$n(u) = 60$



Adding eq. (6), (7), (8),

$(p + q + r) + 3\lambda = 28$ ....(9)

Adding eq. (3), (4), (5)

$(x + y + z) + 2(p + q + r) + 3\lambda = 77$ ....(10)

Let $x + y + z = a$, $p + q + r = b$

then eq. (2) gives

$a + b + \lambda = 52$ ....(11)

Eq.(9) gives $b + 3\lambda = 28$ ....(12)

and eq. (10) gives $a + 2b + 3\lambda = 77$ ....(13)

Doing eq.(11) + (12) – (13) we have

$\lambda = 52 + 28 - 77 = 3$ ....(14)

Hence from eq. (12)

$b = 28 - 3 \times 3 = 19$ ....(15)

From eq. (11), $a = 52 - b - \lambda$

$= 52 - 19 - 3 = 30$ ....(16)

Those who read only one magzine $= x + y + z = 30$

Ans.(b) Since f and g are one-to-one functions, therefore we have

$f(x_1) = f(x_2) \Rightarrow x_1 = x_2$ for all $x_1, x_2 \in R$

$g(y_1) = g(y_2) \Rightarrow y_1 = y_2$ for all $y_1, y_2 \in R$

$(gof)(x_1) = (gof)(x_2) \Rightarrow g(f(x_1)) = g(f(x_2))$

$\Rightarrow f(x_1) = f(x_2)$

$\Rightarrow x_1 = x_2$

Also as f and g are both onto functions, so for each $z \in C$, there exists $y \in B$ such that $g(y) = z$ and $f(x) = y$ for each $x \in A$. Thus $z = g(y) = g(f(x)) = (g \circ f)(x)$. Hence each element $z \in C$ has preimage under $g \circ f$ and therefore $g \circ f$ is an onto function. Since it has been proved that $g \circ f$ is one-one onto function, therefore $(g \circ f)$ exists. This implies that if $g \circ f : A \to C$, then $(g \circ f)^{-1} : C \to A$ or $g^{-1} \circ f^{-1} : C \to A$.

Now $(g \circ f)^{-1}(z) = x \Leftrightarrow (g \circ f)(x) = z$

$\Leftrightarrow g(f(x)) = z$ or $g(y) = z$

$\Leftrightarrow y = g^{-1}(z)$

$\Leftrightarrow f^{-1}(y) = f^{-1}(g^{-1}(z)) = (f^{-1} \circ g^{-1})(z)$

$\Leftrightarrow x = (f^{-1} \circ g^{-1})(z)$.

Hence $(g \circ f)^{-1}(z) = (f^{-1} \circ g^{-1})(z)$

or $(g \circ f)^{-1} = (f^{-1} \circ g^{-1})$.

Q.36 ***(a) Let A = {a, b, c, d, e} and B = {c, e, f, h, k, m} then prove if A and B are finite sets then*** $|A \cup B| = |A| + |B| - |A \cap B|$.

[R.T.U. 2011]

(b) Let 100 of 120 students of mathematics of a college take at least one of the languages Hindi, English and German. Also suppose 65 study Hindi, 45 study English, and 42 study German. If 20 study Hindi and English, 25 study English and German and 15 study Hindi and German. Find the number of students who study all the three languages.

[R.T.U. 2011, 2009]

Ans.(a) $A = \{a, b, c, d, e\}$

$n(A) = |A| = 5$

$B = \{c, e, f, h, k, m\}$

$n(B) = |B| = 6$

Now, $A \cup B = \{a, b, c, d, e, f, h, k, m\}$

$\therefore |A \cup B| = 9$

and $A \cap B = \{c, e\}$

$\therefore |A \cap B| = 2$

Thus, we have

$|A \cup B| = |A| + |B| - |A \cap B|$

$9 = 5 + 6 - 2$ **Hence Proved.**

Ans.(b) Let H, E and G denote the set of students who study Hindi, English and German respectively.

Then,

$n(H \cup E \cup G) = 100$

$n(H) = 65, n(E) = 45, n(G) = 42,$

$n(H \cap E) = 20, n(E \cap G) = 25, n(H \cap G) = 15$

We have to find $n(H \cap E \cap G)$

By the principle of inclusion – exclusion we have

$n(H \cup E \cup G) = n(H) + n(E) + n(G) - n(H \cap E)$

$= -n(E \cap G) - n(H \cap G) + n(H \cap E \cap G)$

$100 = 65 + 45 + 42 - 20 - 15 - 25 + n(H \cap E \cap G)$

$n(H \cap E \cap G) = 160 - 152 = 8$

Thus, 8 students study all three languages.

Q.37 ***Determine whether each of the following function is bijection (one to one and one to many) from R to R.***

(i) $f(x) = -3x + 4$

(ii) $f(x) = -3x^2 + 7$

(iii) $f(x) = x^5 + 1$ *[R.T.U. 2011, MREC 2002]*

Ans. (i) $f(x) = -3x + 4$ to prove one–one for two input if $x_1 \neq x_2$ then $f(x_1) \neq f(x_2)$ for all value of x_1 and x_2 in domain so it is one–one

To prove onto for all values of x in R it will give all the values of R in result so it is onto.

(ii) $f(x) = -3x^2 + 7$

To prove one–one if $x_1 \neq x_2$ then $f(x_1) = f(x_2)$ so this not one–one such as for 1 and –1 it will give result 4. So this is not bijection function.

(iii) $f(x) = x^5 + 1$

To prove one–one if $x_1 \neq x_2$ then if $f(x_1) \neq f(x_2)$ for all values of x_1 and x_2 so this is one–one

To prove onto, for all values for x in R, it will give all values in R. So this is a bijection function.

Q.38 *(a) Show that in the set of integers $I = \{..., -2, -1, 0, 1, 2, ...\}$, the relation $aRb \Rightarrow a \equiv b \pmod n$, $n \in N$, is an equivalence relation.*

(b) Show that an equivalence relation defined in a set A decomposes the set into disjoint classes. *[R.T.U. 2011, 2009]*

Ans. (a) The relation is $a \equiv b \pmod n$ i.e. a is congruent to $b \pmod n$ meaning that

$a - b$ = multiple of $n = np$, $p \in I$

For equivalence relation, we show that given relation is reflexive, symmetric and transitive.

(i) Reflexive: Let $a \in I$, then

$a - a = 0 = 0 \cdot n$ = multiple of n

$\Rightarrow aRa$

$\therefore$ R is reflexive.

(ii) Symmetric: Let $a, b \in I$, then

$aRb \Rightarrow a \equiv b \pmod n \Rightarrow a - b = np, p \in I$

$\Rightarrow b - a = -np = n(-p), -p \in I$

$\Rightarrow b \equiv a \pmod n = bRa$

$\therefore$ R is symmetric.

(iii) Transitive: Let $a, b, c \in I$, and

$aRb \Rightarrow a - b = n p_1$ ....(i)

$bRc \Rightarrow b - c = n p_2$ ....(ii)

(i) + (ii) $\Rightarrow a - c = n(p_1 + p_2), p_1 + p_2 \in I$

$\Rightarrow a \equiv c \pmod n$

$\Rightarrow aRc$,

$\therefore$ R is transitive.

Hence, it is an equivalence relation.

Ans.(b) Let an equivalence relation R be defined in A. Let $a \in A$ and T be a subset of A consisting of all those elements which are equivalent to a *i.e.*,

$T = \{x \mid x \in A \text{ and } a \, R \, x\}$.

Then $a \in T$, for $a \, R \, a$ (R is reflexive). Any two elements of T are equivalent to each other, for if $x, y \in T$, then $x \, R \, a$ and $y \, R \, a$.

Again $x \, R \, a$, $yR \, a \Rightarrow x \, R \, a$, $a \, R \, y$ [R is symmetric]

$\Rightarrow x \, R \, y$

Hence T is an equivalence class.

suppose T_1 be another equivalence class *i.e.*,

$T_1 = \{x \mid x \in A \text{ and } x \, R \, b\}$,

where b is not equivalent to a. Then the classes T and T_1 must be disjoint. For if they have a common element A, $A \, R \, a$ and $s \, R \, b$, so that $b \, R \, a$ which is contrary to our hypotheses.

Now the set A can be decomposed into equivalence classes T, T_1, T_2, ... such that every element of A belongs to one of these classes. Since these classes are mutually disjoint, we obtain the required partition of A.

Q.39 *(a) Let S be any non empty set. Show that $(P(S), \subseteq)$ is a partially ordered set for the relation inclusion.*

(b) Let R be the relation defined on a set of natural numbers N s.t. for $x, y \in N$, $xRy \Leftrightarrow x - y$ is divisible by 3, then show that R is an equivalence relation on N. Find equivalence classes also. *[R.T.U. 2008]*

Ans.(a) Let A_1, A_2, A_3 be three subsets of S so they are elements of P(S).

Relation is reflexive : $A_1 \subseteq A_1$ is true for any set S, so the 'set inclusion' relation is reflexive.

Relation is anti-symmetric : $A_1 \subseteq A_2$ and $A_2 \subseteq A_1$ both be true only when $A_1 = A_2$ so the relation is anti-symmetric.

Relation is transitive : Whenever $A_1 \subseteq A_2$ and $A_2 \subseteq A_3$, both are true then $A_1 \subseteq A_3$ is also true so relation is transitive also. So relation is partial order relation.

Hence $\{P(S), \subseteq\}$ is a partially ordered set for the relation inclusion.

Ans.(b) Here R is $x - y$ is divisible by 3 or $x - y$ is multiple of 3.

Reflexive : Let $x \in N$

$x - x = 0 \times 3$ is a multiple of 3.

$\Rightarrow x \, R \, x \; \forall x \in N$

$\therefore$ R is reflexive.

Symmetric : Let $x, y \in N$

then $x \, R \, y \Rightarrow x \equiv y \pmod 3$

$\Rightarrow x - y$ is divisible by 3.

$\Rightarrow x - y = 3k$

$\Rightarrow y - x = -3k$

$y \equiv x \pmod 3$

$\Rightarrow y \, R \, x$

$\therefore$ R is symmetric.

Transitive: Let $x, y, z \in N$

$x \, R \, y \Rightarrow x - y = 3 k_1$

$y \, R \, z \Rightarrow y - z = 3k_2$

$x - z = 3(k_1 + k_2)$

$x \equiv z \pmod 3$

$x \, R \, z$

$\therefore$ R is transitive.

Hence it is an equivalence relation.

Q.40 *If $f(x) = x + 2$, $g(x) = x - 2$, $h(x) = 3x$ be functions on R, the set of the real numbers then find :*

(i) gof (ii) fog (iii) fof
(iv) gofg (v) foh (vi) hog
(vii) hof (viii) (foh)og *[Raj. Univ. 2006]*

Ans. Given that $f(x) = x + 2$, $g(x) = x - 2$, $h(x) = 3x$

(i) $gof(x) = g[f(x)] = g[x+2] = x+2-2 = x$

(ii) $fog(x) = f[g(x)] = f[x-2] = x-2+2 = x$

(iii) $fof(x) = f[f(x)] = f[x+2] = x+2+2 = x+4$

(iv) $gog(x) = g[g(x)] = g[(x-2)] = x-2-2 = x-4$

(v) $foh(x) = f[h(x)] = f[3x] = 3x+2$

(vi) $hog(x) = h[g(x)] = h[x-2] = 3(x-2) = 3x-6$

(vii) $hof(x) = h[f(x)] = h[x+2] = 3(x+2) = 3x+6$

(viii) $(foh)og(x) = foh[g(x)] = foh[x-2]$
$= f[3(x-2)] = f(3x-6) = 3x-6+2 = 3x-4$

PART-C

Q.41 *Let $R = \{(1, 2), (2, 3), (3, 1)\}$ and $A = \{1, 2, 3\}$. Find reflexive, symmetric and transitive closure of R using-*

(a) Composition of relation R
(b) Composition of matrix relation R
(c) Graphical representation of R *[R.T.U. 2019]*

Ans.(a) The reflexive closure of R, denoted by $R^{(r)}$, is given by

$R^{(r)} = R \cup I_A = \{(1,2),(2,3),(3,1)\} \cup \{(1,1),(2,2),(3,3)\}$
$= \{(1,1),(1,2),(2,2),(2,3),(3,1),(3,3)\}$

The symmetric closure of R, denoted by $R^{(S)}$, is given by

$R^{(S)} = R \cup R^{-1} = \{(1,2),(2,3),(3,1)\} \cup \{(2,1),(3,2),(1,3)\}$
$= \{(1,2),(1,3),(2,1),(2,3),(3,1),(3,2)\}$

Now, $RoR = \{(1,2),(2,3),(3,1)\}$ o $\{(1,2),(2,3),(3,1)\}$

$R^2 = \{(1,3),(2,1),(3,2)\}$

$R^3 = R^2oR = \{(1,3),(2,1),(3,2)\}$ o $\{(1,2),(2,3),(3,1)\}$
$= \{(1,1),(2,2),(3,3)\}$

$R^4 = R^3oR = \{(1,1),(2,2),(3,3)\}$ o $\{(1,2),(2,3),(3,1)\}$
$= \{(1,2),(2,3),(3,1)\} = R$

Thus, $R^5 = R^4oR = RoR = R^2$, $R^6 = R^5oR = RoR = R^2oR = R^3$ and so on.

Hence, the transitive closure of R, denoted by R^*, is given by

$R^* = R \cup R^2 \cup R^3 = \{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)\}$

(b) Let M be the relation matrix of R. Then

$$\therefore \quad M_R = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

The symmetric closure matrix of R, denoted by $M_R^{(S)}$, is given by

$$M_R^{(S)} = M_R \vee M_R^T$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \vee \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

so that $R^{(S)} = \{(1,2),(1,3),(2,1),(2,3),(3,1),(3,2)\}$

The reflexive closure matrix of R, denoted by $M_R^{(r)}$, is given by

$$M_R^{(r)} = M_R \vee I_3$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

so that $R^{(r)} = \{(1,1),(1,2),(2,2),(2,3),(3,1),(3,3)\}$

Now $M_R^2 = M_R \cdot M_R$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$M_R^3 = M_R^2 \cdot M_R$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The transitive closure relation matrix of R, denoted by, M_R^* is given by

$$M_R^* = M_R \vee M_R^2 \vee M_R^3$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \vee \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

so that $R^* = \{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)\}$

(c) The graphical representation of R is shown in Fig.1

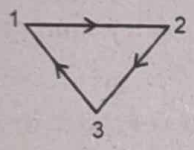


Fig. 1

To find out the reflexive closure representation of R we add all the arrows from points to themselves which is shown in Fig. 2.

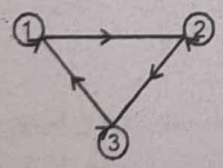


Fig. 2

To find the symmetric closure representation of R, we add missing reverses of all arrows in graphical representation of R. This is shown in Fig. 3.

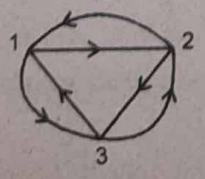


Fig. 3

To find transitive arrow, we add arrow 1 to 1 since $1 \to 2 \to 3 \to 1$. Similarly, 2 to 2 and 3 to 3. Again we add arrow 1 to 3, since $1 \to 2 \to 3$. Similarly, 2 to 1 and 3 to 2. This is shown in Fig. 4.

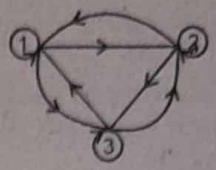


Fig. 4

Q.42 ***What is the Pigeonhole principle and the Extended Pigeonhole principle also prove both.*** ***[R.T.U. 2012]***

OR

State and prove the generalized pigeonhole principle. ***[R.T.U. 2017]***

OR

Explain pigeonhole and extended pigeonhole principle with example. ***[R.T.U. 2010]***

OR

State and prove the Pigeonhole and Generalized Pigeonhole Principles. ***[R.T.U. 2014]***

Ans. Pigeonhole Principle

If n pigeonholes are occupied by n pigeons and $m > n$ then at least one pigeonhole is occupied by more than one pigeon.

The pigeon hole principle is nothing more than the obvious remark : if you have fewer pigeon holes than pigeons and you put every pigeon in a pigeon hole, then there must result at least one pigeon hole with more than one pigeon. It is surprising how useful this can be as a proof strategy.

The pigeonhole principle, also known as *Dirichlet's* box (or drawer) principle, states that, given two natural numbers n and m with $n > m$, if n items are put into m pigeonholes, then at least one pigeonhole must contain more than one item. Another way of stating this would be that m holes can hold at most m objects with one object to a hole; adding another object will force one to reuse one of the holes, provided that m is finite. More formally, the theorem states that there does not exist an injective function on finite sets whose co-domain is smaller than its domain.

The pigeonhole principle is an example of a **counting argument** which can be applied to many formal problems, including ones involving infinite sets that cannot be put into one-to-one correspondence.

Generalizations of the Pigeonhole Principle

A generalized version of this principle states that, if n discrete objects are to be allocated to m containers, then at least one container must hold no fewer than [n/m] objects,

where [x] is the ceiling function, denoting the smallest integer larger than or equal to x.

A probabilistic generalization of the pigeonhole principle states that if n pigeons are randomly put into m pigeonholes with uniform probability 1/m, then at least one pigeonhole will hold more than one pigeon with probability

$$1-\frac{m!}{(m-n)!m^n}=1-\frac{(m)_n}{m^n}\quad\left(\because (m)_n=\frac{\sqrt{m+1}}{\sqrt{m-n+1}}\right)$$

where $(m)_n$ is falling factorial, for $n=0$ and for $n=1$ (and $m>0$), that probability is zero; in other words, if there is just one pigeon, there cannot be conflict. For $n>m$ (more pigeons than pigeonholes) it is one, in which case it coincides with the ordinary pigeonhole principle. But even if the number of pigeons does not exceed the number of pigeonholes ($n \le m$), due to the random nature of the assignment of pigeons to pigeonholes there is often a substantial chance that clashes will occur. For example, if 2 pigeons are randomly assigned to 4 pigeonholes, there is a 25% chance that at least one pigeonhole will hold more than one pigeon; for 5 pigeons and 10 holes, that probability is 69.76%; and for 10 pigeons and 20 holes it is about 93.45%. This problem is treated at much greater length at birthday paradox.

Pigeonhole Principle : Simple Form

Theorem : If $n+1$ objects are put into n boxes, then at least one box contains two or more objects.

Proof : Suppose none of the n boxes contains more than one object. Then the total number of objects would be at most n. This is a contradiction, since there are at least $n+1$ objects.

Example : There are n married couples. How many of the 2n people must be selected in order to guarantee that one has selected a married couple?

Q.43 ***Prove by mathematical induction that $3^n > n^3$ for all integers $n \ge 4$.*** ***[R.T.U. 2017]***

Ans. Let $P(n) = 3^n > n^3$ where $n \ge 4$.

Basic Step : P(n) is true since $3^4 = 81 > 4^3 = 64$

Inductive step : Let P(k) is true for all $k \ge 4$, i.e. $3^k > k^3$.

Then we need to show that P(k+1) is true, i.e. $3^{k+1} > (k+1)^3$

Let us rewrite

$$(k+1)^3 = k^3 + 3k^2 + 3k + 1$$

$$= k^3\left(1+\frac{3}{k}+\frac{3}{k^2}+\frac{1}{k^3}\right)$$

Since $3^k > k^3$ (using P(k)), we would be done if we could also prove that $3>\left(1+\frac{3}{k}+\frac{3}{k^2}+\frac{1}{k^3}\right)$ for $k \ge 4$.

Observe that the function $f(k) = 1+\frac{3}{k}+\frac{3}{k^2}+\frac{1}{k^3}$ decreases as k increases, so that f(k) is largest when k is smallest. In other words, f(4) is the largest value of f(k) where $k \ge 4$.

Since

$$f(4) = 1+\frac{3}{4}+\frac{3}{4^2}+\frac{1}{4^3}$$

$$=\frac{125}{64}$$

is obviously less then 3. We have, for any integer $k \ge 4$,

$$3>\left(1+\frac{3}{k}+\frac{3}{k^2}+\frac{1}{k^3}\right)$$

Thus combining the two facts :

$3>\left(1+\frac{3}{k}+\frac{3}{k^2}+\frac{1}{k^3}\right)$ and $3^k > k^3$ for $k \ge 4$, we can multiply and get

$$3^{k+1} > k^3\left(1+\frac{3}{k}+\frac{3}{k^2}+\frac{1}{k^3}\right)$$

or $3^{k+1} > (k+1)^3$

So P(k+1) is true and then by mathematical induction P(n) is true for all integers $n \ge 4$, i.e. $3^n > n^3$.

Q.44 ***Prove by mathematical induction that $6^{n+2} + 7^{2n+1}$ is divisible by 43 for each positive integer n.*** ***[R.T.U. 2016]***

OR

Define principle of mathematical induction and hence prove that $6^{(n+2)} + 7^{(2n+1)}$ is divisible by 43 for each positive integer n. ***[R.T.U. 2008]***

Ans. Principle of Mathematical Induction : Let p(n) be some statement involving a positive integer n. Suppose it is required to show that p(n) is true for integer n greater than some fixed integer n_0. The method adopted here to prove it is called the *method of mathematical induction*. This method consists of the following three steps :

Basis Step : First it is verified whether p(n) is true for certain number. Generally we take n_0 to have the value one. If p (n) is not true for n = 1 then the least value of n found for which it is true.

Hypothesis : Then it is assumed that p (n) is true for $n=k$.

Inductive Step : Taking p(n) to be true for n = k, it is proved that p(n) is true also for the next value n = (k+1).

Since it has been found to be true for $n = n_0$, so it is true for $n = n_0 + 1$. When it is true for $n = n_0 + 1$, it is true for the next value $n = n_0 + 1 + 1$.

Arranging in this way, it is concluded that p(n) is true for all positive integer values of $n \ge n_0$. The above method

proving a proposition p(n) involving a positive integer n is called the *Method of Mathematical Induction* or the *Principle of Mathematical Induction.*

Proof

Let $p(n)$: 43 divides $6^{n+2}+7^{2n+1}$

Basis Step : Let n = 1, then p(1) is true

$= 6^{1+2}+7^{2+1} = 6^3+7^3 = 216+343$

$= 559$, which is divisible by 43.

Inductive Hypothesis : Let p(k) be true for $k \geq 1$.

$6^{k+2}+7^{2k+1}$ is divisible by 43 (say 43λ).

Inductive Step : We shall show that

$p(k+1)$: $6^{k+3}+7^{2k+3}$ is divisible by 43, is true whenever p(k) is true.

$$6^{k+3}+7^{2k+3} = 6^{k+2}.6^1+7^{2k+1}.7^2$$

$$= 6^{k+2}.6+7^{2k+1}.(43+6)$$

$$= 43.7^{2k+1}+6\left(6^{k+2}+7^{2k+1}\right)$$

$$= 43.7^{2k+1}+6.43\lambda = 43\,(7^{2k+1}+6\lambda)$$

$\Rightarrow$ $6^{k+3}+7^{2k+3}$ is divisible by 43.

Now

$\Rightarrow$ p (k + 1) is true.

Hence by principle of mathematical induction 43 divides $6^{(n+2)}+7^{(2n+1)}$.

Q.45 *(a) Prove that $A-B = A \cap B' = B' \cap A'$*

(b) Consider the following collection of subsets $\{A_1, A_2, A_3\}$ of a set $A = \{1,2,3,4,5,6,7,8,9,10\}$

[a] [{1,6,9}, {2,3,8}, {4,5,7,10}]

[b] {1}, {2,4,8}, {5,7,9}] and

[c] [{1,5}, {2,3,8}, {4,5,6,7,9,10}]

Determine which one is a partition of a set A

(c) Let f, g, h be mapping from N to N when N is the set of naturals such that $f(n) = n + 1$,

$$g(n) = 2n,\ h(n) = \begin{cases} 0, & n \text{ is even} \\ 1, & n \text{ is odd} \end{cases}$$

(i) Show that f, g and h are functions

(ii) Determine fof, fog, hog and (fog) oh

Where 'o' stands for composition of functions

[R.T.U. 2016]

Ans.(a) Prove that $A-B = A \cap B' = B'-A$

Let $x \in A-B \Leftrightarrow x \in A$ and $x \notin B$

$\Leftrightarrow x \notin A'$ and $x \in B'$

$\Leftrightarrow x \in B'$ and $x \notin A'$

$\Leftrightarrow x \in B'-A'$

Thus $A-B = B'-A'$

Again

$x \in A-B \Leftrightarrow x \in A$ and $x \notin B$

$\Leftrightarrow x \in A$ and $x \in B'$

$\Leftrightarrow x \in A \cap B'$

thus $A-B = A \cap B'$

from (1) and (2)

$A-B = A \cap B' = B'-A'$

Ans.(b) $A=\{1,2,3,4,5,6,7,8,9,10\}$

A family of sets A_1, A_2, A_3 is called partition of A if,

(i) $A_j \cap A_k = \phi$ for any $k \neq j$

(ii) Their union completely covers the space, $S = A_1 \cup A_2 \ldots A_n$

[a] $A_1 = \{1,6,9\}$

$A_2 = \{2,3,8\}$

$A_3 = \{4,5,7,10\}$

$A_1 \cap A_2 = \{\phi\}$

$A_1 \cap A_3 = \phi$

$A_2 \cap A_3 = \phi$

Also, $A_1 \cup A_2 \cup A_3 = \{1,2,3,4,5,6,7,8,9,10\} = A$

$\therefore A_1, A_2, A_3$ is partition of A.

[b] $A_1 = \{1\}$

$A_2 = \{2,4,8\}$

$A_3 = \{5,7,9\}$

There, $A_1 \cap A_2 = \phi$

$A_2 \cap A_3 = \phi$

$A_1 \cap A_3 = \phi$

Now, $A_1 \cup A_2 \cup A_3 = \{1,2,4,5,7,8,9\} \neq A$

$\therefore A_1, A_2, A_3$ is not the partition of A.

[c] $A_1 = \{1,5\}$,

$A_2 = \{2,3,8\}$,

$A_3 = \{4,5,6,7,9,10\}$

Now, $A_1 \cap A_2 = \phi$

$A_2 \cap A_3 = \phi$

$A_1 \cap A_2 = \{5\}$

$\Rightarrow$ A_1, A_2 and A_3 is not a partition of A.

Ans.(c) $f(n) = n+1$,

$$g(n) = 2n,\ h(n) = \begin{cases} 0, & n \text{ is ever} \\ 1, & n \text{ is odd} \end{cases}$$

(i) As we know that a relation is a function if it is defined for all values of domain and $f(x) \neq f(y) = x \neq y, \forall x, y \in N$.

f is a function

Here $f(n) = n+1, n \in N$.

Also, f is defined for all $n \in N$ let $f(n_1) \neq f(n_2)$

$$n_1+1 \neq n_2+1 \Rightarrow n_1 \neq n_2$$

$\therefore$ f is a function.

g is a function

Here, $g(n) = 2n$

$\because$ g(n) exists for all $n \in N$ and

$$g(n_1) \neq g(n_2)$$

$$g(n_1) \neq g(n_2) = 2n_1 \neq 2n_2$$

$$\Rightarrow n_1 \neq n_2$$

$\therefore$ g is a function.

h is a function

$$h(n) = \begin{cases} 0, & n \text{ is ever} \\ 1, & n \text{ is odd} \end{cases}$$

$\because 0 \notin N$, so h(n) doesn't exist for even n and then h is not a function.

(ii)
$$fof = (fof)(n) = f(f(n))$$
$$= f(n+1)$$
$$= n+1+1$$
$$= n+2$$
$$(fog)(n) = f(g(n)) = f(2n)$$
$$= 2n+1$$
$$(hog)(n) = h(g(n))$$
$$= h(2n)$$
$$= 0$$
$$(fog)oh = (fog)(h(n))$$

$$= \begin{cases} fog(0), & n \text{ is even} \\ fog(1), & n \text{ is odd} \end{cases} = \begin{cases} f[g(1)], & n \text{ is even} \\ f[g(1)], & n \text{ is odd} \end{cases}$$

$$= \begin{cases} f(0), & n \text{ is even} \\ f(2), & n \text{ is odd} \end{cases} = \begin{cases} 1, & n \text{ is even} \\ 3, & n \text{ is odd} \end{cases}$$

Q.46 (a) ***State and prove the Principle of Inclusion and Exclusion for three sets A,B and C.***

(b) There are 250 students in a computer Institute of these 180 have taken a course in Pascal, 150 have taken a course in C++, 120 have taken a course in Java. Further 80 have taken Pascal and C++, 60 have taken C++ and Java, 40 have taken Pascal and Java and 35 have taken all 3 courses. So find-

(i) How many students have not taken any course?

(ii) How many study at least one of the languages?

(iii) How many students study only Java?

(iv) How many students study Pascal and C++ but not Java?

(c) Let A = {1,1,1,2,2,3,4,4} and B = {1,2,4,4,5,5,5}. Find $A \cup B$, $A \cap B$, $A - B$ and A+B. [R.T.U. 2015]

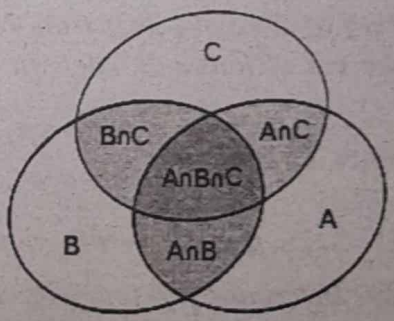
Ans.(a) In combinatorics (combinatorial mathematics), the inclusion–exclusion principle is a counting technique which generalizes the familiar method of obtaining the number of elements in the union of two finite or more sets.

For three sets A, B, C, the inclusion–exclusion principle is

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

Where, A, B, C are three finite sets.

Proof : Using Venn diagram



Now, we can also prove it otherwise :

Lets say we have three sets A, B and C.

A big hint is to prove the result for three sets, A, B, C, given the result for two sets.

$|A \cup B| = |A| + |B| - |A \cap B|$

We start by writing the three-fold union as the union of two sets:

$|A \cup B \cup C| = |(A \cup B) \cup C|$

and so we can use the two-set result on $A \cup B$ and C, so

$|(A \cup B) \cup C| = |A \cup B| + |C| - |(A \cup B) \cap C|$

Now, we can use the two-set result on $|A \cup B|$ to get

$|A \cup B \cup C| = |A \cup B| + |C| - |(A \cup B) \cap C|$

$= |A| + |B| - |A \cup B| + |C| - |(A \cup B) \cap C|$

$= |A| + |B| + |C| - |A \cap B| - |(A \cup B) \cap C|$

Now, use the distributive property on the last term:

$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B|$

$- |(A \cap B)| \cup (B \cap C|$

and use the two-set property yet again on the last term, with sets $A \cap C$ and $B \cap C$ to get

$|(A \cap C) \cup (B \cap C)| = |(A \cap C)| + |(B \cap C)|$

$- |(A \cap C) \cap (B \cap C)|$

$= |(A \cap C)| + |(B \cap C)| - |A \cap B \cap C|$

Finally, we substitute this into our big expression (remembering that it was negated) to get

$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B|$

$- |A \cap C| - |B \cap C| + |A \cap B \cap C|$

Ans.(b) Given that,

Total (N) = 250

Pascal(P) = 180

C++ (C) = 150

Java (J) = 120

PC = 80

CJ = 60

PJ = 40

PCJ = 35

No. of students who have taken atleast one course

$= P \rightarrow C \rightarrow J$

$= P + C + J - PC - CJ - PJ + PCJ$

$= 180 + 150 + 120 - 80 - 60 - 40 + 35$

$= 305$

Looks like the total no. of students given is wrong. Most probably it will be N = 350. In that case,

(i) How many students have not taken any course?

= Total – (No.of students who have taken atleast one course)

= 350 – 305

= 45

(ii) How many study atleast one of the languages?

= no. of students who have taken atleast one course

= 305

(iii) How many study only java?

= J – PJ – CJ + PCJ

= 120 – 60 – 40 + 35

= 55

(iv) How many study Pascal and C++ but not Java?

= PC – PCJ

= 80 – 35

= 45

Ans.(c) Given that,

A = {1,1,1,2,2,3,4,4}

B = {1,2,4,4,5,5,5}

Since, we are performing set operations on A and B, then A and B are sets and therefore,

A = {1,1,1,2,2,3,4,4} = {1,2,3,4}

B = {1,2,4,4,5,5,5} = {1,2,4,5}

Now,

(i) $A \cup B = \{1,2,3,4,5\}$

(ii) $A \cap B = \{1,2,4\}$

(iii) $A - B = A - (A \cap B)$

$= \{1,2,3,4\} - \{1,2,4\}$

$= \{3\}$

(iv) $A + B = A \cup B$ {'+' is nothing but union of sets}

$= \{1,2,3,4,5\}$

Q.47 ***(a) Let R be a relation defined on a set of orderded pairs of positive integers such that for all*** $(x,y),(u,v) \in Z^+ \times Z^+$, (x,y) ***R*** (u, v) ***if and only if*** $\frac{u}{x} = \frac{v}{y}$. ***Determine whether R is an equivalence relation.***

(b) Let $A = \{1,2,3,4\}$ ***and*** $R = \{(a,b): a+b>4\}$ ***be a relation on A. Draw the graph of the relation R.***

(c) Let R be an equivalence relation on a set of positive integers defined by x R y if and only if $x \equiv y \pmod 3$. ***Then, find the equivalence class of 2 and also find the partition generated by the equivalence relation.*** ***[R.T.U. 2015]***

Ans.(a) For a relation to be equivalent, it needs to be reflexive, symmetric and transitive.

Reflexive: A relation is said to be reflexive if it contains (a,a) for all possible a

R is reflexive because (x,y)R(u, v) is contained in R for all combinations of (x,y) because $\frac{u}{x}=\frac{v}{y}=1$.

Symmetry: A relation is said to be symmetric if for every (a,b) pair it contains, it also contains a corresponding (b,a) pair.

R is symmetric because if a pair ((x,y),(u,v)) satisfies the relation (u/x = v/y) then it also satisfies the relation (x/u = y/v) and hence pair ((u,v),(x,y)) is also in the relation.

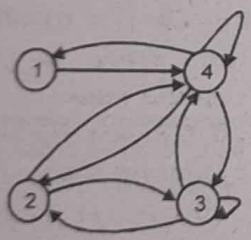
Transitive: A relation is said to be transitive if for every (a,b) and (b,c) pair it contains, it also contains an (a,c) pair.

R is transitive because for all pairs ((x,y),(u,v))&((u,v),(w,z)) it contains, it satisfies the relation x/w = y/z by the mathematical property a= b & b=c implies a = c.

Since, R is reflexive, symmetric and transitive, R is an equivalent relation.

Ans.(b) A = {1, 2, 3, 4} R = {c, b) | a + b > 4}. Draw graph of R.

R contains {(1, 4), (2, 3), (2, 4), (3, 3), (3, 4), (3, 2), (4, 1), (4, 2), (4, 3), (4, 4)}



Graph of R has an edge between vertex i & j if (i, j) ∈ R.

Ans.(c) Here equivalence relation is

$x \equiv y \pmod 3$

Which means on dividing y by 3 the remainder should be equal to reminder when x is divided by 3.

Relation is defmed on set of positive integers.

$x \equiv y \pmod 3$

is read "x is congruent to y modulo 3d"

Here (4, 7) will belong to the relation as dividing 4 by 3 gives remainder 1 & dividing 7 by 3 also gives 1.

R = {(1, 4), (4, 1), (2, 5), (4, 7), (7, 4) ...; (x, y)|x ≡ y mod 3}

Equivalence Class of 2

It is the class of (x, y) from the relation of such y values which relates to x = 2.

As (2, 2), (2, 5), (2, 8), (2, 11) ∈ R

Equivalence Class of 2, [|Z|] = {2, 5, 8, 11, 14, 17, ...}

For final partition by equivalence relation will find all classes,

[|0|] = {3, 6, 9, 12, 15, ...}

[|1|] = {1, 4, 7, 10, 13, 16, ...}

[|2|] = {2, 5, 8, 11, 14, 17, ...}

[|3|] = {3, 6, 9, 12, 15, ...} → similiar to [|0|]

So no further partioning by class. Partition by equivalence relation can be written as

[{1, 4, 7, 10, 13, 16, ...}, {2, 5, 8, 11, 14, 17, ...}, {3, 6, 9, 12, 15, ...}]

Q.48 ***(a) If A, B, C be finite sets then***

$$n(A \cup B \cup B) = n(A) + n(B) + n(C) - n(A \cap B) - (B \cap C) - n(C \cap A) + n(A \cap B \cap C)$$

(b) Participation in sports is compulsory in a school. In a class of 80 students, 60 play football 40 play basket ball. Find

(i) How many play both the games

(ii) Play football only *[R.T.U. 2013]*

Ans. (a) Suppose that $A \cup B = X$

Then $n(X) = n(A) + n(B) - n(A \cap B)$ ...(1)

Also $n(A \cup B \cup C) = n(X \cup C)$

$= n(X) + n(C) - n(X \cap C)$

$= n(X) + n(C) - n[(A \cup B) \cap C]$

$= n(X) + n(C) - n[(A \cap C) \cup (B \cap C)]$

$= n(X) + n(C) - \{n(A \cap C) + n(B \cap C) - n(A \cap C \cap B)\}$ ...(2)

$= n(A) + n(B) + n(C) - n(A \cap B) - n(B \cap C) - n(C \cap A) + n(A \cap B \cap C)$

Ans. (b) Let F and B are the costs of students who play football and basket ball, respectively,

Given

$|F| = 60$

$|B| = 40$

We know that

(i) $|A \cup B| = |A| + |B| - |A \cap B|$

$80 = 60 + 40 - |A \cap B|$.

$20 = |A \cap B|$

So 20 students play both the games.

(ii) Play football only

$\because$ given that 60 play football and as we calculated that 20 students play both games so, students who play football only are,

$= 60 - 20 = 40$ students.

Q.49 *(a) Let R be an equivalence relation an A, and let p be the collection of all distinct equivalence classes [a] for $a \in A$. Then show that P is a partition of A and R is the equivalence relation determine by P.*

(b) In the set of natural number $N = \{1, 2, ...\}$ show that the relation defined as $a\ R\ b \Leftrightarrow a = b^k$ for $a, b, R \in N$ is a partial order relation.

[R.T.U. 2012; Raj. Univ. 2006, 2002, MNIT 2003]

Ans. (a) **Theorem :** Let X be a non-empty set. Then every partition of X induces an equivalence relation on X, and every equivalence relation induces a partition of X.

Proof : Let R be an equivalence relation on X, Let P be the collection of distinct equivalence classes of X wrt R :

$P = \{(x)\} : x \in X.$

It's clear that the union of subsets [x] of X in P is all of X as x ranges over X. If $z \in [x] \cap [y]$ then (x, z), (y, z) $\in$ R, which, by symmetry and transitivity gives (x, y) $\in$ R, so [x] = [y]. It follows that P is a partition of X.

On the other hand if P is a partition of X, define a relation R on X by (x, y) $\in$R if x and y belong to the same subset of X contained in P. Clearly R is reflexive since the union of the subsets of X in P is X. R is clearly symmetric, and finally if x, y $\in$ P and y, z $\in$ P and x, z $\in$ P, so R is transitive.

Since R is reflexive, symmetric and transitive, it follows that R is an equivalence relation. Note the the equivalence classes are exactly the members of P.

Ans. (b) The relation R defined on set of natural numbers is

$a\ R\ b \Leftrightarrow a = b^k \qquad \forall a, b, k \in N$

(i) To show R is reflexive

$\forall a \in N$ we have $a = a^1, 1 \in N$

$\therefore$ R is reflexive

(ii) To show R is antisymmetric we have to prove that if aRb then bRA when a = b

Suppose a R b and bRa .

$\Rightarrow a = b^{k_1}$ and $b = a^{k_2}, k_1 k_2 \in N$

$\therefore a = b^{k_1} = \left(a^{k_2}\right)^{k_1} = a^{k_1 k_2} \quad \therefore k_1 = k_2 = 1$

Since $k_1, k_2 \in N$, we must have $k_1 = k_2 = 1$ thus $a = b$

$\therefore$ R is antisymmetric.

(iii) To show R is transitive

Let aRb and bRc then

$a = b^{k_1}$ and $b = c^{k_2} \quad k_1, k_2 \in N$

$\because a = b^{k_1} = c^{k_1 k_2} \quad as\ k_1 k_2 \in N$

$\Rightarrow a = c^k \qquad \{k = k_1 k_2 \in N\}$

$\Rightarrow a\ R\ c$ $\therefore$ R is transitive

Since R is reflexive , antisymmetirc and transitive .

Hence R is a partial order relation.

Q.50 *(a) Let A, B and C are arbitrary sets, show that :*

(i) $(A - B) - C = A - (B \cup C)$

(ii) $(A - B) - C = (A - C) - B$

(iii) $(A - B) - C = (A - C) - (B - C)$

Do not prove with examples. Give mathematical reasoning.

(b) Let A, B, C be sets. Under what conditions each of the following statements are true ?

(i) $(A - B) \cup (A - C) = A$ (ii) $(A - B) \cup (A - C) = \phi$

(iii) $(A - B) \oplus (A - C) = \phi$

Justify your answer. *[Raj. Univ. 2005, 2003, 1998, 1995]*

Ans. (a) (i) Let $x \in (A - B) - C$

$\Rightarrow x \in (A - B)$ and $x \notin C$

$\Rightarrow (x \in A$ and $x \notin B)$ and $x \notin C$

$\Rightarrow x \in A$ and $x \notin B$ and $x \notin C$

$\Rightarrow x \in A - (B \cup C)$

Hence $(A - B) - C \subseteq A - (B \cup C)$ .....(1)

Again let $x \in A - (B \cup C)$

$\Rightarrow x \in A$ and $x \notin (B \cup C)$

$\Rightarrow$ $x \in A$ and $(x \notin B$ and $x \notin C)$

$\Rightarrow$ $(x \in A$ and $x \notin B)$ and $x \notin C$

$\Rightarrow$ $x \in (A - B)$ and $x \notin C$

$\Rightarrow$ $x \in (A - B) - C$

Hence $A - (B \cup C) \subseteq (A - B) - C$ .....(2)

$\therefore$ from eq. (1) and (2)

$\Rightarrow (A - B) - C = A - (B \cup C)$ **Hence Proved.**

(ii) Let $x \in (A - B) - C$

$\Rightarrow$ $x \in (A - B)$ and $x \notin C$

$\Rightarrow$ $(x \in A$ and $x \notin B)$ and $x \notin C$

$\Rightarrow$ $(x \in A$ and $x \notin C)$ and $x \notin B$

$\Rightarrow$ $x \in (A - C)$ and $x \notin B$

$\Rightarrow$ $x \in (A - C) - B$

Hence $(A - B) - C \subseteq (A - C) - B$ .....(1)

Again Let $x \in (A - C) - B$

$\Rightarrow$ $x \in (A - C)$ and $x \notin B$

$\Rightarrow$ $(x \in A$ and $x \notin C)$ and $x \notin B$

$\Rightarrow$ $(x \in A$ and $x \notin B)$ and $x \notin C$

$\Rightarrow$ $x \in (A - B)$ and $x \notin C \Rightarrow x \in (A - B) - C$

Hence $(A - C) - B \subseteq (A - B) - C$ .....(2)

$\therefore$ from (1) and (2)

$(A - B) - C = (A - C) - B$ **Hence Proved.**

(iii) Let $x \in (A - B) - C$

$\Rightarrow$ $x \in (A - B)$ and $x \notin C$

$(x \in A$ and $x \notin B)$ and $x \notin C$

$\Rightarrow$ $(x \in A$ and $x \notin C)$ and $x \notin B$

$\Rightarrow$ $x \in (A - C)$ and $x \notin (B - C)$

$\Rightarrow$ $x \in (A - C) - (B - C)$

Hence $(A - B) - C \subseteq (A - C) - (B - C)$ .....(1)

Again, let $x \in (A - C) - (B - C)$

$\Rightarrow$ $x \in (A - C)$ and $x \notin (B - C)$

$\Rightarrow$ $(x \in A$ and $x \notin C)$ and $x \notin (B - C)$

$\Rightarrow$ $(x \in A$ and $x \notin B)$ and $x \notin C$

$[\because x \notin C, x \notin (B - C) \Rightarrow x \notin B]$

$\Rightarrow$ $(x \in A$ and $x \notin B)$ and $x \notin C$

$\Rightarrow$ $x \in (A - B)$ and $x \notin C$

$\Rightarrow$ $x \in (A - B) - C$

$\Rightarrow$ $(A - C) - (B - C) \subseteq (A - B) - C$ .....(2)

$\therefore$ From eq. (1) and eq. (2)

$(A - B) - C = (A - C) - (B - C)$ **Hence Proved.**

Ans.(b) (i) $(A - B) \cup (A - C) = A$

It is true if B or C is null set or .....(1)

A and B are disjoint sets or .....(2)

A and C are disjoint sets .....(3)

Since if eq. (1) is true then $B = \phi$ or $C = \phi$

$\Rightarrow A - B = A$ or $A - C = A$

and then $(A - B) \cup (A - C) = A$.

If eq. (2) is true then

$A - B = A$

$\Rightarrow (A - B) \cup (A - C) = A \cup (A - C) = A$

If eq.(3) is true then $A - C = A$

$\Rightarrow (A - B) \cup (A - C) = (A - B) \cup A = A$

(ii) $(A - B) \cup (A - C) = \phi$

It is true if eq. (1) $A = B = C$ or eq. (2) A is a null set *i.e.* $A = \phi$

As if eq. (1) is true *i.e.* $A = B = C$ then $A - B = \phi$ and $A - C = \phi$ so it follows that

$(A - B) \cup (A - C) = \phi \cup \phi = \phi$

If eq. (2) is true *i.e.* $A = \phi$ then $A - B = \phi - B = \phi$

and $A - C = \phi - C = \phi$ so it follows that

$(A - B) \cup (A - C) = \phi \cup \phi = \phi$

(iii) $(A - B) \oplus (A - C) = \phi$

It is true if

(1) $A \subseteq B$ and $A \subseteq C$ or

(2) $A \subseteq B$ and $A \subseteq C$ but $B = C$

As, if eq. (1) is true *i.e.* $A \subseteq B$ and $A \subseteq C = A - B = \phi$ and $A - C = \phi$

So it follows that $(A - B) \oplus (A - C) = \phi \oplus \phi$

$= (\phi \cup \phi) - (\phi \cap \phi) = \phi - \phi = \phi$

If eq. (2) is true *i.e.*

$A \notin B$, $A \notin C$ but $B = C$

then it follows that

$(A - B) \oplus (A - C) = (A - B) \oplus (A - B)$

$[(A - B) \cup (A - B)] - [(A - B) \cap (A - B)] = (A - B) - (A - B) = \phi$

❑❑❑

PROPOSITIONAL LOGIC 2

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 ***Define finite state machines.*** *[R.T.U. 2019]*

Ans. Finite automation as a machine equipped with an input tape. The machine works on a discrete time scale. At every point of time the machine is in one of its states, then it reads the next letter on the tape (the letter under the reading head), or maybe nothing (in the first variation), and then, according to the transition function (depending on the actual state and the letter being read, if any) it goes to a/the next state. It may happen in some variation that there is no transitions defined for the actual state and letter, then the machine gets stuck and cannot continue its run.

Q.2 ***In how many ways can a team of 11 cricketers be chosen for 6 bowlers, 4 wicket keepers and 11 batsmen to give majority of batsmen so that at least 4 bowlers are there and 1 wicketkeeper?*** *[R.T.U. 2019]*

Ans. 1 wicketkeeper can be selected in C(4, 1) ways

4 bowlers chosen = C(6, 4)

Remaining 6 batsmen = C(11, 6)

Total possibilities = C(4, 1) * C(6, 4) * C(11, 6) = 27720

The batsmen has to be majority. So the split cannot be 1 WC, 5 Bowlers, 5 Batsmen. It can only be 1WC, 4 bowlers and 6 batsmen.

Q.3 ***Show that $(p \wedge q) \rightarrow (p \vee q)$ is a tautology.*** *[R.T.U. 2017]*

Ans. $(p \wedge q) \rightarrow (p \vee q)$

First, we construct the truth table

p	q	$p \wedge q$	$p \vee q$	$p \wedge q \rightarrow p \vee q$
T	T	T	T	T
T	F	F	T	T
F	T	F	T	T
F	F	F	F	T

Since in the last column, all are true *i.e.* T therefore $p \wedge q \rightarrow p \vee q$ is a tautology.

Q.4 ***Find PCNF of a statement S whose PDNF is $(p \wedge q \wedge r) \vee (p \wedge q \wedge \sim r) \vee (\sim p \wedge \sim q \wedge r)$.*** *[R.T.U. 2017]*

Ans. First we obtain PDNF of ~s, which is the sum (disjunction) of those minterms which are not present in the given PDNF of s. Hence the PDNF of ~s is

$$(\sim P \wedge \sim q \wedge \sim r) \vee (\sim P \wedge \sim q \wedge r) \vee (P \wedge \sim q \wedge \sim r) \vee (\sim P \wedge q \wedge \sim r)$$

Thus, the PCNF of $S \equiv$ [PDNF of $(\sim s)$], i.e.

$$\equiv \sim ((\sim P \wedge \sim q \wedge \sim r) \vee (\sim P \wedge \sim q \wedge r) \vee (P \wedge \sim q \wedge \sim r) \vee (\sim P \wedge q \wedge \sim r))$$

$$\equiv (P \vee q \vee r) \wedge (p \vee q \vee \sim r) \wedge (\sim P \vee q \vee r) \wedge (P \vee \sim q \vee r)$$

Q.5 ***Explain the following for propositions with example:***

(i) Logical Equivalence

(ii) Tautological Implication

(iii) Normal Forms *[R.T.U. 2015]*

Ans.(i) Logical Equivalence : Any two propositions for wihich the truth table is same are said to be LOGICALLY EQUIVALENT.

Ex. $p \rightarrow q \;\&\; \neg p \vee q$

p	q	$p \to q$	$\neg q$	$(\neg p \vee q)$
F	F	T	T	T
F	T	T	T	T
T	F	F	F	F
T	T	T	F	T

(ii) Tautological Implication : Compound statements which are always true regardless of the truth or false of component statements are called tautologies. Obviously, the truth table of a tautology will contain only T entries in the last column.

Example : The statement $(p \to q) \leftrightarrow (\sim q \to \sim p)$ is a tautology

p	q	$p \to q$	$\sim q$	$\sim p$	$\sim q \to \sim p$	$(p \to q) \leftrightarrow (\sim q \to \sim p)$
T	T	T	F	F	T	T
T	F	F	T	F	F	T
F	T	T	F	T	T	T
F	F	T	T	T	T	T

(iii) Normal Forms : A literal L is either on after P or its negation ($\neg$P). A clause D is a disjunction literals. A formula C is in NORMAL FORM if it is a conjunction of clause.

$L \to P | \neg P$

$D \to L | L \vee D$

$C \to D | D \wedge C$

Example : (a) $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$

(a) $(p \vee r) \wedge (\neg p \vee r) \wedge (p \vee \neg r)$

Q.6 ***Over the universe of animals, let***
P(x) : x is a whale ; Q(x) : x is a fish
R(x) : x lives in water.
Translate the following into English
$\exists x (\sim R(x))$
$\exists x (Q(x) \wedge \sim P(x))$
$\forall x (P(x) \wedge R(x)) \to Q(x)$ *[R.T.U. 2014]*

Ans. $\exists$ x(~R(x)) : There exists an animal which does not lives in water.

$\exists$ x(Q(x) $\wedge$ ~ P(x)) : There exists a fish that is not a whale.

$\exists$ x(P(x) $\wedge$ R(x)) $\to$ Q(x) : Every whale that lives in the water, is a fish.

Q.7 ***Consider the following :***
p : It is hot today
q : The temperature is 35°C
Write in simple sentence the meaning of the following :
(i) $p \vee q$ ***(ii)*** $\sim(p \vee q)$ ***(iii)*** $\sim(p \wedge q)$
(iv) $\sim p \wedge \sim q$ *[R.T.U. 2013]*

Ans. Given

p : It is hot today

q : The temperature is 35°C

(i) $p \vee q$

Either it is hot today or the temperature is 35° C

(ii) $\sim(p \vee q)$

Neither it is hot today nor the temperature is 35° C

(iii) $\sim(p \wedge q)$

It is not hot today and the temperature is not 35° C

(iv) $\sim p \vee \sim q$

It is not hot today and the temperature is not 35° C

PART-B

Q.8 (a) Show that $\sim(pv(\sim p \wedge q)) \equiv (\sim p) \wedge (\sim q)$ ***(without truth table)***

(b) Write contrapositive converse and inverse of the statement "The home team wins whenever it is raining". Also construct the truth table for each statement. *[R.T.U. 2019]*

Ans.(a) $\sim(pv(\sim p \wedge q)) \equiv \sim p \wedge \sim(\sim p \wedge \sim q)$ [De Morgan's Law]

$\equiv \sim p \wedge (\sim\sim p \vee \sim q)$ [De Morgan's Law]

$\equiv \sim p \wedge (p \vee \sim q)$ [Double Negation Law]

$\equiv (\sim p \wedge p) \vee (\sim p \wedge \sim q)$ [Distributive Law]

$\equiv F \vee (\sim p \wedge \sim q)$ [$\because \sim p \wedge p \equiv F$]

$\equiv \sim p \wedge \sim p$

Ans.(b) The given proposition is in the form "q whenever p" such that,

q (Conclusion) : The home team wins.

p (hypothesis) : It is raining.

Converse : $q \to p$ is "If the home team wins then it is raining",

Inverse : $\neg p \to \neg q$ is "If it is not raining then the home team does not win".

Contra positive: $\neg q \to \neg p$ is "If the home team does not win then it is not raining".

p	q	$p \to q$	$q \to p$	$\sim p \to \sim q$	$\sim q \to \sim p$
T	T	T	T	T	T
T	F	F	T	T	F
F	T	T	F	F	T
F	F	T	T	T	T

Q.9 ***Draw the transition diagram of the finite state machine M (I, S, O, S_0, f, g), where I = {a, b}, S = {S_0, S_1}, O = {0, 1} and the transition table is as follows-***

	f		g	
I / S	a	b	a	b
S_0	S_1	S_0	0	1
S_1	S_0	S_1	1	0

Also, find the output string for the input b b a a.

[R.T.U. 2019]

Ans. FSM, M (I, S, O, S_0, f , g)

I = {a, b}

S = {S_0, S_1}

O = {0, 1}

A → I

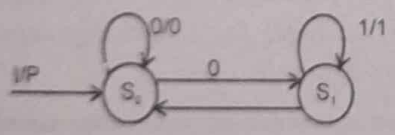
z → O

1. A finite set I of alphabet
2. A finite set S of internal state
3. A finite state Z of O/P symbol
4. An initial state S_0 in S
5. A next state function f from S × I into S.
6. An op state functions of from S × I into Z.

Transition of function f(f : S × I ⇒ P(S))

S_0 - initial state

F - finite state



Transitional diagram (Result)

Output string for bbaa is 1101.

Q.10 ***In a test 70% of the candidate passed in Science, 65% in Mathematics, 27% failed in both Science and Mathematics and 124 passed in both the subjects. Find the total number of candidates for the test.***

[R.T.U. 2019]

Ans. 70% passed in science, 65% passed in mathematics, 27% failed in both.

100 - 27 = 73%

n(S) = 70%, n(M) = 65%, n(S ∩ M) = 73%

n(S∪M) = n(S) + n(M) – n(S ∩ M)

= 70 + 65 – 73 = 62%

∵ 62 passed in both subjects then total no of students = 100

∵ 124 passed in both subjects then total no of students

$\therefore \frac{100}{62} \times 124 = 200.$

Q.11 ***Obtain the Principal disjunctive normal forms of $(p \wedge q) \vee (\sim p \wedge r) \vee (q \wedge r)$.*** *[R.T.U. 2019]*

Ans.

p	q	r	p∧q=a	~p	~p∧r=b	q∧r=c	a∨b∨c
T	T	T	T	F	F	T	T
T	T	F	T	F	F	F	T
T	F	T	F	F	F	F	F
T	F	F	F	F	F	F	F
F	T	T	F	T	T	T	T
F	T	F	F	T	F	F	T
F	F	T	F	T	T	F	T
F	F	F	F	T	F	F	F

PDNF of $(p \wedge q) \vee (\sim p \wedge r) \vee (q \wedge r)$

$= ((p \wedge q) \wedge (r \vee \sim r)) \vee ((\sim p \wedge r) \wedge (q \vee \sim q)) \vee ((q \wedge r) \wedge (p \vee \sim p))$

$= (p \wedge q \wedge r) \vee (p \wedge q \wedge \sim r) \vee (\sim p \wedge r \wedge q) \vee (\sim p \wedge r \wedge \sim q) \vee (q \wedge r \wedge p) \vee (q \wedge r \wedge \sim p)$

$= (p \wedge q \wedge r) \vee (p \wedge q \wedge \sim r) \vee (\sim p \wedge r \wedge q) \vee (\sim p \wedge r \wedge \sim p)$

(Deletion of identical min terms)

Q.12 ***(a) Define Tautology, contradiction and contingency. Determine the contrapositive of each statement:***

(i) If John is a poet, then he is poor.

(ii) Only if Mary studies will she pass the test.

(b) Determine the validity of the argument :

All men are fallible

All kings are Men

Therefore, all kings are fallible. *[R.T.U. 2017]*

Ans.(a) Tautology : A compound statement that is always true for all possible truth values of its propositional variables, is called a tautology. Obviously, its truth table contains only truth value T in the last column.

Example : *Refer to Q.5(ii).*

Contradiction : A compound statement that is always false, is called a contradiction. Obviously its truth table contains only false value in the last column.

Contingency : A statement that is neither a tautology nor a contradiction is called contingency. So its truth table contains both T and F values at least once in its last column.

The contrapositive of $p \to q$ is $\sim q \to \sim p$.

(i) Let p : John is a poet
q : He is poor.
Hence the contrapositive of the given statement is :
If join is not poor, then he is not a poet.

(ii) The given statement is equivalent to "If mary passes the test, then she studied".
Let p : Mary passes the test
q : she studied.
Hence its contrapositive is :
If Mary does not study, then she will not pass the test.

Ans.(b) Let
M(x) : x is a Man
K (x) : x is a king
F (x) : x is fallible
Then the argument takes the form

$$\forall x\left[M(x) \to F(x)\right]$$

$$\frac{\forall x\left[k(x) \to M(x)\right]}{\therefore \forall x\left[k(x) \to F(x)\right]}$$

A formal proof is as follows:

S.No.	Step	Reason
1.	$\forall x\,[M(x) \to F(x)]$	Premise 1
2.	$M(c) \to F(c)$	Step 1 and universal instantiation
3.	$\forall x[k(x) \to M(x)]$	Premise 2
4.	$K(c) \to M(c)$	Step 3 and universal instantiation
5.	$K(c) \to F(c)$	Hypothetical syllogism using step 2 and 4
6.	$\forall x[K(x) \to F(x)]$	Step 5 and universal generalization

Hence the argument is valid.

Q.13 ***(a) Prove that $(\sim p \wedge q) \to [\sim(q \to p)]$ is a tautology with constructing truth table.***

(b) Obtain the principle disjunctive normal form of $(p \wedge q) \vee (\sim p \wedge r) \vee (q \wedge r)$ by constructing truth table. *[R.T.U. 2016]*

Ans.(a) To prove : $(\sim p \wedge q) \to [\sim(q \to p)]$
Truth table is as follows:

p	q	~p	~p ∧ q	~p→(p→q)
T	T	F	F	T
T	F	F	F	T
F	T	T	T	T
F	F	T	F	T

Since all the values are true, it is a tautology.

Ans.(b) *Refer to Q.11.*

Q.14 ***Translate each of these statements into logical expressions using predicates, quantifiers and logical connectives.***

(i) No one is perfect
(ii) Not everyone is perfect
(iii) All your friends are perfect
(iv) One of your friend is perfect
(v) Everyone is your friend and perfect
(vi) Not every body is your friend or someone is not perfect. *[R.T.U. 2016]*

Ans. Let p(x): x is perfect
q(x) : x is your friend
And the universe of discourse be the set of all people.
Then,

(i) $\forall x \sim p(x)$

(ii) $\sim(\forall x(p(x)))$

(iii) $\forall x(q(x) \to p(x))$

(iv) $\exists x(p(x) \wedge q(x))$

(v) $\forall x(p(x) \wedge q(x))$

(vi) $(\forall x \sim q(x)) \vee (\exists x \sim p(x))$

Q.15 ***Explain why predicate logic is better approach than propositional logic for knowledge representation? Give some example also.*** *[R.T.U. 2016]*

Ans. (i) Predicate logic also known as predicate calculus, and quantification theory- is a collection of formal systems used in mathematics, philosophy, linguistics, and computer science. Predicate logic admits quantified variables over non-logical objects and allows the use of sentences that contain such variables, so that rather than just propositions such as Socrates is a man one can have expressions in the form X is a man where X is a variable. This distinguishes it from propositional logic, which does not allow quantifiers.

(ii) Though propositional logic is one of the simplest languages that demonstrates all the important points, it has a very limited ontology, making only the commitment that, the

world consists of facts. This made it difficult to represent even something as simple as the wumpus world. Predicate logic makes a stronger set of ontological commitments. The main one is that the world consists of objects, that is, things with individual identities and properties that distinguish them from other objects.

(iii) First-order logic can also express facts about all of the objects in the universe. This, together with the implication connective from propositional logic, enables one to represent : general laws or rules.

(iv) Firstorder logic is universal in the sense that it can express anything that can be programmed.

(v) It is by far the most studied and best understood scheme yet devised.

Q.16 ***Find the DNF of following:***

(i) $P \to ((p \to Q) \wedge \sim (\sim P \vee \sim P))$

(ii) $\sim (P \to (Q \wedge R))$. ***[R.T.U. 2015]***

Ans. (i) $A : P \circledR ((P \to Q) \wedge \neg (\neg P \vee \neg P))$

A DNF is an OR of AND

we know that $a \to b \equiv \neg a \vee b$

So, $A \equiv \neg P \vee ((P \to Q) \wedge \neg (\neg P \vee \neg P)$

we also know, $a \vee a \equiv a \ \& \ \neg (\neg a) = a$

So $A \equiv \neg P \vee ((P \to Q) \wedge P)$

$A \equiv \neg P \vee ((\neg P \vee Q) \wedge P)$

$A \equiv \neg P \vee ((\neg P \wedge P) \wedge (Q \wedge P))$

$A \equiv \neg P \vee T \vee (Q \wedge P)$ (To true)

$(\therefore a \wedge \neg a$ is a tautology)

$A \equiv \neg (P \to (Q \wedge p)) \to$ DNF

(ii) $B \equiv \neg (P \to (Q \wedge R))$

$B \equiv \neg (\neg P \vee (Q \wedge R))$

$\neg (\neg P \vee Q) \wedge (\neg P \vee R)$

By de Morgan's Law $\neg (a \wedge b) = (\neg a \vee \neg b)$

$B \equiv (\neg (\neg P \vee Q)) \vee (\neg (\neg P \vee R))$

$B \equiv (P \wedge \neg Q) \vee (P \wedge \neg R)$

Q.17 ***Without constructing the truth table, show that $(\sim p \wedge (P \vee Q)) \to Q$ is a tautology.*** ***[R.T.U. 2015]***

Ans. We have,

$(\sim P \wedge (P \vee Q)) \to Q$

$\equiv \ \sim(\sim P \wedge (P \vee Q)) \vee Q$

$\{$using, $A \to B \equiv \sim A \vee B\}$

$\equiv \ (P \vee (\sim(P \vee Q))) \vee Q$

$\equiv \ (P \vee (\sim P \wedge \sim Q)) \vee Q$

$\equiv \ ((P \vee \sim P) \wedge (P \vee \sim Q)) \vee Q$

$\equiv \ (\text{TRUE} \wedge (P \vee \sim Q)) \vee Q$

$\equiv \ (P \vee \sim Q) \vee Q$

$\equiv \ P \vee \sim Q \vee Q$

$\equiv \ \text{TRUE}$

Hence, it is a tautology.

Q.18 ***State the difference between deterministic and non deterministic finite automata.*** ***[R.T.U. 2015, 2014]***

Ans. Deterministic and Non-deterministic Finite Acceptors

S. No.	Deterministic Finite Acceptors	Non- deterministic Finite Acceptors
1.	For every symbol of the alphabet, there is only one state transition.	We do not need to specify how does the NFA react according to some symbol.
2.	Cannot use empty string transition.	Can use empty string transition.
3.	Can be understood as one machine.	Can be understood as multiple little machines computing at the same time.
4.	It will reject the string if it end at other than accepting state.	If all of the branches of NFA dies or rejects the string, we can say that NFA reject the string.
5.	The transition function is single valued.	The transition function is multi-valued.
6.	Checking membership is easy.	Checking membership is difficult.
7.	Construction is difficult.	Construction is easy.
8.	Space required is more.	Space required is comparatively less.
9.	Backtracking is allowed.	Not possible in every case.
10.	Can be constructed for every input and output.	Cannot be constructed for every input and output.
11.	There can be more than one final state.	There can only be one final state.

Q.19 ***Prove $pV(q \Lambda r) \equiv (p V q) \Lambda (p V r)$.*** ***[R.T.U. 2014]***

Ans. $pV(q \Lambda r) \equiv (p V q) \Lambda (p V r)$

p	q	r	$q \wedge r$	$p \vee (q \wedge r)$	$p \vee q$	$p \vee r$	$(p \vee q) \wedge (p \vee r)$
T	T	T	T	T	T	T	T
T	T	F	F	T	T	T	T
T	F	T	F	T	T	T	T
T	F	F	F	T	T	T	T
F	T	T	T	T	T	T	T
F	T	F	F	F	T	F	F
F	F	T	F	F	F	T	F
F	F	F	F	F	F	F	F

Since the column (5) and (8) are identical.

So we have $pV(q \Lambda r) \equiv (p V q) \Lambda (p V r)$.

Q.20 ***Show that propositional formula*** $(p \wedge q) \wedge (r \wedge s) \Rightarrow P$ ***for any propositions p,q,r,s is a tautology.*** *[R.T.U. 2013, 07]*

Ans. Given $(p \wedge q) \wedge (r \wedge s) \Rightarrow P$

p	q	r	s	$(p \wedge q)$	$(r \wedge s)$	$(p \wedge q) \wedge (r \wedge s)$	$(p \wedge q) \wedge (r \wedge s) \rightarrow p$
T	T	T	T	T	T	T	T
T	T	T	F	T	F	F	T
T	T	F	T	T	F	F	T
T	F	T	T	F	T	F	T
F	T	T	T	F	T	F	T
T	T	F	F	T	F	F	T
T	F	F	T	F	F	F	T
F	F	T	F	F	F	F	T
F	T	T	F	F	F	F	T
T	F	T	T	F	T	F	T
F	T	F	F	F	F	F	T
T	F	F	F	F	F	F	T
F	F	F	T	F	F	F	T
F	F	T	F	F	F	F	T
F	T	F	F	F	F	F	T
F	F	F	F	F	F	F	T

$\because (p \wedge q) \wedge (r \wedge s) \rightarrow p$ is always true

Hence proved that the given propositional formula is a tautology.

Q.21 ***Using propositional Logic, prove the validity of the argument.*** $[(p \vee \sim q)] \Rightarrow r \wedge (r \Rightarrow s) \wedge p \Rightarrow s$ *[R.T.U. 2012]*

Ans. Suppose

p : I will study Physics

q : I will study Maths.

r : I will study Accounts

s : I will study Computer.

$(p \vee \sim q)$: I will study Physics or I will not study Maths.

$(p \vee \sim q) \rightarrow r$: If I will study Physics or I will not study Maths, then I will study Accounts.

$r \rightarrow s$: If I will study Accounts, then I will study Computer.

$p \rightarrow s$: If I will study Physics, then I will study Computer.

From the last two arguments we can see that Computer will be studied if either Physics or Accounts is studied. From the second and third argument we can see that

$(p \vee \sim q) \rightarrow s$

Thus, if Physics is studied or Maths is not studied, then Computer is studied is valid because if Physics is studied then by fourth statement Computer is studied and if Maths is not studied then either Physics or Accounts is studied which leads to study of Computer.

Q.22 ***(a) Show that***

$p \rightarrow q \equiv (\sim p) \vee q$

(b) Show that $(p \wedge q) \wedge \sim (p \vee q)$ ***is a contradiction.*** *[R.T.U. 2011]*

Ans. (a)

p	q	$\sim p$	$p \rightarrow q$	$(\sim p) \vee q$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Thus, from truth table we have proved that $p \rightarrow q \equiv (\sim p) \vee q$.

Ans. (b)

p	q	$p \wedge q$	$p \vee q$	$\sim (p \vee q)$	$(p \wedge q) \wedge \sim (p \vee q)$
T	T	T	T	F	F
T	F	F	T	F	F
F	T	F	T	F	F
F	F	F	F	T	F

Since in last column all the false result show that $(p \wedge q) \wedge \sim (p \vee q)$ is a contradiction.

Q.23 ***(a) Obtain principal conjunctive normal form of S where***

$S \equiv (\sim p \rightarrow r) \wedge (q \leftrightarrow p)$.

(b) Check the validity of following argument

$p \vee q$

$p \rightarrow \sim q$

$p \rightarrow r$

---

r

[R.T.U. 2011, 2009]

Ans.(a) For obtaining the PCNF of S, we prepare the truth table

p	q	r	$\sim p$	$\sim p \rightarrow r$	$p \leftrightarrow q$	$(\sim p \rightarrow r) \wedge (p \leftrightarrow q)$
T	T	T	F	T	T	T
T	T	F	F	T	T	T
T	F	T	F	T	F	F
T	F	F	F	T	F	F
F	T	T	T	T	F	F
F	T	F	T	F	F	F
F	F	T	T	T	T	T
F	F	F	T	F	T	F

Here F appears in last column in III, IV, V, VI and VIII row. The required principal conjunctive normal form (PCNF) is $(\sim p \vee q \vee \sim r) \wedge (\sim p \vee q \vee r) \wedge (p \vee \sim q \vee \sim r) \wedge (p \vee \sim q \vee r) \wedge (p \vee q \vee r)$.

Ans.(b) Constructing truth table to verify the given argument is valid or not as :

Row No.	p	q	r	~ q	p_1 p ∨ q	p_2 p → ~ q	p_3 p → r	Q r
0	T	T	T	F	T	F	T	T
1	T	T	F	F	T	F	F	F
2	T	F	T	T	T	T	T	T
3	T	F	F	T	T	T	F	F
4	F	T	T	F	T	F	T	T
5	F	T	F	F	T	T	F	F
6	F	F	T	T	F	T	T	T
7	F	F	F	T	F	T	T	F

Since in 5th row all the three premises namely $p \vee q$, $p \to \sim q$ and $p \to r$ are true (T) but the corresponding truth value of the conclusion is false (F) so invalid.

Q.24 ***Define the Contradiction.*** *[R.T.U. 2011]*

Ans. Contradiction : If a compound statement is false for all value assignments for its component statements, then it is called a contradiction, *i.e.* a compound statement is said to be contradicting its truth value if it is false (F) for all its entries in the truth table. If a statement is a contradiction then its negation will be a tautology.

Example : The statement $p \wedge \sim p$ is a contradiction. The truth table for the given statement is as follows :

p	~p	p ∧ ~p
T	F	F
F	T	F

Q.25 ***Prove that the following implications are tautologies :***

(i) $p \wedge q \to p \vee q$

(ii) $\sim p \to (p \to q)$ *[R.T.U. 2011]*

Ans.(i) $p \wedge q \to p \vee q$: *Refer to Q.3.*

(ii) $\sim p \to (p \to q)$

p	q	~p	p → q	~p → (p → q)
T	T	F	T	T
T	F	F	F	T
F	T	T	T	T
F	F	T	T	T

Since in the last column all entries are true.

So $\sim p \to (p \to q)$ is a tautology.

Q.26 ***Prove the following equivalences :***

(i) $(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$

(ii) $(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$ *[R.T.U. 2010]*

Ans.(i) $(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$

p	q	r	p → q	p → r	p ∧ r	(p → q) ∧ (p → r)	p → (q ∧ r)
T	T	T	T	T	T	T	T
T	T	F	T	F	F	F	F
T	F	T	F	T	F	F	F
T	F	F	F	F	F	F	F
F	T	T	T	T	T	T	T
F	T	F	T	T	F	T	T
F	F	T	T	T	F	T	T
F	F	F	T	T	F	T	T

From the 7 and 8 columns we see that

$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$ **Hence proved**

(ii) $(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$

p	q	r	p → r	q → r	(p ∨ r)	(p → r) ∧ (q → r)	(p ∨ q) → r
T	T	T	T	T	T	T	T
T	T	F	F	F	T	F	F
T	F	T	T	T	T	T	T
T	F	F	F	T	T	F	F
F	T	T	T	T	T	T	T
F	T	F	T	F	F	F	F
F	F	T	T	T	T	T	T
F	F	F	T	T	F	T	T

From the last two columns we see that

$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$ **Hence proved**

PART-C

Q.27 Define fallacy and prove the following-

$(p \wedge q) \vee \sim (p \wedge q)$ ***is*** $\wedge$ ***fallacy*** *[R.T.U. 2019]*

Ans.Fallacy : A compound preposition, that is always false for any assignment of false value to the prepositional variable is called fallacy.

p	q	p ∧ q	~ (p ∧ q)	(p ∧ q) ∨ ~ (p ∧ q)
T	T	T	F	F
T	F	F	T	F
F	T	F	T	F
F	F	F	T	F

$(p \wedge q) \vee \sim (p \wedge q)$ fallacy.

Q.28 *Define tautology and prove the following-*
$\{(p \rightarrow q) \wedge p\} \rightarrow q$ *is tautology* *[R.T.U. 2019]*

Ans. Tautology : A compound preposition that is always true for any assigned of the truth values to the prepositional variable is called Tautology

p	q	$p \rightarrow q$	$(p \rightarrow q) \wedge p$	$(p \rightarrow q) \wedge p \rightarrow q$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

$\{(p \rightarrow q) \wedge p\} \rightarrow q$ Tautology

Q.29 *Describe the block diagram of a finite automaton. Consider the transition system given below :*

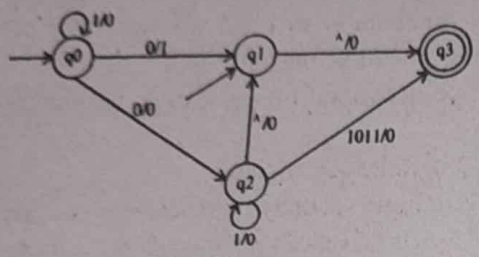


Fig.

Determine the initial states, the final state and the acceptability of 101011 and 111010. *[R.T.U. 2016]*

Ans. Finite Automaton : A finite automaton can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

(i) Q is a finite non-empty set of states.

(ii) Σ is a finite non-empty set of inputs called input alphabets.

(iii) δ is a function which maps $Q \times \Sigma$ into Q and is usually called direct transition function. This is the function which describes the changes of states during the transition. This mapping is usually represented by a transition table or a transition diagram.

(iv) $q_0 \in Q$ is the initial state.

(v) $F \subseteq Q$ is the set of final states. It is assumed here that there may be more than one final state.

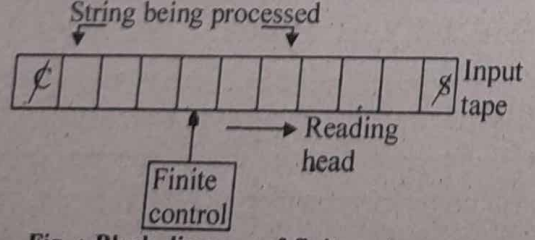


Fig. : Block diagram of finite automaton

(vi) Input Tape : The input tape is divided into squares, each square containing a single symbol from the input alphabet Σ. The end square of the tape contain end-markers ¢ at the left end and $ at the right end. Absence of end-markers indicates that the tape is of infinite length. The left-to-right sequence of symbol between the end markers is the input string to be processed.

(vii) Reading Head : The head examines only one square at a time and can move one square either to the left or to the right. For further analysis, we restrict the movement of R-head only to the right side.

(viii) Finite Control : The input to the finite control will be usually : symbol under the R-head, say a, or the present state of the machine, say q, to give the following outputs : (a) A motion of R-head along the tape to the next square (In some a null move *i.e.* R-head remaining to the same square is permitted); (b) The next state of a finite state machine given by δ (q, a).

Initial states are q_0 and q_1.

Final state is q_3.

For checking the string acceptability, we start the string from initial state. If we reach the final state after completing the string, then we say that this string is accepted by transition system or not.

For string 101011, the path value is $q_0\ q_0\ q_2\ q_3$. Since q_3 is the final state so this string is accepted by above transition system.

For string 111010, there is no path value. So this string is not accepted by the above transition system.

Q.30 *(a) Determine whether the conclusion C follows logically from the premises H_1, H_2 and H_3.*

$H_1 : P \vee Q$

$H_2 : P \rightarrow R$

$$\frac{H_3 : \sim Q \vee S}{C : S \vee R}$$

(b) Explain the followings:

(i) Argument

(ii) Predicates

(iii) Quantifiers *[R.T.U. 2015]*

Ans. (a) We know that,

$A \rightarrow B \equiv \sim A \vee B$

Now we have,

$P \vee Q$

$\sim P \rightarrow Q$ {from H1}

$Q \rightarrow S$

$\sim P \rightarrow S$ {From H3}

$\equiv P \vee S \equiv \sim S \rightarrow P$

Hence,

$\sim S \rightarrow R$

$P \rightarrow R$

$\sim S \rightarrow R \equiv R \vee S$ {from H2}

Hence, yes, the conclusion C follows logically from the premises H1, H2 and H3.

Ans.(b) (i) An arangument is a statement which gives some facts and then claims something. The facts are called a premise & the claim is CONCLUSION. A sound argument is an argument which is valid and which has true premises. A valid argument is an argument in which if the premises are true then conclusion must be true.

(ii) Predicates : A function P which can take two values, TRUE & FALSE

$P : X \rightarrow \{True, False\}$ (X is any set)

(iii) Quantifiers

The way in which sentence can be turned into a proposition is quantification. A *quantifier* is an operator that limits the variables of a proposition. The two quantifiers, are the universal and existential quantifiers are as follows :

(1) The Universal Quantifier : Suppose that P(x) is a propositional function with domain D. The universal quantification of P(x) is the proposition that asserts that P(x) is true for all values of x in the universe of discourse D *i.e.* 'P(x) is true for all values of x in D' written '$\forall x\, P(x)$' and read 'for all x P(x)' or 'for every x P(x)'. The symbol $\forall$ is read as 'for all' or 'for every'.

(2) The Existential Quantifier : With existential quantification, we form a proposition that is true if and only if P(x) is true for at least one value in the universe of discourse.

The existential quantification of P(x) is the proposition. 'There exists an element x in domain D such that P(x) is true' denoted by '$\exists x\, P(x)$' read as 'There exists x such that P(x)' or 'for some x P(x)' or 'There is at least one x such that P(x).' The symbol $\exists$ is read as 'there exists'.

Properties of Quantifiers

The negation of a quantified statement changes the quantifier and also negates the given statement as mentioned below :

(i) $\sim \{\forall x\, P(x)\} \equiv \exists x \sim P(x)$ [De Morgan's law]

(ii) $\sim (\exists x P(x)) \equiv \forall x \sim P(x)$ [De Morgan's law]

(iii) $\exists x P(x) \rightarrow Q(x) \equiv \forall x P(x) \rightarrow \exists x Q(x)$

(iv) $\exists x\, P(x) \rightarrow \forall x Q(x) \equiv \forall x \big(P(x) \rightarrow Q(x)\big)$

(v) $\exists x \big(P(x) \vee Q(x)\big) \equiv \exists x P(x) \vee \exists x Q(x)$

(vi) $\sim \big(\exists x \sim P(x)\big) \equiv \forall x\, P(x)$.

Q.31 ***Discuss Mealy & Moore machines.***

[R.T.U. 2015]

Ans. Finite automata may have outputs corresponding to each transition. There are two types of finite state machine that generate output

(i) Mealy Machine

(ii) Moore Machine

Mealy Machine:

Mealy Machine is an FSM whose output depends on present state as well as present input

It can be described by a six tuple $(Q, \Sigma, O, \delta, X, g_0)$ where :

- Q is finite set of states
- Σ is finite set of symbols called input alphabets
- O is finite set of symbols called output alphabets
- δ is input transition function where $\delta : Q \times \Sigma \rightarrow Q$
- X is output transition function where $X : Q - O$
- g_0 is initial state from where any input is processed ($g_0, \in Q$). State diagram of Mealy machine is shown below :

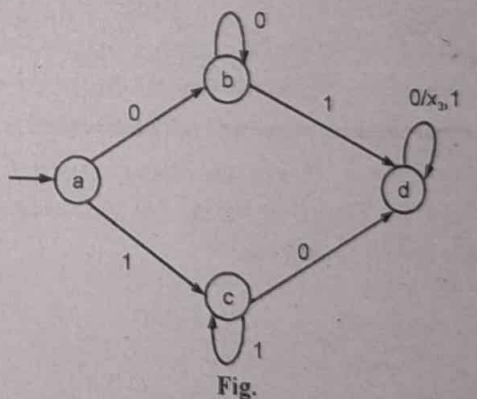


Fig.

Moore Machine:

Moore Machine is an FSM whose outputs depend on only present state. It can be described by a 6 tuple $(Q, \Sigma, O, \delta, X, g_0)$ where :

- Q is a finite set of states
- Σ is a finite set of symbols called input alphabets
- O is a finite set of symbols called output alphabets
- δ is input transition function where $\delta : Q \times \Sigma \rightarrow Q$
- X is the output transition function where $X : Q \times \Sigma \rightarrow 0$

- g_0 is initial state from where any input is processed ($g_0 \in Q$). State diagram is as shown below :

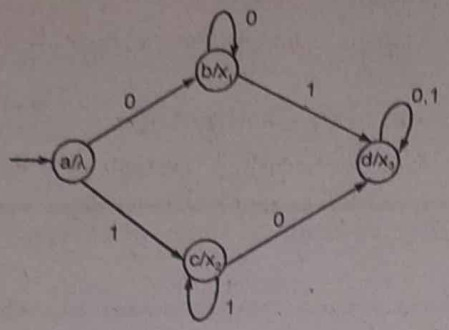


Fig.

Mealy Machine v/s Moore Machine

S. No.	Mealy Machine	Moore Machine
(i)	Output depends both upon present state and present input.	Output depends only upon present state.
(ii)	Generally, it has fewer states than Moore machine.	Generally, it has more states than Mealy machine
(iii)	Output changes at clock edges.	Input change can cause change in output change as soon as logic is done.
(iv)	Mealy machines react faster to inputs.	In moore machines, more logic is needed to decode the output since it has more circuit delays.

Q.32 Consider a Mealy machine given by transition diagram. Construct a moore machine equivalent to this mealy machine.

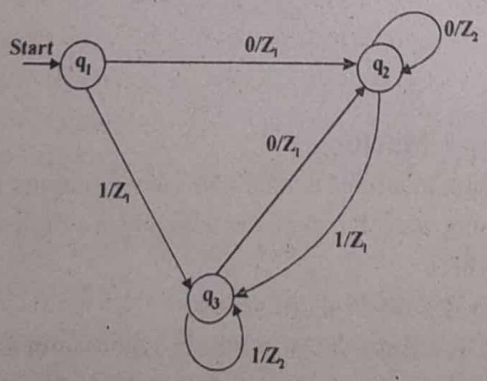


Fig.

[R.T.U. 2013, 2009]

Ans. First of all we have to convert the transition diagram into transition table as follows :

Present state	Next state			
	Input 0	Output	Input 1	Output
→ q_1	q_2	Z_1	q_3	Z_2
q_2	q_2	Z_2	q_3	Z_2
q_3	q_2	Z_1	q_3	Z_2

Step 1 : We look in the next state column for states q_1, q_2 and q_3.

q_1 is associated with no output, q_2 and q_3 are associated with two different outputs *i.e.* Z_1 and Z_2.

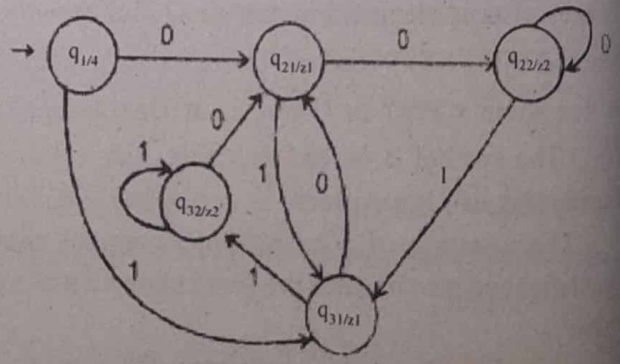
Step 2 : We don't split q_1 now we split q_2 and q_3 into q_{21}, q_{22} and q_{31}, q_{32} respectively. After splitting these state, the transition table we get is as follows :

Present state	Next state			
	Input 0	Output	Input 1	Output
→ q_1	q_{21}	Z_1	q_{31}	Z_2
q_{21}	q_{22}	Z_2	q_{31}	Z_2
q_{22}	q_{22}	Z_2	q_{31}	Z_2
q_{31}	q_{21}	Z_1	q_{32}	Z_1
q_{32}	q_{21}	Z_1	q_{32}	Z_1

Step 3 : Now we rearrange the transition table in such a way that each state in present state column is associated with a single output.

Present state	Next state		Output
	Input 0	Input 1	
→ q_1	q_{21}	q_{31}	0
q_{21}	q_{22}	q_{31}	Z_1
q_{22}	q_{22}	q_{31}	Z_2
q_{31}	q_{21}	q_{32}	Z_1
q_{32}	q_{21}	q_{32}	Z_2

The transition diagram for Moore Machine.



This is the required solution.

Q.33 *Write a note on :*

(a) First Order Logic

(b) Finite state machine as language recognizer

Ans. (a) First Order Logic :

- For some applications, propositional logic is not expressive enough but First-order logic is more expressive: allows representing more complex facts and making more sophisticated inferences.

  For instance, consider the statement "Anyone who drives fast gets a speeding ticket"

- From this, we should be able to conclude "If Joe drives fast, he will get a speeding ticket"
- Similarly, we should be able to conclude "If Rachel drives fast, she will get a speeding ticket" and so on.
- But Propositional Logic does not allow inferences like that because we cannot talk about concepts like "everyone", "someone" etc.
- First-order logic (predicate logic) allows making such kinds of Inferences.

Building Blocks of First-order Logic

- The building blocks of propositional logic were propositions.
- In first order logic, there are three kinds of basic building blocks, constants, variables, predicates.
- **Constants :** refer to specific objects (in a universe of discourse)

  Examples : George, 6, Austin CS311,.....
- **Variables :** range over objects (in a universe of discourse)

  Example : x,y,z, ...
- In universe of discourse is cities in Texas, x can represent Houston, Austin, Dallas, San Antonio....
- **Predicates :** describe properties of objects or relationships between objects.

  Example : ishappy, better than, loves, >...
- Predicates can be applied to both constants and variables.

  Example : ishappy (George), better than (x,y) loves (George, Rachel), x > 3, ...
- A predicate P(x) is true or false depending on whether property P holds for x.

  Example : ishappy (George) is true if George is happy, but false otherwise.

Semantics of First-Order Logic

- In propositional logic, the truth value of formula depends on a truth assignment to variables.
- In FOL, truth value of a formula depends interpretation of predicate symbols and variable over some domain D.
- Consider a FOL formula –P(x)
- A possible interpretation:

  D = (*,o), P(*) = true, P(o) = false, x = *
- Under this interpretation, what's value of –P(x)?
- What about of x = o?.

Ans.(b) Finite state machine as language recognizer :

Let A =(Q, T, a_0, δ, F). It is a finite automation (recognizer), where Q is the finite set of (inner) states, T is the input (or tape) alphabet, $a_0 \in Q$ is the initial state, $F \subseteq Q$ in the set of final (or accepting) states and δ is the transition function as follows.

- $\delta : Q \times (T \cup \{\lambda\}) \rightarrow 2^Q$ (for nondeterministic finite automata with allowed λ-transitions):
- $\delta : Q \times T \rightarrow 2^Q$ (for nondeterministic finite automata without λ-transitions):
- $\delta : Q \times T \rightarrow Q$ (for deterministic finite automata, λ can be partially defined):
- $\delta : Q \times T \rightarrow Q$ (for completely defined deterministic finite automata (it is not allowed that δ is partial function, it must be completely defined.))

One can observe, that the second variation is a special case of the first one (not having λ-transitions. The third variation is a special case of the second one having sets with at most one element as images of the transition function, while the fourth case is more specific allowing sets exactly with one element.

Finite State Machines : *Refer to Q.1.*

There are two widely used ways to present automata; by Cayley tables or by graph. When an automaton is given by a Cayley table, then the 0^{th} line and the 0^{th} column of the table are reserved for the states and for the alphabet, respectively (and it is marked in the 0^{th} element of the 0^{th}

row). In some cases it is more convenient to put the states in the 0^{th} row, while in some cases it is a better choice to put the alphabet there. We will look at both possibilities. The initial state should be the first among the states (it is advisable to mark it by a $\rightarrow$ sign also. The final states should also be marked, they should be circled. The transition function is written into the table: the elements of the set $\delta(q,a)$ are written (if any) in the field of the column and row marked by the state q and by the letter a. In the case when λ-transitions are also allowed, then the 0^{th} row of the column (that contains the symbols of the alphabet) should be extended by the empty word (λ)also. Then λ-transitions can also be indicated in the table.

Automata can also be defined in a graphical way: let the vertices (nodes, that are drawn as circles in this case) of a graph represent the states of the automaton (we may write the names of the states into the circles). The initial state is marked by an arrow going to it not from a node. The accepting states are marked by double circles. The labeled arcs (edges) of the graph represent the transitions of the automaton. If $p \in \delta(q, a)$ for some $p,q \in Q$, $a \in T \cup \{\lambda\}$, then there is an edge from the circle representing state q to the circle representing state p and this edge is labeled by a. (Note that our graph concept is wider here than the usual digraph concept, since it allow multiple edges connecting two states, in most cases these multiple edges are drawn as normal edg having more than 1 labels.)

In this way, implicitly, the alphabet is also given by t graph (only those letters are used in the automaton whi appear as labels on the edges).

In order to provide even more clarification, we prese an example. We describe the same nondeterministic both a table and by a graph.

TQ	$\rightarrow q_0$	q_1	$\subset q_2 \supset$	$\subset q_3 \supset$
a	q_1	q_1	q_2, q_3	-
b	q_0	q_0	-	q2
c	q_0	q_2	-	q_1, q_2, q_3

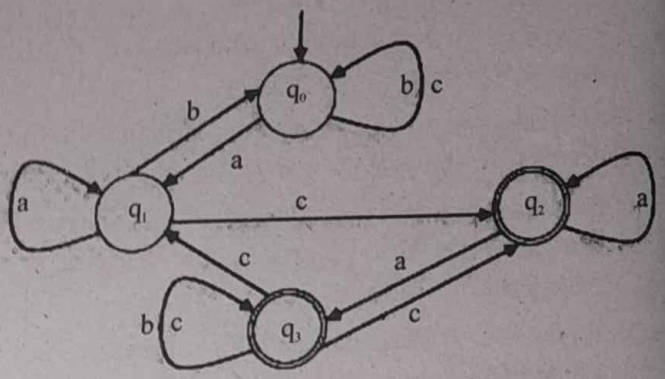


Fig.

POSETS, HASSE DIAGRAM AND LATTICES 3

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 *Prove that α^2 is an even integer, then α is an even integer.* *[R.T.U. 2019]*

Ans. Let a α^2 is an even integer

$\alpha^2 = 2k$, for some integer k

$$\alpha = \frac{2k}{\alpha}$$

So there is an integer $J = \frac{k}{\alpha}$, such that $\alpha = 2J$ α is an even.

Q.2 *Give an example of a partially ordered set which is not a lattice.* *[R.T.U. 2019]*

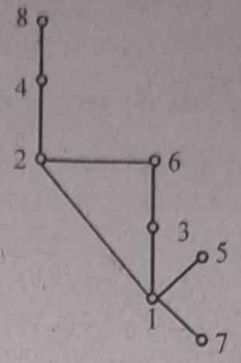
Ans. A partially ordered set $(X, \leq)$ is called a lattice if for every pair of elements $x, y \in X$ both the infimum and supremum of the set $\{x, y\}$ exists. I am trying to get an intuition for how a partially ordered set can fail to be a lattice. In R, for example, once two elements are selected the completeness of the real numbers guarantees the existence of both the infimum and supremum. Now, if we restrict our attention to a nondegenerate interval (a, b) it is clear that no two points in (a, b) have either a supremum or infimum in (a, b).

Q.3 *Draw a Hasse Diagram for (A), (divisibility relation), where*

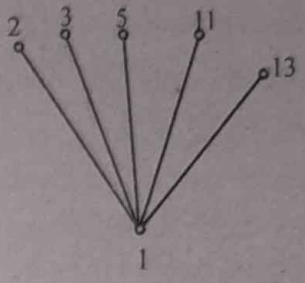
(i) A = {1,2,3,4,5,6,7,8}
(ii) A = {1,2,3,5,11,13};
(iii) A = {2,3,4,5,6,30,60};
(iv) S = {1,2,3,6,12,24,};

Ans.

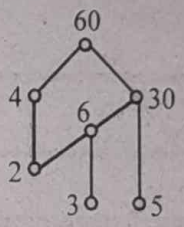
(i) A = {1,2,3,4,5,6,7,8}



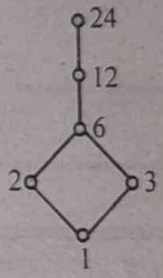
(ii) A = {1,2,3,5,11,13}



(iii) A = {2,3,4,5,6,30,60}



(iv) S = {1,2,3,6,12,24,}



Q.4 *Solve the following:*

(a) In a city, the bus route numbers consists of a natural number less than 100, followed by one of the letters A,B,C,D,E and F. How many different bus routes are possible?

(b) There are 3 question in a question paper. If the questions have 4,3 and 2 solutions respectively, find the total number of solutions.

Ans.(a) The number can be any me of the natural numbers from 1 to 99.

There are 99 choices for the number.

The letter can be chooses in 6 ways

Number of possible bus routes are : $99 \times 6 = 594$

Ans.(b) Here question 1 has 4 solutions, question 2 has 3 solutions and question 3 has 2 solutions.

By the multiplication rule,

Total number of solutions: $4 \times 3 \times 2 = 24$

Q.5 ***Find the generating functions for the following sequences:***

(a) *1,1,1,1,1,1,0,0,0,0,...*

(b) *1,3,3,1,0,0,0,0,...*

Ans.(a) The generating function is:

$$G(x) = 1+1x+1x^2+1x^3+1x^5+0x^6+0x^7+\ldots$$
$$= 1+x+x^2+x^3+x^4+x^5$$

We can apply the formula for the sum of a geometric series to rewrite G(x) as

$$G(x) = \frac{1-x^6}{1-x}$$

Ans.(b) The generating function is

$$G(x) = 1+3x+3x^2+1$$

Using formula

$$G(x) = (1+x)^3$$

PART-B

Q.6 (a) Solve the recurrence relations-

$$a_n - 5a_{n-1} + 6a_{n-2} = 3n^2 - 2n + 1$$

(b) Prove by induction that sum of the cubes of three consecutive integers is divisible by 9.

[R.T.U. 2019]

Ans.(a) $a_n - 5a_{n-1} + 6a_{n-2} = 3n^2 - 2n + 1$

Particular solution of the above is if the form

$$a_n = P_1n^2 + P_2n + P_3$$

$$P_1n^2 + P_2n + P_3 - 5P_1(n-1)^2 - 5P_2(n-1) - 5P_3 + 6P_1(n-2)^2 + 6P_2(n-2) + 6P_3 = 3n^2 - 2n + 1$$

$$\Rightarrow P_1n^2 + P_2n + P_3 - 5P_1(n^2 - 2n + 1) - 5P_2n + 5P_2 - 5P_3 + 6P_1(n^2 - 4n + 4) + 6P_2n - 12P_2 + 6P_3 = 3n^2 - 2n + 1$$

$$\Rightarrow P_1n^2 + P_2n + P_3 - 5P_1n^2 - 10P_1n - 5P_1 - 5P_2n + 5P_2 - 5P_3 + 6P_1n^2 - 24P_1n + 24P_1 + 6P_2n - 12P_2 + 6P_3 = 3n^2 - 2n + 1$$

$$\Rightarrow n^2(P_1 - 5P_1 + 6P_1) + n(P_2 + 10P_1 - 5P_2 - 24P_1 + 6P_2) + (P_3 - 5P_1 + 5P_2 - 5P_3 + 24P_1 - 12P_1 + 6P_3) = 3n^2 - 2n + 1$$

Equating coefficients of n^2, n and constant term, we get

$$P_1 - 5P_1 + 6P_1 = 3$$
$$P_1 + P_1 = 3$$
$$2P_1 = 3$$
$$P_1 = \frac{3}{2}$$

$$P_2 + 10P_1 - 5P_2 - 24P_1 + 6P_2 = -2$$
$$P_2 + 10 \times \frac{3}{2} - 5P_2 - 24 \times \frac{3}{2} + 6P_2 = -2$$
$$P_2 + 15 - 5P_2 - 36 + 6P_2 = -2$$
$$2P_2 = -2 - 15 + 36$$
$$P_2 = \frac{19}{2}$$

$$P_3 - 5P_1 + 5P_2 - 5P_3 + 24P_1 - 24P_1 + 6P_3 = 1$$
$$19P_1 + 2P_3 - 7P_2 = 1$$
$$19 \times \frac{3}{2} + 2P_3 - 7 \times \frac{19}{2} = 1$$
$$19 \times 3 + 4P_3 - 7 \times 19 = 2$$
$$4P_3 = 2 + 7 \times 19 - 19 \times 3$$
$$P_3 = \frac{78}{4}$$

so the particular solution is

$$a_nP = \frac{3}{2}n^2 + \frac{19}{2}n + \frac{78}{4}$$

Ans. (b) $P(n) = m^3 + (m+1)^3 + (m+2)^3$ is divide by 9.

$$P(n) = m^3 + (m+1)^3 + (m+2)^3 = a\lambda$$
$$P(1) = 1^3 + (1+1)^3 + (1+2)^3$$
$$= 1 + 8 + 27 = 36 = 9 \times 4$$

$aX* \rightarrow P(1)$ is true

Let P(m) be true

$$P(m) : m^3 + (m+1)^3 + (m+2)^3 = a\lambda$$

$$G\,P(m+1) : (m+1)^3 + (m+2)^3 + (m+3)^3 = aK$$

$(m+1)^3+(m+2)^3+(m+3)^3=(m+1)^3+(m+2)^3+m^3$
$+9m^3+27m+27$

$\Rightarrow m^3+(m+1)^3+(m+2)^3+9m^3+27m+27=a\lambda+am^2$
$+27m+27$

$\Rightarrow a(\lambda+m^2+3m+3)=9k$

$= \frac{P(1)}{P(m+1)}$ is true.

Q.7 *If the coefficient of $(2r+4)^{th}$ and $(r-2)^{th}$ terms in the expansion of $(1+x)^{18}$ are equal, then find the value of r.*

Ans. The general term of $(1+x)^n$ is $T_{r+1}\ {}_rC x^r$

Hence coefficient of $(2r+4)^{th}$ term will be

$$T_{2r+4}=T_{2r+3+1}={}^{18}C_{2r+3}$$

and coefficient or $(r-2)^{th}$ term will be

$$T_{r-2}=T_{r-3+1}={}^{18}C_{r-3}$$

$${}^{18}C_{2r+3}={}^{18}C_{r-3}$$

$$(2r+3)+(r-3)=18$$

$$(\because {}^nC_r={}^nC_k=r=k \text{ or } r+k=n)$$

$\therefore\ r=6$

Q.8 *In a lattice defined by the following Hasse Diagram, how many complements does the element 'e' have?*

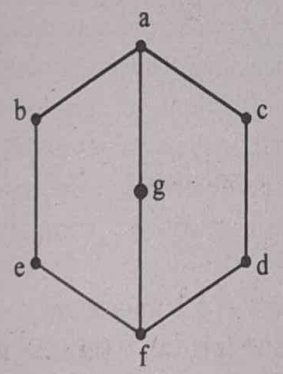


Fig.

Ans. The element e has 3 components – g,c and d.

$e \vee g = a$ and $e \wedge g = f$

$e \vee c = a$ and $e \wedge c = f$

$e \vee d = a$ and $e \wedge d = f$

Q.9 *Which of the following are posets? Explain.*

(a) (Z, =)

(b) (Z, ≠)

(c) (Z, ≥)

(d) (Z, +)

Ans.(a) (Z, =) : This is a poset. The only ordered pairs we will have in this relation is (a,a) for all $a \in z$.

This would mean that the relation is reflexive, anti symmetric and transitive.

(b) (Z, ≠) : This is not a poset because it is not reflexive. We cannot have the order pair (a,a) for all $a \in z$. This relation is also not anti symmetric and not transitive.

(c) (Z, ≥) : This is a poset. For reflexive, we can have the ordered pair (a,a) for all $a \in z$. This is also antisymmetric because consider the ordered pair (a,b) and $a \neq b$, this would mean that a>b. If this is the case, then b>a is not tree and you cannot have (b, a). This is also transitive because if a>b, b>c, and $a \neq b \neq c$. Then it follows that a>c for all $a,b,c \in z$.

(d) (Z, +) : This is not a poset because it is not reflexive. Consider 2+2, since this is not true, we cannot have (2,2). This relation is also not antisymmetric and not transitive.

Q.10 *Use iteration to solve the recurrence relation*

$$a_n = a_{n-1} + n$$

with $a_0 = 4$

Ans. $a_1 = a_0 + 1$

Now $a_2 = a_1 + 2$

But we know what a_1 is

By substitution, we get

$$a_2 = (a_0+1)+2$$

Now, $a_3 = a_2 + 3$

Using our known value of a_2.

$$a_3 = ((a_0+1)+2)+3$$

We notice a pattern. Each time, we take the previous term and add the current index. So,

$$a_n = ((((a_0+1)+2)+3)+\ldots+n-1)+n$$

Regrouping terms, we notice that a_n is just a_0 plus the sum of the integers from 1 to n. So, since $a_0 = 4$

$$a_n = 4 + \frac{n(n+1)}{2}$$

Q.11 *Solve the recurrence relation.*

$$F_n = 5F_{n-1} - 6F_{n-2}$$

Where $F_0 = 1$ *and* $F_1 = 4$

Ans. The characteristic equation of the recurrence relation is

$$x^2 - 5x + 6 = 0$$

So, $(x-3)(x-2) = 0$

Hence, the roots are,

$$x_1 = 3$$

and $$x_2 = 2$$

The roots are real and distinct

Hence $F_n = ax_1^n + bx_2^n$

Here, $F_n = a3^n + b2^n$ (As $x_1 = 3$ and $x_2 = 2$)

Therefore,

$$1 = F_0 = a3^0 + b2^0 = a+b$$

$$4 = F_1 = a3^1 + b2^1 = 3a+2b$$

Solving these two equations, we got a=2 and b = –1

Hence, the final solution is

$$F_n = 2.3^n + (-1).2^n = 2.3^n - 2^n$$

PART-C

Q.12 (a) Define Bounded lattices, complement of an element of a lattices and distributive lattices.

(b) Let (L, ≤) be a bounded distributive Lattice, if an element a ∈ L, has a complement then it is unique. *[R.T.U. 2019]*

Ans.(a) Bounded Lattice : A bounded lattice is an algebraic structure $\mathbb{L} = (L, \wedge, \vee, 0, 1)$, such that $(L, \wedge, \vee)$ is a lattice, and the constants $0, 1 \in L$ satisfy the following:

1. for all $x \in L$, $x \wedge 1 = x$ and $x \vee 1 = 1$,
2. for all $x \in L$, $x \wedge 0 = 0$ and $x \vee 0 = x$.

The element 1 is called the upper bound, or top of $\mathbb{L}$ and the element 0 is called the lower bound or bottom of $\mathbb{L}$.

There is a natural relationship between bounded lattices and bounded lattice-ordered sets. In particular, given a bounded lattice, $(L, \wedge, \vee, 0, 1)$, the lattice-ordered set $(L, \leq)$ that can be defined from the lattice $(L, \wedge, \vee)$ is a bounded lattice-ordered set with upper bound 1 and lower bound 0. Also, one may produce from a bounded lattice-ordered set $(L, \leq)$ a bounded lattice $(L, \wedge, \vee, 0, 1)$ in a pedestrian manner, in essentially the same way one obtains a lattice from a lattice-ordered set. Some authors do not distinguish these structures, but here is one fundamental difference between them: A bounded lattice-ordered set $(L, \leq)$ can have bounded subposets that are also lattice-ordered, but whose bounds are not the same as the bounds of $(L, \leq)$; however, any subalgebra of a bounded lattice $\mathbb{L} = (L, \wedge, \vee, 0, 1)$ is a bounded lattice with the same upper bound and the same lower bound as the bounded lattice $\mathbb{L}$.

Complemented Lattice : A complemented lattice is a bounded lattice (with least element 0 and greatest element 1) in which every element a has a complement, i.e. an element b such that

$a \vee b = 1$ and $a \wedge b = 0$.

Complement of an Element : In general an element may have more than one complement. However, in a (bounded) distributive lattice every element will have at most one complement.A lattice in which every element has exactly one complement is called a uniquely complemented lattice

A lattice with the property that every interval (viewed as a sublattice) is complemented is called a relatively complemented lattice. In other words, a relatively complemented lattice is characterized by the property that for every element a in an interval [c, d] there is an element such that

$a \vee b = d$ and $a \wedge b = c$.

Such an element b is called a complement of *a* relative to the interval.

A distributive lattice is complemented if and only if it is bounded and relatively complemented.The lattice of subspaces of a vector space provide an example of a complemented lattice that is not, in general, distributive.

Distributive Lattice : In mathematics, a distributive lattice is a lattice in which the operations of join and meet distribute over each other. The prototypical examples of such structures are collections of sets for which the lattice operations can be given by set union and intersection. Indeed these lattices of sets describe the scenery completely: every distributive lattice is-up to isomorphism-given as such a lattice of sets.

As in the case of arbitrary lattices, one can choose to consider a distributive lattice L either as a structure of order theory or of universal algebra. In the present situation, the algebraic description appears to be more convenient:

A lattice $(L, \vee, \wedge)$ is distributive if the following additional identity holds for all x, y, and z in L:

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

Viewing lattices as partially ordered sets, this says that the meet operation preserves non-empty finite joins. It is a basic fact of lattice theory that the above condition is equivalent to its dual:

$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ for all x, y, and z in L.

In every lattice, defining $p \leq q$ as usual to mean $p \wedge q = p$, the inequality $x \wedge (y \vee z) \geq (x \wedge y) \vee (x \wedge z)$ holds as well as its dual inequality $x \vee (y \wedge z) \leq (x \vee y) \wedge (x \vee z)$. A lattice is distributive if one of the converse inequalities holds, too. More information on the relationship of this condition to other distributivity conditions of order theory can be found in the article on distributivity (order theory).

Ans.(b) Bounded Lattice : Let $(L, \leq)$ is a lattice. An element I will be used for denote the upper bound (UB) of the set L $(a \in L, a \leq L)$. Obviously I is unique in the lattice, If it exist 0 to denote lower bound (LB) of the set L $(a \in L, 0 \leq a)$. In another words the greatest element in (L, z) if exist.

Complement of a lattice : Let $(L, \leq)$ be a bounded lattice, whose greatest and least elements are 1 and 0 respectively. Then an element $a' \in L$ is called a element of an element $a \in L$ if

$a \vee a' = 1$ and $a \wedge a' = 0$

Also $0' = 1$ and $1' = 0$

Distributive Lattice : A lattice $(L, \wedge, \vee)$ is called a distributive lattice if for any $a, b, c \in L$

(i) $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

(ii) $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

Q.13 ***The matching problem***

N guests arrive at a party. Each person is wearing a hat. We collect all hats and then randomly redistribute the hats, giving each person one of the N hats randomly. What is the probability that at least one person receives his/her own hat?

Hint : Use the inclusion – exclusion principle.

Ans. Let A_i be the event that i^{th} person receives his/her own hat. Then we are interested in finding P(E), where $E = A_1 \cup A_2 \cup A_3 \cup \ldots \cup A_N$. To find P(E), we use the inclusion – exclusion principle. We have.

$$P(E) = P(\cup_{i=1}^{N} A_i)$$

$$= \sum_{i=1}^{N} P(A_i) - \sum_{i,j:i<j} P(A_i \cap A_i) + \sum_{i,j,K:i<j<K} P(A_i \cap A_j \cap A_K) - \ldots + (-1)^{N-1} P(\cap_{i=1}^{N} Ai)$$

Note that there is complete symmetry here, that is, we can write

$P(A_1) = P(A_2) = P(A_3) = \ldots\ldots = P(A_N)$;

$P(A_1 \cap A_2) = P(A_1 \cap A_3) = \ldots\ldots = P(A_2 \cap A_4) = \ldots$

$P(A_1 \cap A_2 \cap A_3) = P(A_1 \cap A_2 \cap A_4)$

$= \ldots = P(A_2 \cap A_4 \cap A_5) = \ldots$;

Thus we have

$$\sum_{i=1}^{N} P(A_i) = NP(A_1);$$

$$\sum_{i,j:i<j=1} P(A_i \cap A_j) = \binom{N}{2} P(A_1 \cap A_2)$$

$$\sum_{i,j,K:i<j<K} P(A_i \cap A_j \cap A_K) = \binom{N}{2} P(A_1 \cap A_2 \cap A_3)$$

Therefore, we have

$$P(E) = NP(A_1) - \binom{N}{2} P(A_1 \cap A_2) + \binom{N}{3} P(A_1 \cap A_2 \cap A_3)$$

$$-\ldots\ldots + \ldots (-1)^{N-1} P(A_1 \cap A_2 \cap A_3 \ldots \cap A_N) \quad \ldots(1)$$

Now, we only need to find $P(A_1)$, $P(A_1 \cap A_2)$, $P(A_1 \cap A_2 \cap A_3)$, etc. to finish solving the problem. To find $P(A_1)$, we have

$$P(A_1) = \frac{|A_1|}{|S|}$$

Here, the sample spare S consists of all possible permutations of N objects (hats). Thus, we have

$|S| = N!$

On the other hand, A_1 consists of all possible permutations of N-1 objects (because the first object is fixed). Thus

$|A_1| = (N-1)!$

Therefore, we have

$$P(A_1) = \frac{|A_1|}{|S|} = \frac{(N-1)!}{N!} = \frac{1}{N}$$

Similarly, we have

$|A_1 \cap A_2| = (N-2)!$

Thus

$$P(A_1 \cap A_2) = \frac{|A_1 \cap A_2|}{|S|} = \frac{(N-2)!}{N!} = \frac{1}{P_{N-2}^{N}}$$

Similarly,

$$P(A_1 \cap A_2 \cap A_3) = \frac{|A_1 \cap A_2 \cap A_3|}{|S|}$$

$$= \frac{(N-3)!}{N!} = \frac{1}{P_{N-3}^{N}}$$

$$P(A_1 \cap A_2 \cap A_3 \cap A_4) = \frac{|A_1 \cap A_2 \cap A_3 \cap A_4|}{|S|}$$

$$= \frac{(N-4)!}{N!} = \frac{1}{P^N_{N-4}}$$

Thus using equation (1) we have

$$P(E) = N.\frac{1}{N} - \binom{N}{2}.\frac{1}{P^N_{N-2}} + \binom{N}{3}.\frac{1}{P^N_{N-2}} - \ldots + (-1)^{N-1}\frac{1}{N!} \quad \ldots(2)$$

By simplifying a little bit, we obtain

$$P(E) = 1 - \frac{1}{2!} + \frac{1}{3!} + \ldots + (-1)^{N-1}\frac{1}{N!}$$

We are done. It is interesting to note what happens when N becomes large. To see that, we should remember the Taylor series expansion of e^x. In particular,

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Letting $x = -1$, we have

$$e^{-1} = 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \ldots$$

Thus, we conclude that as N becomes large, P(E) approaches

$$1 - \frac{1}{e}$$

Q.14 ***Solve the following recurrence relations:-***

(a) $f_n = f_{n-1} + f_{n-2}$
With $f_0 = 0$ *and* $f_1 = 1$

(b) $a_n = -a_{n-1} + 4a_{n-2} + 4a_{n-3}$
With $a_0 = 8$, $a_1 = 6$ *and* $a_2 = 26$

(c) $a_n = a_{n-1} + 2a_{n-2}$
With $a_0 = 2$ *and* $a_1 + 7$

Ans.(a) Since it is linear homogeneous recurrence, first find its characteristic equation

$$r^2 - r - 1 = 0$$

$$r_1 = (1+\sqrt{5})/2$$

and $r_2 = (1-\sqrt{5})/2$

So, by theorem

$$f_n = \alpha_1((1-\sqrt{5})/2)^n + \alpha_2((1-\sqrt{5})/2)^n$$

is a solution

Now, we should find α_1 and α_2 using initial conditions

$$f_0 = \alpha_1 + \alpha_2 = 0$$

$$f_1 = \alpha_1(1+\sqrt{5})/2 + \alpha_2(1-\sqrt{5})/2 = 1$$

So, $\alpha_1 = 1 + /\sqrt{5}$ and $\alpha_2 = -1 + /\sqrt{5}$

$$a_n = 1/\sqrt{5}\ ((1+\sqrt{5})/2^n - 1/\sqrt{5}((1-\sqrt{5})/2)^n$$

is a solution

Ans.(b) Since it is a linear homogeneous recurrence, first find its characteristic equation

$$r^3 + r^2 - 4r - 4 = 0$$

$$(r+1)(r+2)(r-2) = 0$$

$$r_1 = -1,\ r_2 = -2 \text{ and } r_3 = 2$$

So, by theorem $a_n = \alpha_1(-1)^n + \alpha_2(-2)^n + \alpha_3 2^n$ is a solution.

Now, we should find α_1, α_2 and α_3 using initial conditions.

$$a_0 = \alpha_1 + \alpha_2 + \alpha_3 = 8$$

$$a_1 = -\alpha_1 - 2\alpha_2 + 2\alpha_3 = 6$$

$$a_2 = \alpha_1 + 4\alpha_2 + 4\alpha_3 = 26$$

So, $\alpha_1 = 2$, $\alpha_2 = 1$ and $\alpha_3 = 5$

$a_n = 2(-1)^n + (-2)^n + 5 \cdot 2^n$ is a solution.

Ans.(c) Since it is linear homogeneous recurrence, first find its characteristic equation.

$$r^2 - r - 2 = 0$$

$$(r+1)(r-2) = 0$$

$$r_1 = 2 \text{ and } r_2 = -1$$

So, by theorem $a_n = \alpha_1 2^n + \alpha_2(-1)^n$ is a solution.

Now, we should find α_1 and α_2 using initial conditions.

$$a_1 = \alpha_1 + \alpha_2 = 2$$

$$a_1 = \alpha_1 2 + \alpha_2(-1) = 7$$

So, $\alpha_1 = 3$ and $\alpha_2 = -1$

$a_n = 3.2^n - (-1)^n$ is a solution.

Q.15 ***Solve for*** a_n ***given that*** $a_0 = 0$, $a_1 = 6$ ***and*** $a_n = -3a_{n-1} + 10a_{n-2} + 3.2^n$. ***For*** $n \geq Z$.

Ans. Let $G(x) = \sum_{n=0}^{\infty} a_n x^n$ be the generating function for

$a_0, a_1, a_2, \ldots\ldots$

$$G(x) = 6x + \sum_{n=2}^{\infty} a_n x^n$$

$$= 6x + \sum_{n=2}^{\infty} -3a_{n-1} + 10a_{n-2} + 3.2^n \Big) x^n$$

$$= 6x - 3 + \sum_{n=2}^{\infty} a_{n-1}x^n + 10\sum_{n=1}^{\infty} a_{n-2}x^n - 3\sum_{n=2}^{\infty} 2^n x^n$$

$$= 6x - 3x(a_1x + a_2x^2 + \ldots) + 10x^2(a_0 + a_1x + a_2x^2 + \ldots) + 3((2x)^2 + (2x)^3 + (2x)^4 + \ldots)$$

$$= 6x - 3x\,G(x) + 10x^2\,G(x) + 3(2x)^2(1 + (2x) + (2x)^2 \ldots)$$

From the above, we see that

$$G(x) = 6x + G(x)(-3x + 10x^2) + 12x^2 \cdot \frac{1}{1-2x}$$

Therefore,

$$G(x)(1 + 3x - 10x^2) = 6x + \frac{12x^2}{1-2x}$$

$$G(x) = \frac{6x}{1 + 3x - 10x^2} + \frac{12x^2}{(1-2x)(1+3x-10x^2)}$$

Note that $1 + 3x - 10x^2 = (1 - 2x)(1 + 5x)$

Therefore

$$G(x) = \frac{6x}{(1-2x)(1+5x)} + \frac{12x^2}{(1-2x)^2(1+5x)}$$

Using partial fraction, we obtain that

$$\frac{1}{(1-2x)(1+5x)} = \frac{2}{7}\cdot\frac{1}{(1-2x)} + \frac{5}{7}\cdot\frac{1}{(1+5x)}$$

From this, we also obtain that

$$\frac{1}{(1-2x)^2(1+5x)} = \frac{2}{7}\cdot\frac{1}{(1-2x)^2} + \frac{10}{49}\cdot\frac{1}{(1-2x)} + \frac{25}{49}\cdot\frac{1}{(1+5x)}$$

Therefore,

$$G(x) = 6x\left(\frac{2}{7}\cdot\frac{1}{(1-2x)} + \frac{5}{7}\cdot\frac{5}{(1+5x)}\right) + 12x^2\left(\frac{2}{7}\cdot\frac{1}{(1-2x)^2} + \frac{10}{49}\cdot\frac{1}{(1-2x)} + \frac{25}{49}\cdot\frac{1}{(1+5x)}\right)$$

The coeff. of x^n in $6x\left(\frac{2}{7}\frac{1}{(1-2x)} + \frac{5}{7}\frac{5}{(1+5x)}\right)$ equals

6x coeff. of x^{n-1} in $\frac{2}{7}\frac{1}{(1-2x)} + \frac{5}{7}\frac{5}{(1+5x)}$ which equals

$$6\left(\frac{2}{7}\cdot 2^{n-1} + \frac{5}{7}\cdot(-5)^{n-1}\right)$$

The coeff. of x^n in

$$12x^2\left(\frac{2}{7}\frac{1}{(1-2x)^2} + \frac{10}{49}\frac{1}{(1-2x)} + \frac{25}{49}\frac{1}{(1+5x)}\right)$$

equal 12x coeff. of x^{n-2} in

$$\frac{2}{7}\frac{1}{(1-2x)^2} + \frac{10}{49}\frac{1}{(1-2x)} + \frac{25}{49}\frac{1}{(1+5x)}$$

which equals

$$12\left(\frac{2}{7}(n-1).2^{n-2} + \frac{10}{49}2^{n-2} + \frac{25}{49}(-5)^{n-2}\right)$$

Putting the above together, we obtain that.

$$a_n = 6\left(\frac{2}{7}\cdot 2^{n-1} + \frac{5}{7}\cdot(-5)^{n-1}\right) + 12\left(\frac{2}{7}(n-1).2^{n-2} + \frac{10}{49}2^{n-2} + \frac{25}{49}(-5)^{n-2}\right), n \geq 2$$

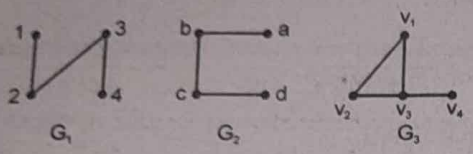
❑❑❑

ALGEBRAIC STRUCTURES 4

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 *Prove that these graphs G_1, G_3 and G_2, G_3 are non-isomorphic*



[R.T.U. 2019]

Ans. No of edges in $G_1 = 3$

No of edges in $G_2 = 3$

No of edges in $G_3 = 4$

As the size of G_1, G_3 and G_2, G_4 are not same so these set of graphs are non-isomorphic.

Q.2 ***Show that the multiplicative group $G = \{1, -1, i, -i\}$ is cyclic. Find its generators.*** *[R.T.U. 2019]*

Ans. $G = \{1, -1, i, -i\}$

$\{i^4, i^2, i^1, (i)^3\}$

$i \in G$

At that same time

$G = \{1, -1, i, -i\}$

$= \{(-i)^4, (-i)^2, (-i)^3, (-i)^1\}$

$-i \in G$

i.e. $\{i$ & $-i\}$ generators of G.

Q.3 ***Write short note on Group.*** *[R.T.U 2012]*

Ans. Group : A group is an algebraic structure $\{G, \perp\}$, where G is a non-empty set and $\perp$ denotes a binary operation $\perp : G \times G \to G$, called the group operation. The notation $\perp(x, y)$ is normally shortened to the infix notation $x \perp y$.

A group must obey the following rules. Let a, b, c be arbitrary elements of G. Then :

- *CLOSURE-* $a \perp b \in G$. This axiom is often omitted because a binary operation is closed by definition.
- *ASSOCIATIVITY-* $(a \perp b) \perp c = a \perp (b \perp c)$.
- *IDENTITY-* There exists an identity (or neutral) element $e \in G$ such that $a \perp e = e \perp a = a$. The identity of G is unique.
- *INVERSE-* For each $a \in G$, there exists an inverse element $x \in G$ such that $a \perp x = x \perp a = e$. The inverse of a is unique.

An abelian group also obeys the additional rule :

- *COMMUTATIVITY-* $a \perp b = b \perp a$.

Q.4 ***Write short note on Field.*** *[R.T.U 2012]*

Ans. Field : Field theory considers sets, such as the real number line, on which all the usual arithmetic properties hold --- those governing addition, subtraction, multiplication and division.

Specifically, a field is a commutative ring in which every nonzero element is assumed to have a multiplicative inverse. Examples include the real number field R, the complex numbers C, the rational numbers.

Fields are important objects of study in algebra since they provide a useful generalization of many number systems, such as the rational numbers, real numbers, and complex numbers. In particular, the usual rules of associativity, commutativity and distributivity hold.

The concept of a field is of use, for example, in defining vectors and matrices, two structures in linear algebra whose components can be elements of an arbitrary field.

Q.5 *Write short note on Ring.* *[R.T.U 2012]*

Ans. Ring : A Ring is a structure that abstracts and generalizes the basic arithmetic operations, specifically the addition and the multiplication.

Briefly, a ring is an abelian group with a second binary operation that is associative and is distributive over the abelian group operation. The abelian group operation is called "addition" and the second binary operation is called "multiplication" in analogy with the integers. One familiar example of a ring is the set of integers. The integers are a commutative ring, since a times b is equal to b times a. The set of polynomials also forms a commutative ring. An example of a non-commutative ring is the ring of square matrices of the same size. Finally, a field (such as the real numbers) is a commutative ring in which one can do "division" by any nonzero element.

The most familiar example of a ring is the set of all integers, Z, consisting of the numbers

$\ldots -4, -3, -2, -1, 0, 1, 2, 3, 4, \ldots$

Mathematically a ring is a set R equipped with two binary operations

$+ : R \times R \to R$ and $: R \times R \to R$ (where $\times$ denotes the Cartesian product), *called addition* and *multiplication*, such that:

- (R, +) is an abelian group with identity element 0, meaning that for all a, b, c in R, the following axioms hold :
- $(a + b) + c = a + (b + c)$ (+ is associative)
- $0 + a = a$ (0 is the identity)
- $a + b = b + a$ (+ is commutative)
- For each a in R there exists $-a$ in R such that $a + (-a) = (-a) + a = 0$ ($-a$ is the inverse element of a)
- Multiplication $\cdot$ is associative:
- $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- Multiplication distributes over addition:
- $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
- $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$.

Q.6 *Write short note on Galois field.* *[R.T.U 2012]*

Ans. Galois Field: A finite field is a field with a finite field order (i.e., number of elements), also called a Galois field.For example, GF(p) is called the prime field of order p, and is the field of residue classes modulo p, where the p elements are denoted 0, 1, ..., $p - 1$. $a = b$ in GF(p) means the same as $a \equiv b \pmod p$

The finite field GF(2) consists of elements 0 and 1 which satisfy the following addition and multiplication tables.

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

Q.7 *Explain Subgroup.*

Ans. Let G be a group and $4 \subseteq G$ be non-empty. If H is also a group under the same operation as G, then H is a sub group of G. if $\{e\} \subseteq H \subseteq G$ then H is proper subgroup of G.

PART-B

Q.8 Define and explain the following by suitable examples-

(i) Cyclic group

(ii) Order of an element in a group

(iii) Field

(iv) Zero divisor of a ring *[R.T.U. 2019]*

Ans.(i) Cyclic Group : In group theory, a branch of abstract algebra, a cyclic group or monogenous group is a group that is generated by a single element. That is, it is a set of invertible elements with a single associative binary operation, and it contains an element g such that every other element of the group may be obtained by repeatedly applying the group operation to g or its inverse. Each element can be written as a power of g in multiplicative notation, or as a multiple of g in additive notation. This element g is called a generator of the group.

Example:

Integer and modular addition

The set of integers Z, with the operation of addition, forms a group. It is an infinite cyclic group, because all integers can be written by repeatedly adding or subtracting the single number 1. In this group, 1 and "1 are the only generators. Every infinite cyclic group is isomorphic to Z.

For every positive integer n, the set of integers modulo n, again with the operation of addition, forms a finite cyclic group, denoted Z/nZ. A modular integer i is a generator of this group if i is relatively prime to n, because these elements can generate all other elements of the group through integer addition. (The number of such generators is $\varphi(n)$, where φ is the Euler totient function.) Every finite cyclic group G is isomorphic to Z/nZ, where n = |G| is the order of the group.

The addition operations on integers and modular integers, used to define the cyclic groups, are the addition operations of commutative rings, also denoted Z and Z/nZ or Z/(n). If p is a prime, then Z/pZ is a finite field, and is usually denoted F_p or GF(p).

(ii) Order of a group : In group theory, a branch of mathematics, the order of a group is its cardinality, that is, the number of elements in its set. The order of an element a of a group, sometimes also period length or period of a, is the smallest positive integer m such that $a^m = e$, where e denotes the identity element of the group, and a^m denotes the product of m copies of a. If no such m exists, a is said to have infinite order.

The order of a group G is denoted by ord(G) or |G|, and the order of an element a is denoted by ord(a) or |a|. The order of an element a is equal to the order of its cyclic subgroup $\langle a \rangle = \{a^k \text{ for k an integer}\}$, the subgroup generated by a. Thus, $|a| = |\langle a \rangle|$.

Lagrange's theorem states that for any subgroup H of G, the order of the subgroup divides the order of the group: |H| is a divisor of |G|. In particular, the order |a| of any element is a divisor of |G|.

Example. The symmetric group S_3 has the following multiplication table.

•	e	s	t	u	v	w
e	e	s	t	u	v	w
s	s	e	v	w	t	u
t	t	u	e	s	w	v
u	u	t	w	v	e	s
v	v	w	s	e	u	t
w	w	v	u	t	s	e

This group has six elements, so $\text{ord}(S_3) = 6$. By definition, the order of the identity, e, is one, since $e^1 = e$. Each of s, t, and w squares to e, so these group elements have order two: $|s| = |t| = |w| = 2$. Finally, u and v have order 3, since $u^3 = vu = e$, and $v^3 = uv = e$.

(iii) Field : *Refer to Q.4.*

Example :

Rational Numbers : Rational numbers have been widely used a long time before the elaboration of the concept of field. They are numbers that can be written as fractions a/b. Where a and b are integers, and $b \neq 0$. The additive inverse of such a fraction is –a/b, and the multiplicative inverse (provided that $a \neq 0$) is b/a, which can be seen as follows :

$$\frac{b}{a}\cdot\frac{a}{b} = \frac{ba}{ab} = 1$$

The abstractly required field axioms reduce to standard properties of rational numbers. For example, the law of distributivity can be proven as follows :

$$\frac{a}{b}\cdot\left(\frac{c}{d}+\frac{e}{f}\right)$$

$$= \frac{a}{b}\cdot\left(\frac{c}{d}\cdot\frac{f}{f}+\frac{e}{f}\cdot\frac{d}{d}\right)$$

$$= \frac{a}{b}\cdot\left(\frac{cf}{df}+\frac{ed}{fd}\right) = \frac{a}{b}\cdot\frac{cf+ed}{df}$$

$$= \frac{a(cf+ed)}{bdf} = \frac{acf}{bdf}+\frac{aed}{bdf} = \frac{ac}{bd}+\frac{ae}{bf}$$

$$= \frac{a}{b}\cdot\frac{c}{d}+\frac{a}{b}\cdot\frac{e}{f}$$

(iv) Zero Divisors in Rings

Definition : Let (R,+, *) be a ring where $0 \in R$ is the identity of +. The element $a \in R\backslash\{0\}$ is said to be a Zero-Divisor of R if there exists a $b \in R\backslash\{0\}$ such that $a * b = 0$ or $b * a = 0$.

For example, consider the ring $(M_{22}, +, *)$ of 2×2 matrices with real coefficients and with the operations of standard matrix additiion +, and standard matrix multiplication *. Recall that the identity of + is the 2×2 zero matrix

$$0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Further consider the matrices $A, B \in M_{22}$ given by

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} \quad \ldots(1)$$

When we multiply the matrices A and B together we have that

$$A * B = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} = 0 \quad \ldots(2)$$

Notice that A is not the identity for + and B is not the identity for +. Therefore, the matrices A and B are zero divisors of M_{22}.

We should bee clear that a ring $(R, +, *)$ need not have any zero divisors. For example, consider the ring $(C, +, *)$ of complex numbers where + is standard addition and * is standard multiplication. We note that the identity of + is $0 = 0 + 0i \in C$. Since $(C, +, *)$ is a commutative ring, then for x, y $\in C\backslash\{0\}$ and where $x = a + bi$ and $y = c + di$ for a, b, c, d $\in R$ we need to consider the following equation.

$$x * y = (a + bi)(c + di) = (ac - bd) + (ad + bc)i = 0 + 0i \quad \ldots(3)$$

Note that this equality holds if and only if :

$$ac = bd \text{ and } ad + bc = 0 \quad \ldots(4)$$

Without loss of generality, assume $x \neq 0$. Then either $a \neq 0$ or $b \neq 0$ or both. Assume $a \neq 0$. Then we can divide both equations by a to get :

$$c = \frac{bd}{a}(*) \text{ and } d + \frac{bc}{a} = 0(**) \quad \ldots(5)$$

Substituting the first equation into the second yields:

$$d + c\frac{b}{a} = 0(*) \quad \ldots(6)$$

$$d + \frac{bd}{a}.\frac{b}{a} = 0$$

$$d + \frac{b^2 d}{a^2} = 0(*)$$

$$d\left(1 + \frac{b^2}{a^2}\right) = 0$$

There are two possibilities in the equation above. Either $d = 0$ or $1 + \frac{b^2}{a^2} = 0$. Clearly $1 + \frac{b^2}{a^2} = 0$ since this would imply that $\left(\frac{b}{a}\right)^2 = -1$. Therefore $d = 0$.

Looking at (**) we see that then $\frac{bc}{a} = 0$ so either $b = 0$ or $c = 0$. If $c = 0$ we have that then $y = c + di = 0$. Meanwhile, if $c \neq 0$ then $b = 0$ and by (*) this implies that $c = 0$ so then $y = c + di = 0$ again. In either case, we see that if $x = a + bi \neq 0 + 0i$ then $y = c + di = 0 + 0i$. Therefore, there exists no zero divisors in the ring $(c, +, *)$

Q.9 ***If {G, *} is a finite cyclic group generated by on element $a \in G$ and is of order n, then $a^n = e$ so that $G = \{a, a^2, \ldots, a^n(=e)\}$. Also n is the least positive integer for which $a^n = e$.*** *[R.T.U. Dec. 2016]*

OR

Prove that if (G, *) is a finite cyclic group generated by an element $a \in G$ and is of order n, then $a^n = e$. Also n is the least positive integer for which $a^n = e$.

Ans. G = <a>, group is generated by a.

G has a order n :

$\Rightarrow G = \{a^0, a^1, a^2, \ldots a^{n-1}\}$

Here, $a^n = a^0$ (because G is cyclic)

Let $a \in G$ such that $a^n \neq e$. Because $|G| = n$, we cannot have order (a) > n, else {e, a, ..., a^n} would all be distinct and this contradicts the order of G.

Let order (a) = m ≤ n. Then $|\langle a\rangle| = m$, the cyclic group generated by a.

By Lagrange theorem

$|\langle a\rangle| / |G| = m/n$

Therefore, n = dm . Then,

$a^n = (a^m)^d = e^d = e$

But, since G is cyclic and G = <a>

Therefore, order (a) = |G|

$G = \{e, a, a^2, \ldots, a^{n-1}\}$

$a^n = e$. Hence, order (a) = n.

Since it is order, it must be minimum such that $a^n = e$

But $p < n$ such that $a^p = e$

Then order (a) = p which is not true as order (a) = n.

Hence, $a^n = e$ and n is the least positive number for which this is possible.

Q.10 ***If S is the set of ordered pairs (a, b) of real numbers and if the binary operations $\oplus$ and $\odot$ are defined by the equations-***

$$(a, b) \oplus (c, d) = (a + c, b + d)$$

and $(a, b) \odot (c, d) = (ac - bd, bc + ad)$

prove that (S, $\oplus$, $\odot$) is a field. *[R.T.U. 2016]*

Ans. S is a set of ordered pairs

$(a, b) \oplus (c, d) = (a+c, b+d)$

$(a, b) \odot (c, d) = (ac - bd, bc + ad)$

1. Closure of S under addition and multiplication :

$(a, b), (c, d) \in S \Rightarrow a, b, c, d \in IR$

$a + c, b + d \in IR \Rightarrow (a + c, b + d) \in S$

$ac, bd, bc, ad \in IR \Rightarrow ac - bd, bc + a\alpha \in IR$

$\Rightarrow (ac - bd, bc + ad) \in S$

Hence, it is closure w.r.t addition and multiplication

2. Associativity of addition and multiplication :

$(a, b) \oplus ((c,d) \oplus (e, f)) = (a, b) \oplus (c+ e, d+ f)$

$= (a + c + e, b+ d + f) = ((a + c, b + d)) \oplus (e, f)$

$= ((a, b) \oplus (c,d)) \oplus (e, f)$

$(a, b) \odot ((c,d) \odot (e,f)) = (a,b) \odot (ce - df, de + cf)$

$= (ace - adf - bde - bcf, bce - bdf, + ade + acf)$

$= (ace - bde - dcf - adf, bce + ade + acf - bdf)$

$= (ac - bd, bc + ad), \odot (e, f)$

$= ((a,b) \odot (c,d)) \odot (e,f)$

3. Commutativity of addition and multiplication :

$(a,b) \oplus (c,d) = (a+c, b+d) = (c + a, d + b)$

$= (c, d) \oplus (a,b)$

$(a,b) \odot (c,d) = (ac - bd, bc + ad) = (ca - db, da + cb)$

$= (c, d) \odot (a, b)$

4. Existence of additive and multiplicative identity :

$(a, b) \oplus (0, 0) = (a + 0, b + 0) = (a, b)$

$(a,b) \odot (1,0) = (a.1 - b \times 0, b.1 + a.0)$

$= (a,b)$

(0,0) and (1,0) are additive and multiplicative identity respectively.

5. *Existence of additive and multiplicative inverse :*

$(a, b) \oplus (-a, -b) = (a - a, b - b) = (0,0)$

$$(a,b) \odot \left(\frac{a}{a^2+b^2}, \frac{-b}{a^2+b^2} \right)$$

$$= \left(\frac{a^2}{a^2+b^2} + \frac{b^2}{a^2+b^2}, \frac{ab}{a^2+b^2} - \frac{ab}{a^2+b^2} \right) = (1,0)$$

Hence, $(-a,-b)$ and $\left(\frac{a}{a^2+b^2}, \frac{-b}{a^2+b^2} \right)$ are additive and multiplicative inverse.

Distributivity of multiplication over addition :

To prove $(a, b), (c,d), (e,f) \in S$

$(a, b) \odot ((e, d) \oplus (c, f)) = ((a, b) \odot (c, d)) \oplus ((a, b) \odot (e, f))$

LHS $= (a, b) \odot (c + e, d + f)$

$= (ac + ae - bd - bf, bc + be + ad + af)$

RHS $= ((a,b) \odot (c, d)) \oplus ((a, b) \odot (e, f))$

$= (ac - bd, bc + ad) \oplus (ae - bf, be + af)$

$= (ac - bd + ae - bf, bc + ad + be + af)$

LHS = RHS **Hence Proved**

Q.11 ***The necessary and sufficient condition for a non-empty subset H of a group {G, *} to be a subgroup is $a, b \in H \Rightarrow a*b^{-1} \in H$.*** *[R.T.U. 2016]*

Ans. The condition is necessary. Suppose H is a subgroup of G and let $a \in H, b \in H$. Now each element of H must possess inverse because H itself is a group.

$b \in H \Rightarrow b^{-1} \in H$

Also, H is closed under composition * in G. Therefore,

$a \in H, b^{-1} \in H \Rightarrow a * b^{-1} \in H$

The condition is sufficient. If it is given $a \in H, b^{-1} \in H \Rightarrow a * b^{-1} \in H$, then we have to prove that H is a subgroup.

(i) Closure property : Let $a, b \in H$ then $b \in H \Rightarrow b^{-1} \in H$.

Therefore, $a \in H, b^{-1} \in H \Rightarrow a * (b^{-1})^{-1} \in H.$

$\Rightarrow a * b \in H.$

H is closed with respect to composition * in G.

(ii) Associative property : Since elements of H are also the elements of G, the composition is associative in H.

(iii) Existence of identity : Since,

$a \in H, a^{-1} \in H \Rightarrow a* a^{-1} \in J = e \in H.$

(iv) Existence of inverse : Let $a \in H$ then

$e \in H, a \in H \Rightarrow e * a^{-1} \in H \Rightarrow a^{-1} \in H.$

Hence, H itself is a group for the composition * in group G.

Q.12 ***Show that $z_5 = \{0,1,2,3,4\}$ is an abelian group for the operation $+_5$ defined as.***

$$a +_5 b = \begin{cases} a+b & \text{if } a+b < 5 \\ a+b-5 & \text{if } a+b \geq 5 \end{cases}$$

[R.T.U. 2015]

Ans. Composition table is

$+_5$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\because$ All entries in the composition table are elements of z_5. So z_5 is closed with respect to addition modulo 5 *i.e.* $+_5$

The composition $+_5$ is associative. If a, b, c are there elements of z_5, then

$a +_5 (b +_5 c) = a +_5 (b + c)$ [$\because b +_5 c \equiv (b+c) \bmod 5$]

= least non-negative remainder when a+ (b+c) is divided by 5.

= least non-negative remainder when (a+b)+c is divided by 5.

$= (a+b) +_5 c = (a+_5 b) +_5 c$

Existence of identity : we have $0 \in z_5$. If a is any element of z_5 than from the composition table, we have $0+_5 a = a +_5 0$

Thus 0 is the identity element .

Existence of inverse : Form the table we get that the inverses of 0,1, 2, 3, 4 are 0, 4, 3, 2, 1 respectively.

The composition is commutative as the corresponding rows and columns in the table are identical.

So $(z_5, +_5)$ is an abelian group.

Q.13 ***Prove that a non-void subset H of a group G is a subgroup***

$$\textit{if } a \in H,\ b \in H \Rightarrow ab^{-1} \in H$$ ***[R.T.U. 2015]***

Ans. Necessary condition : Suppose H is a subgroup of G. Let $a \in H$, $b \in H$. Now each element of H must possess inverse because H itself is a group.

$\therefore\ b \in H \Rightarrow b^{-1} \in H$

Further H must closed w.r. to multiplication i.e. the composition in G. Therefore

$a \in H,\ b^{-1} \in H \Rightarrow ab^{-1} \in H.$

Sufficient condition : Now it is given that $a \in H, b \in H \Rightarrow ab^{-1} \in H$. We are prove that H is a subgroup of G.

Existence of identity : We have

$a \in H,\ a \in H \Rightarrow aa^{-1} \in H$

$\Rightarrow e \in H$

Thus the identity e is an element of H.

Existence of inverse: Let a be any element of H. Then by the given condition we have

$e \in H,\ a \in H \Rightarrow e\ a^{-1} \in H \Rightarrow a^{-1} \in H$

Thus each element of H possess inverse.

Closure property : Let $a, b \in H$

$\because\ b \in H \Rightarrow b^{-1} \in H$

Therefore applying the given condition, we have

$a \in H,\ b^{-1} \in H \Rightarrow a(b^{-1})^{-1} \in H \Rightarrow ab \in H$

Associativity : The elements of H are also the elements of G. The composition in G is associative. Therefore it must also be associative in H.

Hence H itself is a group for the composition in G. Therefore H is a subgroup of G.

PART-C

Q.14 (a) Let (m, *) be a semi group and a ∈ m such that the equations a * u = x and v * a = x have solutions in M for all x ∈ M. Show that (M, *) is a monoid.

(b) Let Δ(G) be the maximum of the degrees of the vertices of a graph G then K(G) ≤ 1 + Δ(G) where K(G) is the chromatic number of graph.

(c) Let G be the set of all non-zero real numbers and Let $a * b = \frac{ab}{2}$, then show that $(G_1, *)$ is an abelian group.

[R.T.U. 2019]

Ans.(a) Given $a * u = x$

and $v * a = x;\ \forall x \in A$ ...(i)

$a \in A$ if we take x = a equation (i) are satisfied.

For some $u = e_l$ and $v = e_m$

$a * b\ e_l = a$ and $e_m * a = a$

Again Let $y \in A$

$y * e_l = (v * a) * e_l = v * (a * e_l) = v * a = y$

$e_m * y = e_m * (a * u) = (e_l * a) * u$

$= a * u = y$

$y * e_l = y$ and $e_m * y = y$

e_l and e_m are right and left identity in A

$e_l = e_m = e$

identity element $e \in A$.

Ans.(b) Proof : Let the no of vertex in a graph is denoted by $|V|$. If $|V| = 1$ then $A(G) = 0$, $K(G) = 1$. So the results holds. Now let k be an integer. and $k \geq 1$. Assume that results hold for all graph with $|V| = k$ vertex.

Let G be a graph with $(k + 1)$ vertex

Let V be any vertex of G and $G_0 = G\{v\}^2$

is a subgraph of G obtained by deleting V from G.

Since G_0 has k vertex so we use induction

$k(G_0) \leq 1 + \Delta(G_0)$

$\Delta(G_0) \leq \Delta(G_0)$

$K(G_0) \leq 1 + \Delta(G)$

So G_0 can be colored with atmost $1 + \Delta(G)$ colors. Since there can be almost $\Delta(G)$ vertices adjacent to V, one of the available $1 + \Delta(G)$ colors remains for V. Thus G can be colored with atmost $1 + \Delta(G)$ colors.

Ans.(c) Let a, b ∈ G Here $a \ast \beta = \frac{ab}{2}, \forall a, b \in G$

(i) Closure : a and b are non zero real numbers ab is also a non zero real number

$\frac{ab}{2}$ is also a non zero real number

$\frac{ab}{2} \in G \Rightarrow (a * b) \in G$

G is closed

(ii) Associativity : Let e a, b, c ∈ G

Then

$$a * (b * c) = a * \left(\frac{bc}{2}\right) = \frac{a(bc)}{2.2} = \frac{abc}{4} \quad \ldots(1)$$

$$(a * b) * c = \left(\frac{ab}{2}\right) * c = \frac{a(bc)}{2.2} = \frac{abc}{4} \quad \ldots(2)$$

From (1) and (2)

$a * (b * c) = (a * b) * c$

G is associative

(iii) Identity : *Let e be the identity element is G.*

$a * e = a \ \forall a \in G.$

$\frac{ae}{2} = a \Rightarrow e = 2 \in G$

2 is identity element is G.

(iv) Inverse : Let a' b c the inverse of $a \in G$.

$a * a' = 2$

$\frac{aa'}{2} = 2 \Rightarrow a' = \frac{4}{a} \in G (a \neq 0)$

$a \in G$, So each elements of G has its inverse in G.

(v) Abelian : Let a, b ∈ G then

$a * b = \frac{ab}{2} = \frac{ba}{2} = b * a$

$a * b = b * a. \ \forall a, b \in G$

Therefore (G, *) is an abelian.

Q.15 (a) Prove that every infinite cyclic group has two and only two generators.

(b) Show that the set $t(g) = \left\{\begin{bmatrix} a & 0 \\ b & 0 \end{bmatrix} a \in z, b \in z\right\}$ ***is an ideal of the ring*** $R = \left\{\begin{bmatrix} a & 0 \\ b & 0 \end{bmatrix} a, b, c \in z\right\}$.

Matrix addition and matrix multiplication being the operations of the system. *[R.T.U. 2014]*

Ans.(a) Let $G = \{a\}$ be an infinite cyclic group generated by a. The elements of G will be integral power of a.

We claim that no two distinct integral power of a can be equal.

For, if possible, let $a^r = a^s$, $r > s$

$\Rightarrow a^r . a^{-s} = a^{s-s}$

$\Rightarrow a^{r-s} = a^0$

$\Rightarrow a^{r-s} = 0$

Since r–s is positive integer

$a^{r-s} = 1 \Rightarrow a^0 = r - s$ finite

So 'a' can't be a generator of an infinite cyclic group G.

Hence $a^r \neq a^s$ unless r = s

Therefor we can write

$G = \{\ldots a^{-4}, a^{-3}, a^{-2}, a^{-1}, a^0, a^1, a^2, \ldots\}$

If a^r is any elment of G we can write $a^r = (a^{-1})^{-r}$

Thus a^{-1} is also a generator of G. To show that a and a^{-1} generator Now if $m \neq 1$ or -1 then a^m can't be generator of G. If a^m is to a generator of G, there must exist an integer k such that $(a^m)k = a$ i.e. $a^{mk} = a$

Now m = 1 or –1 Þ $mk \neq 1$

Therefore two distinct integral powers of 'a' are equal and this contracts that statement we have just proved. Hence a^m cannot be a generator of G if $m \neq 1$ or – 1. Thus G has exactly two generators.

Ans.(b) A subset of ring R is said to be ideal of R.
(let $S \subset R$) is ideal of R
If (I) $a \in S\ b \in S \Rightarrow a - b \in S$
(II) $rS \in S$, $Sr \in S\ \forall\ r \in R, S \in S$

Let $\alpha = \begin{bmatrix} a_1 & 0 \\ b_1 & 0 \end{bmatrix}$

$$\beta = \begin{bmatrix} a_2 & 0 \\ b_2 & 0 \end{bmatrix} \in t(g)$$

$$\alpha - \beta \begin{bmatrix} a_1 - a_2 & 0 \\ b_1 - b_2 & 0 \end{bmatrix} \in t(g)$$

Since $a_1 - a_2, b_1 - b_2 \in Z$

Let $r = \begin{bmatrix} a & 0 \\ b & c \end{bmatrix} \in R$

$$\alpha r = \begin{bmatrix} a & 0 \\ b & 0 \end{bmatrix}\begin{bmatrix} a & 0 \\ b & c \end{bmatrix} = \begin{bmatrix} a^2 & 0 \\ ab & 0 \end{bmatrix} \in t(g)$$

$$r\alpha = \begin{bmatrix} a & 0 \\ b & 0 \end{bmatrix}\begin{bmatrix} a & 0 \\ b & 0 \end{bmatrix} = \begin{bmatrix} a^2 & 0 \\ ab + bc & 0 \end{bmatrix} \in t(g)$$

We have above so that
$\alpha, \beta \in t(g) \Rightarrow \alpha - \beta \in t(g)$
$r\alpha \in t(g)\ \alpha r \in t(g)$
$\forall\ r \in R, a \in S$

Q.16 ***(a) Explain permutation group with example.***
(b) Write out all of the permutations of the set {1, 2, 3, 4}. How many are there in all? Find a sensible way to organize you list.

Ans.(a) A permutation of n distinct object is just a listing of the objects in some order. For example, [c, b, a] is a permutation of the set {a, b, c} of three objects. Likewise, [triangle, melon, airplane] is a permutation of three objects as well. From our mathematical point of view, the objects we use don't actually matter; all we care about is the order they are arranged in. So usually we'll just talk about permutations of the numbers 1 through n. It can think of each number as just counting the objects involved : first object, second object, n^{th} object.

Permutations arise in the world in a many, many ways. For example, suppose you are asked to list preferences amongst a bunch of presidential candidates. The list makes up, from favorite to least favorite, is a permutation of the candidates. In fact, it can use the mathematics of permutations to learn interesting things about different kinds of voting systems.

Another example is a deck of playing cards. In a standard deck, each card appears exactly once. When you shuffle the deck, you are just creating a random permutation of the cards. One can use mathematics related to permutations to answer interesting questions about cards. Like : "How many times do I need to shuffle the deck before it is truly randomized?" The answer, by the way, seems to be 7 for a standard riffle shuffle. But proving that is well beyond the scope fo these notes.

Because permutations are so common, problems involving permutations tend to be very applicable! For example, suppose you have two hundred students in a class and they all hand in an exam. The stack of exams they give you is a permutation of the students; most likely, the list of student scores you keep is alphabetical. This suggests a problem ; What is the fastest way to sort the exams? (In fact, sorting is a fundamental problem in computer science.)

How many permutations are there of a set of n objects? Suppose we try to build a permutation by successively choosing objects. Then there are n choices for the first object, n – 1 choices for the second, and so on, until there is only one choice for the last object. Then to get the total number of possible permutations, we multiply these numbers together, and get $n(n-1)(n-2)\ldots 1$. This number, if you haven't seen it before, is called n-factorial written n!

Ans.(b) Suppose we have some initial ordering of our objects. The letters {a, b, c}, for example, can be organized alphabetically. Then every permutation we can think of as a mising-up of this initial order. In this sense, the permutation is a special kind of function from the set of objects back to itself. A permutation of these objects is then the list [σ(a), σ(b), σ(c)]; this list is called the one line notation for σ.

These permutations-as-functions can be composed : if we think of two permutations σ and τ as different ways to mix up the set, can be mix them up according to σ and then according to τ. Then the composition is specified by the list [τ(σ (a)), τ(σ (b)), τ(σ (c))].

For example, if σ = [2, 3, 1, 4] and τ = [3, 4, 1, 2], then τσ = [4, 3, 1, 2]. (In particular σ(1) = 2, and τ(2) = 4, so the first entry of τσ is 4. The other three entries are computed similarly.) On the other hand, στ = [3, 1, 4, 2]. This is different from τσ. So we see that the group of permutations has elements where fog ≠ gof; we say that S_n is non-commutative. (But remember that nothing in our group definition says that the group needs to be commutative, so this is ok.)

A very nice way to keep track of this mixing-up is the braid notation for a permutation. This simply writes the list of objects in two lines, and draws a line connecting an object on the top to the object it is sent to under the permutation.

Braid diagrams for some permutations. At this point, we can ask whether the permutations with the composition operation are in fact a group. In fact, they are! Let's check. Let σ, τ be permutations of the set $X = \{1, 2, 3, \ldots, n\}$. Then we can specify σ by the list $[\sigma(1), \sigma(2), \ldots, \sigma(n)]$.

Composition of two permutations is again a permutation. Since each permutation contains every element of X exactly once, the composition τoσ must also contain each element of X exactly once. Identity: The permutation $[1, 2, \ldots, n]$ acts as the identity. Inverses : Roughly speaking, if we can mix things up, we can just as easily sort them back out. The 'sorting permutation' of σ is exactly σ^{-1}. Associativity : Suppose we compose three permutations, σ, τ and ρ. In the braid notation, this just means placing the three braids on top of each other top-to-bottom, and then 'forgetting' the two sets of intermediate dots. (TODO : a picture!) Associativity is tantamount to forgetting the two sets of dots in two different orders; the resulting picture is the same either way, so composition of permutations is associative.

Q.17 ***Write short notes on following :***

(a) Monoid

(b) Cosets

(c) Homomorphism and isomorphism.

Ans.(a) Monoid : In abstract algebra, a branch of mathematics, a monoid is an algebraic structure with a single associative binary operation and an identity element.

Monoids are studied in semigroup theory, because they are semigroups with identity. Monoids occur in several branches of mathematics; for instance, they can be regarded as categories with a single object. Thus, they capture the idea of function composition within a set. In fact, all functions from a set into itself from naturally a monoid with respect to function composition. Monoids are also commonly used in computer science, both in its foundational aspects and in practical programming. The set of string built from a given set of character is a free monoid. The transition monoid and systactic monoid are used in describing finite-state machines,, whereas trace monoids and history monoids provide a foundation for process calculi and concurrent computing. Some of the more important results in the study of monoids are the Krohn-Rhodes theorem.

Ans.(b) Cosets

Left Coset : It is very important in group theory (and in abstract algebra more generally). Incidentally, this definition talks about left cosets. Right cosets are important too, but we will consider them another time.

Definition : Let G be a group and $H \subset G$ a subgroup. A left coset of H in G is a subset of G of the form gH for some $g \in G$.

Multiplying element and sets : Of course, the expression gH does not make immediate sense from the group axioms. What is means, by definition, is

$$gH = \{gh \mid h \in H\}$$

To put this another way, the golden rule is this : if you know that $f \in gH$, then we can conclude that there is some $h \in H$ so that $f = gh$.

Here is an example of how the golden rule works.

Applying the golden rule :

Consider $G = S_4$ and $H = \{id, (1, 2)\}$. If $g = (2, 3, 4)$, then $gH = \{(2, 3, 4)(2, 3, 4)(1, 2)\} = \{(2, 3, 4)\}, (1, 3, 4, 2)\}$. Now let $f = (3, 4, 2, 1)$ – this is an element of gH. Which $h \in H$ satisfies $f = gh$?

Or if $g = (1, 3)(2, 4)$, then $gH = \{(1, 3)(2, 4), (1, 4, 2, 3)\}$. If you let $f = (1, 4, 2, 3)$, which $h \in H$ satisfies $f = gh$ this time?

n the box below, compute the two cosets $g_1H \subset S_4$ and $g_2H \subset S_4$ for

$$H = \{id, (1, 2, 3, 4), (1, 3)(2, 4), (1, 4, 3, 2)\}$$

and $g_1 = (1, 3, 2)$, $g_2 = (1, 2, 3\ 4)$

One of two cosets is equal to H itself; the other is disjoint from H.

One coset can have many representatives : Compute another example – in this case the group operation is addition, so we write a + b rather than ab, and similarly cosets are written a + H rather than aH.

Cosets in Z

Let $G = Z$ and $H = 5Z = \{5n \mid n \in Z\} = \{\ldots, -5, 0, 5, 10, \ldots\}$. The coset 2 + H is the set $\{2 + 5n \mid n \in Z\} = \{\ldots, -8, -3, 2, 7, 12, \ldots\}$, which is of course just the set of numbers congruent to 2 mod 5.

We say that 2 is a representative of the coset 2 + H.

Which of the numbers 17, 152, 21, –18, –2 lie in the set 2 + H?

Now calculate 7 + H =

We should see that 7 + H is exactly the same subset of Z as 2 + H. Therefore we can also say that 7 is a representative of 2 + H. The point is that 7 and 2 are the same mod 5. In fact, so are –8, 12, 152 or indeed any other element of the set 2 + H. We can refer to any of them as a representative of this coset.

Ans.(c) Homomorphism and Isomorphism : Let $(G, \cdot)$ and $(G', *)$ be groups. A homomorphism is a set map $\phi : G \to G'$ that preserves the group operation in the respective groups; that is,

$$\phi(a \cdot b) = \phi(a) * \phi(b)$$

for all $a, b \in G$.

Example :

1. Consider the groups $(GL_2(R)$, matrix multiplication), $(R\backslash\{0\}, \times)$. Define $\phi : GL_2(R) \to R\backslash\{0\}$ by

$\phi(A) = \det(A)$.

Then $\phi(AB) = \det(AB) = \det(A)\det(B) = \phi(A)\phi(B)$.

2. If V and W are vector spaces, then $(V, +)$ and $(W, +)$ are groups. Let $T : V \to W$ be any linear transformation. Then T is a group homomorphism. Indeed,

$T(v_1 + v_2) = T(v_1) + T(v_2)$

by linearity.

$\phi(a) = [a]$

Then $\phi(a + b) = [a + b] = [a] + [b] = \phi(a) + \phi(b)$.

3. Consider the groups $(R, +)$ and $(R_{>0}, \times)$.
Define $\phi : R \to R_{>0}$ by

$\phi(a) = 2^a$.

Then $\phi(a + b) = 2^{a+b} = 2^a \cdot 2^b = \phi(a) \cdot \phi(b)$.

4. For any group $(G, \cdot)$, we have identity homomorphism $I : G \to G$ given by

$I(g) = g$

for all $g \in G$.

5. Define $\phi : S_n \to Z/2Z$ by :
$\phi(\sigma) = 1$ if σ is an odd permutation, $\phi(\sigma) = 0$ if σ is an even permutation. Exercise : ϕ is a homomorphism.

Definition : Let $\phi : G \to G'$ be a homomorphism. We say G is the domain of ϕ and G' is the codomain of ϕ.

We now discuss some properties of homomorphisms.

Theorem : Let $\phi : G \to G'$ be a homomorphism. For all $a \in G, n \in Z$, the following hold

1. $\phi(e_G) = e_{G'}$
2. $\phi(a)^{-1} = \phi(a^{-1})$
3. $\phi(a_1 \ldots a_n) = \phi(a_1) \ldots \phi(a_n)$
4. $\phi(a)^n = \phi(a^n)$

Proof :

1. $\phi(e_G) = \phi(e_G)\phi(e_G)$ so multiplying by $\phi(e_G)^{-1}$ (on any side) implies $\phi(e_G) = e_{G'}$.
2. $e_{G'} = \phi(a \cdot a^{-1}) = \phi(a)\phi(a^{-1})$, so $\phi(a)^{-1} = \phi(a^{-1})$.
3. Induction on multiplicative definition.
4. Part 3, where $a_i = a$ for all i.

Definition : A group homomorphism $\phi : G \to G'$ is an isomorphism if ϕ is a bijection. If there is an isomorphism between G and G' we say G and G' are isomorphic. This is denoted by $G \cong G'$.

Example : In the example with different groups, 4 and 5 are isomorphisms.

Remark : If G is a group, the set of bijection $\{\phi : G \to G \mid \phi$ is a bijection$\}$ is a group under composition. Indeed, one can check that if $\phi : G \to G$ and $\psi : G \to G$ are isomorphisms, then $\phi o \psi$ is an isomorphism. The fact that $\phi o \psi$ is a bijection has nothing to do with the group structure. To see that $\phi o \psi$ is a homomorphism, observe

$$\phi(\psi(ab)) = \phi(\psi(a)\ \psi(b)) = \phi(\psi(a))\ \phi(\psi(b)).$$

Furthermore If $\phi : G \to G$ is a bijective homomorphism and we define $\psi : G \to G$ by assigning $\psi(a)$ to the unique value $a' \in G$ such that $\phi(a') = a$, then by definition $\psi o \phi = \phi o \psi = I$, (where I is the identity map). We need to justify that ψ, the candidate inverse for ϕ, is indeed a homomorphism. In this light, if $a', b' \in G$ then there exist a, b such that $\psi(a') = a$ and $\psi(b') = b$, and this means $\phi(a) = a'$ and $\phi(b) = b'$ and hence $\phi(ab) = a'b'$, so $\psi(a'b') = ab = \psi(a')\psi(b')$. Hence $\phi = \psi^{-1}$ an isomorphism of G.

This group of isomorphism from a group G to itself is called the automosphism group of G, and is denoted Aut (G).

Given a homomorphism $\phi : G \to G'$ there are subgroup of each that can indicate to us whether ϕ is injective or surjective.

Definition : Let $\phi : G \to G'$ be a homomorphism. Define $\ker(\phi) = \{g \in G : \phi(g) = e_{G'}\}$. This is called the kernel of ϕ. Define $\text{im}(\phi) = \{\phi(g) : g \in G\}$. This is called the image of ϕ. We usually use the notation $\phi(G)$ for im(G).

Example : If $\phi : GL_2(R) \to R\backslash\{0\}$ is given by $\phi(A) = \det(A)$ then $\ker(\phi(A)) = SL_2(R)$ and $\text{im}(\phi) = R\backslash\{0\}$.

❑❑❑

GRAPH THEORY 5

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 ***Write short note on isomorphism of graphs.***

[R.T.U. 2019]

OR

Define the isomorphic graph with example.

[R.T.U. 2014]

Ans. Isomorphic Graphs : Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are said to be Isomorphic to each other if there exists a bijection mapping f from V_1 to V_2.

i.e., $f : V_1 \to V_2$ such that for each of the vertices v_i, v_j of V_1,

$\{v_i, v_j\} \in E_1 \Rightarrow \{f(v_i), f(v_j)\} \in E_2$

The function f is called an Isomorphism from G_1 to G_2.

It is immediately apparent by the definition of isomorphism that two isomorphic graphs must have

(a) The same number of vertices

(b) The same number of edges

(c) An equal number of vertices with a given degree *i.e.*, same degree sequence.

However, these conditions are by no means sufficient. For instance, the two graphs (given 'below) satisfy all three conditions, yet they are not isomorphic.

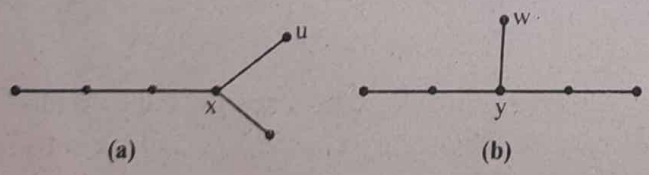


Fig. : Two graphs that are not isomorphic

The graphs in fig. (a) and (b) are not isomorphic can be shown as follows : If the graph (a) were to be isomorphic to the one in (b), vertex x must correspond to y, because there are no other vertices of degree three. Now in (b) there only one pendant vertex, w, adjacent to y. While in (a) the are two pendant vertices, u′ and v, adjacent to x. Thus th adjacency relationship is not preserved. Hence (a) and (b) are not isomorphic.

Q.2 ***Write short note on planar graphs.*** ***[R.T.U. 201***

OR

Define the planar graph with example.

[R.T.U. 201

Ans. Planar Graphs : A graph is called planar if it can b drawn in a plane such that no two edges intersect except a their common end vertices, if any.

Note that, if a graph G has been drawn with crossin edges, it does not mean that G is non–planar. There may b other planar representation of G. For example, following ar planar graphs.

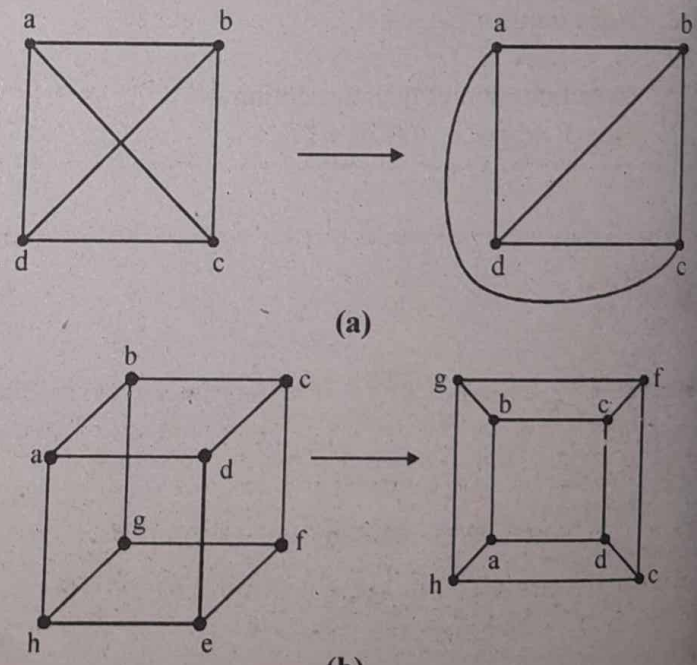


Fig.

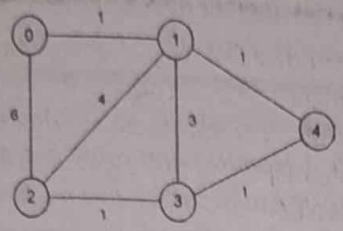
Drawing of a geometric representation of a graph on any surface such that no edges interest is called embedding.

Q.3 ***Define the weighted graph with example.***

[R.T.U. 2015]

Ans. Weighted Graph : A graph having a weight (a number), associated with each edge. Some algorithms require all weights to be nonnegative, integral, positive, etc. These are also known as edge-weighted graph.

Example: The edge weights are 6,1,4,1,3,1,1 from left to right.



Q.4 ***Write short note on Eulerian and Hamiltonian Graphs.***

[R.T.U. 2012]

Ans. Eulerian Graph : A closed walk in a graph G, which passes through every edge of G exactly once and which ends at the first vertex thus forming a closed circuit is called an *Euler line* in G, and a graph that consists of an Euler line is called an *Eulerian graph*.

$v_1\ e_1\ v_2\ e_2\ v_3\ e_3\ v_4\ e_4\ v_2\ e_5\ v_5\ e_6\ v_1$

Hamiltonian Graph : The Hamiltonian graph G is that which has a closed path passing through every vertex exactly once though it may not pass through every edge of G.

PART-B

Q.5 ***Write short note on cut sets.***

[R.T.U. 2019]

Ans. Cut Set : A cut set of a connected graph G is a set S of edges with the following properties

- The removal of all edges in S disconnects G.
- The removal of some (but not all) of edges in S does not disconnects G.

As an example consider the following graph

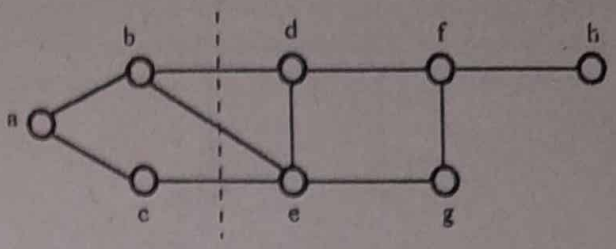


Fig.

We can disconnect G by removing the three edges bd, bc, and ce, but we cannot disconnect it by removing just two of thes edges. Note that a cut set is a set of edges in which no edges is redundant.

Q.6 ***Write short note on vertex connectivity.***

[R.T.U. 2019]

Ans. Vertex Connectivity : The connectivity (or vertex connectivity) K(G) of a connected graph G (other than a complete graph) is the minimum number of vertices whose removal disconnects G. When $K(G) \geq k$, the graph is said to be k-connected (or k-vertex connected). When we remove a vertex, we must also remove the edges incident to it. As an example consider following graphs.

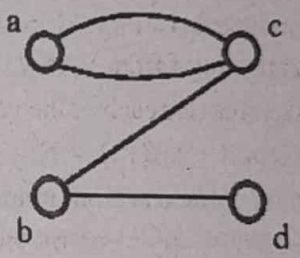


Fig.

The above graph G can be disconnected by removal of single vertex (either b or c). The G has connectivity 1.

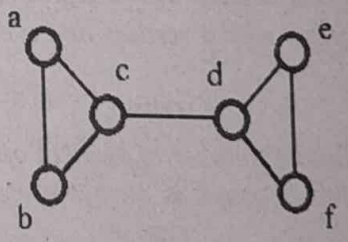


Fig.

The above graph G can be disconnected by removal of single vertex (either c or d). The vertex c or d is a cut-vertex. The G has connectivity 1.

The above G can be disconnected by removing just one vertex i.e., vertex c. The vertex c is the cut-vertex. The G has connectivity 1.

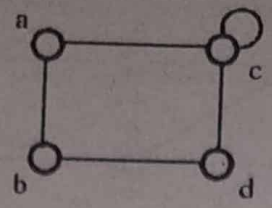


Fig.

The above G cannot be disconnected by removing a single vertex, but the removal of two non-adjacent vertices (such as b and c) disconnects it. The G has connectivity 2.

Q.7 ***Prove that the chromatic number of a graph will not exceed by more than one, the maximum degree of the vertices in a graph.*** *[R.T.U. 2017, 2011]*

Ans. Proof : Let $\Delta(G)$ be the maximum of the degrees of the vertices of a graph G.

Let the number of vertices in a graph is denoted by $|V|$. If $|V| = 1$, then $\Delta(G) = 0$ and $K(G) = 1$, so the result holds. Now let K be an integer and $K \geq 1$. Assume that the result holds for all graph with $|V| = K$ vertices.

Let G be a graph with $(K + 1)$ vertices. Let v be any vertex of G and let $G_0 = G/\{v\}$ is a subgraph of G obtained by deleting v from G.

Since G_0 has K vertices so we can use the induction hypothesis to conclude that

$$K(G_0) \leq 1 + \Delta(G_0)$$

Also, $\Delta(G_0) \leq \Delta(G)$

$\therefore \quad K(G_0) \leq 1 + \Delta(G)$

So G_0 can be colored with at most $1 + \Delta(G)$ colors. Since there can be atmost $\Delta(G)$ vertices adjacent to v, one of the available $1 + \Delta(G)$ colors remains for v. Thus G can be colored with atmost $1 + \Delta(G)$ colors. **Hence proved.**

Q.8 ***Show the total number of odd degree vertices of a (p, q), graph (graph with p vertices and q edges) is even.*** *[R.T.U. 2012]*

OR

Prove that the number of vertices of odd degrees in an undirected graph is always even. *[R.T.U. 2016]*

OR

Prove that the number of odd degree vertices in a graph G is always even.

[R.T.U. 2011, 2009; Raj. Univ. 2007, 2006, 2005]

Ans. Let G (V, E) be a graph. $V_e \subset V$ and $V_o \subset V$ be the set of vertices of even degree and odd degree respectively. The $V_e \cup V_o$. Also let *n* be the number of edges.

$$\sum_{v \in V} \deg(v) = \sum_{v \in V_e} \deg(v) + \sum_{v \in V_o} \deg(v) = 2n$$

Since $\deg(v)$ is even for $v \in V_e$ $\sum_{v \in V_e} \deg(v) = m$ (say) is also even.

$$\Rightarrow \sum_{v \in V_o} \deg(v) = 2n - m = 2n - 2k$$

($m = 2k$ for some integer k)

$= 2(n - k) = 2l$ ($l = n - k$ is an integer)

= an even number

Since all the terms in the sum $\sum_{v \in V_o} \deg(v)$ are odd, there must be an even number of such terms. Thus, number of odd degree vertices is even.

Q.9 ***(a) Give an example of connected graph that has***
(i) A Hamiltonian cycle but no Euler circuit
(ii) A Euler circuit but no Hamiltonian cycle
(b) What is the length of shortest path between the vertices a to z in the following weighted graph

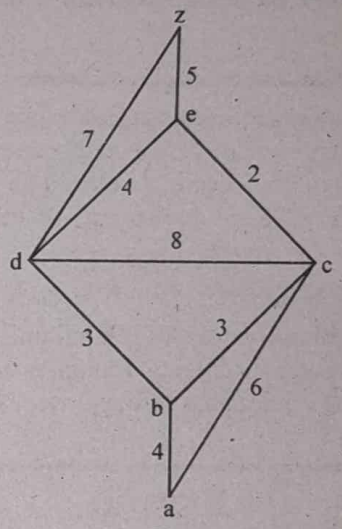


Fig.

[R.T.U. 2016]

Ans.(a)(i) A Hamiltonian cycle but no Euler circuit

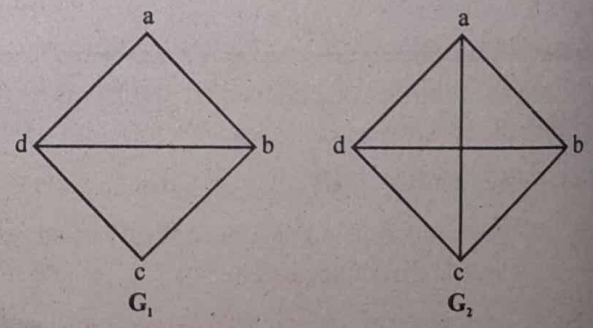


Fig. 1

(ii) Euler circuit but no Hamiltonian cycle

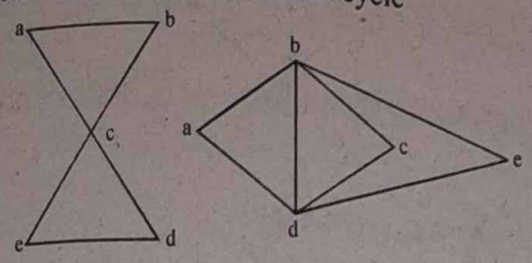


Fig. 2

Ans.(b) The weighted graph is as follows:

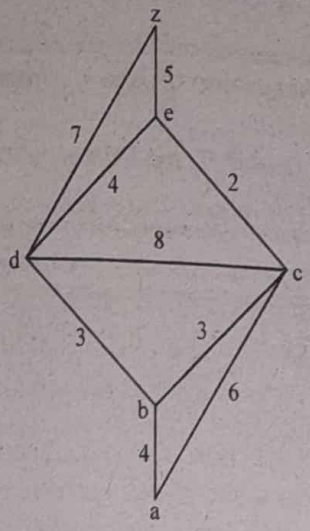


Fig.

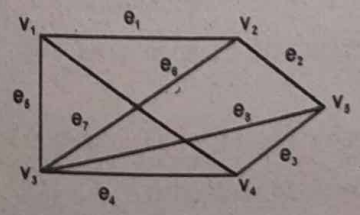
Initially, let $L(v_1) = 0$, $L(v) = \infty$, $v \neq v_1$

Constructing temporary and permanent labels as follows

v	v_1	v_2	v_3	v_4	v_5	v_6
$L_0(v)$	[0]	∞	∞	∞	∞	∞
$L_1(v)$	[0]	[4]	6	∞	∞	∞
$L_2(v)$	[0]	[4]	[6]	7	∞	∞
$L_3(v)$	[0]	[4]	[6]	[7]	8	∞
$L_4(v)$	[0]	[4]	[6]	[7]	[8]	14
$L_5(v)$	[0]	[4]	[6]	[7]	[8]	[13]

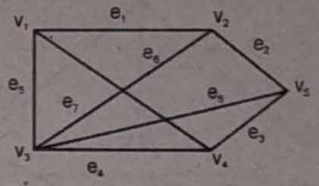
Thus, the length of the shortest path is 13 and the path is $v_1 \rightarrow v_3 \rightarrow v_5 \rightarrow v_6$

Q.10 ***Define spanning tree in a graph. Find five spanning trees for the graph shown in figure and write the sets of branches and chords corresponding to these spanning trees.***

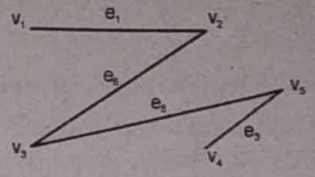


[R.T.U. 2015]

Ans. Spanning tree : A tree is spanning tree of graph G if it spans G, i.e., include every vertex of G and is a subgraph of G (every edge in the tree belongs to G).

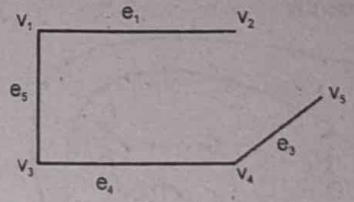


Spanning Tree 1 :



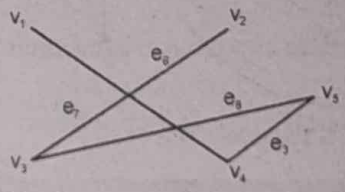
$E = \{e_1, e_3, e_6, e_8\}$

Spanning Tree 2 :



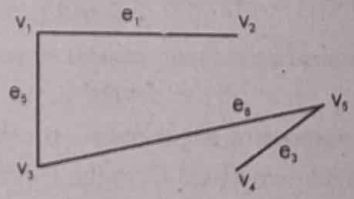
$E = \{e_1, e_3, e_4, e_5\}$

Spanning Tree 3 :



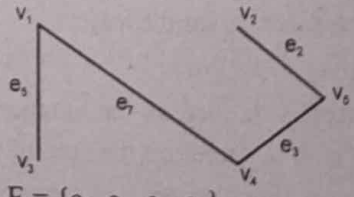
$E = \{e_3, e_6, e_7, e_8\}$

Spanning Tree 4 :



$E = \{e_1, e_3, e_5, e_8\}$

Spanning Tree 5 :



$E = \{e_2, e_3, e_5, e_7\}$

All the 5 are minimum spanning trees (trees with minimum edges) and have no branching sets.

Q.11 ***State the Kuratowski's theorem.*** *[R.T.U. 2013, 2012]*

Ans. Kuratowskis Theorem : A simple graph is planar if and only if it does not contain a subgraph homeomorphic to $K_{3,3}$ or K_5.

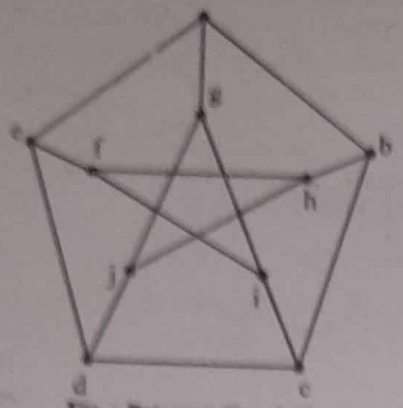


Fig. : Peterson Graph G

For example, the Peterson graph is not planar as it has a subgraph homeomorphic to $K_{3,3}$

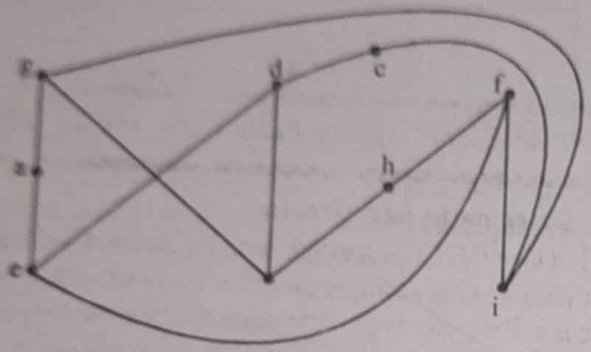


Fig. : Subgraph of G Homeomorphic to $K_{3,3}$

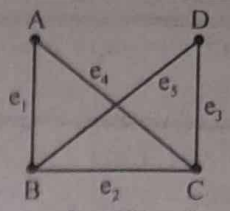
Q.12 ***Show the sum of the degrees of all the vertices in a graph is equal to twice the number of edges in the graph.*** ***[R.T.U. 2011]***

Ans. The given statement, "sum of the degree of all the vertices in G is twice the number of edges in G", is called **Handshaking Lemma** which is derived from the fact that at a gathering the total no. of handshakes is equal to twice the number of hands involved. Replacing 'handshakes' with 'edges' and 'hands' with 'vertices' gives the lemma.

Now proof can be given as follows :

Since degree of vertex is defined as the number of edges (or arcs) connected with it, therefore the sum of the degree counts the total no. of times an edge is connected with a vertex. As every edge is connected with exactly two vertices, so each edge is counted twice at each of its ends. This implies that the sum of the degree equals twice the number of edges.

For example : In the given graph G



deg (A) = 2 deg (B) = 3

deg (C) = 3 deg (D) = 2

The sum of degree equals 10, which as expected twice the number of edges.

Q.13 ***(a) Show that isomorphism of simple graphs is an equivalence relation.***

(b) Suppose that G and H are isomorphic simple graphs. Show that their complementary graphs $\bar{G}$ and $\bar{H}$ are also isomorphic. ***[Raj. Univ. 200***

Ans.(a) Reflexive : Let G be any simple graph then G is isomorphic to itself by the identity function.

$\therefore$ Isomorphism is reflexive.

Symmetric : Let G is isomorphic to another simple graph H. Then there exists a one-to-one correspondence f from G to H that preserves adjacency and non-adjacency. It follows that f^{-1} is a one-to-one correspondence from H to G that preserves adjacency as well as non adjacency.

Hence isomorphism is symmetric.

Transitive : Let G is isomorphic to H and H is isomorphic to K, then there exist bijections f and g from G to H and from H to K that preserve adjacency and non adjacency. It follows that gof is a bijection from G to K that preserves adjacency and non adjacency.

Hence isomorphism is transitive.

Thus, isomorphism is an equivalence relation.

Ans.(b) Let f be an isomorphism from G onto H. Then f is a one-to-one correspondence from G onto H and f also preserves adjacency as well as non adjacency of vertices.

Let ϕ be a mapping from $\bar{G}$ to $\bar{H}$. Further the adjacent vertices in $\bar{G}$ are not adjacent in G also the non adjacent vertices in $\bar{G}$ are adjacent in G, similar statement holds for the graph H.

As G and H are simple so $\bar{G}$ and $\bar{H}$ must be simple graph. Thus ϕ is a one-to-one correspondence from $\bar{G}$ onto $\bar{H}$ and ϕ also preserves the adjacency as well as non-adjacency of vertices.

Hence ϕ is an isomorphism from $\bar{G}$ onto $\bar{H}$. Therefore, $\bar{G}$ and $\bar{H}$ are isomorphic to each other.

Q.14 *Write short note on Graph Homeomorphism, paths and circuits.*

Ans. Graph Homeomorphism : If a graph G has a vertex v of degree 2 and edges (v, v_1), (v, v_2) with $v_1 \neq v_2$, we say that the edges (v, v_1) and (v, v_2) are in series. Deleting such vertex v and replacing (v, v_1) and (v, v_2) with (v_1, v_2) is called a series reduction. For instance, in the third graph of Fig. the edges (v_6, v_2) and (v_6, v_4) are in series. By removing vertex v_6 we get the first graph in the left.

The opposite of a series reduction is an elementary subdivision. It consists of replacing an edge (u, v) with two edges (u, w) and (w, v), where w is a new vertex.

Two graphs are said to be homeomorphic if they are isomorphic or can be reduced to isomorphic graphs by a sequence of series reductions. Equivalently; two graphs are homeomorphic if they can be obtained from the same graph by a sequence of elementary subdivisions.

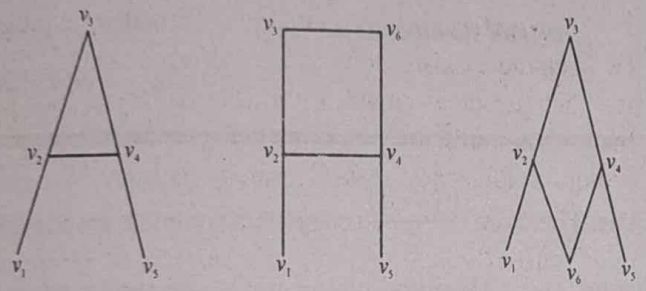


Fig. : Three Homeomorphic Graphs

Note that if a graph G is planar, then all graphs homeomorphic to G are also planar.

Paths and Circuits : A path from v_0 to v_n of length n is a sequence of $(n + 1)$ vertices (v_k) and n edges (e_k) of the form $v_0, e_1, v_1, e_2, v_2, \ldots e_n, v_n$, where each edge e_k connects v_{k-1} with v_k (and points from v_{k-1} to v_k if the edge is directed). The path may be specified by giving only the sequence of edegs $e_1 \ldots e_n$. If there are no multiple edges we can specify the path by giving only the vertices: $v_0, v_1, \ldots, v_n$. The path is a circuit (or cycle) if it begins and ends at the same vertex, *i.e.*, $v_0 = v_n$ and has length greater than zero. A path or circuit is simple if it does not contain the same edge twice.

A graph that does not contain any circuit is called acyclic. The number of edges appearing in the sequence is called the length of the path. A path with no repeated edges is called a trail. A (path or trail) is trivial if it has only one vertex and no edges, otherwise non-trivial. A non–trivial closed trail from a vertex u to itself is called a circuit or a circuit is a closed path of non–zero length from a vertex u to u with no repeated edges. A cycle is a closed path of non zero length from a vertex u to u with no repeated edges and no repeated vertices except initial and terminal vertices. Thus,

- A cycle is a circuit that does not contain any repetition of vertices except the initial and terminal vertices.
- A cycle of length n is called a n–cycle.
- A cycle is called even (or odd) if it contains an even (or odd) number of edges.

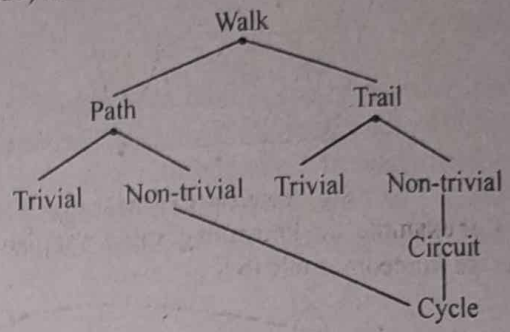


Fig.

Q.15 *Write a short note on Euler Paths and Circuits.*

Ans. Euler paths and circuits

Let G = (V, E) be a graph with no isolated vertices. An Euler path, in G is a simple path that transverses every edge of the graph exactly once. Analogously, an Euler circuit in G is a simple circuit that transverses every edge of the graph exactly once.

Theorem 1 : *If a graph* G *has an Euler circuit then every vertex of the graph has even degree.*

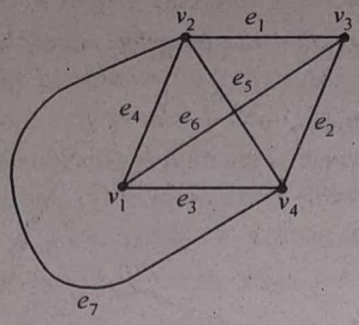
Proof : Let G be a graph with an Euler circuit. Start at some vertex on the circuit and follow the circuit from vertex to vertex, erasing each edge as you go along it. When you go through a vertex you erase one edge going in and one edge going out, or else you erase a loop. Either way, the erasure reduces the degree of the vertex by 2. Eventually every edge gets erased and all the vertices have degree 0. So all vertices must have had even degree to begin with.

It follows from the above theorem that if a graph has a vertex with odd degree then the graph can not have an Euler circuit. The following provide a converse to the above theorem.

Let G be a connected multigraph. Then C contains an Euler circuit if and only if its (G) vertices have even degree. Also, G contains an Euler path from vertex a to vertex b $(\neq a)$ if and only if a and b have odd degree, and all its other vertices have even degree.

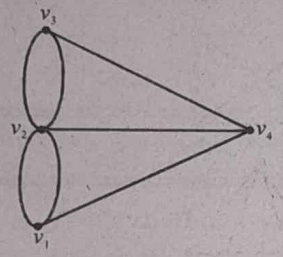
Theorem 2 : (Euler Theorem) : *If all the vertices of a connected graph have even degree, then the graph has an Euler circuit.*

Example 1 : *Show that the following graph has no Euler circuit :*



Solution: Vertices v_1 and v_3 both have degree 3, which is odd. Hence, by the remark following the Theorem 3.11, this graph does not have an Euler circuit.

Example 2 : (Königsberg Seven Bridge Problem) : *Does the following graph has an Euler circuit ?*



Solution: Here deg $(v_1) = 3$, deg $(v_2) = 5$, deg $(v_3) = 3$, deg $(v_4) = 3$.

Since more than two vertices are of odd degree, the given graph does not have an Euler path or circuit.

Procedure to Determine whether a Graph G has an Euler Circuit :

Step I : First list the degrees of all the vertices in the graph.

Step II :

(a) If degree of any vertex is zero, the graph is not connected therefore it cannot have an Euler path or circuit.

(b) If all the degrees are even then G has both Euler path and Euler circuit.

(c) If exactly two vertices are of odd degree then the graph has Euler path only, no Euler circuit.

(d) If degree of more than two vertices is odd, then graph does not have any Euler path or circuit.

FLEURY'S ALGORITHM

This algorithm is used to construct an Euler circuit.

Let G = (V, E) be a graph and C be the Euler circuit.

Step I : Choose a vertex u as the starting vertex.

Step II : Select an edge $e_1 = \{u, v_1\}$. Let $V_C : u, v_1$ and $E_C : e_1$. Remove e_1 from E and let $G_1 = (V, E–\{G\})$ be the resulting subgraph of G.

Step III : Suppose that $V_C = v, u, v_1, v_2, \ldots, v_n$ and E_C $e_1, e_2, \ldots, e_m$ have been constructed so far and that all of these edges and any resulting isolated vertices have been removed from V and E to form G_m.

Since v_m is of even degree and e_m ends at v_m there is an edge e_{m+1} in G_m that incident on v_m. If there are more than one such edges, select one that is not a bridge for G_m, denote the vertex of e_{m+1} other than v_m by v_{m+1} and extend V_C to V_C : $u, v_1, v_2, \ldots v_m, v_{m+1}$ and E_C to E_C: $e_1, e_2, \ldots e_m, e_{m+1}$. Then delete e_{m+1} and any isolated vertices from G_m to G_{m+1}.

Step IV : Repeat step III untill no edges remain in E.

PART-C

Q.16 In a complete graph with n - vertices there are $\frac{(n-1)}{2}$ ***edge disjoint Hamiltonian circuits, if n is an odd number ≥ 3.***

[R.T.U. 2019]

Ans. There are $\frac{n(n-1)}{2}$ edges in a complete graph with n-vertex and a Hamilton circuit in G consist of n edges, So number of edges disjoint Hamilton circuit is G cannot exceed $\frac{n-1}{2}$. That these are $\frac{n-1}{2}$ edges disjoint Hamilton circuits. When n is odd the subgraph is diagram given below is a Hamilton.

Circuits vertex fixed on a circle rotate polygon pattern clockwise $\frac{360}{n-1}, \frac{2\times360}{n-1}, \ldots \frac{n-3}{2}\times\frac{360}{n-1}$ edges.

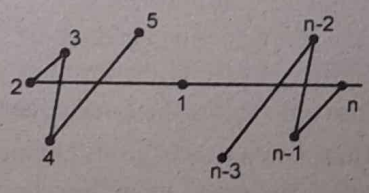
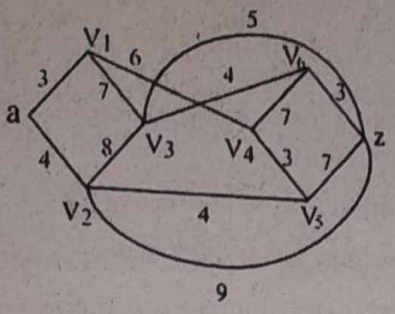


Fig.

Hamilton circuits has no edges in common with any of the previous one. Therefore $\frac{n-3}{2}$ new Hamilton circuits all edges disjoint form fig. and also disjoint among themselves.

Q.17 (a) ***Find the shortest path from a to z in the following graph-***



(b) Suppose that a connected planar graph has 30 vertices, each of degree three. Into how many regions is the plane divided by a planar representation of this graph. ***[R.T.U. 2019]***

Ans.(a) Find the shortest path from a to z

(i) $av_1, v_1v_4, v_4v_6, v_6z$

$3 + 6 + 7 + 3 = 19$

(ii) $av_1, v_1v_3, v_3v_6, v_6z$

$3 + 7 + 4 + 3 = 17$

(iii) $av_1, v_1v_4, v_4v_5, v_5z$

$3 + 6 + 3 + 7 = 19$

(iv) $av_2, v_2v_3, v_3v_6, v_6z$

$4 + 8 + 4 + 3 = 19$

(v) $av_2, v_2v_3, v_3v_6, v_6v_4, v_4v_5, v_5z$

$4 + 8 + 4 + 7 + 3 + 7 = 33$

(vi) av_2, v_2z

$4 + 9 = 13$

(vii) av_1, v_1v_3, v_3z

$3 + 8 + 5 = 16$

(viii) av_2, v_2v_3, v_3z

$4 + 8 + 5 = 17$

shortest path (av_2, v_2z)

Ans.(b) $n = 3$ and $\deg(v) = 3 \times 30 = 90$

sum of degrees of vertex = 2 × No. of edges

$90 = 2e$

$e = 45$

Region according Euler's formula

$n - e + r + 2 = 30$

$r = 17$ regions

Q.18 (a) ***Sketch the complete graphs $k_n, 1 \le n \le 6$.***

(b) Show that the complete digraph with n-nodes has the maximum number of edges i.e. n(n–1) edges, assuming there are no loops.

(c) Draw graph which is Eulerian as well as Hamiltonian. ***[R.T.U. 2017]***

Ans.(a) The graphs of k_1, k_2, k_3, k_4, k_5 and k_6 are as follows :

(i) k_1

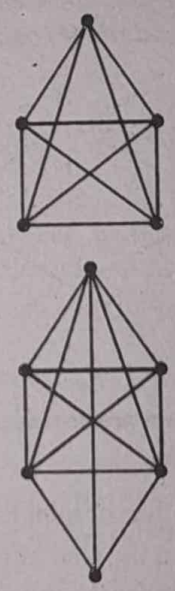
(ii) k_2

(iii) k_3

(iv) k_4

(v) k_5

(vi) k_6



Ans.(b) There are two types of complete digraph, i.e. complete asymmetric digraph and complete symmetric digraph. Out of these two a complete symmetric digraph has more edges than that of complete asymmetric.

Also a complete symmetric digraph with n vertices has

$$2 \times {}^nC_2 \text{ edges i.e.}$$

$$2 \times \frac{n(n-1)}{2} = n(n-1) \text{ edges}$$

Hence Maximum number of edges in a complete digraph is $n(n-1)$.

Ans.(c) Both Euler circuit and Hamiltonian cycle :

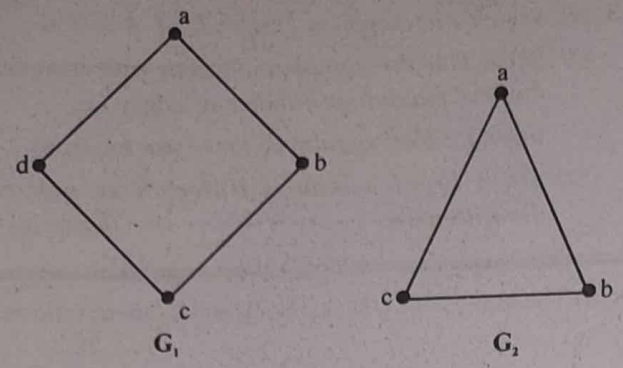


Fig.

In G_1 Euler circuit : a, b, c, d, a
Hamiltonian cycle : a, b, c, d, a

In G_2 Euler circuit and Hamiltonian cycle are a, b, c, a.

Q.19 ***(a) Explain the Minimal Spanning Tree. Also write the Kruskal Algorithm for find Minimal Spanning tree.***

(b) Given the Graph in following figure. Apply Prim's algorithm to obtain the minimal spanning tree.

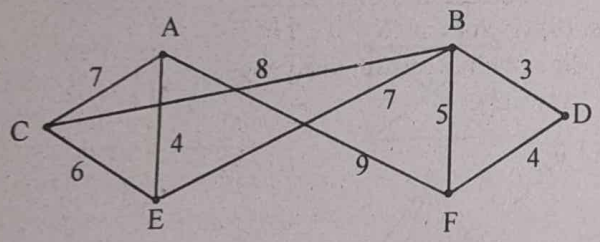


Fig. *[R.T.U. 2012]*

Ans. (a) Minimal Spanning Tree

If a connected weighted tree G, then its minimal spanning tree is a spanning tree of G such that the sum of the weights of its edges is minimum. For instance for the following graph of figure. The spanning tree, shown by thicker lines is the one of minimum weight.

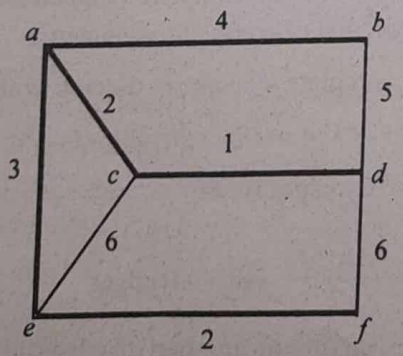


Fig. : Minimum Spanning Tree

Kruskal's algorithm

An algorithm in graph theory that finds a minimal spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimal spanning forest (a minimal spanning tree for each connected component).

Kruskal's algorithm is an example of a greedy algorithm.

This algorithm was written by Joseph Kruskal in 1956.

An algorithm for computing a minimal spanning tree. It maintains a set of partial minimal spanning trees, and repeatedly adds the shortest edge in the graph whose vertices are in different partial minimal spanning trees.

Algorithm of finding minimal spanning tree by Kruskal's algorithm

Step 1 : Create a forest F (a set of trees), where each vertex in the graph is a separate tree

Step 2 : Create a set S containing all the edges in the graph

Step 3 : While S is nonempty

(a) remove an edge with minimum weight from S

(b) if that edge connects two different trees, then add it to the forest, combining two trees into a single tree

(c) otherwise discard that edge

At the termination of the algorithm, the forest has only one component and forms a minimal spanning tree of the graph.

For example determine the minimal spanning tree in the following graph by applying Kruskal's algorithm.

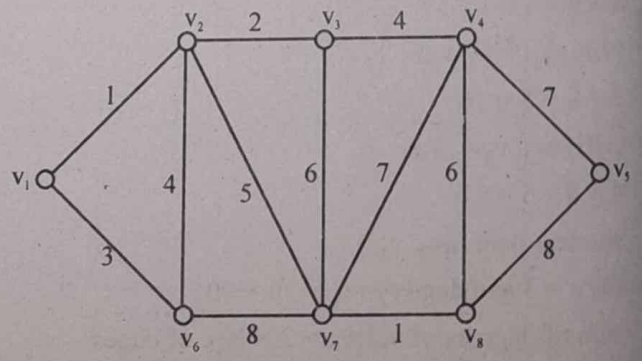
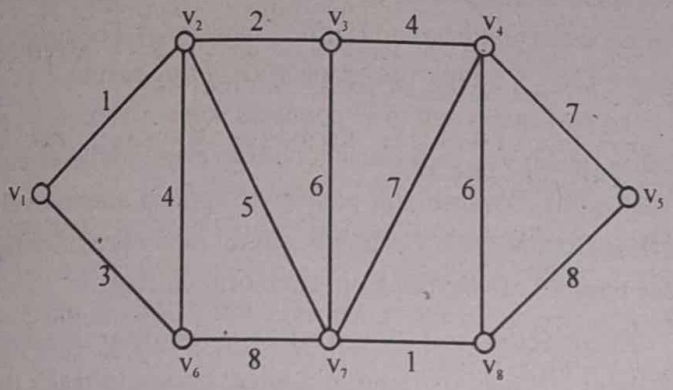


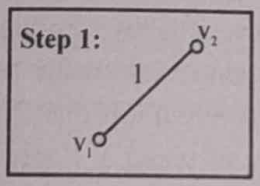
Fig.

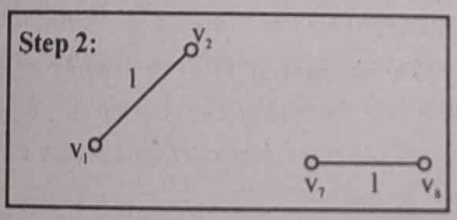
Using the above graph, here are the steps to the minimal spanning tree, using Kruskal's algorithm:

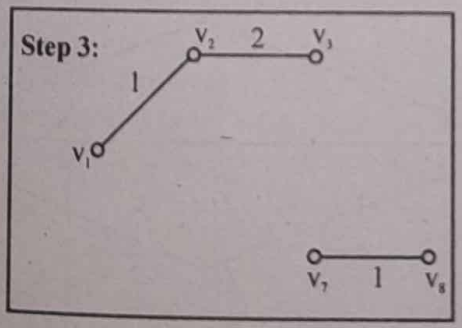
1. v_1 to v_2 –cost is 1 –add to tree
2. v_7 to v_8 –cost is 1 –add to tree
3. v_2 to v_3 – cost is 2 – add to tree

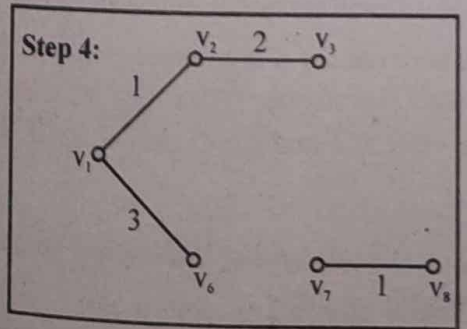
4. v_1 to v_6 – cost is 3 – add to tree
5. v_2 to v_6 – cost is 4 – reject because it forms a circuit
6. v_3 to v_4 – cost is 4 – add to tree
7. v_2 to v_7 – cost is 5 – add to tree
8. v_3 to v_7 – cost is 6 – reject because it forms a circuit
9. v_4 to v_8 – cost is 6 – reject because it forms a circuit
10. v_4 to v_7 – cost is 7 – reject because it forms a circuit
11. v_4 to v_5 – cost is 7 – add to tree
12. We stop here, because n – 1 edge has been added. We are left with the minimal spanning tree, with a total weight of 23.

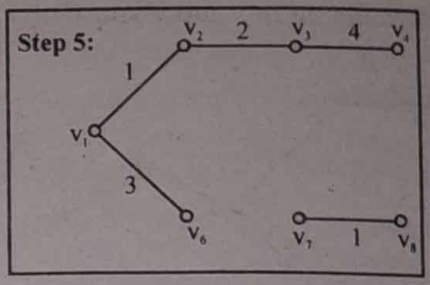


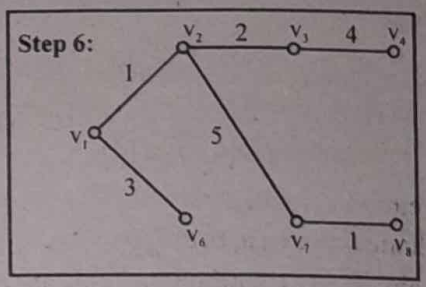


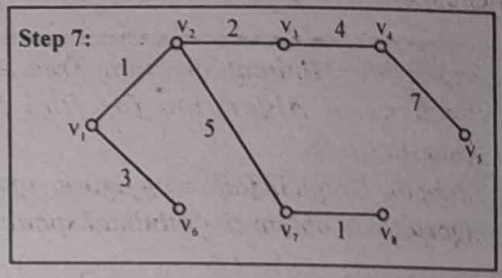








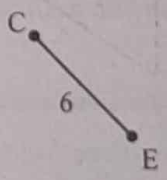




Ans. (b) (i) Start at C

CE is the lowest-weighted edge (6).

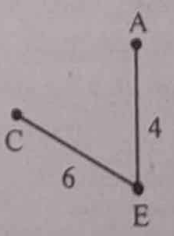
Draw it in.



(ii) From C or E

EA is the lowest-weighted edge (4).

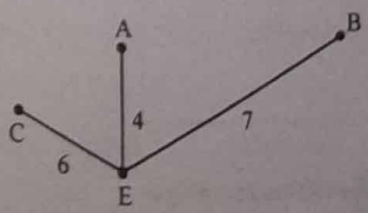
Draw it in.



(iii) From C, E or A

EB is the lowest-weighted edge (7).

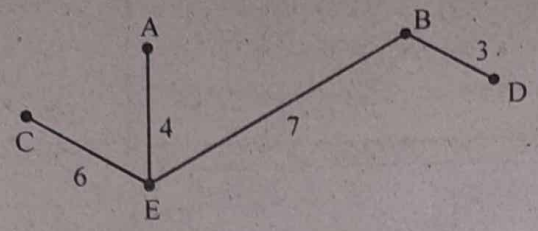
Draw it in.



(iv) From C, E, A or B

BD is the lowest-weighted edge (3).

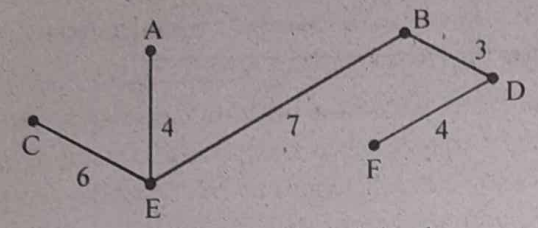
Draw it in.



(v) From C, E, A, B or D

DF is the lowest-weighted edge (4).

Draw it in.



All vertices have now been joined.

The minimum spanning tree is determined.

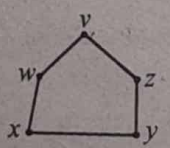
Minimum spanning tree length = 6 + 4 + 7 + 3 + 4

= 24 units.

Q.20 ***Write a detailed note on Hamiltonian path and circuits with example.***

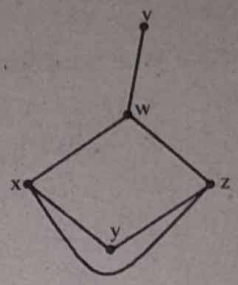
Ans. Hamiltonian Path and Circuits : A Hamiltonian circuit in a graph G is a circuit, that contains each vertex of G once (except for the starting and ending vertex, which occurs twice). A Hamiltonian path in G is a path (not a circuit) that contains each vertex of G once. Note that by deleting an edge in a Hamiltonian circuit we get a Hamiltonian path, so if a graph has a Hamiltonian circuit, then it also has a Hamiltonian path. The converse is not true, *i.e.*, *a* graph may have a Hamiltonian path but not a Hamiltonian circuit.

Example 1 : *Find a Hamiltonian circuit in the graph :*



Solution: *vwxyzv*

Example 2 : *Show that the following graph has a Hamiltonian path but no Hamiltonian circuit:*



Solution: *vwxyz* is a Hamiltonian path. There is no Hamiltonian circuit since no cycle goes through *v*.

In general it is not easy to determine if a given graph has a Hamiltonian path or circuit, although often it is possible to argue that a graph has no Hamiltonian circuit. For instance if G = (V, E) is a bipartite graph with vertex partition $\{V_1, V_2\}$ (so that each edge in G connects some vertex in V_1 to some vertex in V_2), then G cannot have a Hamiltonian circuit if $|V_1| \neq |V_2|$, because any path must contain alternatively vertices from V_1 and V_2, so any circuit in G must have the same number of vertices from each of both sets.

Edge Removal Argument : Another kind of argument consists of removing edges trying to make the degree of every vertex equal two. For instance in the graph of Fig. we cannot remove any edge because that would make the degree of *b*, *e* or *d* less than 2, so it is impossible to reduce the degree of *a* and *c*. Consequently that graph has no Hamiltonian circuit.

Dirac's Theorem : *If G is a simple graph with n vertices with n ≥ 3 such that the degree of every vertex in G is at least n/2, then G has a Hamiltonian circuit.*

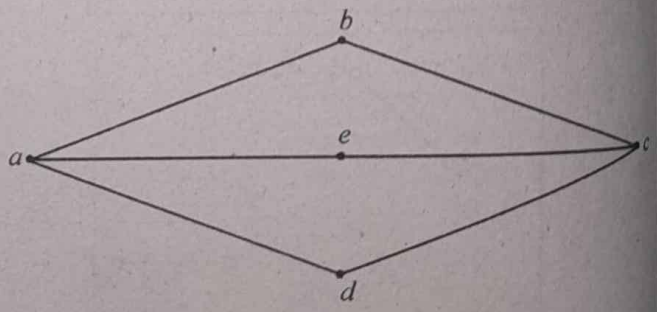
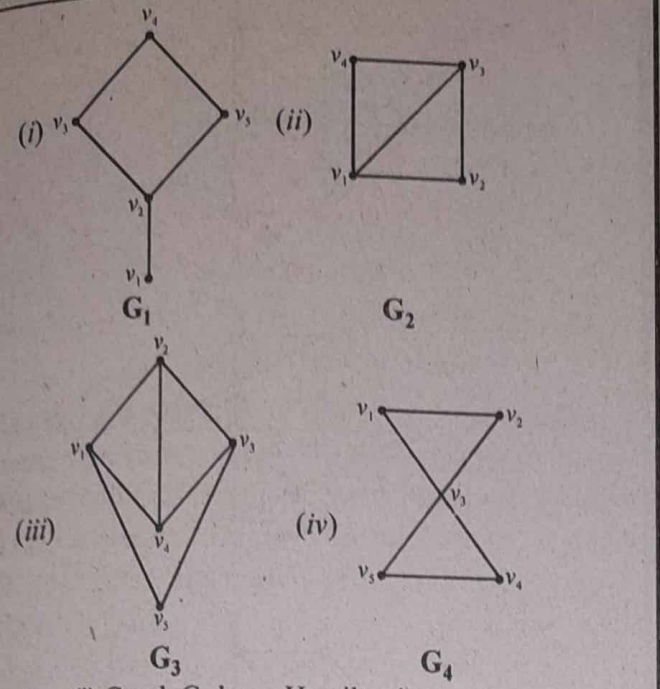


Fig. : Graph without Hamiltonian Circuit

Ore's Theorem : *If G is a simple graph with n vertices with n ≥ 3 such that deg(u) + deg(v) ≥ n for every pair of non-adjacent vertices u and v in G, then G has a Hamiltonian circuit.*

Example 3 : *Which of the following graphs has a Hamiltonian path or cycle :*



Solution: **(i)** Graph G_1 has a Hamiltonian path v_1, v_2, v_3, v_4, v_5 but no Hamiltonian cycle.

(ii) Graph G_2 has a Hamiltonian cycle v_1, v_2, v_3, v_4, v_1.

(iii) Graph G_3 has a Hamiltonian cycle $v_1, v_5, v_3, v_4, v_2, v_1$.

(iv) Graph G_4 has no Hamiltonian path.

Q.21 *Explain shortest path problem with example.*

Ans. Shortest Path : Shortest path between two vertices in a graph is the path of minimum length. Thus, if :

(a) The graph is without weights, the length of path denotes the number of edges in the path and shortest path between two vertices is the path with least number of edges.

(b) The graph is weighted graph, the shortest path between two vertices is the path of minimum length (weight).

Shortest Path Problem : With each edge e of G let there associate a real number $w(e)$, called its weight. Then G, together with these weights on its edges, is called a weighted graph.

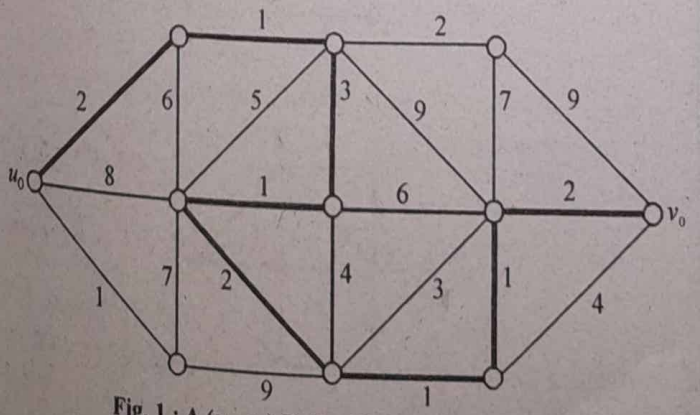


Fig. 1 : A (u_0, v_0) Path of Minimum Weight

Weighted graphs occur frequently in applications of graph theory. In the friendship graph, for example, weight might indicate intensity of friendship; in the communication graph, they could represent the construction of maintenance costs of the various communication links.

If H is a subgraph of a weighted graph, the weight $w(H)$ of H is the sum of the weights $\sum_{e \in E(H)} w(e)$ on its edges. Many optimization problems amount to finding, in a weighted graph, a subgraph of a certain type with minimum (or maximum) weight.

Problem : *Given a railway network connecting various towns, determine a shortest route between two specified towns in the network.*

Here one must find, in a weighted graph, a path of minimum weight connecting two specified vertices u_0 and v_0; the weights represent distances by rail between directly-linked towns, and are therefore non-negative. The path indicated in the graphs of figure 1 is a $(u_0–v_0)$-path of minimum weight.

We now present an algorithm for solving the shortest path problem. For clarity of exposition, we shall refer to the weight of a path in a weighted graph as its length; similarly the minimum weight of a (u, v)-path will be called the distance between u and v and denoted by $d(u, v)$. These definitions coincide with the usual notions of length and distance, when all the weights are equal to one.

It clearly suffices to deal with the shortest path problem for simple graphs; so we shall assume here that G is simple. We shall also assume that all the weights are positive. This, again, is not a serious restriction because, if the weight of an edge is zero, then its ends can be identified. We about the convention that $w(uv) = \infty$ if $uv \notin E$.

The algorithm to be described was discovered by Dijkstra (1959) and, independently, by Whiting and Hiller (1960). It finds not only a shortest (u_0, v_0)-path, but shortest paths from u_0 to all other of G. The basic idea is as follows :

Suppose that S is a proper subset of V such that $u_0 \in S$, and let $\bar{S}$ denote V\S. If $P = u_0 \ldots \bar{u}\,\bar{v}$ is a shortest path from u_0 to $\bar{S}$ then clearly $\bar{u} \in S$ and the $(u_0, \bar{u})$-section of path of P must be a shortest $(u_0, \bar{u})$-path. Therefore

$$d(u_0, \bar{v}) = d(u_0, \bar{u}) + w(\bar{u}\,\bar{v})$$

and the distance from u_0 to $\bar{S}$ is given by the formula

$$d(u_0, \bar{S}) = \min_{\substack{u \in S \\ v \in \bar{S}}} \{d(u_0, u) + w(uv)\} \quad \ldots(i)$$

This formula is the basis of Dijkstra's algorithm. Starting with the set $S_0 = \{u_0\}$, an increasing sequence $S_0, S_1, \ldots, S_{v-1}$ of subsets of V is constructed, in such a way that, at the end of stage i, shortest paths from u_0 to all vertices in S_i are known.

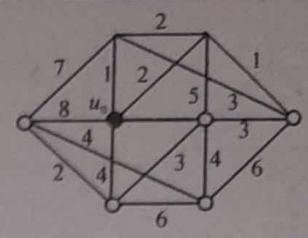
The first step is to determine a vertex nearest to u_0. This is achieved by computing $d(u_0, \bar{S}_0)$ and selecting a vertex $u_1 \in \bar{S}_0$ such that $d(u_0, u_1) = d(u_0, \bar{S}_0)$; by eq.(i)

$$d(u_0, \bar{S}_0) = \min_{\substack{u \in S_0 \\ v \in \bar{S}_0}} \{d(u_0, u) + w(uv)\} = \min_{v \in \bar{S}_0} \{w(u_0, v)\}$$

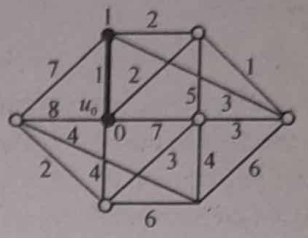
and so $d(u_0, \bar{S}_0)$ is easily computed. We now set $S_1 = \{u_0, u_1\}$ and let P_1 denotes the path u_0, u_1; this is clearly a shortest (u_0, u_1)-path. In general, if the set $S_k = (u_0, u_1, \ldots, u_k)$ and corresponding shortest paths $P_1, P_2, \ldots, P_k$ have already been determined, we compute $d(u_0, \bar{S}_k)$ using eq.(i) and select a vertex $u_{k+1} \in \bar{S}_k$ such that $d(u_0, u_{k+1}) = d(u_0, \bar{S}_k)$. By eq.(i), $d(u_0, u_{k+1}) = d(u_0, u_j) + w(u_j u_{k+1})$ for some $j \le k$; we get a shortest (u_0, u_{k+1})-path by adjoining the edge $u_j u_{k+1}$ to the path P_j.

We illustrate this procedure by considering the weighted graph depicted in figure 2. Shortest paths from u_0 to the remaining vertices are determined in seven stages. At each stage, the vertices to which shortest paths have been found are indicated by solid dots, and each is labeled by its distance from u_0; initially u_0 is labeled 0. The actual shortest paths are indicated by solid lines. Notice that, at each stage, these shortest paths together form a connected graph without cycles; such a graph is called a tree, and we can think of the algorithm as a 'tree-growing' procedure. The final tree, in figure 2, has the property that, for each vertex v, the path connecting u_0 and v is a shortest (u_0, v)-path.

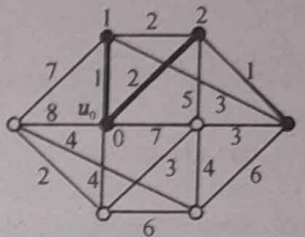
Dijkstra's algorithm is refinement of the procedure. This refinement is motivated by the consideration that, if the minimum in eq.(i) were to be computed from scratch at each stage, many comparisons would be repeated unnecessarily.



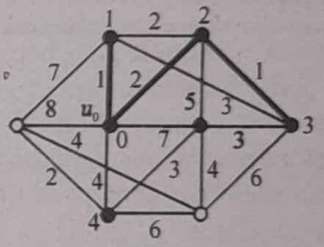
(a)



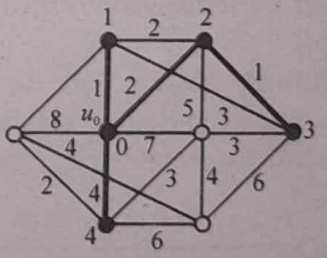
(b)



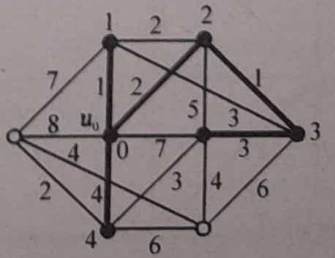
(c)



(d)



(e)



(f)

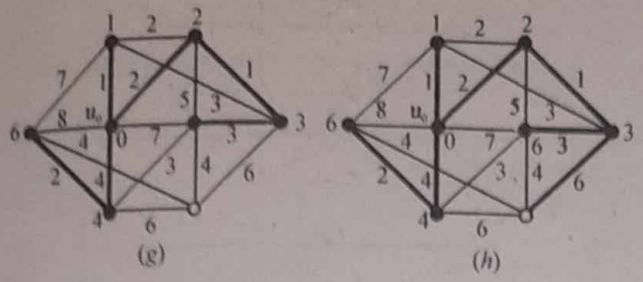


Fig. 2 : Shortest Path Algorithm

To avoid such repetitions, and to retain computational information from one stage to the next, we adopt the following labeling procedure. Throughout the algorithm, each vertex v carries a label $l(v)$ which is an upper bound; $d(u_0, v)$. Initially $l(u_0) = 0$ and $l(v) = \infty$ for $v \neq u_0$ (In actual computation ∞ is replaced by any sufficiently large number.) As the algorithm proceeds, these labels are modified so that, at the end of stage i,

$$l(u) = d(u_0, u) \text{ for } u \in S_i$$

and $$l(v) = \min_{u \in S_{i-1}} \{d(u_0, u) + w(uv)\} \text{ for } v \in \overline{S}_i$$

Dijkstra's Algorithm

1. Set $l(u_0) = 0, l(v) = \infty$ for $v \neq u_0, S_0 = \{u_0\}$ and $i = 0$

2. For each $v \in \overline{S}_i$, replace $l(v)$ by min $\{l(v), l(u_i) + w(u_i v)\}$. Compute $\min_{v \in S_i}\{l(v)\}$ and let u_{i+1} denotes a vertex this minimum is attained.

   Set $S_{i+1} = S_i \cup \{u_{i+1}\}$.

3. If $i = v - 1$, stop. If $i < v - 1$, replace i by $i + 1$ and go to step 2.

When the algorithm terminates, the distance from u_0 to v is given by the final value of the label $l(v)$. (If our interest is in determining the distance to one specific vertex v_0, we stop as soon as some u_j equals v_0.) A flow diagram summarizing this algorithm is shown in figure 3.

As described above, Dijkstra's algorithm determines only the distances from u_0 to all the other vertices, and not the actual shortest paths. These shortest paths can, however, be easily determined by keeping track of the predecessors of vertices in the tree.

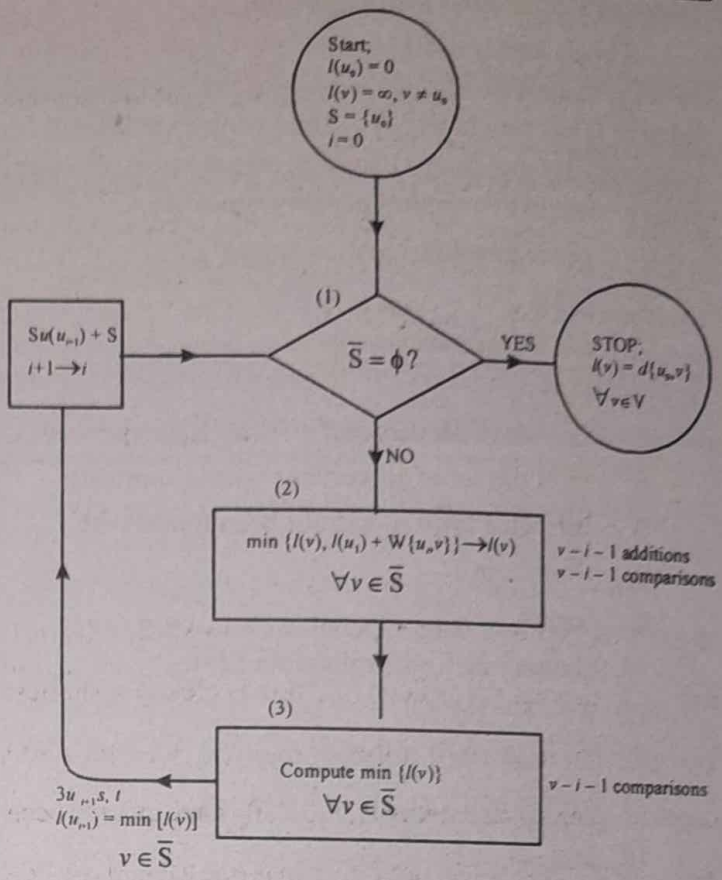


Fig. 3 : Dijkstra's Algorithm

Dijkstra's algorithm is an example of what Edmonds (1965) calls a good algorithm. A graph-theoretic algorithm is good. If the number of computational steps required for its implementation on any graph G is bounded above by a polynomial in v and $\in$ (such as $3v^2\in$). An algorithm whose implementation may require an exponential number of steps (such as 2) might be very inefficient for some large graphs.

To see that Dijkstra's algorithm is good, note that the computations involved in boxes 2 and 3 of the flow diagram, totalled over all iterations, require $v(v-1)/2$ additions and $v(v-1)$ comparisons. One of the questions that is not elaborated upon in the flow diagram is the matter of deciding whether a vertex belongs to 9 or not (box 1). Dreyfus (1969) reports a technique for doing this that requires a total of $(v-1)^2$ comparisons. Hence, if we regard either a comparison or an addition as a basic computational unit, the total number of computations required for this algorithm is approximately $5v^2/2$, and thus of order v^2. (A function $f(v, e)$ is of order $g(v, \in)$ if there exists positive constant c such that $f(v, \in)/g(v, \in) \leq c$ for all v and e,)

Although the shortest path problem can be solved by a good algorithm, there are many problems in graph theory for which no good algorithm is known.

Dijkstra's Shortest-Path Algorithm

This is an algorithm to find the shortest path from a vertex a to another vertex z in a connected weighted graph. Edge (i, j) has weight $w(i, j) > 0$ and vertex x is labeled $L(x)$ (minimum distance from a if known, otherwise ∞). The output is $L(z)$ = length of a minimum path from a to z.

```
1.  procedure dijkstra(w, u0, v0, L)
2.  L(u0) := 0
3.  for all vertices x ≠ u0
4.  L(x) := ∞
5.  T := set of all vertices
6.  {T is the set of all vertices whose shortest}
7.  {distance from u0 has not been found yet}
8.  while v0 in T
9.  begin
10. choose v in T with minimum L(v)
11. T := T – {v}
12. for each x in T adjacent to v
13. L(x) := min{L(x), L(v) + w(v,x)}
14. end
15. return L(v0)
16. end dijkstra
```

For instance consider the graph in Fig.4.

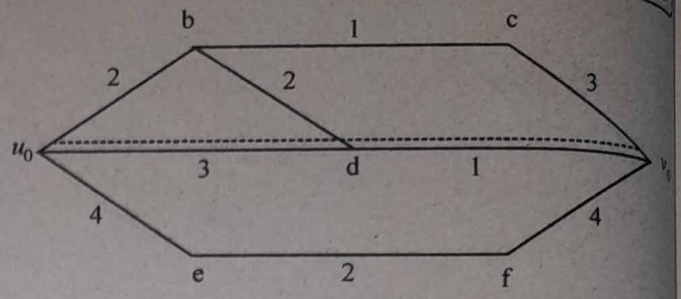


Fig. 4 : Shortest Path from u_0 to v_0

The algorithm would label the vertices in the following way in each iteration (the boxed vertices are the ones removed from T).

The algorithm would label the vertices in the following way in each iteration (the boxed vertices are the ones removed from T).

Table

Iteration	u_0	b	c	d	e	f	v_0
0	0	∞	∞	∞	∞	∞	∞
1	[0]	2	∞	3	4	∞	∞
2	[0]	[2]	3	3	4	∞	∞
3	[0]	[2]	[3]	3	4	∞	6
4	[0]	[2]	[3]	[3]	4	∞	4
5	[0]	[2]	[3]	[3]	[4]	6	4
6	[0]	[2]	[3]	[3]	[4]	6	[4]

At this point the algorithm returns the value 4.

❑❑❑

DATABASE MANAGEMENT SYSTEM

IV C.S.

1 INTRODUCTION TO DATABASE SYSTEMS

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 *Why E-R model used in DBMS?* *[R.T.U. 2019]*

Ans. The Entity-Relationship Model : The entity-relationship (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. The E-R data model is one of several semantic data models; the semantic aspect of the model lies in its representation of the meaning of the data.

Q.2 *What do you mean by Referential Integrity?* *[R.T.U. 2019]*

Ans. Referential Integrity : This constraints identifies any column in the same table or between different tables. For a column to be defined as a foreign key, it should be a defined as a primary key in table.

Which it is referring. One or more columns can be defined as foreign key.

Syntax to define a foreign key at column level.

[CONSTRAINT Constraint_name] REFERENCES

Referenced. Table_name (column_name)

Syntax to define a foreign key at table laud.

[CONSTRAINT constraint_name] FOREIGN KEY (Column_name) references referenced_table name (column_name)

Q.3 *Explain the concepts of Primary key.* *[R.T.U. 2019]*

Ans. Every record must have one unique identifier called the primary key that has a unique value within the table or collection. Primary keys can be concatenated which means that the uniqueness can be made up of one or more fields.

Syntax to define a primary key at column level :

Column name data type [CONSTRAINT constraint_name] PRIMARY KEY

Syntax to define a primary key at table level :

[CONSTRAINT Constraint_name] PRIMARY KEY

[Column_name1, column_name2]

Q.4 *What is Entity? Explain with a suitable example.* *[R.T.U. 2019]*

Ans. Entity : In database systems, objects are referred as entity. *"An entity is a thing or object in the real world that is distinguishable from other things or objects."*

For example, each person is an entity and banks accounts can be considered as entity.

Note : Object and class of OOPs are referred as an entity and an entity set in database system respectively.

Q.5 *What is DBMS?* *[R.T.U. 2019]*

Ans. DBMS : A Database Management system is a collection of interrelated data and collection of programs to access that data. The data describes one particular enterprise. Database systems are ubiquitous today and most people interact either directly or indirectly, with

databases many times everyday. A major purpose of the database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

Underlying the structure of a database is the data model- a collection of conceptual tools for describing data, data relationships, data semantics and data constraints. The entity relationship (E-R) data model is a widely used data model which provides a convenient graphical representation to view data, relationship and constraints.

Q.6 ***What is the difference between logical data independence and physical data independence?*** *[R.T.U. 2016, 2015]*

Ans.

S. No.	Logical Data Independence	Physical Data Independence
1.	Indicates that conceptual/logical schema can be changed without affecting the existing external (view) schemas.	Indicates that the physical storage structures or devices used for storing the data could be changed without any change in the conceptual view or external view.
2.	The change would be absorbed by the mapping between the external and conceptual (logical) levels.	The change would be absorbed by the mapping between the conceptual and internal levels.
3.	Modifications such as the deletion of a conceptual view field or record may require changes in the external view and application program.	Modifications of physical level improve the performance.

Q.7 ***What do you mean by constraints? Explain different types of constraints with examples.*** *[R.T.U. 2016]*

Ans. Constraints : Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table. The whole purpose of constraints is to maintain the data integrity during an update/delete/insert into a table.

Different Types of constraints that can be created in RDBMS

1. Mapping Constraints :

Mapping Cardinalities : Express the number of entities to which another entity can be associated via a relationship. For binary relationship sets between entity sets A and B, the mapping cardinality must be one of:

(i) **One-to-one :** 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.

(ii) **One-to-many :** An entity in A is associated with any number in B. An entity in B is associated with at most one entity in A.

(iii) **Many-to-one :** An entity in A is associated with at most one entity in B. An entity in B is associated with any number in A.

(iv) **Many-to-many :** Entities in A and B are associated with any number from each other.

Q.8 ***Explain physical and external view of data.***

Ans. External View : In the relational model, the external view also presents data as a set of relations. It is tailored to the needs of a particular category of users. Portions of stored data should not be seen by some users and begins to implement a level of security and simplifies the view for these users. For example, students should not see faculty salaries, faculty should not see billing or payment data, etc.

Physical View : The physical view describes the details of how data is stored: files, indices, etc., on the random access disk system. It also typically describes the record layout of files and type of files (hash, b-tree, flat

Q.9 ***Differentiate between candidate keys and super keys.***

Ans. Difference between Candidate Keys and Super Keys

S.No.	Candidate Keys	Super Keys
1.	A candidate key is any set of one or more columns whose combined value is unique through out the table.	An attribute, or group of attributes, that is sufficient to distinguish every tuple in the relation from every other one.
2.	Each candidate key is not called super key and no component of a candidate key is allowed to be null.	Each super key is called a candidate key.

Q.10 ***Explain the difference between partial & total participation.***

Ans. Difference between Partial and Total Participation

S.No.	Partial Participation	Total Participation
1.	All instances need not participate.	Every entity instance must be connected through the relationship to another instance of the other participating entity types.
2.	Represented by single line from entity rectangle to relationship diamond.	Represented by double line from entity rectangle to relationship diamond.

Q.11 *Differentiate between entity and entity sets.*

Ans. Difference between Entity and Entity Sets

S. No.	Entity	Entity Set
1.	Entities are the people, places, things, events and concepts of interest to an organization.	Entity set represents collection of things. Example : An employee entity set might represent a collection of all the employees that work for an organization.
2.	Entity can be concrete like employee, book, project etc. and can be abstract or a concept. Example : title, job company etc.	Entity set need not be disjoint. For example : the entity set employee (all employee of bank) and the entity set customer (all customer of the bank) may have members in common.

PART-B

Q.12 *What is E-R model? What are the features of E-R model? Draw and explain E-R model for Library Management System.* *[R.T.U. 2019]*

Ans. E-R Model : The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called *entities* and of *relationships* among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. For example, each person is an entity and bank accounts can be considered as entities.

Entities are described in a database by a set of **attributes**. For example, the attributes *account-number* and *balance* may describe one particular account in a bank and they form attributes of the *account* entity set. Similarly, attributes *customer-name*, *customer-street* address and *customer-city* may describe a *customer* entity.

An extra attribute *customer-id* is used to uniquely identify customers (since it may be possible to have two customers with the same name, street address and city).

A unique customer identifier must be assigned to each customer. In the United States, many enterprises use the social-security number of a person (a unique number the U.S. government assigns to every person in the United States) as a customer identifier.

A **relationship** is an association among several entities. For example, a *depositor* relationship associates a customer with each account that she has. The set of all entities of the same type and the set of all relationships of the same type are termed an **entity set** and **relationship set**, respectively.

The overall logical structure (schema) of a database can be expressed graphically by an *E-R diagram*, which is built up from the following components -

(i) **Rectangles :** Which represent entity sets.
(ii) **Ellipses :** Which represent attributes.
(iii) **Diamonds :** Which represent relationships among entity sets.
(iv) **Lines :** Which link attributes to entity sets and entity sets to relationships.

Each component is labeled with the entity or relationship that it represents.

As an illustration, consider part of a database banking system consisting of customers and of the accounts that these customers have. Figure 1 shows the corresponding E-R diagram.

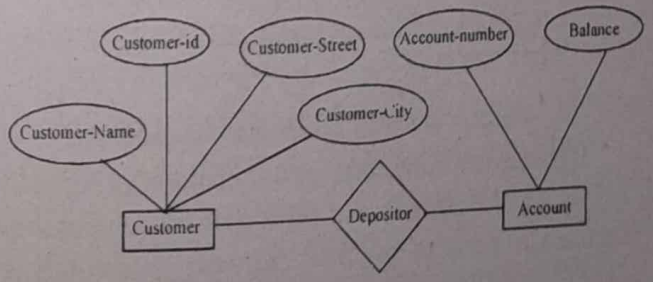


Fig. 1 : A simple E-R Diagram

The E-R diagram indicates that there are two entity sets, *customer* and *account*, with attributes. The diagram also shows a relationship *depositor* between customer and account.

In addition to entities and relationships, the E-R model represents certain constraints to which the contents of a database must conform. One important constraint is **mapping cardinalities**, which express the number of entities to which another entity can be associated via a relationship set. For example - if each account must belong to only one customer, the E-R model can express that constraint.

E-R diagrams also provide a way to indicate more complex constraints on the number of times each entity participates in relationships in a relationship set. An edge between an entity set and a binary relationship set can have an associated minimum and maximum cardinality, shown in the form *l..h*, where *l* is the minimum and *h* is the maximum cardinality. A minimum value of 1 indicates total participation of the entity set in the relationship set. A maximum value of 1 indicates that the entity participates in at most one relationship, while a maximum value * indicates no limit.

Note that a label 1..* on an edge is equivalent to a double line.

For example, consider Figure 2. The edge between *loan* and *borrower* has a cardinality constraint of 1..1, meaning the minimum and the maximum cardinality both are 1. That is, each loan must have exactly one associated customer. The limit 0..* on the edge from *customer* to *borrower* indicates that a customer can have zero or more loans. Thus, the relationship *borrower* is one to many from *customer* to *loan* and further the participation of *loan* in *borrower* is total.

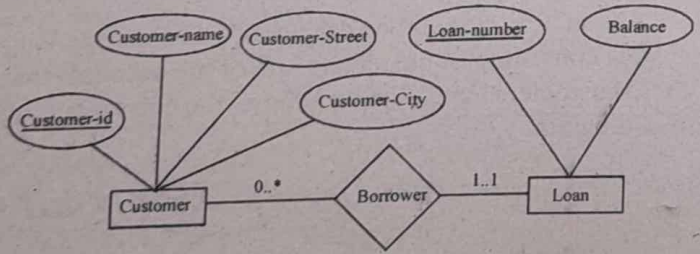


Fig. 2 : Cardinality Limits on Relationship Sets

It is easy to misinterpret the 0..* on the edge between *customer* and *borrower* and think that the relationship *borrower* is many to one from *customer* to *loan*-this is exactly the reverse of the correct interpretation.

If both edges from a binary relationship have maximum value of 1, the relationship is one to one. If had specified a cardinality limit of 1..* on the edge betw *customer* and *borrower*, we would be saying that e customer must have at least one loan.

E-R Notations :

1. E entity set =

customer , employee , loan , branch , accoun

These are entity sets.

2. weak entity → Loan payment are weak entity.

3. Relationship set → Shows the relationship

4. A ← Show primary set

Customer_id, branch name, account-no. These primary keys.

5. ( ) – All the attributes are shown through ell

6. (A) – Derived attribute

Employment-length is derived attribute.

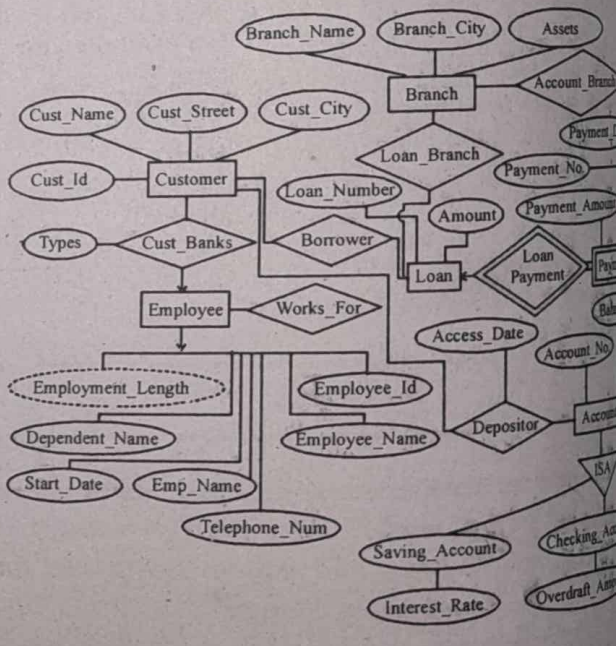
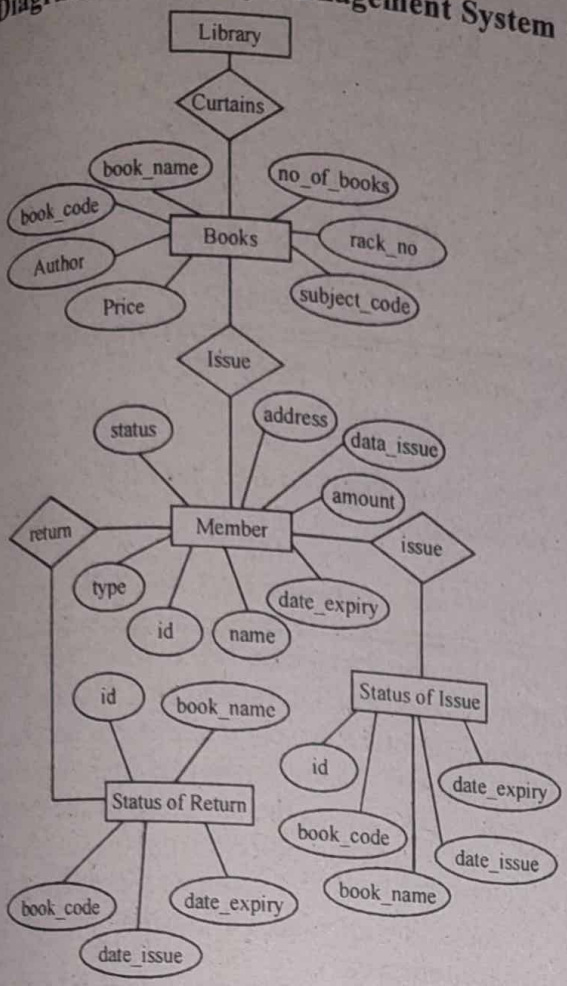


Fig. 3 : E-R Diagram

The E-R model defines the conceptual view database. E-R model is a high-level conceptual mode database design.

ER Diagram for Library Management System :



Useful in designing Real World Database

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

Then we have the notion of entity sets. An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

It is this important aspect of the entities and the design of E-R Model, that make the Entity-Relationship Model useful in designing Real-World Databases, because real-world objects can directly be taken as entities. There real-world properties such as age or name for a student can directly be taken as attributes for entities. Thus, E-R Model is useful for designing real world databases.

Convert ER-Diagram into Tables :

1. Members :

Column Name	Id_no (PK)	Name	Address	Date of Issue	Date of Expiry	Status
Data Type	Text	Text	Text	Date/time	Date/time	Text

2. Add_Books :

Column Name	Book _name	Book _code (PK)	Author	Date _of _arrival	Price	Rack _no	No _of _books	Subject _code
Data Type	Text	Text	Text	Date/ time	Date/ time	Text	Text	Text

3. Issue :

Column Name	Id_no (FK)	Book_Name	Issue_date	Due_date
Data Type	Text	Text	Date/time	Date/time

Q.13 ***Differentiate between file system and DBMS. Explain the ternary relationship with a suitable example.*** ***[R.T.U. 2019]***

Ans. File System vs DBMS - Difference between File System and DBMS

File Management System	Database Management System
File System is a general, easy-to-use system to store general files which require less security and constraints.	Database management system is used when security constraints are high.
Data Redundancy is more in file management system.	Data Redundancy is less in database management system.
Data Inconsistency is more in file system.	Data Inconsistency is less in database management system.
Centralisation is hard to get when it comes to File Management System.	Centralisation is achieved in Database Management System.
User locates the physical address of the files to access data in File Management System.	In Database Management System, user is unaware of physical address where data is stored.
Security is low in File Management System.	Security is high in Database Management System.
File Management System stores unstructured data as isolated data files/entities.	Database Management System stores structured data which have well defined constraints and interrelation.

Ternary Relationship : Suppose we want to keep track of which person got which degree from which University. We could represent this with a ternary relationship as follows :

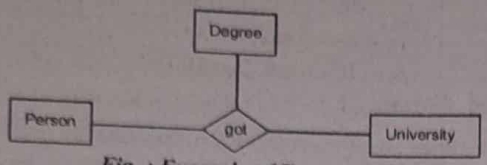


Fig. : Example of Ternary Relationship

Q.14 ***Draw and explain architecture of RDBMS.***

[R.T.U. 2017]

Ans. Architecture of RDBMS : There are five major components that are exercised in a typical interaction with an RDBMS:

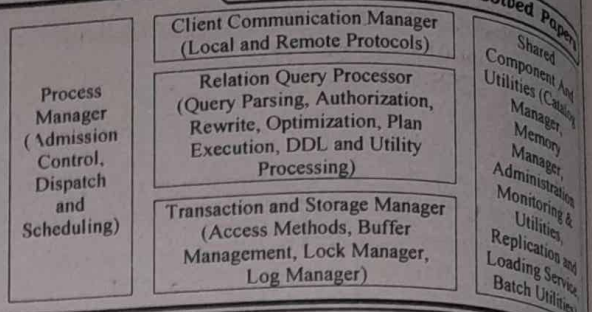
(1) Client Communication Manager : In order to communicate with a database an application needs to make a connection with a database over a network. An application establishes a connection with the Client Communication Manager. This component enables communication between various database clients through both local and remote protocols. Its main responsibility is to remember communication state, return data and control messages (result codes, errors) as well as forward the client's request to other parts of the DBMS.

(2) Process Manager : The process manager is responsible for providing a "thread of computation" for each database request from a database client. It links the threads data and control output to the appropriate communication manager client. The first decision to be made by the process manager is to determine if enough system resources are available to execute the query or defer the same until a later time.

(3) Relational Query Processor : On receiving a request to process a query the relation query processor first checks if the user is authorised to run the query. It then compiles the query into an interim query plan which is further optimised. The resulting plan is executed by the "plan executor" which eventually makes use of the transaction and storage monitor.

(4) Transaction and Storage Manager : Once a query is parsed it retrieves the requested data from the Transaction and Storage Manager. It the gatekeeper to all data access and manipulation calls. The transactional and storage manager also make sure that ACID properties of a transaction are adhered too and thus the need for a lock and log manager.

(5) Shared Components and Utilities : There are a number of shared components and utilities that are essential for a database to run.



Q.15 ***Contrast between DDL and DML.***

OR ***[R.T.U. 2017]***

Differential between DDL and DML using syntax for them. ***[R.T.U. 2014]***

OR

Differentiate between DDL and DML. ***[R.T.U. 2014]***

Ans. Difference between DDL and DML : Data Definition Language (DDL) statements are used to define the database structure or schema. Some examples:

- CREATE: to create objects in the database
- ALTER : alters the structure of the database
- DROP : delete objects from the database
- TRUNCATE : remove all records from a table, including all spaces allocated for the records are removed
- COMMENT : add comments to the data dictionary
- RENAME : rename an object

Data Manipulation Language (DML) statements are used for managing data within schema objects. Some examples:

- SELECT : retrieve data from the database
- INSERT : insert data into a table
- UPDATE : updates existing data within a table
- DELETE : deletes all records from a table, the space for the records remain
- MERGE : UPSERT operation (insert or update)

Transaction control statements manage changes made by DML statements. The transaction control statements are:

COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

All transaction control statements, except certain forms of the COMMIT and ROLLBACK commands, are supported in PL/SQL.

Session control statements dynamically manage the properties of a user session. These statements do not implicitly commit the current transaction.

PL/SQL does not support session control statements. The session control statements are:

ALTER SESSION

SET ROLE

- CALL : call a PL/SQL or Java subprogram
- EXPLAIN PLAN : explain access path to data
- LOCK TABLE : control concurrency

DDL Commands Syntax

The Create Table command

```
CREATE TABLE [schema.]table
   ( { column datatype [DEFAULT expr]
[column_constraint] ... | table_constraint}
   [, { column datatype [DEFAULT expr]
[column_constraint] ... | table_constraint} ]...)
   [AS subquery]
```

The DROP Command

Syntax:

```
DROP TABLE <table_name>
```

Example:

```
DROP TABLE Student;
```

It will destroy the table and all data which will be recorded in it

DML Command Syntax

Viewing data in the table (Select Command) :Once data has been inserted into a table, the next most logical operation would be to view what has been inserted. The SELECT SQL verb is used to achieve this.

All rows and all columns

Syntax: SELECT * FROM Table_name;

eg: Select * from Student; It will show all the table records.

SELECT First_name, DOB FROM STUDENT WHERE Reg_no = 'S101'; Cover it by single inverted comma if its datatype is varchar or char.

This Command will show one row. because you have given condition for only one row and particular records. If condition which has given in WHERE Clause is true then records will be fetched otherwise it will show no records selected.

Q.16 *Discuss types of DBMS.* *[R.T.U. 2017]*

Ans.Types and Classification of Database Management System : On the basis of data model, we have five types of DBMS:

(1) **Relational Database :** This is the most popular data model used in industries. It is based on the SQL. They are table oriented which means data is stored in different access control tables, each has the key field whose task is to identify each row. The tables or the files with the data are called as relations that help in designating the row or record, and columns are referred to attributes or fields. Few examples are MYSQL(Oracle, open source), Oracle database (Oracle), Microsoft SQL server(Microsoft) and DB2(IBM).

(2) **Object Oriented Database :** The information here is in the form of the object as used in object oriented programming. It adds the database functionality to object programming languages. It requires less code, use more natural data and also code bases are easy to maintain. Examples are ObjectDB (ObjectDB software).

(3) **Object Relational Database :** Relational DBMS are evolving continuously and they have been incorporating many concepts developed in object database leading to a new class called extended relational database or object relational database.

(4) **Hierarchical Database :** In this, the information about the groups of parent or child relationships is present in the records which is similar to the structure of a tree. Here the data follows a series of records, set of values attached to it. They are used in industry on mainframe platforms. Examples are IMS(IBM), Windows registry(Microsoft).

(5) **Network Database :** Mainly used on a large digital computers. If there are more connections, then this database is efficient. They are similar to hierarchical database, they look like a cobweb or interconnected network of records. Examples are CA-IDMS (COMPUTER associates), IMAGE(HP).

While on the basis of number of users, we have two types of DBMS:

(i) **Single User :** As the name itself indicates it can support only one user at a time. It is mostly used with the personal computer on which the data resides accessible to a single person. The user may design, maintain and write the database programs.

(ii) **Multiple Users :** It supports multiple users concurrently. Data can be both integrated and shared,a database should be integrated when the same information is not need be recorded in two places. For example a student in the college should have the database containing his information. It must be accessible to all the departments related to him. For example the library department and the fee section department should have information about student's database. So in such case, we can integrate and even though database resides in only one place both the departments will have the access to it.

Q.17 ***Explain advantage of DBMS over file system.*** *[R.T.U. 2017]*

OR

What do you mean by DBMS? Explain the advantages of Database management system over file management system. *[R.T.U. 2016]*

OR

Explain the advantages of Database Management System over File Management System. *[R.T.U. 2015]*

OR

List out four significant differences between a file processing system and a DBMS. *[Raj. Univ. 2007]*

OR

Advantage of DBMS over File Processing System. *[R.T.U. 2010]*

Ans. DBMS : *Refer to Q.5.*

Advantages of Database Management System Over File Management System :

1. As we do not have 1000 GB main memory (primary memory) to store the data, so we store the data in some permanent storage device (secondary memory) like magnetic disk or magnetic tape etc. Hence, file oriented system fails in primary memory cases and we apply database management system to store the data files permanently.
2. Suppose if we have a large amount of primary memory on a 16 bit or 32 bit computer system, then a problem can occur in file based system to use the data by direct or random addressing. Also we cannot call more than 2GB or 4GB of data direct to the primary memory at a time. So, we need a database program to identify the data.
3. Some programs are too lengthy and complex due to which they cannot store large amount of data in the files related to the operating systems. But a database system makes it simple and fast.
4. We cannot change and access file-oriented data simultaneously, so we have requirement of such type of system (DBMS) which can be used to access the large amount of data concurrently.
5. Also we cannot recall or recover the file oriented data, but centralized database management can solve such type of problem in DBMS.
6. File oriented operating system provide only a password mechanism for security, but this is not successful in case of number of users are accessing the same data by using the same login.

At the end, we can say that DBMS is a piece of software that is designed to make the processing fast and easier.

Q.18 ***Why query processor component of database system is important? Briefly discuss about components of query processor.*** *[R.T.U. 2016, 20...]*

Ans. Query Processor Component : The query processor components are important because it helps a database system simplify and facilitate access to data. High level views help to achieve this goal; with them, users of the system are not burdened unnecessarily with the physical details of the implementation of the system. However, quick processing of updates and queries is important. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

These components are used in evaluating DDL and DML queries and includes following components.

(i) DML Compiler : It first attemps to transform user's request into an equivalent but more efficient form and then translates that into a set of low level instructions that can be used by query evaluation engine.

(ii) Embedded DML Pre-Compiler : It Converts DML statements embedded in an application program to normal procedure calls in host language. This pre compiler consult DML compiler to generate the appropriate code.

(iii) DDL Interpreter : It makes data dictionary which contains metadata.

(iv) Query Evaluation Engine : It executes low instructions generated by the DML compiler.

Q.19 ***Briefly discuss the history of database systems.*** *[R.T.U. 201...]*

Ans. Database Systems : The history of database management systems begins around the time when computers began taking off. In 1960's, the concept of the database was put in use and also began to grow in commercial use and it was this rise that was an interest to a man named Charles W. Bachman. He invented the database management system. Charles Bachman was an industrial researcher, beginning a career at Dow Chemical where he became the data processing manager in 1950's before leaving to work at General Electric in 1960.

It was in 1960 that Bachman came up with the Integrated Database System, the very first DBMS. This got the proverbial ball rolling, as Bachman found the Database Talk Group along with the group that gave the

standardization to the programming language of COBOL. Not to be left out, IBM created their own database system, known IMS, for that of NASA's Apollo space program. Both of these are now known as the precursors of navigational databases.

The earlier invented DBMS was not easy to use. Thus, in 1970's a man named Edgar Codd thought of a way to make things a bit easier. Codd worked for IBM and felt that there had to be a way to make things easier when using these DBMS. He wrote a paper entitled, A Relational Model of Data for Large Shared Data Banks, in which he proposed replacing these current systems with that of tables and rows. This concept became relational DBMS.

IBM started working on a prototype system loosely based on Codd's concepts in the early 1970's. The first version was ready in 1974/5, and then started work on multi-table systems in which the data could be split so that all of the data for a record (some of which is optional) did not have to be stored in a single large "chunk". Subsequent multi-user versions were tested by customers in 1978 and 1979, and by the time a standardized query language (SQL) had been added.

Then in 1990's, object-oriented programming was on its peak, which gets reflected in databases also bringing in picture Object Databases and Object-Relational Databases.

Q.20 ***Explain the notational conventions used in the ER model.*** ***[R.T.U. 2015]***

Ans. All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. The symbols used for the basic ER constructs are:

- Entities are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.
- Attributes are represented by ellipses.
- A solid line connecting two entities represents relationships. The name of the relationship is written above the line. Relationship names should be verbs and diamond sign is used to represent relationship sets.
- Attributes, when included, are listed inside the entity rectangle. Attributes, which are identifiers, are underlined. Attribute names should be singular nouns.
- Multi-valued attributes are represented by double ellipses.
- Directed line is used to indicate one occurrence and undirected line is used to indicate many occurrences in a relation.

The symbols used to design an E-R diagram is shown below.

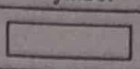
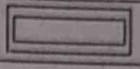
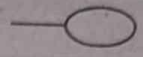
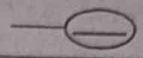
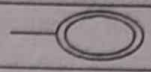
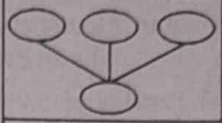
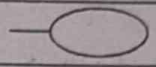
Symbol	Meaning
	Entity Type
	Weak Entity Type
	Relationship Type
	Identifying Relationship Type
	Attribute
	Key Attribute
	Multivaluated Attribute
	Composite Attribute
	Derived Attribute

Fig.

Q.21 ***Describe the following terms with examples:***
(i) Owner entity sets
(ii) Identifying relationship
(iii) Discriminator
and design E–R diagram to model them. ***[R.T.U. 2014]***

Ans. (i) Owner Entity Sets

(i) The identifying relationship will be one-to-many from owner entity set to weak entity set.

(ii) The participation of owner entity set in the identifying relationship will be partial and the participation of the weak entity set in the identifying relationship will be total.

As shown in fig.1 set E_2 is said to be existence-dependent on E_1 and E_1 is said to be the owner entity set of E_2. The relationship R between E_2 and E_1 is called identifying relationship. The weak entity set E_2 will have a set of attributes called its discriminator, which together with the primary key of E_1 will form the primary key of E_2.

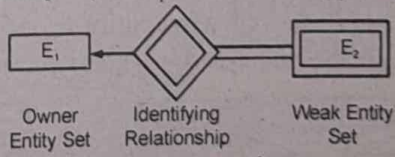


Fig.1 : Owner entity sets

(ii) Identifying Relationships

For a weak entity set to be meaningful, it must be associated with another entity set, called the identifying or owner entity set. Every weak entity must be associated

with an identifying entity; that is, the weak entity set is said to be existence dependent on the identifying entity set. The identifying entity set is said to own the weak entity set that is identifies. The relationship associating the weak entity set with the identifying entity set is called the identifying relationship.

The identifying relationship is many-to-one from the weak entity set to the identifying entity set, and the participation of the weak entity set in the relationship is total . The identifying relationship set should not have any descriptive attributes, since any such attributes can instead be associated with the weak entity set.

Consider the relationship between APARTMENT and BUILDING as shown in fig.2. In reality, two apartments built in two buildings may have the same apartment number. To distinguish between these two apartments, you may need to attach the building number to the apartment number. In such a case, we can say that the apartment id depends on the existence of the building id. That is, the identifier for the APARTMENT entity should include the identifier for the BUILDING entity. Such a relationship is called the identifying relationships.

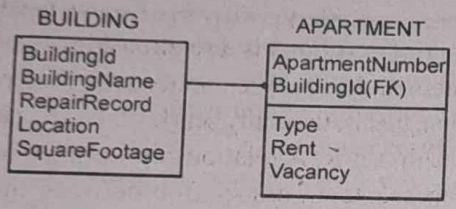


Fig. 2 : Identifying relationship

The identifying relationship is represented by a solid line. The parent entity is represented by a square and the id dependent entity is represented by a round-cornered square.

(iii) Discriminator : The weak entity set is dependent on a strong entity set and has a discriminator.

The discriminator of a weak entity set is an attribute or a set of attributes whose value uniquely identifies each weak entity from a set of weak entities related to one strong entity whose primary key value is given. For example, the discriminator of the weak entity set departures (fig.3) is the attribute date, since for each flight the date values are different.

The primary key of the weak entity set is formed by the primary key of the strong entity set plus its discriminator.

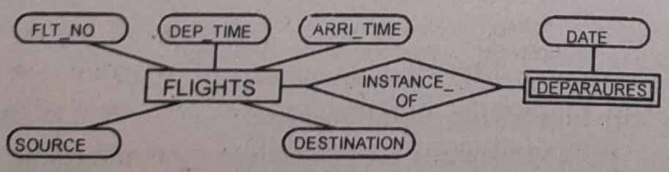


Fig.3 : E-R diagram for a weak entity set which is dependent on a strong entity set

Example : Suppose an entity set EMPLOYEE (EMP_ID, EMP_NAME, SALARY, DEPENDENTS) has an attribute DEPENDENT which is multi-valued i.e. an employee may have none or one or more than dependents. This situation can be best modeled as follows

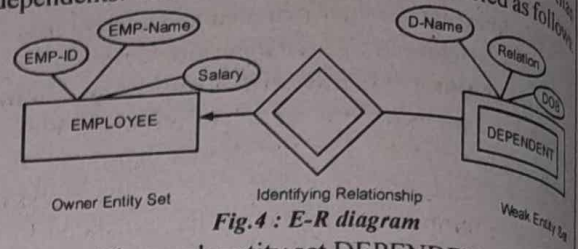


Fig.4 : E-R diagram

- The weak entity set DEPENDENT is existence dependent on the strong entity set EMPLOYEE.
- The weak entity set "DEPENDENT" has a discriminator attribute D_NAME, which along with primary key EMP_ID of EMPLOYEE forms primary key of the weak entity set DEPENDENT. In E-R diagram, the Discriminator (also called partial key) of a weak entity set is marked by underlining with a broken line.

Q.22 ***What is view in DBMS? List two reasons why we may choose to define a view and list two major problems with processing update operations expressed in terms of views.*** *[R.T.U. 2013]*

Ans. View Level : It is the highest level of abstraction which describes different views of the entire database. These views are designed according to the requirements of user who wants to access only a part of the database. A database may have several views, according to the demand of individual user or the group of users. The data in these views are not exactly stored in DBMS but they are computed using specification of view described by user. An analogy to the concept of data types in programming language may clarify the distinction among levels of abstraction. Most high-level programming languages support the notion of a record type.

At the view level, computer users see a set of application programs that hide details of the data type. Similarly, at the view level, several views of the database are defined, and database users see these views. In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing parts of the database.

Customer_Loan		
Cust_ID	:	101
Loan_No	:	1011
Amount_in_Dollars	:	8755.00

View

Two reasons why we may choose to define a view :

(a) Security conditions may require that the entire logical database should not visible to all users.

(b) We may wish to create a personalized collection of relations that is better matched to a certain user's intuition than is the actual logical model.

Two major problems with processing update operations expressed in terms of views : Views present significant problems if updates are expressed with them. The difficulty is that a modification to the database expressed in terms of a view must be translated to a modification to the actual relations in the logical model of the database :

(a) Since the view may not have all the attributes of the underlying tables, insertion of a tuple into the view will insert tuples into the underlying tables, with those attributes not participating in the view getting null values. This may not be desirable, especially if the attribute in question is part of the primary key of the table.

(b) If a view is a join of several underlying tables and an insertion results tuples with nulls in the join columns, the desired effect of the insertion will not be achieved. In other words, an update to a view may not be expressible at all as updates to base relations.

Q.23 *Write short notes on :*

(i) Participation constraints.

(ii) Role in ER-data model

(iii) Identifying Relationship. *[R.T.U. 2013]*

Ans.(i) Participation Constraints : The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R. If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be partial. For example, we expect every loan entity to be related to at least one customer through the borrower relationship. Therefore the participation of loan in the relationship set borrower is total. In contrast, an individual can be a bank customer whether or not she has a loan with the bank. Hence, it is possible that only some of the customer entities are related to the loan entity set through the borrower relationship, and the participation of customer in the borrower relationship set is therefore partial.

Ans.(ii) The Entity-Relationship Model : *Refer to Q.1.* The E-R data model employs three basic notations entity sets, relationship sets and attributes.

Entity Sets : An entity is a " thing" or "object" in the real world that is distinguishable from all other objects.

For example each person in an enterprise is an entity. An entity has a set of properties and the values for some set of properties may uniquely identify an entity.

An entity set is a set of entities of the same type that share the same properties or attributes. Entity sets do not need to be digitised. An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set.

Each entity has a value for each of its attributes.

A database thus includes a collection of entity sets, each of which contains any number of entities of the same type.

Relationship Sets : A relationship is an association among several entities.

A relationship set is a set of relationships of the same type. Finally, it is a mathematical relation on $n \geq 2$ entity sets. If $E_1, E_2, \ldots E_n$ are entity sets, then relationship set R is a subset of

$$\{(e_1, e_2, \ldots e_n)/e, e_1 \in E_1, e_2 \in E_2 \ldots, e_n \in E_n\}$$

where $(e_1, e_2 \ldots e_n)$ is a relationship.

The association between entity sets is referred to as participation, that is, the entity sets $E_1, E_2 \ldots E_n$ participate in relationship set R. A relationship instance in an E-R schema represents an association between the named entities in the real world enterprise.

A relationship may also have attributes called **descriptive** attributes.

Attributes : For each attribute, there is a set of permitted values, called the domain or value set of that attribute.

Formally, an attribute of an entity set is a function that maps from the entity set into domain. Since an entity set may have several attributes, each entity can be described by a set of (attribute, data value) pairs, one pair for each attribute of the entity set.

An attribute, as used in the E-R model, can be characterized by the following attribute types :

(a) Simple and composite attributes

(b) Single-valued and multi-valued attributes

(c) Derived attribute

An E-R diagram can express the overall logical structure of a database graphically. E-R diagrams are simple and clear qualities that may well account in large part for the widespread use of the E-R model.

Ans.(iii) Identifying Relationship : *Refer to Q.21(ii).*

PART-C

Q.24 (a) ***Explain the aggregation v/s ternary relationship in detail.*** *[R.T.U. 2019]*

OR

Differentiate between Ternary relationship and aggregation. *[R.T.U. 2015, Dec.2013]*

(b) ***Explain the entity v/s attribute in detail.*** *[R.T.U. 2019]*

OR

Differentiate between entity and attribute. *[R.T.U. 2015, Dec.2013]*

Ans.(a) Difference between Ternary Relationship and Aggregation :

Ternary Relationship : *Refer to Q.13.*

Aggregation : Aggregation used when we have to model a relationship involving another relationship set.

Aggregation allows us to treat a relationship set as an entity set for purposes of participating in (other) relationships.

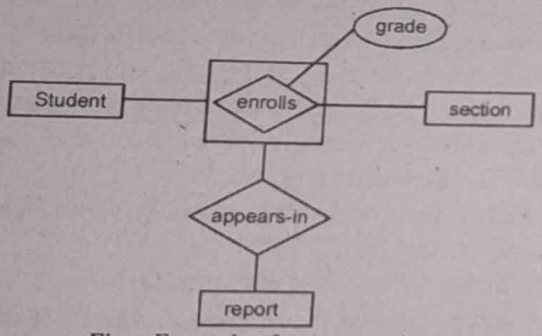


Fig. : Example of a Aggregation

Ans.(b) Difference between Entity and Attributes :

Entity : *Refer to Q.4.*

Attributes : Entities are described in a database by a set of properties. These properties are referred as attributes.

For example, the attributes account-number and balance may describe one particular account in a bank and they form attributes of the account entity set. Similarly, attributes customer-name, customer-street address and customer-city may describe a customer entity.

Q.25 ***Explain inventory management system.*** *[R.T.U. 2017]*

Ans. Inventory Management System : The Inventory Management System is a real-time inventory database capable of connecting multiple stores. This can be used to track the inventory of a single store, or to manage the distribution of stock between several branches of a large franchise. However, the system merely records sales and restocking data and provides notification of low stock at any location through email at a specified interval. The goal is to reduce the strain of tracking rather than to handle all store maintenance. Further features may include the ability to generate reports of sales, but again the interpretation is left to the management. In addition, since theft does occasionally occur, the system provides solutions for confirming the store inventory and for correcting stock quantities.

Typically, a company maintains one or both types of inventory; stock items and non-stock items. Stock items are stored products or parts that are ready for sale. Non-stock items are items that are used by the company, such as office supplies.

You can use an Inventory Management System to:

- Identify, store, and track stock and non-stock items.
- Identify and track prices in multiple currencies.
- Identify and store items that require special handling such as refrigeration.
- Identify items that require quality analysis or testing.
- Determine obsolete items.
- Identify and account for broken or defective parts.

A typical Inventory Management System may provide following business processes:

(1) Item classification : Provides for numerous purchasing, sales, and distribution classifications to report on purchasing or sales activity using many different facets of item characteristics and to determine how products move through or reside within the warehouse.

(2) Unit of measure conversions : Enables to define package size and the relationships among packages and performs standard conversions, such as pounds to ounces or count to dozens.

(3) Dual units of measure : Enables to maintain inventory and perform transactions for items in two units of measure.

(4) Manufacturing information : Enables to define the elements of items to enhance inventory planning and lead time forecasting. As companies move toward leaner inventories, such accurately forecasting is critical to successful operations.

(5) Item grade and potency information : Enables to monitor grade and potency, which is crucial in

industries such as food and drug manufacturing. In many cases, recording and tracking processes are strictly regulated, and noncompliance can result in stiff penalties. Furthermore, regulatory agencies require extensive documentation.

(6) **Issues :** Issues are typically used to remove inventory from a location. You can enter an issue for damaged goods, marketing demonstration, or internal use.

(7) **Adjustments :** Adjustments are used to reconcile discrepancies between physical inventory counts and on-hand system quantities. You can enter an adjustment for shrinkage, unrecorded gain, and initial balances.

(8) **Transfers :** Transfers are used to document the movement of an item. You can enter a transfer to record movement from location to location, from vehicle to location, or from plant to plant.

(9) **Physical inventories :** Physical inventories provide cycle count and tag count to conduct periodic physical inventory reconciliations.

The Inventory Management System's Database may contain the following tables:

(i) Basic information about each warehouse location, such as zones and level of detail.

(ii) Basic information about each item. For example, item number, description, search keys, category codes, default units of measure, etc.

(iii) Default item information. For example, each item's process and dimension groups, other parameters common to every unit of that item in the warehouse.

(iv) Inventory cost records.

(v) History of all inventory movements.

(vi) Predefined messages that print on documents such as sales orders and purchase orders.

(vii) Order types (sales, procurement, and so on) and the order statuses at which the system creates a request.

Q.26 ***Draw ER diagram of any one of the following and explain each component of this ER diagram. Library management system.*** ***[R.T.U. 2017]***

OR

What is the role of ER model in database design? Draw an ER diagram for library management system and convert ER-diagram into tables.

Ans. *Refer to Q.12.*

Q.27 ***Draw the diagram of system structure of DBMS. Write down the main function of each component.*** ***[R.T.U. 2016]***

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

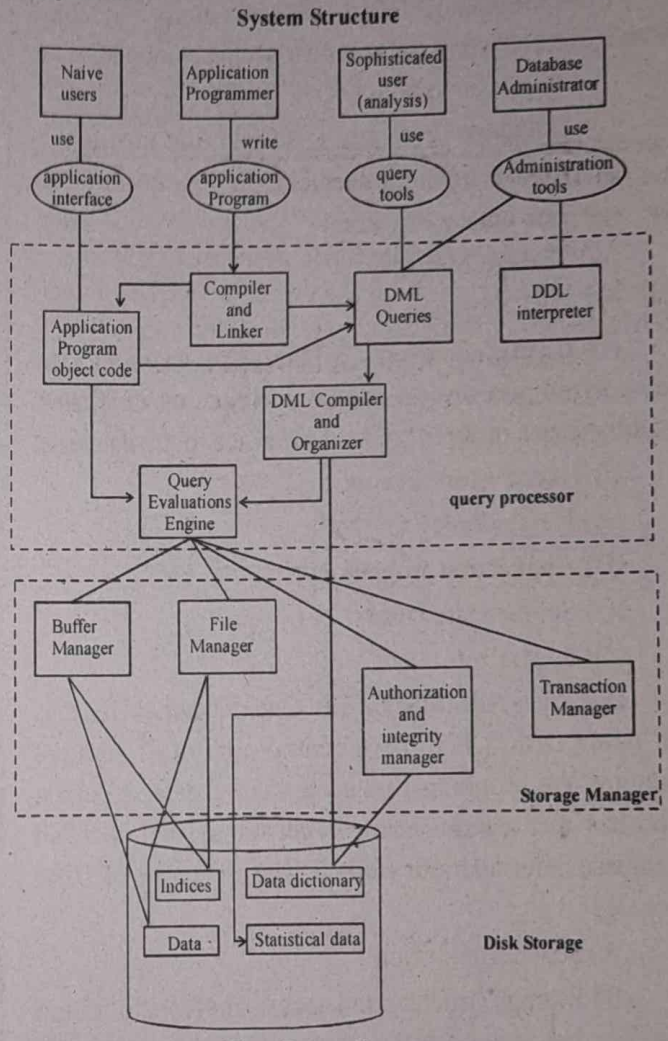


Fig. : System Structure

The storage manager is important because databases typically require a large amount of storage space. Separate databases range in size from hundreds of gigabytes to, for the largest database, terabytes of data. A gigabyte is 1000 mega bytes (1 billion bytes) and a terabyte is 1 million mega bytes (1 trillion bytes). Since main memory of computer cannot store this much information, the information is stored on disks. Data is moved between disk storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of the central processing unit, it is imperative that

the database system structures the data so as to minimize the need to move data between disk and main memory.

The *query processor* is important because it helps the database system to simplify and facilitate access to data. High level views helps to achieve this goal; with them, users of the system are not bounding unnecessarily with the physical details for the implementation of the database system to translate updates and queries written in a non-procedural language, at the logical level, into an efficient sequence of operations at the physical level.

Components of DBMS Environment : Database Management System environment's main components are as :

(i) Database Users and Database Administrator
(ii) Transaction Management
(iii) The Query Processor
(iv) Storage Manager
(v) Disk Storage

(i) Database Users : Database users can be categorized into several classes and each class of users usually uses a different type of interface to the database.

Different users are as :

(A) Naive users
(B) Application programmer
(C) Sophisticated users
(D) Specialized users

Database Administrator : One of the main reasons for using DBMS is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called Database Administrator (DBA). The functions of DBA includes :

(A) Schema definition
(B) Storage structure and access- method definition
(C) Schema and Physical –Organization modification
(D) Granting of authorization for data access
(E) Routine maintenance

(ii) Transaction Management : The transaction management subsystem is responsible for ensuring that the database remains in a consistent (correct) state despite system failures. The transaction manager also ensures that concurrent transaction execution proceed without conflicting.

(iii) The Query Processor : The query processor subsystem compiles and executes DDL and DML statements.

The query processor components include :

(A) DDL interpreter
(B) DML compiler
(C) Query evaluation engine

(iv) Storage Manager : The storage manager subsystem provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.

The storage manager components include :

(A) Authorization and integrity manager
(B) Transaction manager
(C) File manager
(D) Buffer manager

(v) Disk Storage : The storage manager implements several data structures as part of the physical system implementation.

(A) Date files
(B) Data dictionary
(C) Indices

Q.28 ***Differentiate specialization and generalization.*** *[R.T.U. 2010, 2008]*

OR

Explain concept of specialization and generalization in E-R model. *[R.T.U. 2016]*

OR

Write short note on specialization and generalization in E-R model.

[Raj. Univ. 2007, 2006, 2000]

Ans. Although the basic E-R concept can model most database features, some aspect of a database may be more optly expressed by certain extensions Specialization and Generalization.

Specialization : An entity set may include subgrouping of entities that are distinct in some way from the atttributes that are not shared by all the entities in the entity set. The E-R model provides a means for representing these distinct entity grouping.

Consider an entity set person with attributes name, street and city. A person may be further classified as one of the following :

- Customer
- Employee

Each of these person types is described by a set of attributes that includes all the attributes of entity set person plus possibility additional attributes. For example- customer entities may be described further by the attributes customer-id whereas employee entities may be described further by the attributes employee-id and salary. The process of designing subgrouping within an entity set is called *specialization*. The specialization of person allows us to distinguish among person according to whether they are employee or customers.

An another example, suppose the bank wishes to divide accounts need two categories, checking account and saving account. Saving accounts need a minimum balance, but the bank may set interest rates differently for different customers, offering better rates favored customers. Checking account must be recorded.

The bank could then create two specialization of account, namely saving-account and checking-account. The entity set saving-account would have all the attributes account and an additional attribute overdraft account.

We can apply specialization repeatedly to refine a design schema for instance bank employees. It may be further classified as one of the following :

- officer
- teller
- secretry

Each of these employee types is described by a set of attributes that includes all the attributes of entity set employee plus additional attributes. For example- officer entites may be described further by the attributes office-number and hours-per-week and secretary entities by the attributes hours-per week further secretary entities may participate in a relationship secretary for which identifiers and employees are assisted by a secretary.

An entity set may be specialized by more than one distinguishing feature. In our example the distinguishing feature among employee entities is the job the employee performs. Another coexistent specialization could be based on whether the person is a temporary- employee and permanent- employee when more than one specialization is formed on an entity set, a particular entity may belong to multiple who is a secretary.

In terms of an E-R diagram, specialization is depicted by a triangle component labeled IS A, as figure shows. The label IS A stands for "is a" for example that a customer "is a" person. The IS A relationship may also be referred to as a superclass-subclass relationship. Higher and lower level entity sets are depicted as regular entity sets i.e. as rectangles containing the name of the entity set.

Generalization : It is simple inversion of specialization. The refinement from an initial entity set into successive level of entity subgroupings represent a top-down design process in which distinctions are made explicit. The design process may also proceed in a bottom-up manner, in which multiple entity sets are synthesized into higher-level entity set on the basis of common features. The database designer may have first identified a customer entity set with the attributes like name, street, city and customer-id and an employee entity set with the attributes like name, street, city, employee-id and salary.

There are similarities between the customers entity set and the employee entity set in the sense that they have several attributes in common. This commonality can be expressed by generalization, which is a containment relationship that exists between a higher-level entity set and one or more lower-level entity sets. Higher and lower-level entity sets also may be designed by the terms superclass and subclass, respectively the person entity set is the superclass of the customer and employee subclasses.

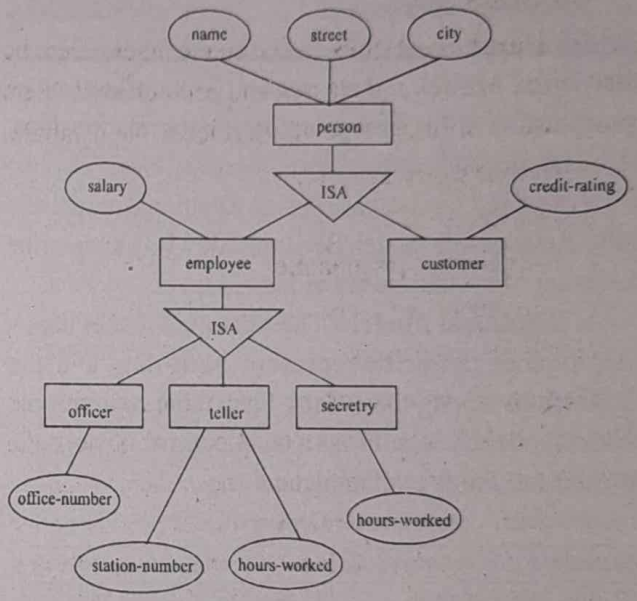


Fig. : Specialization and generalization

For all practical purpose generalization is a simple inversion of specialization we will apply both process, in combination in the course of designing the E–R schema for an enterprise. In terms of the E-R diagram itself we do not distinguish between specialization and generalization new level of entity representation will be distinguished or synthesized as the design schema comes to express fully the database application and the requirement of the database. Difference in the two approaches may be characterized by their starting point and overall goal.

Specialization stems from a single entity set; it emphasizes differences among entities within the set by creating distinct lower-level entity sets. These lower-level entity sets may have attributes or may participate in the

relationship they do not apply to all the entities in the higher level entity set. Indeed the reason a designer applies specialization is to represent such distinctive features. If customer and employee neither have attributes that person entities do not have nor participate in different relationship than those in which person entities participate there would be no need to specialization the person entity set.

Generalization proceeds from the recognition that a number of entity set share some common features. On the basis of their commonalities generalization synthesizes these entity sets into a single higher level set.

Q.29 ***Explain network and object oriented model. What are the roles of these models in database design?*** ***[R.T.U.2015, Dec.2013]***

OR

Define data model. Explain similarities and differences among network, hierarchical and relational data model. ***[R.T.U. 2013]***

Ans. Data Model : Data model is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints. Generally two models are used for database design Relational model and Entity-Relationship model. Both provide a way to describe the design of a database at the logical level.

1. Relational Model : The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns and each column has a unique name. Given table represents a simple relational database:

Table 1 : Customer

Customer-Id	*Customer-Name*	*Customer-Street*	*Customer-City*
192–83–7465	Johnson	12 Alma St.	Palo Alto
019–28–3746	Smith	4 North St.	Rye
677–89–9011	Hayes	3 Main St.	Harrison
182–73–6091	Turner	123 Putnam Ave.	Stamford
321–12–3123	Jones	100 Main St.	Harrison
336–66–9999	Lindsay	175 Park Ave.	Pittsfield
019–28–3746	Smith	72 North St.	Rye

Table 2 : Account

Account-Number	*Balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

Table 3 : Depositor

Customer-Id	*Account-Number*
192–83–7465	A-101
192–83–7465	A-201
019–28–3746	A-215
677–89–9011	A-102
182–73–6091	A-305
321–12–3123	A-217
336–66–9999	A-222
019–28–3746	A-201

Table 1. shows details of bank customers, table 2 shows accounts and table 3 shows which accounts belong to which customers.

The first table, the *customer* table, shows, for example- that the customer identified by customer-id 192-83-7465 is named Johnson and lives at 12 Alma St. in Palo Alto.

The table 2, *account*, shows, for example- that account A-101 has a balance of $500 and A-201 has a balance of $900.

The table 3 shows which accounts belong to which customers. For example- account number A-101 belongs to the customer whose customer-id is 192-83-7465, namely Johnson and customers 192-83-7465 (Johnson) and 019-28-3746 (Smith) share account number A-201 (they may share a business venture).

The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields or attributes. The columns of the table correspond to the attributes of the record type.

It is not hard to see how tables may be stored in files. For instance, a special character (such as a comma) may be used to delimit the different attributes of a record and another special character (such as a newline character) may be used to delimit records. The relational model hides such low-level implementation details from database developers and users.

The relational data model is the most widely used data model and a vast majority of current database systems are based on the relational model.

The relational model is at a lower level of abstraction than the E-R model. Database designs are often carried out in the E-R model and then translated to the relational

model. For example, it is easy to see that the tables *customer* and *account* correspond to the entity sets of the same name, while the table *depositor* corresponds to the relationship set *depositor*. We also note that it is possible to create schemas in the relational model that have problems such as unnecessarily duplicated information. For example, suppose we store *account-number* as an attribute of the *customer* record. Then, to represent the fact that accounts A-101 and A-201 both belong to customer Johnson (with customer-id 192-83-7465), we would need to store two rows in the *customer* table. The values for *customer-name, customer-street* and *customer-city* for Johnson would get unnecessarily duplicated in the two rows.

2. Other Data Models : The **Object-Oriented data model** is another data model that has seen increasing attention. The object-oriented model can be seen as extending the E-R model with notions of encapsulation, methods and object identity.

Historically, two other data models, **the network data model** and the **hierarchical data model,** preceded the relation of data model. These models were tied closely to the underlying implementation and complicated the task of modeling data.

Semi-structured data models permit the specification of data where indivisible data items of the same type may have different sets of attributes. This is in contrast with the data models, where every data item of a particular type must have the same set of attributes. The extensible mark up language (XML) is widely used to represent semi-structured data.

Similarities and Differences Among Different Data Models :

S. No.	Hierarchical Data Model	Network Data Model	Relational Data Model
1.	Relationship between records is of the parent child type (Trees).	Relationship between records is expressed in the form of pointers or links (Graphs).	Relationship between records is represented by a relation that contains a key for each record involved in the relationship.
2.	Many to many relationship cannot be expressed in this model.	Many to many relationship can also be implemented.	Many to many relationship can be easily implemented.
3.	It is a simple, straight forward and natural method of implementing record relationships.	Record relationship implementation is very complex due to the use of pointers.	Relationship implementation is very easy through the use of a key or composite key field(s).
4.	This type of model is useful only when there is some hierarchical character in the database.	Network model is useful for representing such records which have many to many relationships.	Relational model is useful for representing most of the real world objects and relationship among them.
5.	In order to represent links among records, pointers are used. Thus relations among records are physical.	In network model also, the record relations are physical.	Relational model does not maintain physical connection among records. Data is organized logically in the form of rows and columns and stored in table.
6.	Searching for a record is very difficult since one can retrieve a child only after going through its parent record.	Searching a record is easy since there are multiple access paths to data elements.	A unique indexed key field is used to search for a data element.

Similarities and Differences Based on the Operations Performed :

Operation	Hierarchical Model	Network Model	Relational Model
1. Insert operation	We cannot insert the information of a child who does not have any parent.	It does not suffer from any insert anomaly.	It does not suffer from any insert anomaly.
2. Update operation	There are multiple occurrences of child records, which lead to problem of inconsistency during the update operation.	This model is free from update anomalies because there is only a single occurrence for each record set.	This model is also free from update anomalies because it removes the redundancy of data by proper designing through normalization process.

3. Delete operation	It is based on parent child relationship and deletion of parent results in deletion of child records also. This is known as a cascading effect.	It is based on many-to-many (M:N) relationship which make it free from delete anomalies.	It is also free from delete anomalies as the information is stored in different tables.
4. Retrieve operation	Retrieve algorithms are complex and asymmetric.	Retrieve algorithms are complex and symmetric.	Retrieve algorithms are simple and symmetric.

Network Model and its role in Database Design : The network data model preseded the relation of data model. This model was tied closely to the underlying implementation and complicated the task of data modelling.

(i) Relationship between records is expressed in the form of pointer or links (graphs).

(ii) Many-to-many relationship can also be implemented. *"Network model is useful for representing such records which have m-to-m relationships."*

(iii) Searching a record is easy since these are multiple access paths to a data element.

(iv) It does not suffer from any insert anomaly.

(v) This model is free from update and delete anomalies.

Object Oriented Model and its Role in Database Design : The object-oriented data model extends the representation of entities by adding notions of encapsulation, methods (functions), and object identity. The object-oriented model is based on the OOP paradigm and overcome the relational model's restrictions.

- The object-oriented data model has been developed to deal with several new type of applications.
- It is based on the concept of encapsulating in an object, the data and the code that operates on those data.

The main concepts of the object-oriented data model are as follows :

(i) Object : An object corresponds to an entity in the E-R model. In general, an object has associated with it.

- A set of variables
- A set of messages
- A set of methods

(ii) Class : The group of similar objects is ca class. Examples of classes are employees, custom accounts, and loan etc.

(iii) Inheritance : The concept of a class hier is similar to the concept of specialization hierarchy in entity relationship model. For example, employees customers can be represented by classes that specializations of a person class.

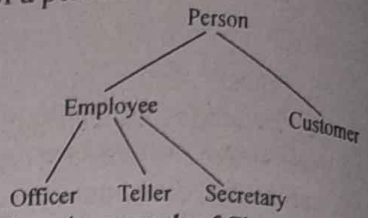


Fig. : An example of Class Hierarchy

Q.30 ***(a) Compare the file system and DBMS on basis of following :***

(i) Integrity

(ii) Difficulty in accessing data

(iii) Concurrently access anomalies

(b) Describe various functions of Database Administrator.

(c) Explain the database design process.

(d) Using neat diagram and examples show that database system hides details of data stored and maintained. *[R.T.U. 20*

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

Integrity of data means that data in database is always accurate, such that incorrect information cannot be stored in database. In order to maintain the integrity of data, some integrity constraints are enforced on the database. A DBMS should provide capabilities for defining and enforcing the constraints. So, we can conclude that integrity constraint can be easily enforced in centralized DBMS system as compared to file system.

(ii) Difficulty in Accessing Data: The conventional file processing environments do not allow needed data to be retrieved in a convenient and efficient manner. Better data retrieval system must be developed for general use. Consider the airline reservation system. If the senior management of company wants to access the information

of all its customers who are living in the same postal code, it has to be done manually because current file processing system does not allow the user to obtain this information. So in the above case, there are two options. Either the application programmer has to write a new application program to satisfy the unusual request or could get this information manually .In former case, it doesn't guarantee that the same query will be asked and same application program would be used in future.

If a query changes, a new application program should be written to get the needed information.

The database system utilizes different sophisticated techniques to access the stored data very efficiently. The database systems provide query languages or report writers that allow the users to ask adhoc queries to obtain the needed information immediately, without the requirement to write application programs (as in case of file system), that access the information from the database. This is possible due to integration in database systems.

(iii) **Concurrent Access Anomalies :** In order to speed up the performance of the system and faster response to applications, many systems allow the user to update the data concurrently. Suppose two users located at different locations wants to book the tickets, there might be situation that both of the people will be given the same seat because the data is stored in multiple locations and both of them will be given a seat from individual copy of the data.

Therefore there should be some protection mechanisms to avoid this concurrent updates. DBMS systems provide mechanisms to provide concurrent access of data to multiple users. Database can manage concurrent data access effectively. It ensures no interference between users that would not result any loss of information or loss of integrity.

Ans. (b) Database Administrator (DBA) : One of the main reasons for using DBMS is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a Database Administrator (DBA).

The functions of a DBA

(i) **Schema definition :** The DBA creates the original database schema by executing a set of data definition statements in the DDL.

(ii) Storage structure and access-method definition.

(iii) **Schema and physical-organization modification :** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization or to alter the physical organization to improve performance.

(iv) **Granting of authorization for data access :** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

(v) **Routine maintenance :** Examples of the database administrator's routine maintenance activities are :

- Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
- Ensuring that enough free disk space is available for normal operations and upgrading disk space as required.
- Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

Ans. (c) Database Design Process : Database design is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data defination language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity.

Design Process

1. **Determine the Purpose of Database :** This helps prepare for the remaining steps.
2. **Find and Organize the Information Required:** Gather all of the types of information to record in the database, such as product name and order number.
3. **Divide the Information into Tables :** Divide information items into major entities or subjects, such as products or orders. Each subject then becomes a table.
4. **Turn Information Item into Columns :** Decide what information needs to be stored in each table. Each item becomes a field, and is displayed as a column in the table. For example, an employees table might include field such as last name and hire date.
5. **Specify Primary Keys :** Choose each table's primary key. The primary key is a column, or a set of columns, that is used to uniquely identify each row. An example might be product ID or order ID.

6. **Set up the Table Relationships :** Look at each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.
7. **Refine the Design :** Analyze the design for errors. Create tables and add a few records of sample data. Check if results come from the tables as expected. Make adjustments to the design, as needed.
8. **Apply the Normalization Rules :** Apply the data normalization rules to see if tables are structured correctly. Make adjustments to the tables.

Ans. (d) A major purpose of a database system is to provide users with an **abstract view** of the data *i.e.* the system hides certain details is of how the data are stored and maintained.

Levels of Abstraction : For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify user's interactions with the system.

1. Physical level : The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

2. Logical level : The next-higher level of abstraction describes what data are stored in the database and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

3. View level : The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

For example, in a Pascal-like language, we may declare a record as follows:

```
type customer = record
                  customer_id: string;
                  customer_name: string;
                  customer_street: string;
                  customer_city: string;
                end;
```

This code defines a new record type called customer with four fields. Each field has a name and a type associated with it. A banking enterprise may have several such record types, including

(a) *account*, with fields *account-number* and *balance*

(b) *employee*, with fields *employee-name* and *salary*

At the **physical level,** a customer, account, or employee record can be described as a block of consecutive storage locations (for example, words or bytes).

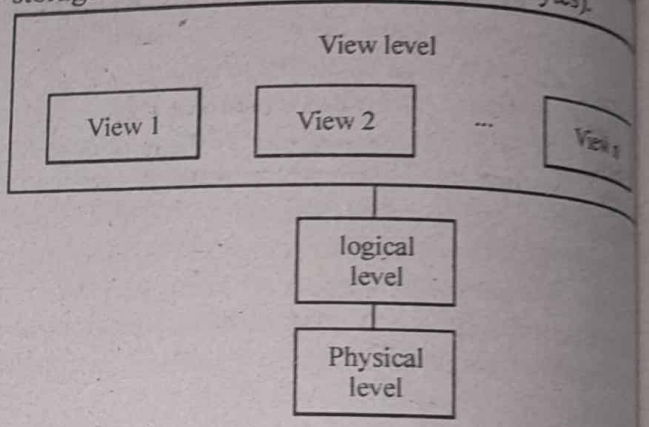


Fig. : The three levels of data abstraction.

The language compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the **logical level**, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

Finally, at the **view level**, computer users see a set of application programs that hide details of the data types. Similarly, at the view level, several views of the database are defined and database users see these views. In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database. For example, tellers in a bank see only that part of the database that has information on customer accounts; they cannot access information about salaries of employees.

DataBase Schemas : A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time corresponds to an instance of a database schema.

Database systems have several schemas, partitioned according to the levels of abstraction. The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called subschemas, that describe different views of the database.

Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. The physical schema is hidden beneath the logical schema and can usually be changed easily without affecting application programs. Application programs are said to exhibit physical data independence if they do not depend on the physical schema and thus need not be rewritten if the physical schema changes.

Q.31 *Consider following transaction that transfer $50 from account A to account B.*

Ti : read (A) : reading A from database

A = A – 50 :

write (A) : updating A in database

read (B) :

B = B + 50 :

write (B) :

using the above example describe the following problems in file system :

(i) Atomicity

(ii) Inconsistency *[R.T.U. 2014]*

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

Suppose that, just before the execution of transaction T_i, the values of accounts A and B are $1000 and $2000, respectively. Now suppose that, during the execution of transaction T_i, a failure occurs that prevents T_i from completing its execution successfully. Further, suppose that the failure happened after the write (A) operation but before the write(B) operation. In this case, the values of accounts A and B reflected in the database are $950 and $2000. The system destroyed $50 as a result of this failure. In particular, we note that the sum A + B is no longer preserved.

Thus, because of the failure, the state of the system no longer reflects a real state of the world that the database is supposed to capture. We term such a state an inconsistent state. We must ensure that such inconsistencies are not visible in a database system. Note, however, that the system must at some point be in an inconsistent state. Even if transaction T_i is executed to completion, there exists a point at which the value of account A is $950 and the value of account B is $2000, which is clearly an inconsistent state. This state, however, is eventually replaced by the consistent state where the value of account A is $950, and the value of account B is $2050. Thus, if the transaction never started or was guaranteed to complete, such an inconsistent state would not be visible except during the execution of the transaction. That is the reason for the atomicity requirement. If the atomicity property is present, all actions of the transaction are reflected in the database, or none are.

The basic idea behind ensuring atomicity is this: The database system keeps track (on disk) of the old values of any data on which a transaction performs a write. This information is written to a file called the log. If the transaction does not complete its execution, the database system restores the old values from the log to make it appear as though the transaction never executed. Ensuring atomicity is the responsibility of the database system; specifically, it is handled by a component of the database called the recovery system.

(ii) Inconsistency : Data inconsistency means different copies of the same data are not matching. That means different versions of same basic data are existing. This occurs as the result of update operations that are not updating the same data stored at different places.

Example : Address information of a customer is recorded differently in different files . Same data which has been repeated at several places may not match after it has been updated at some places. in above example: the sum of A and B is changed by the execution of the transaction if inconsistency occur. Suppose that, just before

the execution of transaction T_i, the values of accounts A and B are \$1000 and \$2000, respectively. Now suppose that, during the execution of transaction T_i, a failure occurs that prevents T_i from completing its execution successfully. Further, suppose that the failure happened after the write(A) and write(B) operation. In this case, the values of accounts A and B are changed. e.g. sum of balances of all accounts, minus sum of loan amounts must unequal value of cash-in-hand .

Consistency Requirement in Above Example : The sum of A and B is unchanged by the execution of the transaction . In general, consistency requirements include. Explicitly specified integrity constraints such as primary keys and foreign keys . Implicit integrity constraints - e.g. sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand . A transaction must see a consistent database. During transaction execution the database may be temporarily inconsistent. When the transaction completes successfully the database must be consistent.

Q.32 ***A university registrar office maintain data about course, course offering, student, instructor, enrolment and grade. Construct an E–R diagram for office and convert E–R diagram into tables.***

[R.T.U. 2014]

Ans. Given that the main entity sets are student, course, course-offering, and instructor. The entity set course-offering is a weak entity set dependent on course. The assumptions made are :

(a) A class meets only at one particular place and time. This E-R diagram cannot model a class meeting at different places at different times.

(b) There is no guarantee that the database does not have two classes meeting at the same place and time.

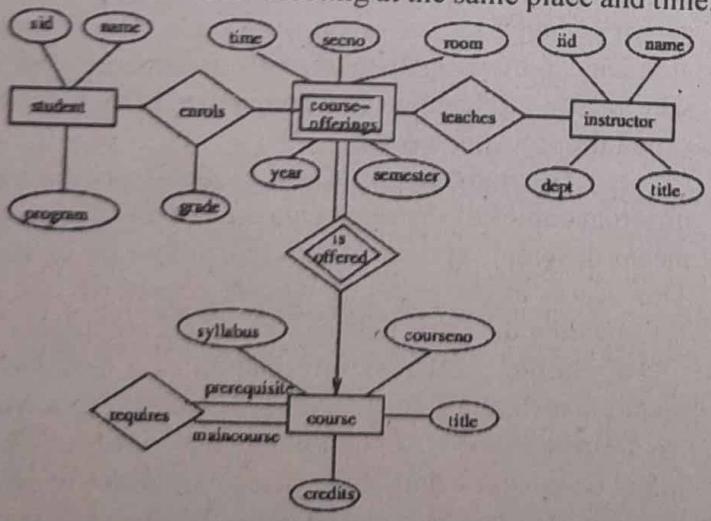


Fig. : E-R diagram for a university

University registrar's tables:

student (student-id, name, program)

course (course-no, title, syllabus, credits)

course-offering (course-no, sec-no, year, semester, time, room)

instructor (instructor-id, name, dept, title)

enrols (student-id, course-no, sec-no, semester, year, grade)

teaches (course-no, sec-no, semester, year, instructor-id)

requires (main-course, pre-requisite)

Table 1 : Student

Student-id	Name	Program
S-112	A	P-1
S-117	B	P-2
S-120	C	P-3

Table 2 : Course

Course-No	Title	Syllabus	Credits
11234	C++	RTU	5
11671	TEF	RTU	7
11841	OOP's	RTU	8

Table 3 : Course-offering

Course-No	Sec-No	Year	Semester	Time	Room
11234	D	First	Fifth	12:30	12
11671	E	Second	Sixth	2:15	13
11841	F	Third	Seventh	3:30	14

Table 4 : Instructor

Instructor-Id	Name	Dept	Title
I-45	AAA	EE	C++
I-86	BBB	EC	TEF
I-13	CCC	CS	OOP's

Table 5 : Enrols

Student-Id	Course-No	Sec-No	Semester	Year	Grade
S-112	11234	D	Fifth	First	a
S-117	11671	E	Sixth	Second	b
S-120	11841	F	Seventh	Third	c

Table 6 : Teaches

Course-No	Sec-No	Semester	Year	Instructor-Id
11234	D	Fifth	First	I-45
11671	E	Sixth	Second	I-86
11841	F	Seventh	Third	I-13

Table 7 : Requires

Main-Course	Pre-Requisite
BCA	12^{th}
MCA	BCA
B.Tech.	12^{th} + Science

Q.33(a) ***Describe the structure of a DBMS. How it is different from RDBMS?*** ***[R.T.U. Dec.2013]***

OR

Describe the five components of DBMS environment and discuss how they relate to each other. ***[R.T.U. 2012]***

(b) ***Explain DBA and data dictionary in brief.*** ***[R.T.U. Dec.2013]***

Ans.(a) Structure of DBMS : *Refer to Q.27.*

Difference between DBMS and RDBMS

S. No.	DBMS	RDBMS
1.	Database Management systems were introduced in 1960s by Charles.	Relational Database Management Systems were in 1970 by Dr. E.F.Codd.
2.	During introduction it followed the navigational modes (Navigational DBMS) for data storage and fetching.	This model uses relationship between tables using primary keys, foreign keys and indexes.
3.	Data fetching is slower for complex and large amount of data.	Comparatively faster because of its relational model.
4.	Used for applications using small amount of data.	Used for complex and large amount of data.
5.	DBMS supports 3 rules of E.F. CODD out off 12 rules.	RDBMS supports minimum 6 rules of E.F. CODD.
6.	DBMS does not support distributed databases.	RDBMS supports distributed databases.
7.	DBMS contains only flat data.	RDBMS contains some relation between entities.
8.	DBMS supports single user.	RDBMS supports multiple user
9.	In DBMS Normalization process will not be present so data redundancy is common in this model.	In RDBMS, normalization process will be present to check the database table consistency. Keys and indexes are used in the tables to avoid redundancy.
10.	Example systems are dBase, Microsoft Access.	Example systems are SQL Server, Oracle.

Ans. (b) Data Base Administrator (DBA) : *Refer to Q.30(b).*

Data Dictionary : Data dictionary stores metadata (data about data) about the structure of the database, in particular the scheme of the database.

A relational-database system needs to maintain data about the relation, such as the schemes of the relation. This information is called the data dictionary or system catalog. Among the types of information that the system must store are these :

(i) Names of the relations
(ii) Names of the attributes of each relation.
(iii) Domains and lengths of attributes
(iv) Names of views defined on the database and definiation of those views.
(v) Integrity constraints.
(vi) Names of authorized users.
(vii) Accounting information about users.
(viii) Passwords of users.
(ix) No. of tuples in each relation.
(x) Method of storage organization.

Q.34 ***What is RDBMS? Describe integrity constraints in relational data model.*** ***[R.T.U. 2013]***

Ans. Relational Data Base Management System (RDBMS) : A relational database consists of a collection of tables, each having a unique name. A row in a table represents a relationship among a set of value. A table is a entity set and row is an entity. Thus a table represents a collection of relationships.

There is a direct correspondence between the concept of a table and the mathematical concept of a relation, from which the relational data model takes its name.

Basic Structure

Consider the employee table. It has 3 column headers : emp_ID, emp_Name and salary. If we follows the terminology of the relational data model, we refer to these headers as attributes. For each attributes, there is set of permitted values called the domain of the attribute. For the attribute emp_Name, the domain is the set of all employee names. Let D_1 denote the set of all employee IDs, D_2 the set of all employee name and D_3 the set of all salary.

Then, any row of employee must consist of a 3-tuple (V_1, V_2, V_3) where V_1 is an employee id (*i.e.* V_1 is in domain D_1) V_2 is a employee name (*i.e.* V_2 is in domain D_2) and V_3 is a salary (*i.e.* V_3 is in domain D_3)

$V_1 \in D_1, V_2 \in D_2, V_3 \in D_3$

In general employee will contain only a subset of the set of all possible rows. Therefore, employee is a subset of

$D_1 * D_2 * D_3$

In general a table of n- attributes must be a subset of

$D_1 * D_2 * .... * D_{n-1} * D_n$

$X_{i=1}^{n} D_i$ (all possible rows)

Integrity Constraints

An Integrity constraint is a condition that is enforced automatically by the DBMS and whose violation prevents the data from being stored in the database. The DBMS enforces integrity constraints in that it only permits legal instances to be stored in the database. The key constraints and the referential integrity constraints are identified as the two minimum constraints that must be enforced by the DBMS.

Constraints can be Defined in Two Ways

1. The constraints can be specified immediately after the column definition. This is called column-level definition.
2. The constraints can be specified after all the columns are defined. This is called table-level definition.

Example of Integrity Constraints

1. No two account can have the same account number.
2. An account number cannot be null.

(i) Primary Key : *Refer to Q.3.*

(ii) Referential Integrity : *Refer to Q.2.*

(iii) SQL Not Null Constraint : This constraint ensures all_rows in the table contain a definite value for the column-which is specified as not null. Which means a null value is not allowed.

Syntax to define a not null constraint :

[CONSTRAINT constraint_name] Not Null

(iv) SQL Unique Key : This constraint ensures that a column or a group of columns in each row have a distinct value.

A columns () can have a null value but the values cannot be duplicated.

Syntax to define a unique key at column level :

[CONSTRAINT constraint_name] UNIQUE

Syntax to define a unique key at table level:

[CONSTRAINT constraint_name] UNIQ (column_name)

(v) SQL Check Constraint : This constraint defin business rule on a column. All the rows must satisfy rule. This constraint can be applied for a single colum a group of columns.

Syntax to define a check constraint :

[CONSTRAINT Constraint_name] CHE (Condition)

Q.35 ***Discuss the various type of keys that are used relational model.*** *[R.T.U. Dec.20*

Ans. Various types of Keys : A key is a single attrib or a combination of attributes whose values uniqu identify each tuple in that relation.

In other words, no two entities in an entity set allowed to have exactly the same value for all attribu A key allows us to identity a set of attributes that sufficient to distinguish relationship from each other.

There are various types of keys :

(i) Super Key
(ii) Candidate Key
(iii) Primary Key
(iv) Alternate Key
(v) Foreign Key

We can understand these keys with the help following example.

Table : Student

St_Name	St_Roll No.	Aadhar No	Sem	Email	Street	City	Dept No
Dheer	143	71234569	VII	Dheer@xyz.com	Sitapura	Jaipur	5
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table : Department

Dept No.	Dept Name	HOD	Dept_location
5	CS	M.K.	1st Floor
⋮	⋮	⋮	⋮

(i) Super Key : A Super key is a set of one or mor attributes that, taken collectively, allow us to identi uniquely an entity in the entity set.

For example :

The St_Roll No. atribute of the student table sufficient to distinguish are student from another. Thus St_Roll No. is a superkey. Similarly, the combination St_Name and St_Roll No. is a super key. The St_Nam is not a super key, because several student might hav the same name i.e. In our example (student table) followin are Super Keys.

St_Roll No

St_Roll No, St_Name ✓
St_Roll No, City ✓
St_Roll No, Sem ✓
St_Name ✗
St_Name, Sem ✗
St_Name, Email ✓
St_Name, DeptNo ✗
Email ✓
Aadhar No. ✓
Aadhar No, St_Name ✓

Suppose that students from same street having different names then

Street ✗
St_Name ✗
St_Name, Street ✓
St_Name, Street, City ✓
St_Name, Street, Sem ✓

(ii) Candidate Key : Minimal superkeys are called candidate keys.

In our example (student table) following are candidate keys.

St_Roll No ✓
Aadhar No. ✓
Email ✓

And if in our system, students from same street are having different names then,

St_Name, Street ✓

But these super keys are not candidate keys

St_Roll No, St_Name ✗ {St_Roll No. alone is sufficient to distinguish.
St_Name, Street, City ✗ {St_Name, street is sufficient to distinguish
St_Roll No., Email ✗ {St_Roll No. or Email is sufficient to distinguish.

i.e. A attribute or a set of minimum attributes which are sufficient to distinguish is called candidate key and combination of other attributes with these attributes can not be a candidate key but it would be a super key.

(iii) Primary Key : The candidate key which is never change or really change is choosen as a primary key.

In our example (student table), among four candidate keys (St_Roll No, St_Name, Street, Email, Aadhar No.) The Aadhar No. never change, so we would select Aadhar no. as a primary key.

(iv) Alternate Keys : The candidate keys which are not choosen as primary key are called alternate keys.

In our example, remaining candidate keys St_Name, Street, Roll No. Email are alternate keys.

(v) Foreign Key : The attribute or set of attributes which are used to link two tables are called foreign key.

In our example, dept No. is a foreign key which is used to link tables student and department. In department table, dept No. is a primary key but it is not a primary key i-student table.

❑❑❑

RELATIONSHIP ALGEBRA AND CALCULAS 2

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 *Explain the role of Triggers in SQL programming.* *[R.T.U. 2019]*

Ans. Triggers : A trigger is a statement that the system executes automatically as a side effect of a modification to the database.

Triggers are special type of stored procedures executed automatically when certain events take place. There are different types of triggers for update, for insert and for delete.

Each trigger is associated with a single database table. Triggers are parameterless procedure that are triggered or fired by the event and not by choice. They cannot have parameters to design a trigger mechanism, we must meet two requirements. Those are-

1. Specify when a trigger is to be executed. This is broken up into an event that causes the trigger to be checked and a condition that must be satisfied for trigger execution to proceed.
2. Specify the actions to be taken when trigger executes.

Q.2 *What is the meaning of sub-query in terms of SQL?* *[R.T.U. 2019]*

Ans. SQL - Sub Queries

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

Q.3 *Consider the relation schema:*
Works(person_name, company_name, salary)
Lives(person_name, street, city)
Located-in(company_name, city)
Managers(person_name, manager_name)
where manager_name refers to person_name.
Give the relational algebra for the following queries:
(i) List the names of the persons work for the company 'SBC' along with the cities they live in.
(ii) Find the name of the persons who live in the same city and same street as their manager.
(iii) Find the persons whose salaries are more than the salary of everybody who works for the company 'SBC'. *[R.T.U. 2019]*

Ans.

(i) $\Pi_{person\_name,\ city}$ (Lives $\bowtie$ ($\sigma_{company\_name = \text{"SBC"}}$ (Works)

(ii) Π_{person_name} ((Lives $\bowtie$ Managers) $\bowtie_{(manager\_name = Lives2.person\_name \wedge Lives.street = Lives2.street \wedge Lives.city = Lives2.city)}$ (ρ_{Lives2} (Lives)))

(iii) Π_{person_name}(Works) – ($\Pi_{Works.person\_name}$ (Works $\bowtie_{(Works.salaryd \leq Works2.salary \wedge Works2.company\_name = \text{"SBC"}}$ ρ_{Works2}(Works)))

Q.4 *Explain aggregate operators.*

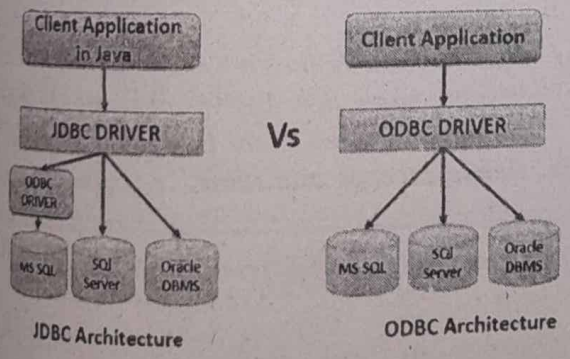
Ans. Aggregate Operators : The aggregation operators perform mathematical operations like Average, Aggregate, Count, Max, Min and Sum, on the numeric property of the elements in the collection.

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

PART-B

Q.5 ***What are the difference between JDBC and ODBC?*** *[R.T.U. 2019]*

Ans. Typically, software applications are written in a specific programming language (such as Java, C#, etc.), while databases accept queries in some other database specific language (such as SQL). Therefore, when a software application needs to access data in a database, an interface that can translate languages to each other (application and database) is required. Otherwise, application programmers need to learn and incorporate database specific languages within their applications. ODBC (Open Database Connectivity) and JDBC (Java DatabBase Connectivity) are two interfaces that solve this specific problem. ODBC is a platform, language and operating system independent interface that can be used for this purpose. Similarly, JDBC is a data API for the Java programming language. Java programmers can use JDBC-to-ODBC bridge to talk to any ODBC compliant database.



JDBC Architecture ODBC Architecture

What is ODBC?

ODBC is an interface to access database management systems (DBMS). ODBC was developed by SQL Access Group in 1992 at a time there were no standard medium to communicate between a database and an application. It does not depend on a specific programming language or a database system or an operating system. Programmers can use ODBC interface to write applications that can query data from any database, regardless of the environment it is running on or the type of DBMS it uses.

Because ODBC driver acts as a translator between the application and the database, ODBC is able to achieve the language and platform independence. This means that the application is relieved of the burden of knowing the database specific language. Instead it will only know and use the ODBS syntax and the driver will translate the query to the database in a language it can understand. Then, the results are returned in a format that can be understood by the application. ODBC software API can be used with both relational and non relational database systems. Another major advantage of having ODBC as a universal middleware between an application and a database is that every time the database specification changes, the software does not need to be updated. Only an update to the ODBC driver would be sufficient.

What is JDBC?

JDBC is a Data API developed for Java programming language. It was released with JDK 1.1 by Sun Microsystems (Java's initial owners). And its current version is JDBC 4.0 (currently distributed with JAVA SE6). Java.sql and javax.sql packages contain the JDBC classes. It is an interface that helps a client to access a database system, by providing methods to query and update data in the databases. JDBC is more suitable for object oriented databases. We can access any ODBC-compliant database by using the JDBC-to-ODBC bridge.

What is the difference between ODBC and JDBC?

ODBC is an open interface which can be used by any application to communicate with any database system, while JDBC is an interface that can be used by Java applications to access databases. Therefore, unlike JDBC, ODBC is language independent. But by using JDBC-to-ODBC bridge Java applications can also talk to any ODBC compliant database.

Comparison Chart

Basis for Comparison	JDBC	ODBC
Basic	JDBC is language and platform dependent (Java Specific).	ODBC is language and platform independent.
Full form	Java Database Connectivity.	Open Database Connectivity.
Code	Code is easy to understand.	Code is complex.

Q.6 ***What do you mean by Null Values? Explain Dynamic SQL in detail.*** *[R.T.U. 2019]*

Ans. Null Values : In databases a common issue is what value or placeholder do you use to represent a missing values. In SQL, this is solved with null. It is used to signify missing or unknown values. The keyword NULL is used to indicate these values. NULL really isn't a specific value as much as it is an indicator. Don't think of NULL as similar to zero or blank, it isn't the same. Zero (0) and blanks " ", are values.

Dynamic SQL : In embedded SQL, the SQL statements to be executed are known at the time of coding the program statements. This is called as static SQL. However, in some cases, the SQL statements to be executed are not known when the program is written. They might be constructed based on user options or, in fact, be written by the user directly in the form SELECT-FROM-WHERE. Such SQL queries, which are not known prior to their execution, from dynamic SQL.

EXECUTE IMMEDIATE <SQL statement>

The EXECUTE IMMEDIATE portion tells the operational environment that the statement belongs to dynamic SQL category.

For example, consider the following dynamic SQL statements. Here, we store the dynamic SQL INSERT statement into a variable called My_statement and pass that variable at the time of execution of dynamic SQL.

My_statement = 'INSERT INTO Student VALUES (10, 'Cryptography', 'Atul', 'THM', 'Security')'

EXECUTE IMMEDIATE: My_statement

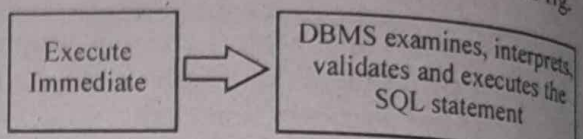
Dynamic SQL is useful in the case of online applications. In such cases, it may not be possible every time to anticipate the kind of queries that the user wants to execute. The user may want to construct them at execution time and execute them immediately. For instance, in a shopping cart application on the Internet, the user may choose to buy 0, 1 or 10 items. In such cases, dynamic SQL would facilitate an elegant creation of a query to process these many items. It may not be possible to achieve the same result using embedded SQL.

There is a variation of the dynamic SQL scheme. We can treat an SQL statement as a temporary object and create it repeatedly. The advantages offered by this approach are as follows :

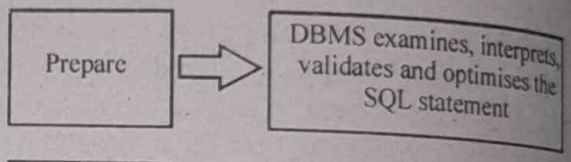
- We can write queries.
- Performance can be far better. This is because the statement is parsed, validated and optimised in advance of execution time.
- Using parameters, the same statement can be executed in various ways.

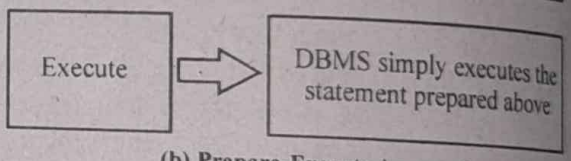
In this model, two statements are required PREPARE and EXECUTE.

The PREPARE statement creates an SQL object that contains the SQL statement to be executed. When this statement is executed, the DBMS examines, interprets, validates and optimises it. It is stored until an EXECUTE (and not EXECUTE IMMEDIATE) statement is encountered. At this stage, the statement is actually executed. This approach is contrasted with the earlier (EXECUTE IMMEDIATE) approach in fig.



(a) Execute Immediate Approach





(b) Prepare-Execute Approach

Fig. : Dynamic SQL approaches

Q.7 ***Write SQL queries for following operations:***

(a) Create student registration table and insert records in it.

(b) Update records based on a key.

(c) Display name and Roll numbers of students who have scored more than 60% marks.

(d) Delete records and table. *[R.T.U. 2017]*

Ans.(a) Create Student registration table and insert records in it :

```
CREATE TABLE Student_Reg
(
 RollNo int,
 Name Varchar(50),
 Course Varchar(10),
 Branch Varchar(10),
```

```
    Score float,
    PRIMARY KEY(RollNo)
);
INSERT into Student_Reg
VALUES (1, Alok, B.Tech, CSE, 70.5);
```

Ans.(b) Update records based on a key

```
UPDATE Student_Reg
SET Course = 'M.Tech'
WHERE Marks > 70.0;
```

Ans.(c) Display name and roll no of students who have scored more than 60% marks

```
SELECT RollNo, Name
FROM Student_Reg
WHERE Marks > 60;
```

Ans.(d) Delete records and table

```
//Delete Records
DELETE from Student_Reg
WHERE Marks < 33
//Delete Table
DROP Student_Reg
```

Q8 *Consider the employee database given below :*

Employee (emp_name, street, city)
Works (emp_name, company_name, salary)
Company (company_name, city)
Manager (emp_name, manager_name)

Give an expression in SQL for each of the following queries :

(i) Modify the database so that Jones now lives in New town.

(ii) Give all managers of first bank corporation a 10 Percent raise.

(iii) Give all managers of First bank corporation a 10 percent raise unless the salary becomes greater than $100,000. In such cases give only a 3 percent raise.

(iv) Find the names of all employees in the database who live in the same city as the company for which they work. *[R.T.U. 2016]*

Ans.(i)

```
update employee
    set city='newtown'
    where emp_name = 'jones';
```

Ans.(ii)

```
update works T
    set salary = salary + salary * 10/100
    where company-name = 'First Bank Corporation';
```

Ans.(iii)

```
Update works T
    set T.salary = T.salary * 1.03
    where T.emp_name in (salary manager_name
                         from manages)
    and T.salary * 1.1 > 100000
    and T.company_name = First Bank Corporation
    update works T
    set T.salary = T.salary * 1.1
    where T.emp_name in (select manager_name
                         from manages)
    and T.salary * 1.1 <= 100000
    and T.company_name = 'First Bank Corporation'
```

SQL-92 provides a **case** operation, using which a more concise solution can be obtained.

```
update works T
set T.salary = T.salary *
(case
      when (T.salary * 1.1 > 100000) then 1.03
else 1.1
)
where T.emp_name in (select manager_name
from manages) and
T.company_name = 'First Bank Corporation'
```

Ans.(iv)

```
select e.emp_name
    from employee e, works w, company c
    where e.emp_name = w.emp_name and
    c.city = c.city and
    w.company_name = c.company_name
```

Q.9 *Consider the following tables:*

Branch (Branch_No, street, city, pincode)
Staff (Staff_No, Fname, position, sex, DOB, Salary, Branch_No)

Answer the following queries using SQL commands:

(i) List all staff with a salary between Rs. 20,000 and Rs. 30,000 of branch office Delhi or Jaipur.

(ii) Find the number of staff working in each branch and sum of their salaries.

(iii) Find all staff whose salary is larger than the salary of at least one member of staff branch 'BO3'.

(iv) Give all staff a 3% pay increases.

[R.T.U. 2015]

Ans.(i)
```
SELECT * FROM Staff, Branch
WHERE
Staff.Branch_no = Branch.Branch_no
AND Salary BETWEEN 20000 AND 30000
AND City = 'Delhi' OR City='Jaipur'
```
(ii)
```
SELECT Branch_no, COUNT(Staff_no)
as "staff_count", SUM(Salary)
FROM Staff
GROUP BY Branch_no
```
(iii)
```
SELECT Staff_no FROM Staff
WHERE Salary > SOME
(SELECT Salary
FROM Staff
WHERE Branch_no='BO3')
```
(iv)
```
UPDATE Staff
SET Salary = Salary + (0.03 *.Salary)
```

Q.10 ***How would you use the feature of nested queries in SQL to develop complex queries? Give examples.*** *[R.T.U. 2015]*

Ans. An SQL subquery is a query that is nested inside another query such as SELECT, INSERT, UPDATE or DELETE. In addition, an SQL subquery can be nested inside another subquery.

An SQL subquery is also called an inner query while the query that contains the subquery is called an outer query.

Let's take a look at the following subquery that returns employees who are located in the offices in the USA.

- The subquery returns all *offices codes* of the offices that locate in the USA.
- The outer query selects the last name and first name of employees whose office code is in the result set returned by the subquery.

Outer Query **Subquery or Inner Query**

```
SELECT lastname, firstname
FROM emplyees
WHERE OfficeCode IN (SELECT officeCode
                     FROM offices
                     WHERE country = 'USA')
```

Another example, we can use comparison operators, like, =, >, <, etc., to compare a single value returned by the subquery with the expression in the WHERE clause.

Consider following payments table :

- Payments(customerNumber, checkNumber, paymentDate, amount)

Consider the following query :

```
SELECT customerNumber,
       checkNumber,
       amount
FROM payments
WHERE amount = (
    SELECT MAX(amount)
    FROM payments
);
```

Q.11 ***Explain Relationship algebra joints.***

[R.T.U. 2016]

[Note : This should be relational algebra joins.]

OR

Discuss the difference between five join operation: theta join, equi join, natural join, outer join and semi join. *[R.T.U. Dec.2013]*

Ans. Relational Algebra Join Operations

Join is combination of cartesian product followed by selection process. Join operation pairs two tuples from different relations if and only if the given join condition is satisfied.

Following are the different types of joins:

1. Theta Join
2. Equi Join
3. Natural Join
4. Outer Join
5. Semi Join

A theta join allows for arbitrary comparison relationships (Such as $\geq$).

- An equi join is a theta join using the equality operator.
- A natural join is an equi join on attributes that have the same name in each relationship.
- We will now discuss them one by one to understand the difference between them.

1. Theta (θ) Join

Theta join is the join condition. Theta joins combines tuples from different relations provided they satisfy the theta condition.

Notation :

$R1 \bowtie_\theta R2$

R1 and R2 are relations with their attributes (A1, A2, ..., An) and (B1, B2, ..., Bn) such that no attribute matches that is $R1 \cap R2 = \phi$. Here θ is condition in form of set of conditions C.

Theta join can use all kinds of comparison operators.

Table : Student Relation

SID	Name	Std
101	Dheer Singh	10
102	Saveen	11

Table : Subjects Relation

Class	Subject
10	Math
10	English
11	Music
11	Sports

$Student\_Detail = STUDENT \bowtie_{Student.Std = Subject.Class} SUBJECT$

Table : Output of Theta Join

SID	Name	Std	Class	Subject
101	Dheer Singh	10	10	Math
101	Dheer Singh	10	10	English
102	Sawra	11	11	Music
102	Sawra	11	11	Sports

2. Equi-Join

When Theta join uses only **equality** comparison operator it is said to be Equi-Join. The above example corresponds to equi-join.

3. Natural Join ($\bowtie$)

Natural join does not use any comparison operator. It does not concatenate the way Cartesian product does. Instead, Natural Join can only be performed if the there is at least one common attribute exists between relation. Those attributes must have same name and domain.

Natural join acts on those matching attributes where the values of attributes in both relation is same.

Table 1 : Relation Courses

SID	Course	Dept
CS01	Database	CS
ME01	Mechanics	ME_2
EE01	Electronics	EE

Table 2 : Relation HOD

Dept	Head
CS	Shailesh Aravatiya
ME	Kavita
EE	P. Mangal

Table 3 : Relation Courses $\bowtie$ HOD

Dept	CID	Course	Head
CS	CS01	Database	Shailesh Aravatiya
ME	ME01	Mechanics	Kavita
EE	EE01	Electronics	P.Mangal

4. Outer Joins

All joins mentioned above, that is Theta Join, Equi Join and Natural Join are called inner-joins. An inner-join process includes only tuples with matching attributes, rest are discarded in resulting relation. There exists methods by which all tuples of any relation are included in the resulting relation.

There are three kinds of outer joins :

(a) Left Outer Join ($R \supset\!\bowtie S$)

All tuples of Left relation, R, are included in the resulting relation and if there exists tuples in R without any matching tuple in S then the S-attributes of resulting relation are made NULL.

Table : Left Relation

A	B
100	Database
101	Mechanics
102	Electronics

Table : Right Relation

A	B
100	Shailesh Aravatiya
102	Kavita
104	P. Mangal

Table : Left Outer Join Output Courses $\supset\!\bowtie$ HOD

A	B	C	D
100	Database	100	Shailesh Aravatiya
101	Mechanics	—	—
102	Electronics	102	Kavita

(b) Right Outer Join : ($R \supset\!\bowtie S$)

All tuples of the Right relation, S, are included in the resulting relation and if there exists tuples in S without any matching tuples in R then the R-attributes of resulting relation are made NULL.

Table : Right Outer Join Output

A	B	C	D
100	Database	100	Shailesh Aravatiya
102	Electronics	102	Kavita
—	—	104	P. Mangal

(c) Full Outer Join : (R⟗S)

All tuples of the both participating relations are included in the resulting relation and if there no mathcing tuples for both relations, their respective unmatched attributes are made NULL.

Table : Full Outer Join Output Courses ⟗ HOD

A	B	C	D
100	Database	100	Shailesh Aravatiya
101	Mechanics	——	—
102	Electronics	102	Kavita
—	—	104	P. Mangal

5. Semi Join

In semi join, first we take the natural join of two relations then we project the attributes of first table only. So after join and matching the common attribute of both relations only attributes of first relation are projected.

Example :

Table : Faculty

facId	facName	Dept	salary	rank
F234	Monika	CSE	21000	lecturer
F235	Amidya	ENG	23000	Asso Prof
F236	Neelam	CSE	27000	Asso Prof
F237	Manju	IT	32000	Professor

Table : Course

Crs Code	CrS Title	Fld
C3456	TOC	F234
C3457	FM	
C3458	DBMS	F236
C3459	OS	F237

If we take the semi join of two relations faculty and course then the resulting relation would be as under :

Table : Semi join operation on Faculty ⋉ Course

facId	facName	Dept	Salary	Rank
F234	Monika	CSE	21000	lecturer
F236	Neelam	CSE	27000	Asso Prof
F237	Manju	IT	32000	Professor

Q.12 *Consider following schemas :*
Project (Pid, Pname, dept–no)
Works–on (emp–id, Pid, hours)
Employee (emp–id, ename, address, salary)
Department (dept–no, dname)
Write relational algebra syntax for following:
(i) For each employee working on a project with Pname of '231 project', retrieve the name of the employee and his/her salary.
(ii) Retrieve the name of each employee who works on all project controlled by department number 5.
(iii) For each project on which more than two employee work retrieve the project number, the project name and the number of employee who work on the project. [R.T.U. 2014]

Ans. (i) $PROJ\_NAME \leftarrow \sigma_{Pname = '231\ project'}(Project)$

$PROJ\_WORKS \leftarrow (PROJ\_NAME \bowtie_{pid = pid} WORKS\text{-}on)$

$WORKS\_EMP \leftarrow (PROJ\_WORKS \bowtie_{emp\_id = emp\_id} Employee)$

$RESULT\ \pi_{ename,\ salary}(WORKS\_EMP)$

(ii) $DEPT5\_PROJS(pid) \leftarrow \pi_{pid}(\sigma_{dept\text{-}no\ =\ 5}(Project))$

$EMP\_PROJ\ (emp\_id, pid) \leftarrow \pi_{emp\_id,\ pid}(WORKS\text{-}on)$

$RESULT\_EMP\_IDS \leftarrow (EMP\_PROJ \div DEPT5\_PROJS)$

$RESULT \leftarrow \pi_{ename}(RESULT\_EMP\_IDS \times EMPLOYEE)$

(iii) $T_1\ (pid, No\text{-}of\text{-}employees) \leftarrow {}_{pid}\Im_{COUNT,\ emp\_id}(WORKS\text{-}on)$

$T_2 \leftarrow \sigma_{No\text{-}of\text{-}employees\ >\ 2}(T_1)$

$RESULT \leftarrow \pi_{Pname,\ pid}(T_2 \times Project)$

Q.13 *Explain following operations in relational algebra with suitable examples :*
(i) Division
(ii) Grouping

Ans. (i) The Division Operation : The division operation is very useful for some queries that include phrases like "for or "every". Division operation is denoted by ÷ or / and general form of division operation is

$$R \div S \text{ or } R/S$$

Here R and S are two relations and schema of S, S_s is subset of schema of R, R, $S_s \subset R_s$. The result of R ÷ S is a relation defined on the schema $R_s - S_s$, that is it

contains only those attributes of first relation R that are in second relation.

To understand division operation consider two relation instances A and B in which (exactly) two fields x and y and B has just one field y with the same domain as Result of A ÷ B will contain all the x values such that for every y value in B, the a tuple <x, y> in A. Division operation is illustrated in figure.

It helps to think of A as a relation listing emp-id and dept-no where the employee ever worked and of the B relation as listing of departments. Now consider a query employee-id who have worked in all the departments". It is shown as query 19 – result of this shows that emp-id 1976 is the only one who has worked in – three departments or we can say 1976 is the only emp-id value for which we have in A relation with all the deptid values in B relation.

It can be written as

EmpID	DeptNo.
1976	1
1976	4
1976	3
1283	4
1283	3
2211	3

Relation A instance

DeptNo
1
3
4

Relation B instance

EmpID
1976

A/B

(ii) Grouping : Queries may require grouping the tuples in the database on the basis of values stored in some column and then applying aggregate function on groups. Such queries can be solved by using grouping operator denoted by G and then applying the aggregate function. The general form of such queries is

$$<\text{grouping attributes}> G_{<\text{function list}>} (R)$$

For example, consider a query "find out maximum salary in each department." To solve this query first we have to make groups of employees, DeptNo wise and then find out the maximum salary for each department.

$$\text{DeptNo } g_{\text{Max (salary)}} (\text{Employee})$$

Consider another example of such queries "Find out the number of employees working in each department and maximum salary for each department". We would write the expression.

$$\text{DeptNo } g_{\text{count (empid-No), max (salary)}} (\text{Employee})$$

Table 1

DeptNo	Avg of Salary	Dept No
1	15000	1
3	9833	3
4	9667	4

Table 2

Count of Employees	Max of Salary
1	1200
3	15000
3	11500

PART-C

Q.14 ***What is embedded SQL?***

Write the following queries in SQL by considering the employee data base ...

(a) Find all the employees in database who live in the same cities as the companies for which they work.

(b) Find all the employees who earn more than the average salary. ***[R.T.U. 2019]***

Ans. Embedded SQL : Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language, which is known as a host language. Because the host language cannot parse SQL, the inserted SQL is parsed by an embedded SQL preprocessor. Embedded SQL is a robust and convenient method of combining the computing power of a programming language with SQL's specialized data management and manipulation capabilities. The SQL standard defines embedding of SQL as embedded SQL and the language in which SQL queries are embedded is referred as host language .The result of the query is made available to the program one tuple (record) at a time .To identify embedded SQL requests to the preprocessor, we use EXEC SQL statement.

EXEC SQL, embedded SQL statement, END-EXEC

Note : A semi-colon is used instead of END-EXEC when SQL is embedded in C or Pascal.

Embedded SQL statements: declare cursor, open, and fetch statements.

```
EXEC SQL
        declare c cursor for
        select cname, ccity
        from deposit, customer
        where deposit.cname = customer.cname
        and deposit.balance > :amount
        END-EXEC
```

where amount is a host-language variable.

```
EXEC SQL open c END-EXEC
```

This statement causes the DB system to execute the query and to save the results within a temporary relation. A series of fetch statement are executed to make tuples of the results available to the program.

Need to Access a Database Using a General Purpose Programming Language : Impedance mismatch is the term used to refer to the problems that occur because of differences between the database model and the programming language model.

Impedance mismatch is less of a problem when a special database programming language is designed that uses the same data model and data types as the database model. One example of such a language is Oracle's PL/SQL. For object databases, the object data model is quite similar to the data model of the Java programming language, so the impedance mismatch is greatly reduced when Java is used as the host language for accessing a Java-compatible object database. Several database programming languages have been implemented as research prototypes.

Most database access in practical situations is through software programs that implement database applications. This software is usually developed in a general purpose programming language such as Java, COBAL or C/C++. The database language such as SQL is, therefore, embedded into general-purpose programming language for accessing the database. When database statements are included in a program, the general-purpose programming language is called the host language, whereas the database language-SRL, in our case is called the data sub-language.

Embedding database commands in a general-purpose programming language : In this approach, database statements are embedded into the host programming language, but they are identified by a special prefix. For example, the prefix for embedded SQL is the string EXEC SQL, which precedes all SQL commands in a host language program. A precompiler or preproccessor scans the source program code to identify database statements and extract them for processing by the DBMS. They are replaced in the program by function calls to the DBMS-generated code. This technique is generally referred to as embedded SQL.

(a) selectE.person_name

fromEmployee as E, Works as W, Company as C

whereE.person_name=W.person_name and E.city=C.city

andW.company_name=C.company_name

(b)SELECT * FROM employees

WHERE salary >

ALL(SELECT avg(salary)FROM employees GROUP BY department_id);

Q.15 ***Explain following operations in relational algebra:***

(a) Selection

(b) Projection

(c) Join

(d) Rename. *[R.T.U. 2017]*

Ans.(a) Select Operation : The select operation selects tuples that satisfies a given predicate. We use the lowercase Greek letter sigma (σ) to denote selection. The predicate appears as a subscript to σ. The argument relation is in parenthesis after the σ.

Loan-number	Branch-Name	Amount
L - 11	Round Hill	900
L - 14	Down town	1500
L -15	Perryridge	1500
L - 16	Perryridge	1300
L - 17	Downtown	1000
L - 23	Red wood	2000
L - 93	Mianus	500

To select those tuples of the loan relation where branch is "perryridge".

$\sigma_{branch\_name} = \text{"Perryridge"}$ (loan)

We can find all tuples in which the amount loan is more than $1200.

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

Thus to find those tuples pertaining to loan of more than $1200 made by the Perryridge branch,

$\sigma_{branch\_name} = \text{"Perryridge"} <amount> 1200$ (loan)

Ans.(b) Project (π) : We have a schema named the loan scheme before which we give a formal definition of the tuple relational calculus, we return to some of the queries for which we wrote relational.

Table 1 : The loan relation

loan-number	branch-name	amount
L-ll	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1500
L-23	Redwood	2000
L-93	Mianus	500

The Project Operation : Suppose we want to list all loan numbers and the amount of the loans, but do not care about the branch name. The project operation allows us to produce this relation. The project operation is a unary operation that returns its argument relation, with certain attributes left out. Since a relation is a set, any duplicate rows are eliminated. Projection is denoted by the uppercase Greek letter pi (Π). We list those attributes that we wish to appear in the result as a subscript to Π. The argument relation follows in parenthesis. Thus, we write the query to list all loan numbers and the amount of the loan as :

$$\Pi_{\text{loan-number, amount}} (\text{Loan})$$

Table 2 : Loan number and the amount of the loan

loan-number	amount
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1500
L-23	2000
L-93	500

Ans.(c) Natural Join ($\bowtie$) : *Refer to Q. 11.*

Ans.(d) The Rename Operation : The resultant relation obtained from any relation algebra expression does not have any name, that we can use to refer to it. It is sometimes very useful to be able to give name to the result of a relational algebra expression. The rename operator, denoted by the lower case Greek letter rho (ρ) is used to do this task. The general form of rename operation is:

$$\rho_x(E)$$

Here x is the new name given to the resultant relation of relational algebra expression E. Rename operator can be used to get a relation R under a new name as relation name R itself is considered to be a relational algebra expression. For example if we use

$$\rho_{Emp} (\text{Employee})$$

Now we can refer Employee relation with it's new name Emp.

A second form of the rename operations is as follows. Assume that a relational algebra expression E has 'n' columns. Then the expression

$$\rho_x (A_1, A_2, \ldots\ldots A_n) (E)$$

returns the result of expression E under the name x and with the columns renamed to $A_1, A_2 \ldots\ldots A_n$. To illustrate rename operation consider the query 7 "Find names of all the employees working in dept-no 4" we can rewrite this query in two steps as:

$$\rho_{\text{emp-d4}} (\sigma_{\text{Dept-No} = 4} (\text{Employee}))$$

$$\rho_{\text{Ename (emp-d4)}}$$

Q.16 ***Explain the difference between relational algebra and relational calculus.*** ***[R.T.U. 2017]***

OR

State the difference between tuple and domain relational calculus. ***[R.T.U. 2017]***

OR

Differentiate relational algebra and relational calculus. ***[R.T.U. 2016]***

Ans. Difference between relational algebra and relational calculus :

Relational Algebra : A basic expression in the relational algebra consists of either one of the following :

(i) A relation in the database

(ii) A constant relation

A constant relation is written by listing its tuples within { }, for example { (A-101, Downtown, 500) (A-215, Mianus, 700) }.

Relational Calculus :

1. Domain Relational Calculus : A second form of relational calculus, called domain relational calculus, uses domain variables that take on values from an attribute domain, rather than values for an entire tuple. The domain relational calculus however is closely related to the tuple relational calculus.

(i) Formal Definitions : An expression in the domain relational calculus is of the form :

$$\{ < x_1, x_2. \ldots x_n > | p(x_1, x_2 \ldots\ldots x_n)\}$$

Where $x_1, x_2 \ldots\ldots x_n$ represent domain variables. 'p' represent a formula composed of atoms. An atom in the domain relational calculus has one of the following forms:

$<x_1, x_2 \ldots x_n > \in$ r, where 'r' is a relation on 'n' attributes and $x_1, x_2 \ldots x_n$ are domain variables or domain constants.

- x⊖ y, where x and y are domain variables and ⊖is a comparison operator.
- x ⊖ C, where x is a domain variable, ⊖ is a comparison operator and 'C' is a constant in the domain of the attribute for which 'x' is a domain variable.

The formula are composed by these atoms using following rules :

- An atom is formula.
- If p_1 is a formula, then $\neg p_1$ and (p_1) are also formula.
- If p_1 and p_2 are formula, then $\neg p_1 \vee p_2$, $p_1 \vee p_2$ and $p_1 \Rightarrow p_2$ are also formula. If $p_1(x)$ is formula in x, where x is a domain variable, then $\exists x\, p_1(x)$ and $\forall(p_1(x)$ are also formulae.

Example : Find all employee working in dept-no = 3

$\{<e_n\ n,\ d,\ mn,\ d_j,\ s, d_n> \mid <e_n\ n,\ d,\ mn,\ d_j,\ s,\ d_n> \in$ employee $\wedge\ d_n = 3\}$

This query requires all seven columns of employee relation, so we need to seven domain variables that are e_n, n, d, mn, d_j, s, d_n. The formula specifies that the collection of variables must belong to employee and d_n which is domain variable for dept-no attribute must be 3.

(ii) Safety of Expression : In tuple relational calculus that some queries of type $\{t|\neg(t \in R)\}$ are not safe as they produce infinite relation. We can define safety of expression explicity and for that we are defining domain of a tuple relational formula p, dom(p) as the values that appear in tuples of a relation mentioned in p and all the values referenced by p. For example, dom(t ∈ Employee ∧ salary > 10000) is the set having 10,000 and all the values appearing in employee.

An expression $\{t|(p(t)\}$ is safe if all values of the result are from dom (p).

So the expression $\{t|\neg(t \in R)\}$ is unsafe as dom($\neg(t \in R)$) is the set of all the values appearing in R. (R is relation specified in p). Similar situation may arise in domain relational calculus. An expression such as :

$\{< a, b, c > \mid \neg(< a, b, c) \in R)\}$ is unsafe, as it allows values that are not in the domain of expression. For the domain relational calculus we define some rules for safety of expression.

$\{<x_1, x_2, \ldots, x_n> \mid P(x_1, x_2, \ldots x_n)\}$

(i) All values that appear in tuples of the expression are values from dom (p).

(ii) For every "These exists" subformula of the form $\exists x\,(p_1(x)$, the subformula is true if and only if there is a value x in dom (p_1) such that $p_1(x)$ is true.

(iii) For every 'for all' subformula of the form $\forall x\,(p_1(x))$, the subformula is true if and only if $p_1(x)$ is true for all values x from dom (p_1).

These rules helps in avoiding test of infinitely many posibilities as we restrict our attention only on the values appeared in dom (p).

All the domain relational calculus queries given in the example are safe.

2. Tuple Relational Calculus : When we write a relational-algebra expression, we provide a sequence of procedures that generate the answer to our query. The tuple relational calculus is a non-procedural query language. It describes the desired information without giving a specific procedure for obtaining that information.

(i) Formal Definition : A query in the tuple relational calculus is expressed as :

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

that is, it is the set of all tuples t such that predicate 'P' is true for t. We use t[A] to denote the value of tuple on attribute A and we use $t \in r$ to denote that tuple 't' is in relation 'r'.

A tuple-relational-calculus formula is built up out of atoms. An atom has one of the following forms :

$S \in r$, where 'S' is a tuple variable and 'r' is a relation.

S[x] – u[y], where S and u are tuple variables. x is an attribute on which 'S' is defined, y is an attribute on which 'y' is defined.

S[x] ⊖C, where 'S' is a tuple variable, x is an attribute on which 'S' is defined, ⊖ is a comparison operator and C is constant in the domain of attribute x.

The tuple relational calculus is restricted to safe expressions is equivalent in expressive power to the basic relational algebra. Thus for every relational expression using only the basic operation.

We build up formulae from atoms using the following rules:

- An atom is a formula.
- If p_1 is a formula, then so are $\neg p_1$ and (p_1).
- If p_1 and p_2 are formulae then so are $p_1 \vee p_2$, $p_1 \wedge p_2$ and $p_1 \Rightarrow p_2$.
- If $p_1(S)$ is a formula containing a free variable and r is a relation, then $\exists S \in r\,(p_1)(S)$ and $\forall S \in r\,(p_1)(S)$ are also formula.

Note : Expressions like $\{t \mid \neg(t \in r)\}$ are called unsafe as it generates infinite tuples. There are infinite tuples not belonging to r.

(ii) Expressive power of Language : Tuple relational calculus restricted to safe expressions and domain relational calculus also restricted to safe expressions are as powerful as relational algebra. If a query language can express all the queries we can express in relational algebra, it is said to be relationally complete. So domain relational calculus and tuple relational calculus are relationally complete.

A practical query langauage is expected to be relationally complete, in addition commercial query languages typically support features that allow us to express some queries that cannot be expressed in relational algebra.

Q.17 ***Explain Triggers with the help of suitable example.*** *[R.T.U. 2017]*

OR

What is trigger? How do we create triggers on a database? Show some syntax. *[R.T.U. 2016, 2014]*

OR

Write short note on Triggers.

[Raj. Univ. 2007, 06, 05]

Ans. Triggers : *Refer to Q.1.*

Triggers are persistent and are accessible to all database operations. Once we enter a trigger into the database, the database system takes on the responsibility of executing it whenever the specified event occurs and the corresponding condition is satisfied.

Triggers are useful mechanism for alerting humans or for starting certain tasks automatically when certain conditions are met.

Components of Trigger

Part	Description	Possible Value
Trigger Timing	When the trigger Fires in relation to the triggering event	BEFORE AFTER
Triggering event	Which data manipulation operation on the table or view causes the trigger to fire	INSERT DELETE UPDATE
Triggering Type	How many times the trigger body executes	Statement Row
Trigger Body	What action the trigger performs	Complete PL/SQL block

Syntax of Creating Statement Trigger

The syntax for creating a trigger is:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
    {BEFORE | AFTER | INSTEAD OF }
    {INSERT [OR] | UPDATE [OR] | DELETE}
    [OF col_name]
    ON table_name
    [REFERENCING OLD AS o NEW AS n]
    [FOR EACH ROW]
WHEN (condition)
DECLARE
    Declaration-statements
BEGIN
    Executable- statements
EXCEPTION
    Exception-handling- statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name: Creates or replaces an existing trigger with the trigger name.
- (BEFORE | AFTER | INSTEAD OF) : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE): This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Creating DDL Event Triggers

```
CREATE or REPLACE trigger CREATE_DB_
OBJECT_ AUDIT
    after create on schema
    begin
    call INSERT_AUDIT_RECORD (ora_dict_obj_
name);
    end;
    /
```

As shown in this example, you can reference system attributes such as the object name.

To protect the objects within a schema you may want to create a trigger that is executed for each attempted drop table command. That trigger will have to be a BEFORE DROP trigger.

Startup and Shutdown Trigger

```
CREATE or REPLACE trigger PIN_ON_STARTUP
after startup on database
begin
DBMS_SHARED_POOL.KEEP (
"SYS.STANDARD", "P");
end;
/
```

This example shows a simple trigger that will be executed immediately after a database startup. You can modify the list of packages in the trigger body to include those most used by your applications.

Startup and shutdown triggers can access the ora_instance_num, ora_database_name, ora_ login_user, and ora_sysevent attributes.

Q.18 ***Explain Embedded SQL and Dynamic SQL.*** *[R.T.U. 2016]*

Ans. Embedded SQL : *Refer to Q.14.*

Dynamic SQL : *Refer to Q.6.*

Q.19 ***(a) Consider following schemas :***
Passengers (Name, Address, Age)
Reservations (Name, FlightNum, Seat)
Flights (FlightNum, DepartCity, DestinationCity, MinutesLate, DepartureTime, ArrivalTime)

(i) Get the name of passengers who had reservation on a flight that was more than 30 minutes late.

(ii) Get the names of passengers who had reservations on all flights that were more than 60 minutes late.

(iii) Get the names of pairs of passengers, who are of the same age.

(b) Discuss various types of inner join operation. *[R.T.U. 2016]*

Ans.(a)

(i) Temp 1 := $\sigma_{MinutesLate>30}$ Flights
Temp 2 := Reservations/$_{FlightName=FlightNum}$ Temp 1
RESULT := Π_{Name} Temp 2

(ii) Temp1 := $\sigma_{MinutesLate>60}$ Flights
Temp 2 := $\Pi_{Flight\ Num}$ Temp 1
Temp 3 := $\Pi_{Name,\ FlightNum}$ Reservations
RESULT := Temp 3/Temp 2

(iii) Temp 0 := Passenger [Name1, Address1, Age1]
Temp 1 := Passenger × Temp0
Temp 2 := $\sigma_{Age=Age1}$ AND$_{Name<>Name1}$ Temp1
RESULT : = $\Pi_{Name1,\ Name2}$ Temp 2

Ans.(b) An inner join requires each row in the two joined tables to have matching column values and is a commonly used join operation in applications but should not be assumed to be the best choice in all situations. Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join predicate. When the join-predicate is satisfied by matching non-NULL values, column values for each matched pair of rows of A and B are combined into a result row. There are mainly two types of inner join operations :

Equi Join : An equi-join is a specific type of comparator-based join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as<) disqualifies a join as an equi-join. We can write equi-join as below.

```
SELECT *
FROM employee, department
WHERE employee.
Department ID = department. DepartmentID;
```

Natural Join : The natural join is a special case of equi-join. Natural jon $(\bowtie)$ is a binary operator that is written as $(R \bowtie S)$ where R and S are relations. The result o the natural join is the set of all combinations of tuples in and S that are equal on their common attribute name The natural join is arguably one of the most import operators since it is the relational counterpart of log AND. Example of Natural Join-

```
SELECT*
FROM employee NATURAL JOIN
department;
```

Q.20 Discuss the various fundamental operations in relational algebra with suitable example. *[R.T.U. 2015]*

OR

Explain expressive power of algebra and calculus. *[R.T.U. Dec.2013, 2011, 2008, Raj. Univ. 2005, 2003, 2001]*

Ans. Expressive power of algebra and calculus :

The expressive power of basic relational algebra is

- **Select (σ) : Unary Operation :** The selection, project and rename operations are called unary operations, because they operate on one relation.

(i) Select Operation : *Refer to Q.15(a).*

(ii) Project (π) : *Refer to Q.15(b).*

(iii) Rename (ρ) : *Refer to Q.15(d).*

- **Binary Operations :** Join and Division are example of binary operation. These operate on two relations.

(i) Division (÷) : The division operation denoted by '÷' is suited to queries that include the phrase "for all"

r ÷ s

Let r and s be relations on schemas R and S respectively where

$R = (A_1 \ldots\ldots, A_m, B_1 \ldots\ldots, B_n)$

$S = (B_1 \ldots\ldots, B_n)$

The result of r ÷ s is a relation on schemas

$R - S = (A_1 \ldots\ldots, A_m)$

$r \div s = \{t / t \in \pi_{R-S}(r) \wedge \forall u \in S(tu \in r)\}$

Ex : Relations r, s:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
∈	6
∈	1
β	2

r

B
1
2

S

r ÷ s =

A
α
β

(ii) The Natural Join Operation (⋈) : The natural join is a binary operation that allows us to combine certain selection and a cartesian product into one operation. It is denoted by the join symbol '⋈'. The natural join operation forms a cartesian forcing equality and those attributes that appear in both relation schemas and finally removes duplicate attributes.

A frequent type of Join connects two relations by:

(i) Equating attributes of the same name and

(ii) Projecting out one copy of each pair of equated attributes i.e. remove duplicate attributes called natural Join

denoted by $R3 = R1 \bowtie R2$

Ex: Sells

Shop	Candy	Price
Sharma	Eclairs	2.50
Sharma	Swaad	2.75
Gupta	Eclairs	2.80
Gupta	Minto	3.00

Shop

Shop	address
Sharma's	Malviya Nagar
Gupta's	Bajaj Nagar

Shoppings = Sells ⋈ Shops

Shoppings

Shops	Candy	Price	Address
Sharma's	Eclairs	2.50	Malviya Nagar
Sharma's	Swaad	2.75	Malviya Nagar
Gupta's	Eclairs	2.50	Bajaj Nagar
Gupta's	Minto	3.00	Bajaj Nagar

(iii) Union (∪) : Consider a query to find the names of all bank customers who have either an account or a loan or both. Note that the customer relation does not contain the information, since a customer does not need to have either an account or a loan at the bank. To answer this query, we need the information in the depositor relation (Table 3) and in the borrower relation (Table 4). We know how to find the names of all customers with a loan in the bank:

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

We also know how to find the names of all customers with an account in the bank:

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

To answer the query, we need the **union** of these two sets that is, we need all customer names that appear in either or both of the two relations. We find these data

by the binary operation union, denoted, as in set theory, by ∪. So the expression needed is

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

The result relation for this query appears in figure. Notice that there are 10 tuples in the result, even though there are seven distinct borrowers and six depositors. This apparent discrepancy occurs because Smith, Jones and Hayes are borrowers as well as depositors. Since relations are sets, duplicate values are eliminated.

Table : The Depositor Relation

customer-name	account-number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Table : The Borrower Relation

customer-name	loan-number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Williams	L-17

customer-name

Adams
Curry
Hayes
Jackson
Jones
Smith
Williams
Lindsay
Johnson
Turner

Fig. : Names of all customers who have either a loan or an account.

(iv) Set Diff. (–) : The set difference operation is used to find tuples that are in one relation but are not in another.

(v) Cartesian Product (×) : The cartesian product operation, denoted by a cross (×), allows us to combine information from any two relation. We write the cartesian product of relations r_1 and r_2 as $r_1 \times r_2$.

For example

The relation schema for r = borrower × loan is

(borrower. customer_name, borrower.loan_number, loan.branch_name, loan. loan-number, loan.amount)

With this schema, we can distinguish borrow.loan_number from loan. loan_number.

(vi) Set intersection (∩) : $r \cap S = r - (r - S)$

The extended relational algebra allows

(a) Generalized Projection (g) : It extends the projection operation by allowing arithmetic functions to be used in the projection list.

(b) Aggregate Functions : It allows G_{sum}, G_{ang}, G_{min}, G_{max} and $G_{dint\text{-}district}$ functions.

(c) Outer-join : *Refer to Q.11.*

Q.21 ***Consider the schemas and write the SQL syntax for mention statements (i), (ii) and (iii).***

Project (Pid, Pname, dept–no)

Works–on (emp–id, Pid, hours)

Employee (emp–id, ename, address, salary)

Department (dept–no, dname)

(i) For each employee working on a project with Pname of '231 Project', retrieve the name of the employee and his/her salary.

(ii) Retrieve the name of each employee who works on all project controlled by department number 5.

(iii) For each project on which more than two employee work, retrieve the project number, the project name and the number of employee who work on the project.

[R.T.U. 2014]

Ans.(i) SELECT ename, salary
FROM Employee, Works-on, Project
WHERE emp-id = emp-id AND pid = pid AND Pname = '231 Project';

(ii) SELECT ename
FROM Employee
WHERE ((SELECT pid
FROM Works-on

```
                WHERE   emp-id = emp-id
                CONTAINS
                (SELECT pid
                FROM    Project
                WHERE   dept-no = 5));
(iii) SELECT    pid, Pname, COUNT (*)
      FROM      Project, Works-on
      WHERE     pid = pid
      GROUP BY  pid, Pname
      HAVING    COUNT (*) > 2;
```

Q.22 ***What is JDBC? Explain establishing a connection to the database, create statement, execute query and iterate result set in JDBC.***

[R.T.U. 2014]

Ans. JDBC : *Refer to Q.5.*

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database.

(1) Establish a JDBC Connection : The programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps :

(i)Import JDBC Packages : The Import statements tell the Java compiler where to find the classes you reference in your code and are placed at the very beginning of your source code.To use the standard JDBC package, which allows you to select, insert, update, and delete data in SQL tables, add the following imports to your source code.

```
import java.sql.* ;  // for standard JDBC programs
import java.math.* ; // for BigDecimal and BigInteger support
```

(ii)Register JDBC Driver : You must register the driver in your program before you use it. Registering the driver is the process by which the Oracle driver's class file is loaded into the memory, so it can be utilized as an implementation of the JDBC interfaces.

You can register a driver in one of two ways.

Approach I - Class.forName() : The most common approach to register a driver is to use Java's Class.for Name() method, to dynamically load the driver's class file into memory, which automatically registers it.

Approach II - DriverManager.registerDriver() : The second approach you can use to register a driver, is to use the staticDriverManager.registerDriver() method.

(iii)Database URL Formulation : After you've loaded the driver, you can establish a connection using theDriverManager.getConnection() method. For easy reference, let me list the three overloaded DriverManager.getConnection() methods :

- getConnection(String url)
- getConnection(String url, Properties prop)
- getConnection(String url, String user, String password)

(iv) Create Connection Object

We have listed down three forms of DriverManager.getConnection() method to create a connection object.Once a connection is obtained we can interact with the database. The JDBC Statement, CallableStatement, and PreparedStatement interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database.

(2) Creating Statement : Before you can use a Statement object to execute a SQL statement, you need to create one using the Connection object's createStatement( ) method, as in the following example :

```
stmt = conn.createStatement( );
```

(i)The PreparedStatement Objects

The PreparedStatement interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object.

This statement gives you the flexibility of supplying arguments dynamically.

```
pstmt = conn.prepareStatement(SQL);
```

(ii)The CallableStatement Objects

Just as a Connection object creates the Statement and PreparedStatement objects, it also creates the CallableStatement object, which would be used to execute

a call to a database stored procedure. Three types of parameters exist: IN, OUT, and INOUT. The PreparedStatement object only uses the IN parameter. The CallableStatement object can use all the three.

```
CallableStatement cstmt = null;
String SQL = "{call getEmpName (?, ?)}";
cstmt = conn.prepareCall (SQL);
```

(3) Execute Query

Once you've created a Statement object, you can then use it to execute an SQL statement with one of its three execute methods.

- **boolean execute (String SQL):** Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
- **int executeUpdate (String SQL):** Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected : for example, an INSERT, UPDATE, or DELETE statement.
- **ResultSet executeQuery (String SQL):** Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

(4) Result Set

The SQL statements that read data from a database query, return the data in a result set. The SELECT statement is the standard way to select rows from a database and view them in a result set. The java.sql.ResultSet interface represents the result set of a database query.

A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

The methods of the ResultSet interface can be broken down into three categories :

- **Navigational Methods :** Used to move the cursor around.
- **Get Methods :** Used to view the data in the columns of the current row being pointed by the cursor.
- **Update Methods :** Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

The cursor is movable based on the properties of the ResultSet. These properties are designated when the corresponding Statement that generates the ResultSet is created.

Q.23 ***(a) Consider the following database schema***

employee (ename, street, city)

Works (ename, company_name, salary)

Company (company_name, city)

manager (ename, mgr_name)

Give regular expression for each of the following statements :

(i) Find the company with most employees.

(ii) Find the company with the smallest payroll.

(iii) Give all employees of ABC company a 10-percent salary raise.

(iv) Delete all tuples in the works relation for employee of XYZ, company.

(v) Find the names and cities of residence of all employees who works for ABC company.

(b) Explain any 3 set-operators and 3-Aggregate operators in SQL with the help of suitable examples. **[R.T.U. 2013]**

Ans. (a) Regular Expression is :

(i) $t_1 \leftarrow {}_{company\_name}G_{count\text{-}distinct}\ ename\ (works)$

$t_2 \leftarrow \max_{num\_employees}(\rho_{company\_strength\ (company\_name,\ num\_employees)}(t_1))$

$\Pi_{company\_name}(\rho_{t3\ (company\_name,\ num\_employees)}(t_1) \rhd\lhd \rho_{t4(num\_employees)}(t_2))$

(ii) $t_1 \leftarrow {}_{company\_name}G_{Sum}\ salary\ (works)$

$t_2 \leftarrow \min_{payroll}(\rho_{company\_payroll\ (company\_name,\ payroll)}(t_1))$

$\Pi_{company\_name}(\rho_{t3\ (company\_name,\ payroll)}(t_1) \rhd\lhd \rho_{t4\ (payroll)}(t_2))$

(iii) works ← $\Pi_{ename,\ company\_name,\ 1.1*salary}(\sigma_{(company\_name = "ABC")}$ (works) ∪ (works − $\sigma_{company\_name = "ABC"}$ (works))

(iv) works ← works − $\sigma_{company\_name = "XYZ"}$ (works)

(v) $\Pi_{ename,\ city}$ (employee ⋈ ($\sigma_{company\_name = "ABC"}$ (works))

Ans. (b) Set Operations

The SQL operations **union, intersect,** and **expect** operate on relations and correspond to the relational-algebra operations ∪, ∩, and −. Like union, intersection, and set difference in relational algebra, the relations participating in the operations must be compatible; that is, they must have the same set of attributes.

Let us demonstrate how several of the example queries can be written in SQL. We shall now construct queries involving the **union, intersect,** and **except** operations of two sets: the set of all customers who have an account at the bank, which can be derived by

select *customer_name*

from *depositor*

and the set of customers who have a loan at the bank, which can be derived by

select *customer_name*

from *borrower*

In our discussion that follows, we shall refer to the relation obtained as the result of the preceding queries as d and b, respectively.

1. The Union Operation

To find all the bank customers having a loan, an account, or both at the bank, we write

(**select** *customer_name*

from *depositor*)

union

(**select** *customer_name*

from *borrower*)

The **union** operation automatically eliminates duplicates, unlike the **select** clause. Thus, in the preceding query, if a customer-say, jones-has several accounts or loans (or both) at the bank, then Jones will appear only once in the result. If we want to retain all duplicates, we must write **union all** in place of **union** :

(**select** *customer_name*

from *depositor*)

union all

(**select** *customer _name*

from *borrower*)

The number of duplicate tuples in the result is equal to the total number of duplicates that appear in both *d* and *b*. Thus, if Jones has three accounts and two loans at the bank, then there will be five tuples with the name Jones in the result.

2. The intersect Operation

To find all customers who have both a loan and an account at the bank, we write

(**select distinct** *customer_name*

from *depositor*)

intersect

(**select distinct** *customer_name*

from *borrower*)

The **intersect** operation automatically eliminates duplicates. Thus, in the preceding query, if a customer–say, Jones–has several accounts and loans at the bank, then loans will appear only once in the result.

If we want to retain all duplicates, we must write intersect all in place of intersect :

(**select** *customer_name*

from *depositor*)

intersect all

(**select** *customer_name*

from *borrower*)

The number of duplicate tuples that appear in the result is equal to the minimum number of duplicates in both *d* and *b*. Thus, if Jones has three accounts and two loans at the bank, then there will be two tuples with the name Jones in the result.

3. The Except Operation

To find all customers who have an account but no loan at the bank, we write

(**select distinct** *customer_name*

from *depositor*)

except

(**select** *customer_name*

from *borrower*)

The **except** operation automatically eliminates duplicates. Thus, in the preceding query, a tuple with customer name Jones will appear (exactly once) in the result only if jones has an account at the bank, but has no loan at the bank.

If we want to retain all duplicates, we must write except all in place of except :

(**select** *customer_name*

from *depositor*)

except all

(**select** *customer_name*

from *borrower*)

The number of duplicate copies of a tuple in the result is equal to the number of duplicate copies of the tuple in depositor minus the number of duplicate copies of the tuple in borrower , provided that the difference is positive. Thus, if Jones has three accounts and one loan at the bank, then there will be two tuples with the name Jones in the result. If, instead, this customer has two accounts and three loans at the bank, there will be no tuple with the name Jones in the result.

The aggregate operators supported by SQL are :

COUNT, SUM, AVG, MIN, MAX .

(i) COUNT (A): The number of values in the column A

(ii) SUM (A): The sum of all values in column A

(iii) AVG (A): The average of all values in column A

(iv) MAX(A): The maximum value in column A

(v) MIN(A): The minimum value in column A

Using the COUNT operator

Count the number of sailors

```
SELECT COUNT (*)
FROM Sailors S;
```

Count the number of different sailor names

```
SELECT COUNT (DISTINCT S.sname)
FROM Sailors S;
```

Example of SUM operator

Find the Sum of ages of all sailors with a rating of 10

```
SELECT SUM (S. age)
FROM Sailors S
WHERE S. rating = 10;
```

Example of AVG Operator

Find the average age of all sailors with rating 10

```
SELECT AVG (S. age)
FROM Sailors S
WHERE S. rating = 10
```

SCHEMA REFINEMENT AND NORMAL FORMS 3

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 *What is the purpose of normalization in DBMS?* *[R.T.U. 2019]*

OR

Discuss the need of normalization.

Ans. Need of Normalization : When you normalize a database, you have four goals: arranging data into logical groupings such that each group describes a small part of the whole; minimizing the amount of duplicate data stored in a database; organizing the data such that, when you modify it, you make the change in only one place; and building a database in which you can access and manipulate the data quickly and efficiently without compromising the integrity of the data in storage.

Data normalization helps you design new databases to meet these goals or to test databases to see whether they meet the goals. Sometimes database designers refer to these goals in terms such as data integrity, referential integrity, or keyed data access. Ideally, you normalize data before you create database tables. However, you can also use these techniques to test an existing database.

Q.2 *Decompose the schema R = (A, B, C, D, E) into (A, B, C) and (A, D, E). Also show that this decomposition is a lossless-join decomposition if the following set F of functional dependencies holds :*

$A \rightarrow BC,\ CD \rightarrow E,\ B \rightarrow D,\ E \rightarrow A$ *[R.T.U. 2015, 2013]*

OR

Suppose we decompose the scheme R (A, B, C, D, E,) into (A, B, C) and (A, D, E). Show that this is a lossless-join decomposition if the Set F = {A → BC, CD→E, B→D, E→A} of functional dependencies hold. *[R.T.U. 2011]*

Ans. A decomposition $\{R_1, R_2\}$ is lossless-join decomposition if

$R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2$

Let $R_1 = (A, B, C)$ and $R_2 = (A, D, E)$

So, $R_1 \cap R_2 = A$

Since A is a candidate key. There $R_1 \cap R_2 \rightarrow R_1$ so this is a lossless-join decomposition.

Q.3 *What is normalization.*

Ans. Normalization : Database normalization or data normalization, is a technique to organize the contents of the tables for transactional databases and data warehouses. Normalization is part of successful database design; without normalization, database systems can be inaccurate, slow, and inefficient, and they might not produce the data you expect.

Q.4 *Write advantages of normaliztion.*

Ans. Advantages of Normalization:

1. Avoids data modification (INSERT/DELETE/ UPDATE) anomalies as each data item lives in one place.
2. Greater flexibility in getting the expected data.
3. Normalization is conceptually cleaner and easier to maintain and change as your needs change.
4. Fewer null values and less opportunity for inconsistency.
5. A better handle on database security.

Q.5 *Explain Decomposition.*

Ans. Decomposition : A functional decomposition is the process of breaking down the functions of an organization into progressively greater (finer and finer) levels of detail. In decomposition, one function is described in greater detail by a set of other supporting functions.

The decomposition of a relation scheme R consists of replacing the relation schema by two or more relation schemas that each contain a subset of the attributes of R and together include all attributes in R.

Decomposition helps in eliminating some of the problems of bad design such as redundancy, inconsistencies and anomalies.

There are two types of decomposition :

1. Lossy Decomposition
2. Lossless Join Decomposition

PART-B

Q.6 What is Normalization? Also explain functional dependencies with a suitable example.

[R.T.U. 2019]

Ans. Normalization : *Refer to Q.3.*

Functional Dependencies : The single most important concept in a relational schema design is that of functional dependency. A functional dependency is a constraint between two set of attributes in a relation from a database.

Given a relation R a set of attributes x in R is said to functionally determine. Another attribute y also in R, (written $x \rightarrow y$) if and only if each x value is associated with at most one y value. We call x the determinant set and y the dependent attribute. Thus given a tuple and the values of the attribute in x. One can determine the corresponding value of the y attribute.

A functional depending set 'S' is irreducible if the set has three following properties :

1. Each right set of a functional dependency of 'S' contains only one attribute.
2. Each left set of functional dependency of 'S'is irreducible. It means that reducing any one attribute from left set would not change the content of S.
3. Reducing any functional dependency will change the content of S.

A functional dependency, denoted by $x \rightarrow y$, between two sets of attributes x and y that are subsets of the attributes of relation R.

Functional dependencies and keys : We say that a set of one or more attributes

{A_1, ... A_n} is a key for a relation R if :

1. Those attributes functionally determine all other attributes of the relation.
2. No proper subset of those attributes functionally determines all other attributes of R.

A set of attributes that contains a key is called a superkey. Thus every key is a superkey but not every key is minimal. If a relation has more than one key, one of the keys is designed as the **primary key**.

Closures : Let a relation 'R' has some functional dependencies 'F' specified. The closure is the set of all functional dependencies that may be logically derived from F.

'F' is the set of most obvious and important functional dependencies and F^+, the closure is the set of all the functional dependencies including F and those that can be deduced from F^+. The closure is important and may be needed in finding one or more candidate keys of the relation.

A set of rules that may be used to infer additional dependencies was proposed by Armstrong in 1974. These rules are complete set of rules called Axioms or inference rules such that all possible functional dependencies may be derived from them. The rules are :

1. **Reflexivity Rule :** If 'x' is a set of attributes and 'y' is a subset of 'x', then $x \rightarrow y$ holds.
2. **Augmentation Rule :** If $x \rightarrow y$ holds and 'ω' is the set of attributes, then $\omega x \rightarrow \omega y$ holds.
3. **Transitivity Rule :** If $x \rightarrow y$ and $y \rightarrow z$ hold, then $x \rightarrow z$ holds.

These rules are called Armstrong's Axioms.

The most important additional axioms are :

1. **Union Rule :** If $x \rightarrow y$ and $x \rightarrow z$ hold then $x \rightarrow yz$ holds.
2. **Decomposition Rule :** If $x \rightarrow yz$ holds, so do $x \rightarrow y$ and $x \rightarrow z$.
3. **Pseudotransitivity Rule :** If $x \rightarrow y$ and $\omega y \rightarrow z$ hold then so does $\omega x \rightarrow z$.

Full Functional Dependencies (FFD):

A functional dependency $X \rightarrow Y$ is full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.

If all the non-key attributes of the entity completely and functionally depend on the key attribute of the same entity so it is known as full Functional Dependencies.

For example, we have Student table (Entity) having 4 column (attribute).

(1) Sid
(2) Sname
(3) Add
(4) CourseName

The Sname, Add and CourseName are the non-key attribute which completely depend on Sid (Key Attribute).

If we want to retrive any information about student we only need key attribute i.e. Sid and rest of the information we can get on the basis of key attribute.

So here all the attribute (Sname, Add, CourseName) fully depend on key attribute, and hence it is called full Fuctional Dependencies.

Q.7 ***Explain functional dependencies with the help of suitable examples.*** *[R.T.U. 2017]*

OR

Describe the concept of full functional dependency. *[R.T.U. 2015]*

OR

Define Functional Dependency. Explain Armstrong's axioms or rules, with examples. *[R.T.U. 2015]*

OR

Describe the concept of full functional dependency (FFD). *[R.T.U. Dec. 2013, 2008]*

Ans. Functional Dependencies : *Refer to Q.6.*

Q.8 ***What is Decomposition? Explain Lossy and Lossless join decomposition.*** *[R.T.U. 2016]*

Ans. Decomposition : *Refer to Q.5.*

Lossy Decomposition : "The decomposition of relation R into R1 and R2 is **lossy** when the join of R1 and R2 does not yield the same relation as in R."

One of the disadvantages of decomposition into two or more relational schemes (or tables) is that some information is lost during retrieval of original relation or table.

Consider that we have table STUDENT with three attributes roll_no , sname and department.

STUDENT:

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

This relation is decomposed into two relations no_name and name_dept :

No_name

Roll_no	Sname
111	parimal
222	parimal

Name_dept

Sname	Dept
parimal	COMPUTER
parimal	ELECTRICAL

In lossy decomposition ,spurious tuples are generated when a natural join is applied to the relations in the decomposition.

stu_joined :

Roll_no	Sname	Dept
111	parimal	COMPUTER
111	parimal	ELECTRICAL
222	parimal	COMPUTER
222	parimal	ELECTRICAL

The above decomposition is a bad decomposition or Lossy decomposition.

Lossless Join Decomposition : "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."

A relational table is decomposed (or factored) into two or more smaller tables, in such a way that the designer can capture the precise content of the original table by joining the decomposed parts. This is called lossless-join (or non-additive join) decomposition.

This is also refferd as non-additive decomposition.

The lossless-join decomposition is always defined with respect to a specific set F of dependencies.

Consider that we have table STUDENT with three attribute roll_no , sname and department.

STUDENT :

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

This relation is decomposed into two relation Stu_name and Stu_dept :

Stu_name

Roll_no	Sname
111	parimal
222	parimal

Stu_dept

Roll_no	Dept
111	COMPUTER
222	ELECTRICAL

Now ,when these two relations are joined on the common column 'roll_no' ,the resultant relation will look like stu_joined.

stu_joined :

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

In lossless decomposition, no any spurious tuples are generated when a natural joined is applied to the relations in the decomposition.

Q.9 *We are given a schema S = (A, B, C, D, E). The F of functional dependencies is {A→B, BC→E, ED→A}*

(a) List all candidate keys for S.

(b) Is S in 3NF? Why?

(c) Is S in BCNF? Why?

(d) Find canonical cover F_c of F.

[R.T.U. 2016, 2014]

Ans. (a) Diagrammatic representation of the FD's is

$A \rightarrow B$

$C \rightarrow E$

$D \rightarrow A$

Finding the candidate keys (CKs)

'A' uniquely can't determine all the attributes. It needs 'C' and 'D'. Therefore CK is {A, C, D}.

Similarly if we have 'B' and 'C', we need 'D' to determine all other attributes. So CK is {B, C, D}.

Finally, if we have 'E' and 'D', we need 'C' to determine all other attributes.

So, CK is {C, E, D}

Thus, we have 3CKs

{A, C, D}

{B, C, D}

{C, E, D}

Non-prime attributes are 'C' and 'D'.

(b) Now to check all NFs

(i) Any given relation by definition of a relation is already in 1NF.

(ii) To find out 2NF

Is there any composite key?

Ans. Yes

Then does any part of our composite keys independently determine any of the non-prime attributes?

Ans. 'C' and 'D' cannot be independently determined by any part of any CK. So 2NF condition is passed.

(iii) To find out 3NF

Does the relation have any non-primes?

Ans. Yes

Then, is there a non-prime that determines (point to) another non-prime?

Ans. Neither 'C' determines (points to) 'D' nor vice versa.

So, 3NF condition is passed.

(c) To find out BCNF

In all given FDs, are all the determinents also candidate keys?

No, neither 'A' nor 'BC' nor 'ED' are candidate keys. Therefore BCNF condition does not pass.

∴ Highest NF is 3NF

(d) In the given dependencies no any redundency. So the canonical cover F_c of F is

$A \rightarrow B$

$C \rightarrow E$

$D \rightarrow A$

Q.10 *Compute F^+ of the following set F of Functional Dependency for relation schema R = (A,B,C,D,E)*

$A \rightarrow BC$, $CD \rightarrow E$, $B \rightarrow D$, $E \rightarrow A$.

List the candidate key for R. *[R.T.U. 2013]*

Ans. Note : It is not reasonable to expect students to enumerate all of F^+. Some shorthand representation of the result should be acceptable as long as the nontrivial members of F^+ are found.

Starting with $A \rightarrow BC$, we can conclude: $A \rightarrow B$ and $A \rightarrow C$

Since $A \rightarrow B$ and $B \rightarrow D$, $A \rightarrow D$	(decomposition, transitive)
Since $A \rightarrow CD$ and $CD \rightarrow E$, $A \rightarrow E$	(union, decomposition, transitive)
Since $A \rightarrow A$, We have	(reflexive)
$A \rightarrow ABCDE$ from the above steps	(union)
Since $E \rightarrow A$, $E \rightarrow ABCDE$	(transitive)
Since $CD \rightarrow E$, $CD \rightarrow ABCDE$	(transitive)
Since $B \rightarrow D$ and $BC \rightarrow CD$, $BC \rightarrow ABCD$	(augmentative, transitive)
Also, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow D$, etc.	

Therefore, any functional dependency with A, E, BC, or CD on the left hand side of the arrow is in F^+, no

matter which other attributes appear in the FD. Allow * to represent any set of attributes in R, then F^+ is $BD \rightarrow B$, $BD \rightarrow D$, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow BD$, $B \rightarrow D$, $B \rightarrow B$, $B \rightarrow BD$, and all FDs of the form $A^* \rightarrow \alpha$, $BC^* \rightarrow \alpha$, $CD^* \rightarrow \alpha$, $E^* \rightarrow \alpha$ where α is any subset of {A, B, C, D, E}. The candidate keys are A, BC, CD, and E.

PART-C

Q.11 *Explain Boyce-Codd normal form and 3-NF in detail.* *[R.T.U. 2019]*

Ans. BCNF (Boyce-Code Normal Form)

One of the most desirable normal forms that we can obtain is Boyce-Code Normal Form (BCNF). A relation schema R is in BCNF with respect to a set of functional dependencies if for all functional dependencies in F^+ of a form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds :

- $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \subseteq \alpha$).
- α is a superkey for schema R.

A database design is in BCNF if each member of the set of relation schema that constitutes the design is in BCNF.

Often testing of a relation to see if it satisfies BCNF can be simplified.

- To check if a nontrivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF, compute $\alpha+$ and verify that it includes all attributes of R; that is, it is a superkey of R.
- To check if a relation schema R is in BCNF, it suffices to check only the dependencies in the given set F for violation of BCNF, rather than to check all dependencies in F^+.

Example of BCNF

Stud-ID	S-Name	Subject	Grade
1001	Axay	Physics	A
1001	Axay	Chemistry	C
1001	Axay	Maths	C
1002	Sparsh	Physics	A
1002	Sparsh	Chemistry	A
1002	Sparsh	Maths	B

In this relation following FD_s exist :

S_Name, Subject → Grade

Stud_ID, Subject → Grade

S_Name → Stud_ID

Stud_ID → S_Name

Moreover, in this relation two candidate keys (S_Name, Subject) and (Stud_ID, Subject) exist, which are composite keys and contain a common attribute subject. This relation is in 3NF, however a lot of data repetition is there in terms of S-Name and Stud_ID. So for this relation to be in BCNF, we will have to do the decomposition. The rule for decomposition for a relation R, which is not in BCNF, is as follows :

Let $x \subseteq R$, A be the single attribute in R_1 and $x \rightarrow A$ be a FD that causes a violation of BCNF. Decompose R into R – A and X_A.

So for above relation to be in BCNF, the decomposition will be as follows :

Stud_ID	S_Name
1001	Axay
1002	Sparsh

Third Normal Form (3NF) : There are schemas where a BCNF decomposition cannot be dependency preserving. For such schemas, we have two alternatives if we wish to check if an update violates any functional dependencies :

- Pay the extra cost of computing joins to test for violations.
- Use an alternative decomposition, third normal form (3NF), which we present below, which makes testing of updates cheaper, unlike BCNF, 3NF decompositions may contain some redundancy in the decomposed schema.

Definition : BCNF requires that all nontrival dependencies be of the form $\alpha \rightarrow \beta$, where α is a superkey. 3NF relates this constraint slightly by allowing nontrivial functional dependencies whose left side is not a superkey.

A relation schema R is in third normal form (3NF) with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$ at least one of the following holds:

- $\alpha \rightarrow \beta$ is a trivial functional dependency.
- α is a separately for R.
- Each attribute A in $\beta \rightarrow \alpha$ is contained in a candidate key for R.

The two alternatives are the same as the two alternatives in the definition of BCNF. The third alternative of the 3NF definition seems rather unintuitive and it is not obvious why it is useful. It represents, in some sense, a

minimal relaxation of the BCNF conditions that helps to ensure that every schema has a dependency preserving decomposition into 3NF. Its purpose will become more clear.

Third Normal Form :

Following synthesis algorithm, R_1 has a candidate key

$R_1 = ISQ$, $R_2 = SD$, $R_3 = IB$, $R_4 = BO$

decomposition of R into R_1, R_2, R_3 and R_4 is lossless and dependency preserving.

Q.12 *(a) Explain 3rd NF with suitable example.* *[R.T.U. 2017]*

(b) Explain BCNF with suitable example. *[R.T.U. 2017]*

Ans.(a) *Refer to Q.11.*

Ans.(b) *Refer to Q.11.*

Q.13 *Discuss the purpose of BCNF and describe how BCNF different from 3NF. Provide an example to illustrate your answer.* *[R.T.U. 2016, 2012]*

OR

Why BCNF to be considered stricter than 3NF? Explain decomposition of non-BCNF scheme into BCNF scheme. *[R.T.U. 2015]*

OR

Define BCNF. How does it differ from 3 NF? Why it is considered a stronger form of 3 NF? *[R.T.U. Dec. 2013, 2008]*

OR

Why BCNF considered to be stricter than 3NF? How is non BCNF scheme decomposed into BCNF scheme? *[R.T.U. 2011]*

OR

Discuss the purpose of BCNF and describe how BCNF different from 3NF. Provide an example to illustrate your answer.

[R.T.U. 2010; Raj. Univ. 2005, 2003]

Ans.BCNF (Boyce-Code Normal Form) : *Refer to Q.11.*

Third Normal Form (3NF) : *Refer to Q.11.*

Comparison of BCNF and 3NF: Of the two normal forms for relational database schemas, 3NF and BCNF, there are advantages to 3NF in that we know that it is always possible to obtain a 3NF design without sacrificing a lossless-join or dependency preservation. Nevertheless, there are disadvantages to 3NF. If we do not eliminate all transitive relation schema dependencies, we may have to use null values to represent some of the possible meaningful relationship among data items and there is the problem of repetition of information.

The goals of database design with functional dependencies are :

1. BCNF
2. Lossless-join
3. Dependency preservation

Since it is not possible to satisfy all three, we may be forced to choose between BCNF and dependency preservation with 3NF.

BCNF Vs 3NF

BCNF is an extended form of 3NF. If a relation is in BCNF then it must be in 3NF. In BCNF we extend the concept up to all the candidate keys of the relation, which are concatenated and two or more of the candidate share a common attribute.

"To be in BCNF, a table must only have candidate key as determinants".

Boyce - Code Normal Form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter form of 3NF, because every relation in BCNF is also in 3NF. The formal definition of BCNF differs slightly from the definition of 3NF. A relation schema R is in BCNF if whenever a nontrivial functional dependency $x \rightarrow A$ holds in R, then x is a super key of R.

Decomposition of Non-BCNF into BCNF :

One of the desirable normal form that eliminates all the redundancy which can be discovered and based on functional dependency is BCNF.

A relation schema R is in BCNF with aspect to a set F of functional dependency if, all the functional dependencies in f^+ of the form $\alpha \rightarrow \beta$ where

$\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds :

- $\alpha \rightarrow \beta$ is a trivial functional dependency (*i.e.*, $\beta \subseteq \alpha$)
- α is a super key for schema R

Example : Consider an example that is not in BCNF.

bar (–) loan = (Custid, loanno, amount). The functional dependency loan_no $\rightarrow$ amount holds on bar_loan, but loan_ number is not a super key (because a loan may be made to a consortium of many customers).

If we decompose the bar_loan relation into borrower and loan. Then, the borrower schema is in BCNF because no non-trivial functional dependency holds on it. The loan schema has one non-trivial functional dependency

that holds, loan_number → amount, but loan_no is a super key (in this primary key) for loan. Thus, loan is in BCNF.

Example of BCNF : *Refer to Q.11.*

Example : Suppose we have a database for an instrument loan, consisting of the following attributes : B (Broker), O (office of the Broker), I (Investor), S (stock), Q (Quantity of stock owned by an investor) and D (dividend paid by a stock), with the functional dependencies S → D, I → B, I → Q and B → Q.

(i) Find a key for the relation schema R = BOSQID

Ans. Attribute closure

$S^+ = D$

$I^+ = BO$

$(IS)^+ = BOQDI$ S, contains all attributes

∴ (IS) is a key.

(ii) How many key does relation schema R have?

Ans. (IS) is the only key as described above in (i).

(iii) Find a lossless-join decomposition of R into Boyce-Codd Normal Form. Find a decomposition of R into third Normal Form having a lossless-join and preserving dependencies

Ans. BCNF

S → D is non-trivial and S is not a key

R is not in BCNF. Decompose R

$R_1 = (R - \beta) = BOSQI$, $R_2 = (\alpha, \beta) = (S, D)$

$R_1 \cap R_2 \rightarrow R_2$, R_1: I → B is non-trivial. I is not a key, R_1 is not in BCNF, decompose R_1 into $R_3 = (R_1 - \beta) = OSQI$ and $R_4 = (\alpha, \beta) = (I, \beta)$

Lossless decomposition of R into

$R_2 = SD$, $R_3 = OSQI$ and $R_4 = IB$ is in BCNF.

Q.34 ***(a) What is Bad database? Explain with examples insert, update and delete anomalies in database.***

(b) Consider the schema R = (A, B, C, D, E) with a set F of functional dependencies {ED→D, A→BC, E→A, E→B, B→C}. Find canonical cover for F. ***[R.T.U. 2014]***

Ans. (a) Bad Database : The purpose of database design is to arrange the corporate data fields into an organised structure such that it generates set of relationships and stores information without unnecessary redundancy. In fact, the redundancy and database consistency are the most important logical criteria in database design. A bad database design may result into repetitive data and information and an inability to represent desired information. It is, therefore, important to examine the relationships that exist among the data of an entity to refine the database design.

A poorly planned database may make it tougher to enter or update new records or it may allow many mistakes to get in. You also see this difference when you're trying to pull information from the database, which, of course, is the whole reason you're going to the trouble o creating and managing that database in the first place.

Customer Number : 1454834 Customer : W. Coyote General Delivery Falling Rocks, AZ 84211 (599) 555-9345			Terms : Net 30 Ship Via : USPS Order Date : 21/10/2015	
Product No.	**Description**	**Quantity**	**Unit Price**	**Extended Amount**
SPR-2290	Super strength spring	2	24.00	$48.00
STR-67	Foot straps, leather	2	2.50	$5.00
HLM-45	Deluxe Crash Helmet	1	67.80	$67.88
KPR-1	Rocket, solid fuel	1	128, 200.40	$120,200.40
KLT-7	Emergency Location Transmitter	1	79.80	**Free Gift**
Total Amount				$128,321.28

Fig. : Invoice from acme industries

Insert Anomaly : The insert anomaly refers to a situation wherein one,cannot insert a new tuple into a relation because of an artificial dependency on another relation. The error that has caused the anomaly is that attributes of two different entities are mixed into the same relation. Referring to fig., we see that the ID, name, and address of the customer are included in the invoice view. Were we to merely make a relation from this view as it is, and eventually a table from the relation, we would soon discover that we could not insert a new customer into the database unless they had bought something. This is because all the customer data is embedded in the invoice.

Delete Anomaly : The delete anomaly is just the opposite of the insert anomaly. It refers to a situation wherein a deletion of data about one particular entity causes unintended loss of data that characterizes another entity. In the case of the acme industries invoice, if we delete the last invoice that belongs to a particular customer, we lose all the data related to that customer. Again, this is because data from two entities (customers and invoices) would be incorrectly mixed into a single relation if we merely implemented the invoice as a table without applying the normalization process to the relation.

Update Anomaly : The update anomaly refers to a situation where an update of a single data value requires multiple tuples (rows) of data to be updated. In our invoice example, if we wanted to change the customer's address,

we would have to change it on every single invoice for the customer. This is because the customer address would be redundantly stored in every invoice for the customer. To make matters worse, redundant data provides the golden opportunity to update many copies of the data, but miss a few of them, which results in inconsistent data. The mantra of the skilled database designer is, for each attribute, capture it once, store it once, and use that one copy everywhere.

Ans. (b) First combine $E \rightarrow A$ and $E \rightarrow B$ into $E \rightarrow AB$.

Set is now $\{ED \rightarrow D, A \rightarrow BC, E \rightarrow AB, B \rightarrow C\}$

C is extraneous in $A \rightarrow BC$

Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ an the other dependencies.

Yes : using transitivity on $A \rightarrow B$ and $B \rightarrow C$.

Set is now $\{A \rightarrow B, B \rightarrow C, ED \rightarrow D, E \rightarrow AB\}$

B is extraneous in $E \rightarrow AB$

Check if $E \rightarrow B$ is logically implied by $E \rightarrow A$ an other dependencies.

Yes : using transitivity on $E \rightarrow A$ and $A \rightarrow B$.

Set is now $\{A \rightarrow B, B \rightarrow C, E \rightarrow A, ED \rightarrow D\}$

Thus, the canonical cover is

$\{A \rightarrow B, B \rightarrow C, E \rightarrow A, ED \rightarrow D\}$

❑❑

TRANSACTION PROCESSING 4

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 *What is the need of serializability in transaction processing?* *[R.T.U. 2019]*

OR

What is serializability?

Ans. Serializability is the classical concurrency scheme. It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order.

Q.2 *What is concurrency?* *[R.T.U. 2019]*

OR

What is a concurrent execution of transaction?

Ans. Concurrent execution of transaction means multiple transactions execute/run concurrently in RDBMS with each transaction doing its atomic unit of work for the operations encapsulated in the particular transaction.

Q.3 *What is cascadeless schedule?*

Ans. A cascadeless schedule (also known as recoverable schedule) is one where, for each pair of transactions Ti and Tj such that Tj reads a data item previously written by Ti, the commit operation of Ti appears before the read operation of Tj.

Q.4 *What is graphical form of recoverable schedule?*

Ans.

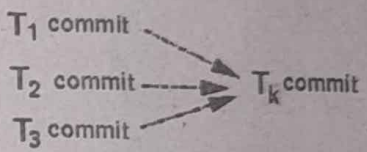
If:

T_1 $w_1(A)$ ------► $r_k(A)$

T_2 $w_2(B)$ ------► $r_k(B)$ T_k

T_3 $w_3(C)$ ------► $r_k(C)$

Then:



The transaction T_k ***must*** **commit** ***only after*** the transactions T_1, T_2 and T_3 have **committed.**

Q.5 *What is use of serialization graph?*

Ans. Serialization graph is used to test the serializability of a schedule.

PART-B

Q.6 *Write a short note on transaction properties and recoverable schedules.* *[R.T.U. 2019]*

Ans. Transaction Properties : The four basic properties of a Transaction Processing System are referred to as the ACID properties. Atomicity, Consistency, Isolation and Durability. These properties are used to test that a transaction is never unfinished, the data held in the system

is always consistent, concurrent transactions are independent and the effects of a transaction continue after the transaction is completed.

Atomicity : The transaction must happen completely or it should be undone completely. If a transaction fails, the effects of all operations that make up the transaction need to be cancelled and the data needs to be returned to its original previous state. If we look at the first part of the word "Atomicity", atom, the term suggests its meaning at the time the concept was first used. An atom is considered the smallest item of matter that can exist.

An example of how a transaction processing system handles atomicity is a student making a purchase from the school canteen. Money changes hands and the student later realizes that the drink is past its use by date. As there are no other drinks available, the student returns the drink and the canteen refunds the purchase price. This transaction is not recorded and so, no transaction occurred.

Consistency : For a successful transaction to take place, all parties must agree on the facts of the exchange. When a successful transaction is completed, the data should be in a consistent state. This means that all data should be able to be accounted for and that each step of the transaction is carried out in the same way each time. This ensures that data is correct for each part of the transaction.

An example of this property would see the sale of two drinks in the canteen should increase the takings by the canteen and decrease the stock held in the canteen by two drinks. The steps are to exchange the drinks for money. The two should balance so that the data is consistent.

Durability : The effects of a completed transaction should always be durable. In a large organization, transactions need to be recorded so that they can be checked at any time in the future. Backups need to be kept of this data. This is also part of an organizations responsibility to pay tax and meet other obligations.

Isolation : Transaction must be independent of each other. This doesn't actually happen. The user just believes that it does. Isolation involves the appearance of treating each transaction separately and keeping the data for each transaction separate.

The aim of isolation is to prevent the same data being treated twice. For example, an organization selling tickets to rock concerts such as Ticketed need to e they do not sell the same seat to more than one per There may be many outlets selling tickets as well a online booking website. Customers who access the on booking site need to be sure that their transaction separate. It is vital that no one else can book the s seat through another agent. Isolation is a bigger issu the Transaction Processing System becomes larger a becomes difficult to make it appear as if each transact is being handled sequentially.

Recoverable Schedule : *Refer to Q.3 and Q.4.*

Q.7 ***What is the states of transactions? Expla briefly.***

Ans. A transaction goes through many different stat throughout its life cycle. These states are called a transaction states. Transaction states are as follows :

(i) Active state
(ii) Partially committed state
(iii) Committed state
(iv) Failed state
(v) Aborted state
(vi) Terminated state

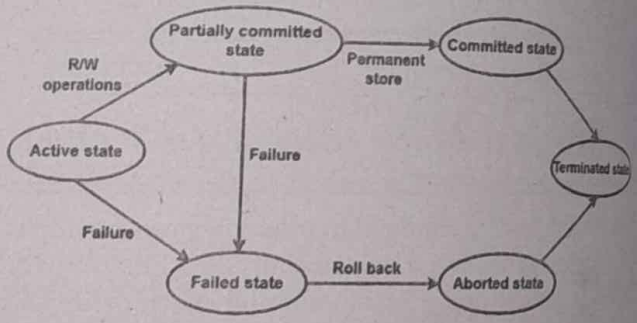


Fig. Transaction States in DBMS

(i) Active State : This is the first state in the life cycle of a transaction. A transaction is called in an active state as long as its instructions are getting executed. All the changes made by the transaction now are stored in the buffer in main memory.

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

ii) Committed State: After all the changes made by e transaction have been successfully stored into the atabase, it enters into a committed state. Now, the ansaction is considered to be fully committed.

ote : After a transaction has entered the committed ate, it is not possible to roll back the transaction. In other ords, it is not possible to undo the changes that have een made by the transaction. This is because the system updated into a new consistent state. The only way to ndo the changes is by carrying out another transaction alled as compensating transaction that performs the verse operations.

v) Failed State : When a transaction is getting executed the active state or partially committed state and some ilure occurs due to which it becomes impossible to ntinue the execution, it enters into a failed state.

) Aborted State : After the transaction has failed and tered into a failed state, all the changes made by it have be undone. To undo the changes made by the transaction, becomes necessary to roll back the transaction. After e transaction has rolled back completely, it enters into aborted state.

i) Terminated State : This is the last state in the life cle of a transaction. After entering the committed state aborted state, the transaction finally enters into a rminated state where its life cycle finally comes to an d.

8 *What is the properties of transaction processing?*

ns. *Refer to Q.6.*

PART-C

9 *Explain conflict v/s view serializability in detail.*

[R.T.U. 2019]

ns. Conflict Serializability : Instructions Ii and Ij, of ansactions Ti and Tj respectively, conflict if and only if ere exists some item P accessed by both Ii and Ij, and least one of these instructions wrote P.

Consider the below operations :

i. Ii = read(P), Ij = read(P). Ii and Ij don't conflict.

ii. Ii = read(P), Ij = write(P). They conflict.

iii. Ii = write(P), Ij = read(P). They conflict.

iv. Ii = write(P), Ij = write(P). They conflict.

A conflict between Ii and Ij forces a temporal order between them.

If Ii and Ij are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.

If a schedule Scan be transformed in to a schedule S by a series of swaps of non-conflicting instructions, then S and S' are conflict equivalent.

In other words a schedule Sis conflict serializable if it is conflict equivalent to a serial schedule.

Example of a schedule that is not conflict serializable:

T3	T4
Read(P)	
	Write(P)
Write(P)	

The instructions cannot be swapped in the above schedule to obtain either the serial schedule < T 3 , T 4 >, or the serial schedule < T 4 ,T 3 >.

A serial schedule T2 follows T1, by a series of swaps of non-conflicting instructions making the below Schedule confict serializable.

T1	T2
Read(X)	
Write(X)	
	Read(X)
	Write(X)
Read(Y)	
Write(Y)	
	Read(Y)
	Write(Y)

View Serializability : S and S' are view equivalent if the following three conditions are met:

i. For each data item P, if transaction Ti reads the initial value of P in schedule S, then transaction Ti must, in schedule S', also read the initial value of P.

ii. For each data item P, if transaction Ti executes read (P)in schedule S, and that value was produced by transaction Tj, then transaction Ti must in

schedule S' also read the value of P that was produced by transaction Tj.

iii. For each data item P, the transaction that performs the final write(P) operation in schedule S must perform the final write(P) operation in schedule S'.

View equivalence is also based purely on reads and writes alone.

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

Every conflict serializable schedule is also view serializable.

Every view serializable schedule which is not conflict serializable has blind writes.

T3	T4	T6
Read(P)		
	Write(P)	
Write(P)		
		Write(P)

Q.10 ***What is cascadeless schedule? Why is cascadeless ness of schedules desirable? Are there any circumstances under which it would be desirable to allow non-cascadeless schedules? Explain and justify your answer.***

[R.T.U. 2019]

Ans. Cascadeless schedule : *Refer to Q.3.*

Recoverability : A recoverable schedule is one where, for each pair of Transaction T_i and T_j such that T_j reads data item previously written by T_i the commit operation of T_i appears before the commit operation T_j.

T8	T9	
read(A)		T9 is dependent on T8
write(A)		Non recoverable schedule if T9 commits before T8
	read(A)	
read(B)		

Suppose that the system allows T9 to commit immediately after execution of read(A) instruction. Thus T9 commit befofe T8 does.

Now suppose that T8 fails before it commits. Since T9 has read the value of data item A written by T8 we must abort T9 to ensure transaction Atomicity.

However, T9 has already committed and cannot be aborted. Thus we have a situation where it is impossible to recover correctly from the failure of T8.

Cascadeless schedules

T10	T11	T12
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

Transaction T10 writes a value of A that is read by Transaction T11. Transaction T11 writes a value of A that is read by Transaction T12. Suppose at this point T10 fails. T10 must be rolled back, since T11 is dependent on T10, T11 must be rolled back, T12 is dependent on T11, T12 must be rolled back.

This phenomenon, in which a single transaction failure leads to a series of transaction rollbacks is called Cascading rollback.

- Cascading rollback is undesirable, since it leads to the undoing of a significant amount of work.
- It is desirable to restrict the schedules to those where cascading rollbacks cannot occur, Such schedules are called Cascadeless Schedules.
- Formally, a cascadeless schedule is one where for each pair of transaction T_i and T_j such that T_j reads data item, previously written by T_i the commit operation of T_i appears before the read operation of T_j.

Every Cascadeless schedule is also recoverable schedule.

Q.11 ***Describe conflict serializability and view serializability with examples?***

Ans. *Refer to Q.9.*

□□□

CONCURRENCY CONTROL 5

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 *How many kinds of locks present in lock-based protocols?*

Ans. **Locks are of two types :**

Binary Locks : A lock on a data item can be in two states; it is either locked or unlocked.

Shared/Exclusive : This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

Q.2 *What is the definition of timestamp-based protocols?*

Ans. The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp. Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one. In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last 'read and write' operation was performed on the data item.

Q.3 *Write down the three phase present in the validation based protocol?*

Ans. In the validation based protocol, the transaction is executed in the following three phases:

(i) Read phase : In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.

(ii) Validation phase : In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.

(iii) Write phase : If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Q.4 *What is deadlock handling?*

Ans. Deadlock is a state of a database system having two or more transactions, when each transaction is waiting for a data item that is being locked by some other

transaction. A deadlock can be indicated by a cycle in the wait-for-graph. This is a directed graph in which the vertices denote transactions and the edges denote waits for data items.

PART-B

Q.5 *What is shadow paging? Explain in detail.*

[R.T.U. 2019]

Ans. In computer science, shadow paging is a technique for providing atomicity and durability (two of the ACID properties) in database systems. A page in this context refers to a unit of physical storage (probably on a hard disk), typically of the order of 1 to 64 KB. Shadow paging is a copy-on-write technique for avoiding in-place updates of pages. Instead, when a page is to be modified, a shadow page is allocated. Since the shadow page has no references (from other pages on disk), it can be modified liberally, without concern for consistency constraints, etc. When the page is ready to become durable, all pages that referred to the original are updated to refer to the new replacement page instead. Because the page is "activated" only when it is ready, it is atomic.

If the referring pages must also be updated via shadow paging, this procedure may recur many times, becoming quite costly. One solution, employed by the WAFL file system (Write Anywhere File Layout) is to be lazy about making pages durable (i.e. write-behind caching). This increases performance significantly by avoiding many writes on hotspots high up in the referential hierarchy (e.g.: a file system superblock) at the cost of high commit latency.

Write-ahead logging is a more popular solution that uses in-place updates. Shadow paging is similar to the old master–new master batch processing technique used in mainframe database systems. In these systems, the output of each batch run (possibly a day's work) was written to two separate disks or other form of storage medium. One was kept for backup, and the other was used as the starting point for the next day's work. Shadow paging is also similar to purely functional data structures, in that in-place updates are avoided.

Q.6 *How many types are present in database failure?*

Ans. There are many types of failures that can affect database processing. Some failures affect the memory only, while others involve secondary storage. Following are the types of failure:

Hardware Failures: Hardware failures may include memory errors, disk crashes, bad disk sectors, disk errors and so on. Hardware failures can also be attributed to design errors, inadequate (poor) quality control during fabrication, overloading (use of under-capacity components) and wear out of mechanical parts.

Software Failures: Software failures may include failures related to software's such as, operating system, DBMS software, application programs and so on.

It's the one call no database administrator wants to receive: There's been a database failure. Wherever they are, whatever they are doing, they must drop everything and head to the office ASAP. The company's whole livelihood relies on their ability to get the data back and get the systems up and running as quickly as possible. And while everyone dreams of the day they can be their company's savior, in the case of database failures, a true hero is the one that never receives that dreaded call.

There are several types of database failure. While there's always a risk, with some time and care, you can protect your company from some of the most common types of database failure.

Three Types of Database Failures

1. System Crash: Mayday indeed. When the system crashes, it's a race towards reinstating affected processes or – worse yet – data recovery.

A system crash usually refers to any kind of bug or hardware malfunction in the operating system or the database software. It can bring the processing of transaction to a halt and can even cause the loss of content residing on volatile storage such as main memory, cache memory, RAM, etc.

There are many reasons why database systems might crash. Maintaining strict security and maintenance routines and protocols in the database, its host system and the network should guarantee minimal downtime. In the event something did happen, the DBA must have a

documented crisis plan to restore and reinstate the database as quickly as possible.

2. Media Failure : This one is very risky. Media failures are caused by a head crash or unreadable media. These types of data failures are considered one of the most serious because it is possible for entire data loss. Media failures usually leave database systems unavailable for several hours until recovery is complete, especially in applications with large devices and high transaction volume.

The best way to prevent this type of database failure is to protect your data with adequate malware protection and backing up your data frequently.

3. Application Software Errors : When the resource limit is exceeded, bad input, logical or internal errors occur, or any other factors related to the application software is compromised, transactions can fail giving way to database failure.

It is generally recommended that application software errors are minimized in the software code during conception and the software engineering process. It is better for developers to put mechanisms and controls into place during the design of the architecture and coding operation, than trying to remedy mistakes later. Asides from failures, malicious code may exploit known vulnerabilities, especially those associated with a particular programming software tool or known human software coding error.

Q.7 *What is the technique of log-based recovery? Explain briefly in detail?*

Ans. Atomicity property of DBMS states that either all the operations of transactions must be performed or none. The modifications done by an aborted transaction should not be visible to database and the modifications done by committed transaction should be visible.

To achieve our goal of atomicity, user must first output to stable storage information describing the modifications, without modifying the database itself. This information can help us ensure that all modifications performed by committed transactions are reflected in the database. This information can also help us ensure that no modifications made by an aborted transaction persist in the database.

Log and log records : The log is a sequence of log records, recording all the update activities in the database. In a stable storage, logs for each transaction are maintained. Any operation which is performed on the database is recorded is on the log. Prior to performing any modification to database, an update log record is created to reflect that modification.

An update log record represented as: <Ti, Xj, V1, V2> has these fields:

Transaction identifier: Unique Identifier of the transaction that performed the write operation.

- **Data item :** Unique identifier of the data item written.
- **Old value :** Value of data item prior to write.
- **New value :** Value of data item after write operation.

Other type of log records is:

<Ti start> : It contains information about when a transaction Ti starts.

<Ti commit> : It contains information about when a transaction Ti commits.

<Ti abort> : It contains information about when a transaction Ti aborts.

Undo and Redo Operations : Because all database modifications must be preceded by creation of log record, the system has available both the old value prior to modification of data item and new value that is to be written for data item. This allows system to perform redo and undo operations as appropriate:

Undo: using a log record sets the data item specified in log record to old value.

Redo: using a log record sets the data item specified in log record to new value.

The database can be modified using two approaches :

(i) Deferred Modification Technique : If the transaction does not modify the database until it has partially committed, it is said to use deferred modification technique.

(ii) Immediate Modification Technique : If database modification occurs while transaction is still active, it is said to use immediate modification technique.

Recovery using Log records : After a system crash has occurred, the system consults the log to determine which transactions need to be redone and which need to be undone.

Transaction Ti needs to be undone if the log contains the record <Ti start> but does not contain either the record <Ti commit> or the record <Ti abort>.

Transaction Ti needs to be redone if log contains record <Ti start> and either the record <Ti commit> or the record <Ti abort>.

Use of Checkpoints : When a system crash occurs, user must consult the log. In principle, that needs to search the entire log to determine this information. There are two major difficulties with this approach:

The search process is time-consuming.

Most of the transactions that, according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will cause recovery to take longer.

To reduce these types of overhead, user introduce checkpoints. A log record of the form <checkpoint L> is used to represent a checkpoint in log where L is a list of transactions active at the time of the checkpoint. When a checkpoint log record is added to log all the transactions that have committed before this checkpoint have <T commit> log record before the checkpoint record. Any database modifications made by Ti are written to the database either prior to the checkpoint or as part of the checkpoint itself. Thus, at recovery time, there is no need to perform a redo operation on Ti.

After a system crash has occurred, the system examines the log to find the last <checkpoint L> record. The redo or undo operations need to be applied only to transactions in L, and to all transactions that started execution after the record was written to the log. Let us denote this set of transactions as T. Same rules of undo and redo are applicable on T as mentioned in Recovery using Log records part.

Note that user need to only examine the part of the log starting with the last checkpoint log record to find the set of transactions T, and to find out whether a commit or abort record occurs in the log for each transaction in T. For example, consider the set of transactions {T0, T1, . . ., T100}. Suppose that the most recent checkpoint took place during the execution of transaction T67 and T69 while T68 and all transactions with subscripts lower than 67 completed before the checkpoint. Thus, only transactions T67, T69, . . ., T100 need to be considered during the recovery scheme. Each of them needs to be redone if it has completed (that is, either committed or aborted); otherwise, it was incomplete, and needs to be undone.

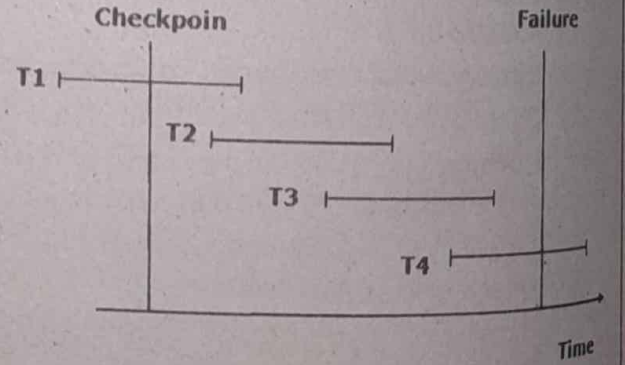
PART-C

Q.8 ***Why must lock and unlock be atomic operations? Explain recovery related data structure in detail. Also explain deadlock handling.***

[R.T.U. 20

Ans. Lock and unlock must be atomic operations because otherwise it may be possible for two transactions to obtain an exclusive lock on the same object, thereby destroying the principles of 2PL. A lock is held over a long duration and a latch is released immediately after the physical read or writes operation is completed. Convoy is a queue of waiting transactions. It occurs when a transaction holding a heavily-used lock is suspended by the operating system and every other transactions that needs this lock is queued.

Recovery using Checkpoint : In the following manner a recovery system recovers the database from this failure:



The recovery system reads log files from the end to start. It reads log files from T4 to T1.

Recovery system maintains two lists, a redo-list and an undo-list.

The transaction is put into redo state if the recovery system sees a log with <Tn, Start> and <Tn, Commit> or

just <Tn, Commit>. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.

For example: In the log file, transaction T2 and T3 will have <Tn, Start> and <Tn, Commit>. The T1 transaction will have only <Tn, commit> in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.

The transaction is put into undo state if the recovery system sees a log with <Tn, Start> but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.

For example: Transaction T4 will have <Tn, Start>. So T4 will be put into undo list since this transaction is not yet complete and failed amid.

Deadlock Handling : *Refer to Q.4.*

Q.9 ***Why use concurrent transaction with recovery process? Explain recovery using with checkpoint?***

Ans. Whenever more than one transaction is being executed, then the interleaved of logs occur. During recovery, it would become difficult for the recovery system to backtrack all logs and then start recovering. To ease this situation, 'checkpoint' concept is used by most DBMS. As we have discussed checkpoint in Transaction Processing Concept of this tutorial, so you can go through the concepts again to make things more clear.

Checkpoint : The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk. The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log files will be created.

When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on.

The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed.

Recovery using Checkpoint : *Refer to Q.8.*

Q.10 ***Defined the whole concurrency control types and examples?***

Ans. In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories.

- Lock based protocols
- Time stamp based protocols

Lock-based Protocols: Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds.

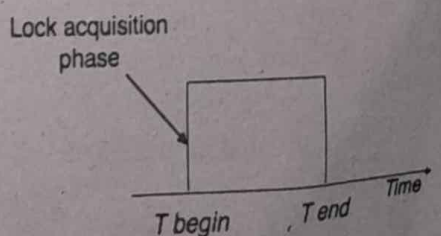
Binary Locks : A lock on a data item can be in two states; it is either locked or unlocked.

Shared/exclusive : This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

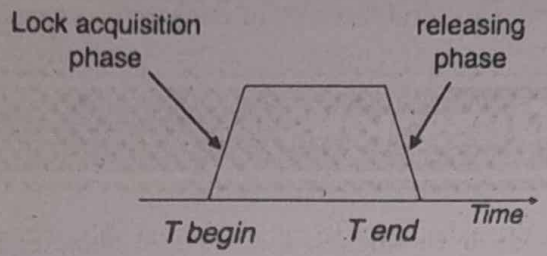
There are four types of lock protocols :

Simplistic Lock Protocol: Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the 'write' operation.

Pre-claiming Lock Protocol: Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.

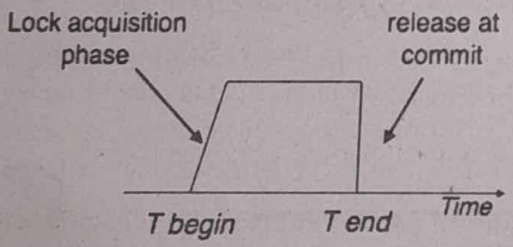


Two-Phase Locking 2PL: This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.



Two-phase locking has two phases, one is growing, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released. To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

Strict Two-Phase Locking: The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.



Strict-2PL does not have cascading abort as 2PL does.

Timestamp-based Protocols: The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one. In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last 'read and write' operation was performed on the data item.

Timestamp Ordering Protocol : The timestamp ordering protocol ensures serializability among transactions in their conflicting read and writes operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

The timestamp of transaction Ti is denoted as TS(Ti).

Read time-stamp of data-item X is denoted by R-timestamp(X).

Write time-stamp of data-item X is denoted by W-timestamp(X).

Timestamp ordering protocol works as follows :

If a transaction Ti issues a read(X) operation :

If TS(Ti) < W-timestamp(X)

Operation rejected.

If TS(Ti) >= W-timestamp(X)

Operation executed.

All data-item timestamps updated.

If a transaction Ti issues a write(X) operation :

If TS(Ti) < R-timestamp(X)

Operation rejected.

If TS(Ti) < W-timestamp(X)

Operation rejected and Ti rolled back.

Otherwise, operation executed.

Thomas' Write Rule : This rule states if TS(Ti) < W-timestamp(X), then the operation is rejected and Ti is rolled back.

Time-stamp ordering rules can be modified to make the schedule view serializable.

Instead of making Ti rolled back, the 'write' operation itself is ignored.

❑❑❑