INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR

B. TECH 5TH SEMESTER EXAMINATION, DECEMBER 2021

Database Management Systems [CS 3102]

Name: Abhiroop Mukherjee

Examination Roll No. : 510519109

G-Suite ID : 510519109.abhirup@students.iiests.ac.in

1, 2, ~~3~~, 4, ~~5~~

Q1) a) **Physical Data Independence.**

→ The capacity of a DBMS to allow change of internal schema without affecting conceptual or ~~external~~ external schema is called the Physical Data Independence.

Eg: consider we change HDD of a computer running a DBMS to a bigger size with all previous data, this should not affect any schemas or views in ~~DBMS~~ DBMS.

b) **Cardinality**

→ cardinality ~~with~~ with respect to DBMS refers to the uniqueness of data value contained in the column

High Cardinality → most of the data in a column are unique

low cardinality → there are a lot of duplicate data in a column

c) Primary Key

→ Before going into primary key, let's define what a key is

→ If $R(A_1, A_2, \ldots A_n)$ is a relation with functional dependencies F & X is a subset of R, we say X is a key if

i) $X \rightarrow A_1, A_2 \ldots A_n$ is implied in F

ii) R is not functionally dependant on any proper subset of X

→ There can exist multiple keys for a relation. The set of those keys are called candidate key

→ The key chosen by Database designed from the set of candidate keys is called the primary key.

d) 3NF

A relation scheme R is in 3NF, if whenever $X \rightarrow A$ holds in R & A is not in X, then either X is superkey of R or A is a prime attribute.

i.e 3NF is 2NF + constraint (no non prime → non prime)

e) Lock compatibility matrix

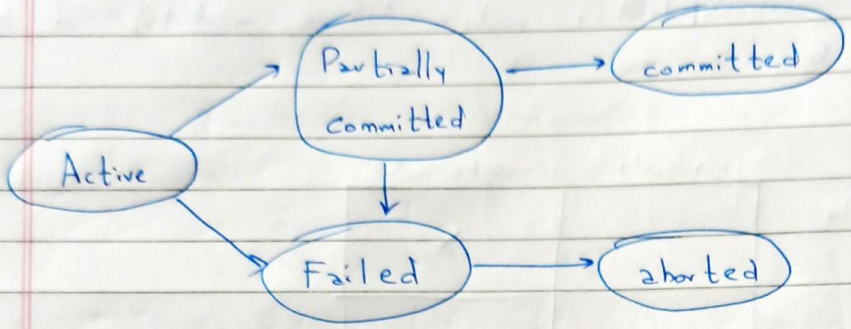
→ Lock compatibility matrix defines what type of locks can be made over each other.

	read only	write only
read-only		X
write only	X	X

→ This matrix means that ~~from a transaction~~ if a data is read-lock by a Transaction T_i, some other transcation T_j can also put a read lock on it (but not write lock)

→ but if data is write locked by T_i, By T_j cant put read nor write lock on it.

Q3) a) State Transition Diagram of a Transaction



→ Partially committed state of a transaction means that the execution of transaction is done ~~and if all, goes DBMS~~

→ If all DBMS checks is passed by the transaction, then the transaction goes into committed state. Changes made by the transaction is now permanent

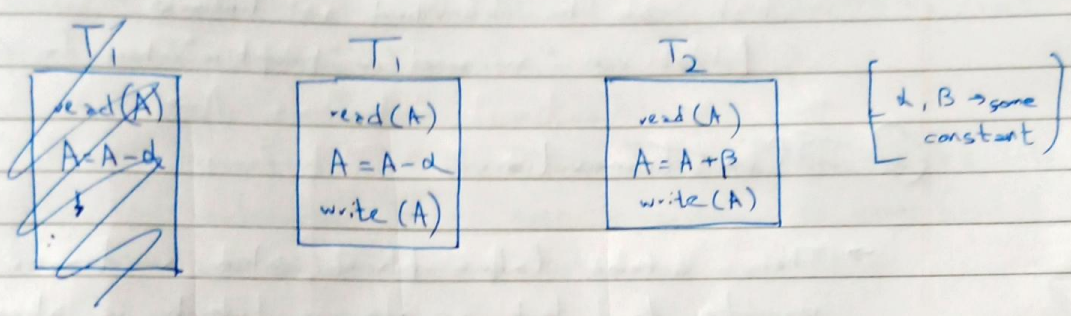
→ If for some reason the DBMS checks fail, the ~~transac~~ transaction goes int Failed State, all it's changes are rolled back via recovery manger & then the transaction is aborted

→ User of DBMS can not redo the transaction after necessary changes so that update to DBMS ~~can be~~ happens.

(a)

Q3) b) **Lost Update Problem**

consider two transactions



→ when T_1 & T_2 happens parallely, this could happen
[assume initially A = 1000]

T_1	T_2	DBMS data.
read(A) [A=1000]		
	read(A) [A=1000]	A = 1000
[A=1000-α] A = A - α		
	A = A + β [A=1000+β]	
write(A)		A = 1000 - α
	write(A)	A = 1000 + β

↓
end result ⇒ A = 1000 + β

→ here we see that ~~about~~ the observed result is different from what was expected, update of T_1 is lost

→ Here ~~consist~~ consistence of a Database is compromised, loss of ~~da~~ update immidiately puts a ~~data~~base to an inconsistent state. [C of ACID]

Q3) c) Wait - Die Scheme

→ Wait - Die scheme a method to solve deadlock which can be described as follows
[didn't use "avoidance" nor "prevention" here for Q3d]

i) ~~If a transaction~~

→ Theis scheme takes place when a transaction requests to lock a resource which is already held by some other transaction.

→ If that case arrives, we have two possibility

i) If T_i, which is requesting a conflicting lock is older than T_j, i.e $TS(T_i) < TS(T_j)$.

→ Then T_i waits for T_j to ~~close~~ release it's ~~this~~ lock then locks the resource for itself.

ii) If $TS(T_i) > TS(T_j)$ [T_i younger than T_j], then T_i is killed and restarted again sometime with same ~~Time~~ time stamp as before.

TLDR: older transaction ~~##~~ waits, but newer transaction is killed.

d) → Wait - Die Scheme is a deadlock prevention scheme.

→ This is due to the fact that this schemes does it work after the transaction is issued & it either kills / wait an already running Transaction.

→ Deadlock prevention, on the other hand actively looks into transaction instruction before even executing to find if any deadlock could occur or not

Eg: Wait-for graph creates a "dependency graph" and actively tries to avoid loops in dependency graph to avoid deadlocks.

Q5) a) Multivalued Dependency

(i) → for a dependency $A \to B$, if for a single value $a \in A$, if there exist multiple values of $b \in B$, then the functional dependency $A \to B$ is said to be a multivalued dependency

(ii) → For a multivalue dependency to exist, we must need at least columns to exist in a relation

Why? → one column relation can't have functional dependency

→ two column relation can have functional dependency but can't have multivalue dependency

→ If we try to add multivalue dependency properties of functional dependency will get violated.

Eg consider relation

A	B
a	1
b	2
c	3
d	4

→ this has an fd $A \to B$, meaning that if

$t_i(A) = t_j(A)$, implies $t_i(B) = t_j(B)$

→ if we try to ~~insert~~ make multivalue dependency by adding (2,4), it will violate the definition of functional definition

(iii) for a three attribute relation R(A,B,C), if there is a multivalue dependency between A & B and ~~A~~ A & C, then B & C should be independent of each other.

→ A relation R having a multivalued dependency can be in BCNF form, but it cannot be in 4NF form

Q5) b) Undo

→ undo'ing a Transaction means that we restore all the data items edited by the transaction to it's previous value before the start of execution

Eg consider a transaction

T
read (A)
A = A - 50
write (A)
read (B)
B = B - 100
write (B)

and consider ~~init~~ before execution of T, A = 1000, B = 500

So

T	Data
read (A) [A = 1000]	A = 1000, B = 500
A = A - 50 [A = 950]	
write (A)	A = 950, B = 500
read (B) [B = 500]	
B = B - 100 [B = 400]	
write (B)	A = 950, B = 400

→ now after execution of T, if we undo T, we will have to revert A to 1000 & B to 500.

Redo

→ redo'ing a transaction means setting the value of all data item updated by transaction to the new values.

Eg: consider the same transaction and same data A & B

→ redo'ing T means we set A = 950 & B = 400

→ Redo operation is idem potent, this means that if we redo multiple times, it will have same effect as redoing T once, i.e A = 950 & B = 400 will be constant even after multiple redos.

→ Redo & Undo operation are used in Log based Crash Recovery to maintain the ACID [consistent & durable] property of a Database.