Lossless Decomposition = Non-additive decomposition
↳ no new ~~tt~~ tuple is added & row is removed.

Page No.
Date :

$R_1 \Rightarrow$ Candidate Key table
↳ complete candidate key derives some other is in R_1.

2NF
↓
Removing Partial Dependency

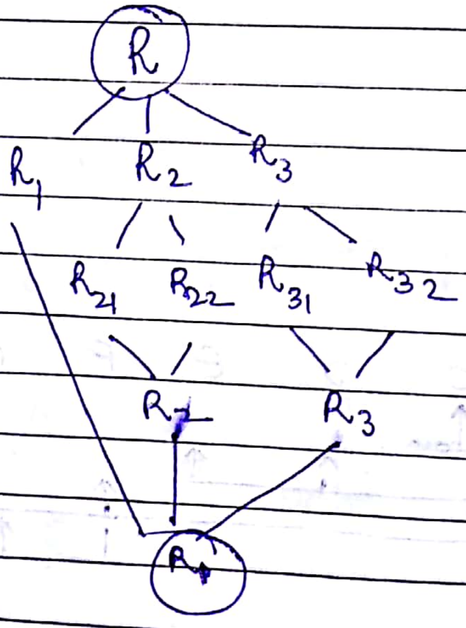
R
→ R_1 ($\underline{A}$, $\underline{B}$, C)
→ R_2 ($\underline{A}$, D, E, I, J) → as D identifies I, J is therefore in new relation.
~~→ R_3 (B, F)~~
~~→ R_4 (B, [illegible])~~
→ R_3 ($\underline{B}$, F, G, H)

~~A → DE~~
~~D → IJ~~

3NF

$R_1 \rightarrow R_1$ ($\underline{A}$, $\underline{B}$, C)
$R_2 \rightarrow R_{21}$ ($\underline{A}$, D, E)
$\quad\rightarrow R_{22}$ ($\underline{D}$, I, J)
$R_3 \rightarrow R_{31}$ ($\underline{B}$, F)
$\quad\rightarrow R_{32}$ ($\underline{F}$, G, H)

} Is also in BCNF.



From top to bottom, we have R are same
⇒ It is lossless decomposition

Instruction - Procedure to complete a task.
↳ direc" to carry out a ~~procedure~~
- → 0 address → accumulator → system register if no register availas
- → 1 address → 1 register
- → 2 " → 2 register

TRANSACTION MANAGEMENT :-

Opcode (operation code)
↳ arithmetic operations.

~~Transaction Management~~

+	A	B	C
opcode	Add1	Add2	Add3

→ 3 address instruction

in register

eg: C = A + B

For 1 address instruction,

opcode

Accumulator → sort of buffer but ~~not~~ a register of a system

Phases in Instruction
- → Fetch (Get value of variables)
- → Decode (what type value is)
- → Execution

Transaction is a task which is a logical unit of work.
(→ a set of instructions)

A complete task is a transaction

eg: A → B
10Rs

1 transaction. (complete task, can't be partial) ↓ always 100% executed can't be <100.

- R(A) → Read
- A = A - 10
- W(A) → Write
- R(B)
- B = B + 10
- W(B)

→ Failure according to client but according to programmer something done (84.3%)

→ If fail here then not updated

→ Atomicity → No partial possible but 100% unit only possible

but instructions can be <100% executed

4 Properties of Transaction :-

A C I D

Transaction Management → A → Atomicity

C → Consistency ↳ A + I + D

I → ~~Isolation~~ Isolation

D → Durability

If a transaction holds A, I, D, then it would be consistent and DB are always consistent but transaction decides consistency

Isolation → but physically also → at one time only 1 work
→ Physical (Its process affect other transaction)
→ Logical (Result does not affect any other transaction)

Durability
↳ If a change in DB is made, then that change must remain in the DB to be accessed forever.
↳ eg: data in pendrive.

Concurrency Management ; if multiple transactions are being carried out simultaneously and it manages and isolates each & every transaction from each other.
Isolation

Recovery Management → manages to recover the previous state.
Durability

Process	Program
→ Set of instruc^n	→ Set of Processes
→ Active	→ Passive

↳ as a program may have some process active other non-active.

~~STAGES OF TRANSACTIONS~~

A transaction is a small unit of work of a program and it may contain several low level tasks. A transaction in a database system must maintain atomicity, consistency, isolation and durability. These are commonly known as ACID Properties.

Atomicity : This property states that a transaction must be treated as an atomic unit i.e. either all of its operations are executed or none. There must be no state in a DB where a transaction is left partially completed.

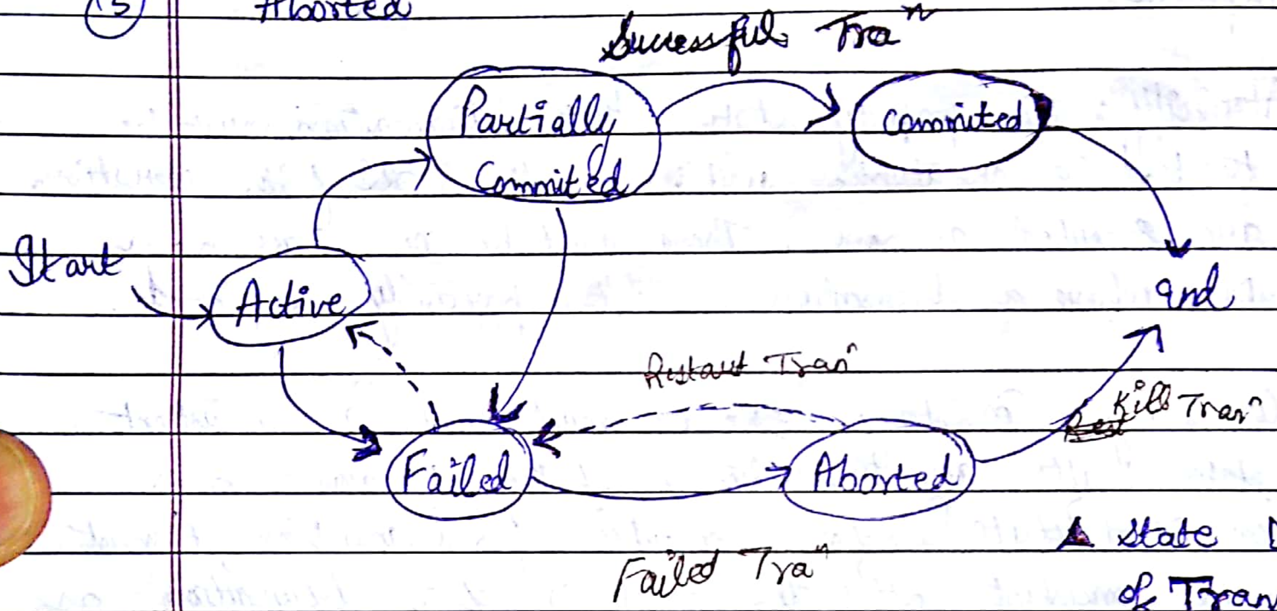
Consistency : Consistency ~~the~~ DB must remain in a consistent state after any transaction. If the DB was in a consistent state before execution of a transaction, it must remain consistent after the execution of a transaction as well.

Isolation : In a DB, where more than 1 transaction are being executed simultaneously and in parallel. The property of isolation means that, all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of other transactions.

Durability : The DB durable to ~~the~~ hold all its latest updates even if the system fails or restarts. If a transaction update a data ~~or~~ in a DB and commits then the DB will hold the modified data.

STATE OF A TRANSACTION :-

1. Active
2. Partially Commited
3. Commited
4. Failed
5. Aborted



A State Diagram of Transaction

SCHEDULE :-

Serializability : multiple transactions running in a multiple - programming environment.

Schedule : set of instructions running in a chronological order.

Serial schedule : Whichever transaction comes first is executed first.

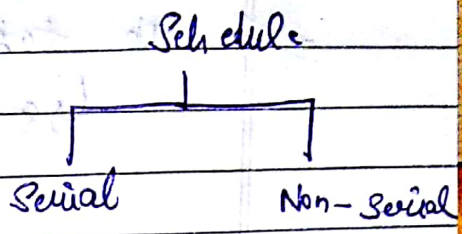
Page No.
Date :

T

S1	S2	S3
R(A)	R(A)	
W(A)	W(A)	
	R(B)	
R(B)	W(B)	
W(B)		

leave for some other process → Non-Serial Schedule

→ Serial Schedule



When multiple transactions try to write same memory, then inconsistency and schedule mangement

Serializability :-

When multiple transactions are beig executed by the OS in a multi-programming environment, there are possibilities that instructions of one transaction are interleaved with some other transaction-

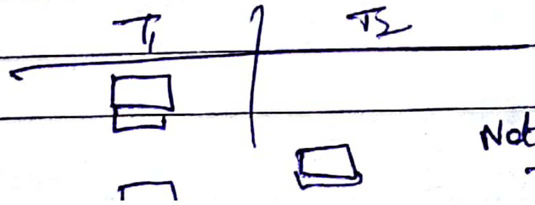
Schedule :-

A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it each comprising of a number of instructions.

Serial Schedule :-

It is a schedule in which transactions are aligned in ~~a~~ such a way that one transaction is executed first. When the Ist transaction completes its cycle, then the next transaction is executed. Transaction are ordered one after the other. This type of schedule is called serial schedule.

First come first serve ↳ order



Not possible in serial

We take serial schedule as benchmark.

↳ every other scheduling is compared to serial schedule.

➤ In a multi-transaction environment, serial schedule are considered as benchmark.
The execution sequence of an instruction in a transaction cannot be changed but two transactions can have their instructions executed in a random fashion. This execution does no harm if 2 trans^n are mutually independent and working on different set of data but in case of these 2 trans^n if working on same set of data then results may vary ~~&~~ ~~are~~ become unpredictable. This ever-varying result may bring the DB into an inconsistent state.

To resolve this problem we allow parallel execution of a transaction schedule if its transactions are either serializable or have some equivalence relation among them.