The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

$X \rightarrow Y$

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency

1. Trivial functional dependency

$A \rightarrow B$ has trivial functional dependency if B is a subset of A.

The following dependencies are also trivial like: $A \rightarrow A$, $B \rightarrow B$

Example:

Consider a table with two columns Employee_Id and Employee_Name.
{Employee_id, Employee_Name} → Employee_Id is a trivial functiona
l dependency as
Employee_Id is a subset of {Employee_Id, Employee_Name}.
Also, Employee_Id → Employee_Id and Employee_Name → Employe
e_Name are trivial dependencies too.

2. Non-trivial functional dependency

A → B has a non-trivial functional dependency if B is not a subset of A.
When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

ID → Name,
Name → DOB

Inference Rule (IR):

The Armstrong's axioms are the basic inference rule.

Armstrong's axioms are used to conclude functional dependencies on a relational database.

The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.

Using the inference rule, we can derive additional functional dependency from the initial set.

The Functional dependency has 6 types of inference rule:

1. Reflexive Rule (IR_1)

In the reflexive rule, if Y is a subset of X, then X determines Y.

If $X \supseteq Y$ then $X \rightarrow Y$

Example:

X = {a, b, c, d, e}

Y = {a, b, c}

2. Augmentation Rule (IR_2)

The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.

If $X \rightarrow Y$ then $XZ \rightarrow YZ$

Example:

For R(ABCD), **if** $A \rightarrow B$ then $AC \rightarrow BC$

3. Transitive Rule (IR_3)

In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.

If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

4. Union Rule (IR_4)

Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.

If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$

Proof:

1. $X \rightarrow Y$ (given)
2. $X \rightarrow Z$ (given)
3. $X \rightarrow XY$ (using IR_2 on 1 by augmentation with X. Where $XX = X$)
4. $XY \rightarrow YZ$ (using IR_2 on 2 by augmentation with Y)
5. $X \rightarrow YZ$ (using IR_3 on 3 and 4)

5. Decomposition Rule (IR_5)

Decomposition rule is also known as project rule. It is the reverse of union rule.

This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.

If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$

Proof:

1. $X \rightarrow YZ$ (given)
2. $YZ \rightarrow Y$ (using IR_1 Rule)
3. $X \rightarrow Y$ (using IR_3 on 1 and 2)

6. Pseudo transitive Rule (IR_6)

In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.

If $X \rightarrow Y$ and $YZ \rightarrow W$ then $XZ \rightarrow W$

Proof:

1. $X \rightarrow Y$ (given)
2. $WY \rightarrow Z$ (given)
3. $WX \rightarrow WY$ (using IR_2 on 1 by augmenting with W)
4. $WX \rightarrow Z$ (using IR_3 on 3 and 2)

Normalization:

1. Normalization is the process of organizing the data in the database.

2. Normalization is used to minimize the redundancy from a relation or set of relations.

3. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.

4. Normalization divides the larger table into smaller and links them using relationships.

5. The normal form is used to reduce redundancy from the database table.

6. The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows.

Anomalies :

Insert Anomaly: Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.

Delete Anomaly: The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.

Update Anomaly: The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

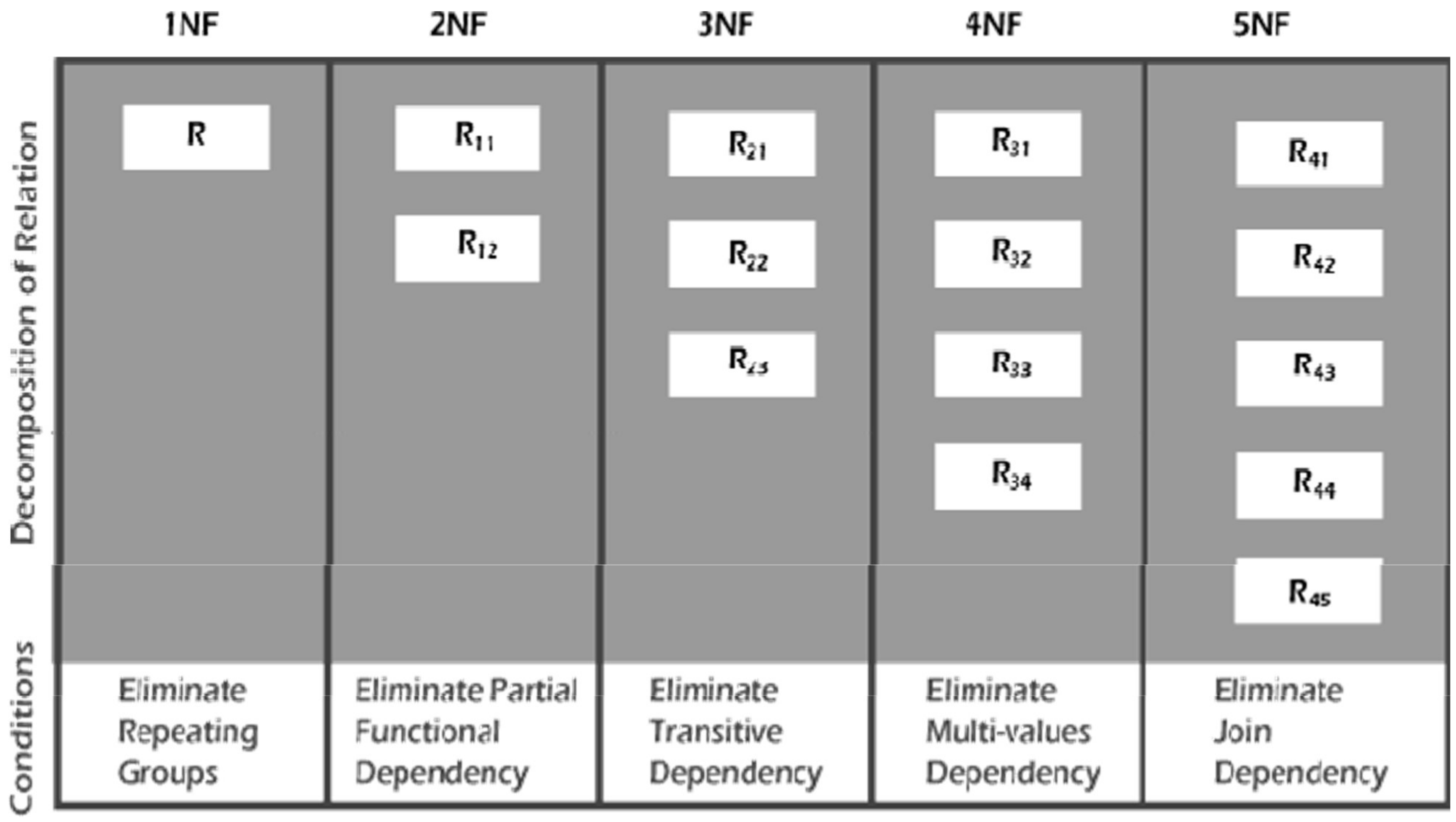
Types of Normal Forms:

Basic:

- 1NF
- 2NF
- 3NF
- BCNF

Others:

- 4NF
- 5NF


1NF
2NF
3NF
4NF
5NF
Decomposition of Relation
Conditions
R
R11
R12
R21
R22
R23
R31
R32
R33
R34
R41
R42
R43
R44
R45
Eliminate Repeating Groups
Eliminate Partial Functional Dependency
Eliminate Transitive Dependency
Eliminate Multi-values Dependency
Eliminate Join Dependency

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

First Normal Form (1NF)

1. A relation will be 1NF if it contains an atomic value.
2. It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
3. First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar

Functional Dependency

A functional dependency X->Y in a relation holds if two tuples having same value for X also have same value for Y i.e X uniquely determines Y.

E-ID	**E-NAME**	**E-CITY**	**E-STATE**
E001	John	Delhi	Delhi
E002	Mary	Delhi	Delhi
E003	John	Noida	U.P.

In EMPLOYEE relation,

FD **E-ID->E-NAME** holds because for each E-ID, there is a unique value of E-NAME.

FD **E-ID->E-CITY** and **E-CITY->E-STATE** also holds.

FD E-NAME->E-ID **does not hold** because E-NAME 'John' is not uniquely determining E-ID. There are 2 E-IDs corresponding to John (E001 and E003).

Candidate Key-

1. A set of minimal attribute(s) that can identify each tuple uniquely in the given relation is called as a candidate key.
2. A minimal super key is called as a candidate key.
3. For any given relation, It is possible to have multiple candidate keys.
4. There exists no general formula to find the total number of candidate keys of a given relation.

Finding Candidate Keys-

Step-01:

   a. Determine all essential attributes of the given relation.
   b. Essential attributes are those attributes which are not present on RHS of any functional dependency.
   c. Essential attributes are always a part of every candidate key.
   d. This is because they can not be determined by other attributes.

Step-02:

The remaining attributes of the relation are non-essential attributes.
This is because they can be determined by using essential attributes.
Now, following two cases are possible-

Case-01:

If all essential attributes together can determine all remaining non-essential attributes, then-
The combination of essential attributes is the candidate key.
It is the only possible candidate key.

Case-02:

If all essential attributes together can not determine all remaining non-essential attributes, then-
The set of essential attributes and some non-essential attributes will be the candidate key(s).
In this case, multiple candidate keys are possible.
To find the candidate keys, we check different combinations of essential and non-essential attributes.

Let R = (A, B, C, D, E, F) be a relation scheme with the following dependencies-
$C \rightarrow F$
$E \rightarrow A$
$EC \rightarrow D$
$A \rightarrow B$
Find key.

Essential attributes of the relation are- C and E.
So, attributes C and E will definitely be a part of every candidate key.
We will check if the essential attributes together can determine all remaining
non-essential attributes.
To check, we find the closure of CE.
$\{ CE \}^+$
$= \{ C , E \}$
$= \{ C , E , F \}$ (Using $C \rightarrow F$)
$= \{ A , C , E , F \}$ (Using $E \rightarrow A$)
$= \{ A , C , D , E , F \}$ (Using $EC \rightarrow D$)
$= \{ A , B , C , D , E , F \}$ (Using $A \rightarrow B$)
We conclude that CE can determine all the attributes of the given relation.
So CE only candidate key.

Total Number of Super Keys-

There are total 6 attributes in the given relation of which-
There are 2 essential attributes- C and E.
Remaining 4 attributes are non-essential attributes.
Essential attributes will be definitely present in every key.
Non-essential attributes may or may not be taken in every super key
So, number of super keys possible = 2 x 2 x 2 x 2 = 16.
Thus, total number of super keys possible = 16.

Let R = (A, B, C, D, E) be a relation scheme with the following dependencies-

$AB \rightarrow C$

$C \rightarrow D$

$B \rightarrow E$

Determine the total number of candidate keys and super keys.

AB is the only possible candidate key of the relation
total number of super keys possible = 8

Consider the relation scheme R(E, F, G, H, I, J, K, L, M, N) and the set of functional dependencies-

$\{ E, F \} \rightarrow \{ G \}$

$\{ F \} \rightarrow \{ I, J \}$

$\{ E, H \} \rightarrow \{ K, L \}$

$\{ K \} \rightarrow \{ M \}$

$\{ L \} \rightarrow \{ N \}$

Determine the total number of candidate keys and super keys.

EFH is the only possible candidate key of the relation.
total number of super keys possible = 128.

Consider the relation scheme R(A, B, C, D, E, H) Find candidate key.

$A \rightarrow B$

$BC \rightarrow D$

$E \rightarrow C$

$D \rightarrow A$

Ans: AEH, BEH, DEH

Given R(X Y Z W) and FD= { $XYZ \rightarrow W$, $XY \rightarrow ZW$ and $X \rightarrow YZW$ }
Find candidate key

Ans: X

Give R(A, B, C, D) and Set of Functional Dependency FD = { $AB \rightarrow CD$, $C \rightarrow A$, $D \rightarrow B$}. The question is to calculate the candidate key and no. of candidate key in above relation R using a given set of FDs.

FOUR Candidate Keys: AB, AD, BC, and CD

Relational Decomposition

1. When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
2. In a database, it breaks the table into multiple tables.
3. If the relation has no proper decomposition, then it may lead to problems like loss of information.
4. Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Types of Decomposition

1. Lossless Decomposition

a. If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
b. The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
c. The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

Dept_ID	Emp_ID	Dept_Name
Dpt1	E001	Operations
Dpt2	E002	HR
Dpt3	E003	Finance

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

Properties of Decomposition

It must have the following properties:

1. Decomposition Must be Lossless
2. Dependency Preservation
3. Lack of Data Redundancy

Following conditions must hold: To check for lossless join decomposition using Functional Dependency set.

1. Attribute(R1) U Attribute (R2) = Attribute (R)
2. Attribute (R1) ∩ Attribute (R2) ≠ Φ
3. Attribute (R1) ∩ Attribute (R2) -> Attribute (R1)

   or

   Attribute (R1) ∩ Attribute (R2) -> Attribute (R2)

Let us take an example of a relation R (X, Y, Z, W) with Functional Dependency set {X -> Y Z} is decomposed into relation R1(X, Y, Z) and relation R2(X, W) which is a lossless join decomposition as:

First condition holds true as Attribute (R1) U Attribute (R2) = (X, Y, Z) U (X, W) = (X, Y, Z, W) = Attribute (R).

The second condition holds as Attribute (R1) ∩ Attribute (R2) = (X, Y, Z) ∩ (X, W) ≠ Φ

The third condition holds as Attribute (R1) ∩ Attribute (R2) = X is a key of R1(X, Y, Z) because X -> Y Z is given.

Question 1: Consider a relation schema R(X Y Z W P) (above table R) is decomposed into R1(X Y Z) and R2(Z W P). determine whether the above R1 and R2 are Lossless or Lossy?

Cond 1: satisfied as Attribute(R1) U Attribute (R2) = Attribute (R) = (X Y Z W P)
Cond 2: satisfied as Attribute (R1) $\cap$ Attribute (R2) $\neq \Phi$ = (Z)
Cond 3: Not satisfied as common attribute Z is not key in any relation R1 or R2 (you can check from table values of column Z is repeating)

Question 2: Consider a relation schema R(X Y Z W P) (above table R) is decomposed into R1(X Y Z W) and R2(X Z W P). determine whether the above R1 and R2 are Lossless or Lossy?

Cond 1: satisfied as Attribute(R1) U Attribute (R2) = Attribute (R) = (X Y Z W P)
Cond 2: satisfied as Attribute (R1) $\cap$ Attribute (R2) $\neq \Phi$ = (X Z W)
Cond 3: satisfied as common attribute X Z W is key (we can check from table values of column W is unique and any combination of W will also be unique)

Question 2 . Given a relational schema R = { SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS } and the decomposed table
R1 = { SSN, ENAME }
R2 = { PNUMBER, PNAME, PLOCATION }
R3 = { SSN, PNUMBER, HOURS }
FD = { SSN → ENAME, PNUMBER → { PNAME, PLOCATION}, { SSN, PNUMBER } → HOURS }.
Identify whether the given decomposition of R, R1 R@, and R3 is lossless or lossy decomposition?

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R1	a1	a2	b13	b14	b15	b16
R2	b11	b22	a3	a4	a5	b26
R3	a1	b32	a3	b34	b35	a6

	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R1	a1	a2	b13	b14	b15	b16
R2	b11	b22	a3	a4	a5	b26
R3	a1	a2	a3	a4	a5	a6

Give R(X, Y, Z, W) and Set of Functional Dependency FD = { X → Y, Y → Z, Z → X}. The question is to calculate the candidate key and no. of candidate key in above relation R using a given set of FDs.

Candidate keys WX, WY, WZ.

Give R(X, Y, Z, W) and Set of Functional Dependency FD = { XY → ZW, W → X}. The question is to calculate the candidate key and no. of candidate key in above relation R using a given set of FDs.

Candidate keys Y X, Y W.

Give R(P, Q, R, S, T, U) and Set of Functional Dependency FD = { PQ → R, R → S, Q → PT}. The question is to calculate the candidate key and no. of candidate key in above relation R using a given set of FDs.

Candidate key QU

Give R(A, B, C, D) and Set of Functional Dependency FD = { AB → CD, C → A, D → B}. The question is to calculate the candidate key and no. of candidate key in above relation R using a given set of FDs.

Candidate Keys: AB, AD, BC, and CD

Definition of 3NF:

A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.

3NF is used to reduce the data duplication. It is also used to achieve the data integrity.

If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

- X is a super key.
- Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Definition of BCNF:

BCNF is the advance version of 3NF. It is stricter than 3NF.

A table is in BCNF if every functional dependency X → Y, X is the super key of the table.

For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Given a relation R(A, B, C, D) and Functional Dependency set FD = { AB $\rightarrow$ CD, B $\rightarrow$ C }, determine whether the given R is in 2NF? If not convert it into 2 NF.

Key : AB Not in 2NF

a) R1(B, C)

b) R2(A, B, D)

Given a relation R(P, Q, R, S, T) and Functional Dependency set FD = { PQ $\rightarrow$ R, S $\rightarrow$ T }, determine whether the given R is in 2NF? If not convert it into 2 NF.

Key : PQS Not in 2NF

a) R1(P, Q, R)

b) R2(S, T)

c) R3(P, Q, S)

Given a relation R(P, Q, R, S, T, U, V, W, X, Y) and Functional Dependency set FD = { PQ $\rightarrow$ R, PS $\rightarrow$ VW, QS $\rightarrow$ TU, P $\rightarrow$ X, W $\rightarrow$ Y }, determine whether the given R is in 2NF? If not convert it into 2 NF.

Key : PQS Not in 2NF

a) R1(P, Q, R)
b) R2(P, S, V, W)
c) R3(Q, S, T, U)
d) R4(P, X)
e) R5(W, Y)
f) R6(P, Q, S)

Given a relation R(A, B, C, D, E) and Functional Dependency set FD = { A $\rightarrow$ B, B $\rightarrow$ E, C $\rightarrow$ D}, determine whether the given R is in 2NF? If not convert it into 2 NF.

Key : AC Not in 2NF

R1(A, B, E)

R2(C, D) R3(A, C)

Given a relation R(X, Y, Z) and Functional Dependency set FD = { X → Y and Y → Z }, determine whether the given R is in 3NF? If not convert it into 3 NF.

Key : X Not in 3NF

a) R1(X, Y)
b) R2(Y, Z)

Given a relation R(X, Y, Z, W, P) and Functional Dependency set FD = { X → Y, Y → P, and Z → W}, determine whether the given R is in 3NF? If not convert it into 3 NF.

Key : X Z Not in 3NF

R1(X, Y) R2(Y, P)

R3(Z, W) R4(X, Z)

Given a relation R(P, Q, R, S, T, U, V, W, X, Y) and Functional Dependency set FD = { PQ → R, P → ST, Q → U, U → VW, and S → XY}, determine whether the given R is in 3NF? If not convert it into 3 NF.

Key : PQ Not in 3NF

R1(P, S, T)

R2(Q, U)

R3(U, V, W)

R4(S, X, Y)

R5(P, Q, R)

Given a relation R(X, Y, Z) and Functional Dependency set FD = { XY → Z and Z → Y }, determine whether the given R is in BCNF? If not convert it into BCNF.

Key : XY and XZ
R1(Z, Y)
R2(X, Z)

Given a relation R(X, Y, Z) and Functional Dependency set FD = { X → Y and Y → Z }, determine whether the given R is in BCNF? If not convert it into BCNF.

Key : X
R1(X, Y)
R2(Y, Z)

Given a relation R(P, Q, R, S, T) and Functional Dependency set FD = { QR → PST, S → Q }, determine given R is in which normal form?

Key : RS and RQ
3NF

Given a relation R(A, B, C) and Functional Dependency set FD = { A → B, B → C, and C → A}, determine given R is in which normal form?

Key : A B and C
BCNF

Canonical Cover

A canonical cover or irreducible a set of functional dependencies FD is a simplified set of FD that has a similar closure as the original set FD.

Extraneous attributes: An attribute of an FD is said to be extraneous if we can remove it without changing the closure of the set of FD.

Example: Given a relational Schema R(A, B, C, D) and set of Function Dependency FD = { B → A, AD → BC, C → ABD }. Find the canonical cover?

1. Decompose the FD using decomposition rule(Armstrong Axiom).

B → A

AD → B (using decomposition inference rule on AD → BC)

AD → C (using decomposition inference rule on AD → BC)

C → A (using decomposition inference rule on C → ABD)

C → B (using decomposition inference rule on C → ABD)

C → D (using decomposition inference rule on C → ABD)

Now set of FD = { B → A, AD → B, AD → C, C → A, C → B, C → D }

2. The next step is to find closure of the left side of each of the given FD by including that FD and excluding that FD, if closure in both cases are same then that FD is redundant and we remove that FD from the given set, otherwise if both the closures are different then we do not exclude that FD.

1a. Closure B+ = BA using FD = { **B → A**, AD → B, AD → C, C → A, C → B, C → D }

1b. Closure B+ = B using FD = { AD → B, AD → C, C → A, C → B, C → D }

From 1 a and 1 b, we found that both the Closure(by including **B → A** and excluding **B → A**) are not equivalent, hence FD B → A is important and cannot be removed from the set of FD.

2 a. Closure AD+ = ADBC using FD = { B →A, AD → B, AD → C, C → A, C → B, C → D }

2 b. Closure AD+ = ADCB using FD = { B → A, AD → C, C → A, C → B, C → D }

From 2 a and 2 b, we found that both the Closure (by including AD → B and excluding AD → B) are equivalent, hence FD AD → B is not important and can be removed from the set of FD.

Hence resultant FD = { B → A, AD → C, C → A, C → B, C → D }

3 a. Closure AD+ = ADCB using FD = { B →A, **AD** → **C**, C → A, C → B, C → D }

3 b. Closure AD+ = AD using FD = { B → A, C → A, C → B, C → D }

From 3 a and 3 b, we found that both the Closure (by including **AD** → **C** and excluding **AD** → **C**) are not equivalent, hence FD AD → C is important and cannot be removed from the set of FD.

Hence resultant FD = { B → A, AD → C, C → A, C → B, C → D }

4 a. Closure C+ = CABD using FD = { B →A, AD → C, **C** → **A**, C → B, C → D }

4 b. Closure C+ = CBDA using FD = { B → A, AD → C, C → B, C → D }

From 4 a and 4 b, we found that both the Closure (by including **C** → **A** and excluding **C** → **A**) are equivalent, hence FD **C** → **A** is not important and can be removed from the set of FD.

Hence resultant FD = { B → A, AD → C, C → B, C → D }

5 a. Closure C+ = CBDA using FD = { B →A, AD → C, **C** → **B**, C → D }

5 b. Closure C+ = CD using FD = { B → A, AD → C, C → D }

From 5 a and 5 b, we found that both the Closure (by including **C** → **B** and excluding **C** → **B**) are not equivalent, hence FD **C** → **B** is important and cannot be removed from the set of FD.

Hence resultant FD = { B → A, AD → C, C → B, C → D }
6 a. Closure C+ = CDBA using FD = { B →A, AD → C, C → B, **C** → **D** }
6 b. Closure C+ = CBA using FD = { B → A, AD → C, C → B }
From 6 a and 6 b, we found that both the Closure(by including **C** → **D** and excluding **C** → **D**) are not equivalent, hence FD **C** → **D** is important and cannot be removed from the set of FD.
Hence resultant FD = { B → A, AD → C, C → B, C → D }
Since FD = { B → A, AD → C, C → B, C → D } is resultant FD, now we have checked the redundancy of attribute, since the left side of FD AD → C has two attributes, let's check their importance, i.e. whether they both are important or only one.
Closure AD+ = ADCB using FD = { B →A, **AD** → **C**, C → B, C → D }
Closure A+ = A using FD = { B →A, **AD** → **C**, C → B, C → D }
Closure D+ = D using FD = { B →A, **AD** → **C**, C → B, C → D }
Since the closure of AD+, A+, D+ that we found are not all equivalent, hence in FD AD → C, both A and D are important attributes and cannot be removed.
Hence resultant FD = { B → A, AD → C, C → B, C → D } and we can rewrite as
FD = { B → A, AD → C, C → BD } is Canonical Cover of FD = { B → A, AD → BC, C → ABD }.

Given a relational Schema R(W, X, Y, Z) and set of Function Dependency FD = { $W \rightarrow X$, $Y \rightarrow X$, $Z \rightarrow WXY$, $WY \rightarrow Z$ }. Find the canonical cover?

FD = { $W \rightarrow X$, $Y \rightarrow X$, $Z \rightarrow WY$, $WY \rightarrow Z$ } is Canonical Cover of FD = { $W \rightarrow X$, $Y \rightarrow X$, $Z \rightarrow WXY$, $WY \rightarrow Z$ }.

Given a relational Schema R(V, W, X, Y, Z) and set of Function Dependency FD = { $V \rightarrow W$, $VW \rightarrow X$, $Y \rightarrow VXZ$ }. Find the canonical cover?

FD = { $V \rightarrow WX$, $Y \rightarrow VZ$ } is Canonical Cover of FD = { $V \rightarrow W$, $VW \rightarrow X$, $Y \rightarrow VXZ$ }.

EQUIVALENCE OF FUNCTIONAL DEPENDENCY

Two or more than two sets of functional dependencies are called equivalence if the right-hand side of one set of functional dependency can be determined using the second FD set, similarly the right-hand side of the second FD set can be determined using the first FD set.

Q 1: Given a relational schema R(X, Y, Z, W, V) set of functional dependencies P and Q such that:
$P = \{ X \rightarrow Y, XY \rightarrow Z, W \rightarrow XZ, W \rightarrow V\}$ and $Q = \{ X \rightarrow YZ, W \rightarrow XV \}$
using FD sets P and Q are equivalence or not.

determine the right-hand side of the FD set of P using FD set Q.
Let's find closure of the left side of each FD of P using FD Q.

$$X+ = XYZ \text{ (using } X \rightarrow YZ)$$
$$XY+ = XYZ \text{ (using } X \rightarrow YZ)$$
$$W+ = WXVYZ \text{ (using } W \rightarrow XV \text{ and } X \rightarrow YZ)$$
$$W+ = WXVYZ \text{ (using } W \rightarrow XV \text{ and } X \rightarrow YZ)$$

Now compare closure of each X, XY, W and W calculated using FD Q with the right-hand side of FD P. Closure of each X, XY, W and W has all the attributes which are on the right-hand side of each FD of P. Hence, we can say P is a subset of Q.

determine the right-hand side of the FD set of Q using FD set P.
Let us find closure of the left side of each FD of Q using FD P.

$$X+ = XYZ \text{ (using } X \rightarrow Y \text{ and } XY \rightarrow Z)$$

$$W+ = WXZVY \text{ (using } W \rightarrow XZ, W \rightarrow V, \text{ and } X \rightarrow Y)$$

Now compare closure of each X, W calculated using FD P with the right-hand side of FD Q. Closure of each X and W has all the attributes which are on the right-hand side of each FD of Q. Hence, we can say **Q is a subset of P**

Given a relational schema R(A, B, C, D) set of functional dependencies P and Q such that:
P = { A → B, B → C, C → D } and Q = { A → BC, C → D } using FD sets P and Q are equivalence or not.

Not equivalence. Q is a subset of P but P is not a subset of Q

Given a relational schema R(X, Y, Z) set of functional dependencies P and Q such that:
P = { X → Y, Y → Z, Z → X } and Q = { X → YZ, Y → X, Z → X } using FD sets P and Q are equivalence or not.

equivalence. Q is a subset of P and P is a subset of Q

Multi-valued Dependency

- Multi-valued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multi-valued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.
In this case, these two columns can be called as multi-valued dependent on BIKE_MODEL. The representation of these dependencies is shown below:
BIKE_MODEL → → MANUF_YEAR
BIKE_MODEL → → COLOR
This can be read as "BIKE_MODEL multi determined MANUF_YEAR" and "BIKE_MODEL multi determined COLOR".

Fourth normal form (4NF)

A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.

For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Join Dependency

1. Join decomposition is a further generalization of Multi-valued dependencies.
2. If the join of R1 and R2 over C is equal to relation R, then we can say that a join dependency (JD) exists.
3. Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
4. Alternatively, R1 and R2 are a lossless decomposition of R.
5. A JD ⋈ {R1, R2,..., Rn} is said to hold over a relation R if R1, R2,....., Rn is a lossless-join decomposition.
6. The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R.
7. Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Fifth normal form (5NF)

A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
5NF is also known as Project-join normal form (PJ/NF).

P

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

1.
A→BC
BC→AD
D→E
Highest NF

2. Consider a relation R (A, B, C, D, E, F, G, H), where each attribute is atomic, and the following functional dependencies exist.
CH → G, A → BC, B → CFH, E → A, F → EG

3. Consider the following functional dependencies in a database:
Date_of_Birth → Age
Age → Eligibility
Name → Roll_number
Roll_number → Name
Course_number → Course_name
Course_number → Instructor
(Roll_number, Course_number) → Grade

4. Book (title, Author, Catalog no, publisher, year, price)
Collection (title, Author, Catalog no)

The following functional dependencies:
title, Author → Catalog no
Catalog no → title Author publisher year
publisher title year → price
Assume (Author, Title) is the key for both schemas.

BOOK is 2NF and COLLECTION is BCNF

Q. R (ABCD)

$\{AB \rightarrow CD, D \rightarrow A\}$

Assume that it does not contain any multi-valued attribute .

Find the highest normal form of this relation.

Q. R (ABCDEF)

$\{A \rightarrow BCDEF, BC \rightarrow ADEF, B \rightarrow F, D \rightarrow E\}$

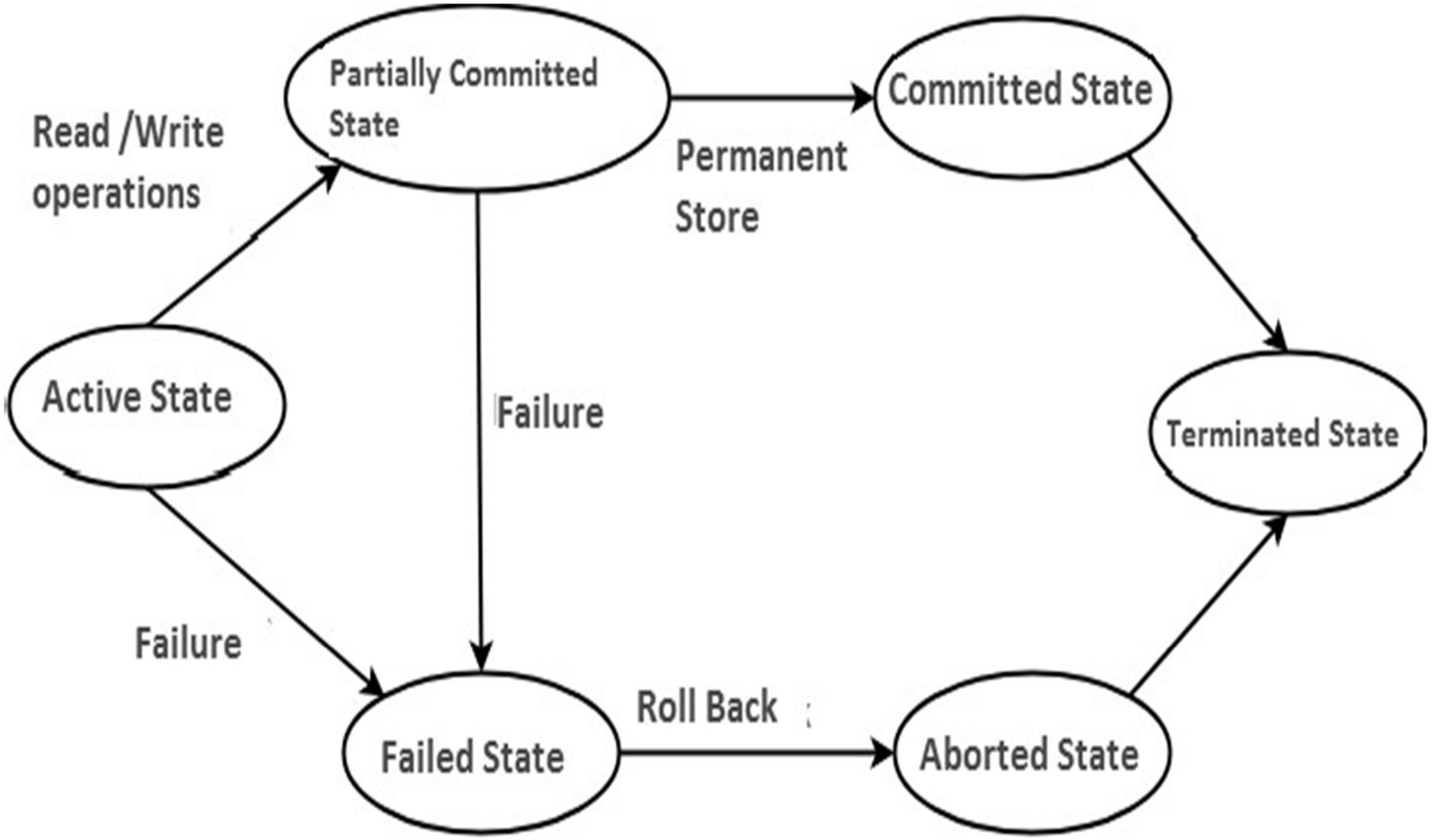
Assume it is in 1NF .

Find the highest normal form of this relation.

R (ABCDEFGH)

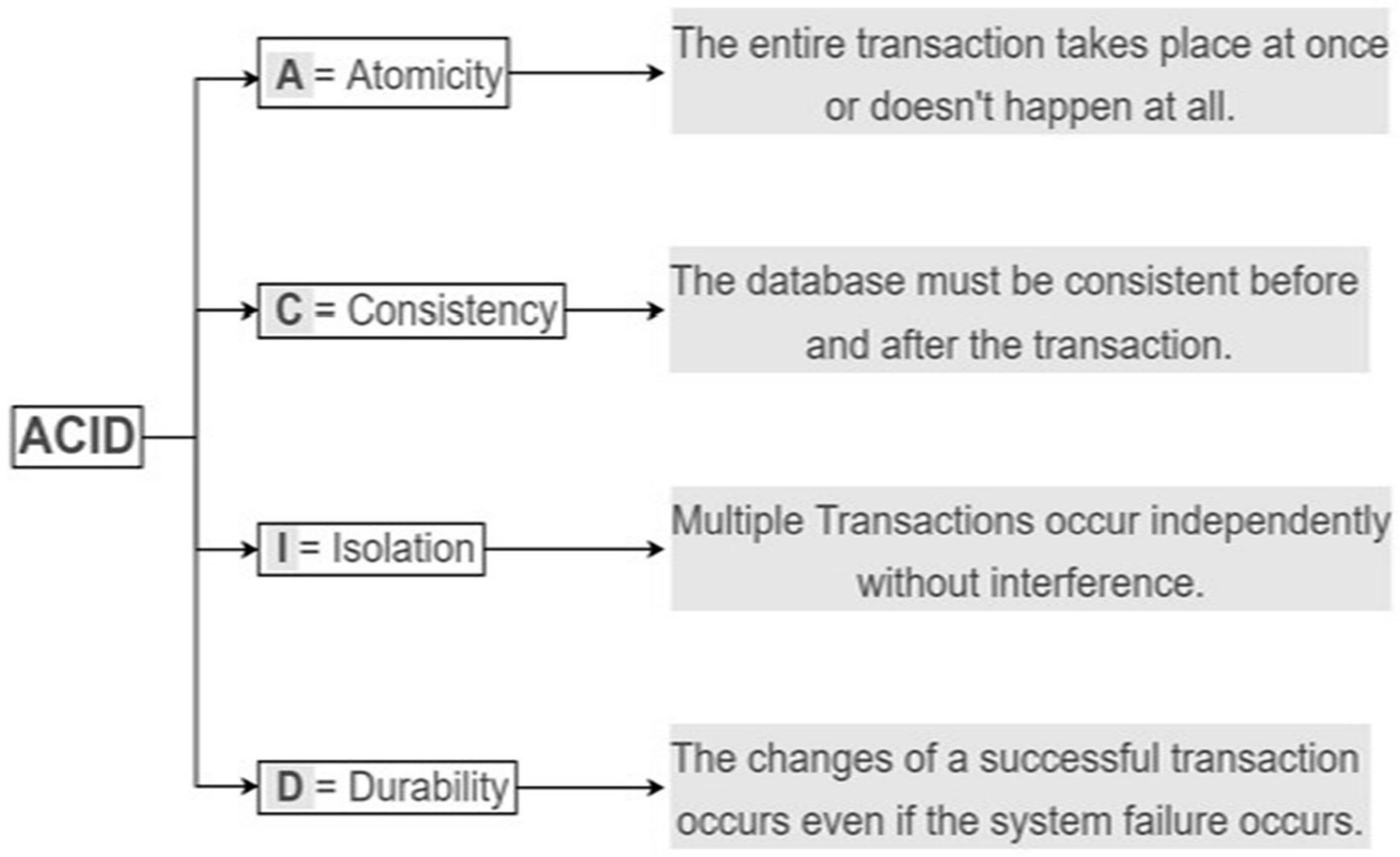
$\{ABC \rightarrow DE, E \rightarrow FG, H \rightarrow G, G \rightarrow H, ABC \rightarrow EF\}$

Find the highest normal form of this relation



Transaction States in DBMS

ACID Properties in DBMS



The entire transaction takes place at once or doesn't happen at all.

The database must be consistent before and after the transaction.

Multiple Transactions occur independently without interference.

The changes of a successful transaction occurs even if the system failure occurs.

Atomicity

It states that all operations of the transaction take place at once if not, the transaction is aborted.

There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2

Read(A)
A:= A-100
Write(A)
	Read(B)
	Y:= Y+100
	Write(B)

1. After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.
2. If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B.
3. This shows the inconsistent database state.
4. In order to ensure correctness of database state, the transaction must be executed in entirety.

<u>Consistency</u>

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.
- **For example:** The total amount must be maintained before or after the transaction.

  Total before T occurs = 600+300=900
  Total after T occurs= 500+400=900

- Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

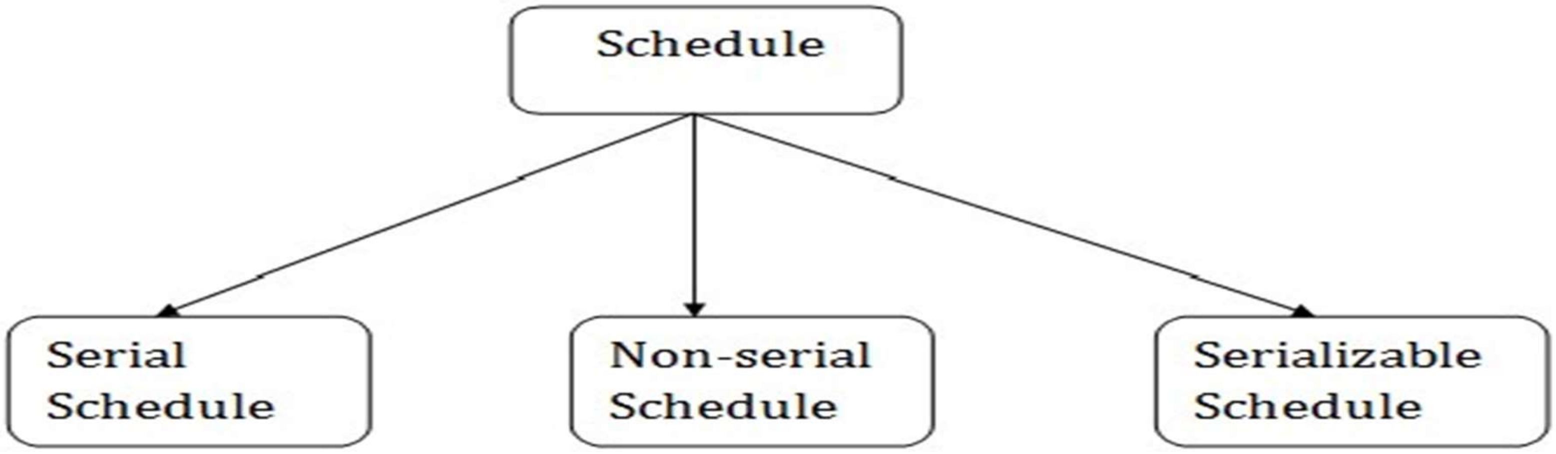
- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

Schedule

A series of operation from one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transaction.



1. Serial Schedule

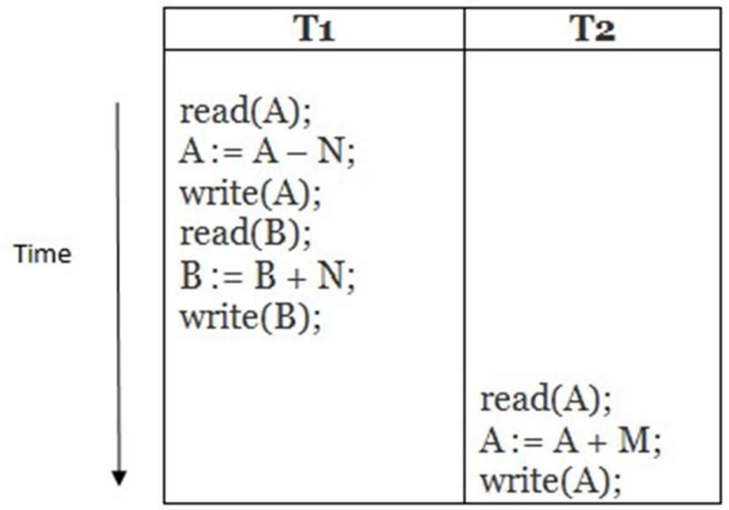
The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

Execute all the operations of T1 which was followed by all the operations of T2.

Execute all the operations of T1 which was followed by all the operations of T2.

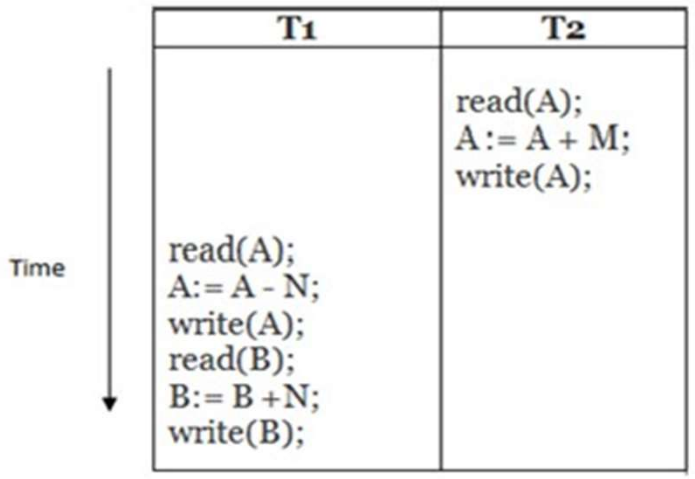
(a)



T1	T2
read(A);	
A := A – N;	
write(A);	
read(B);	
B := B + N;	
write(B);	
	read(A);
	A := A + M;
	write(A);

Schedule A

(b)



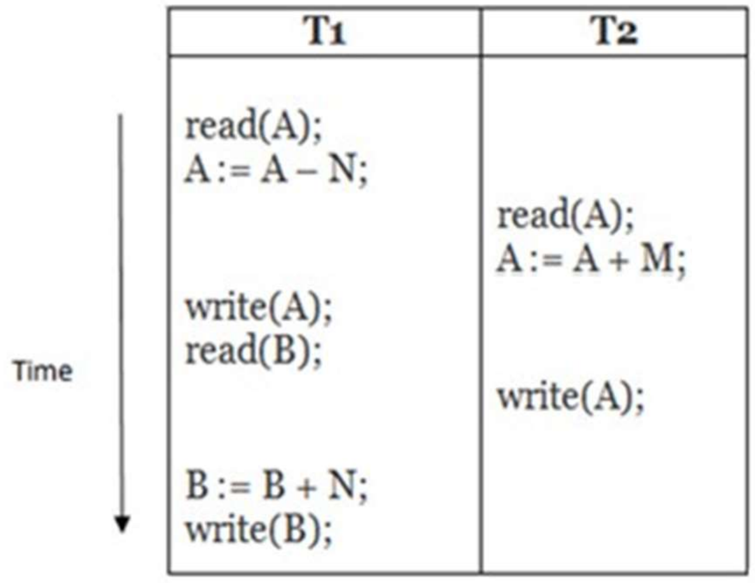
T1	T2
	read(A);
	A := A + M;
	write(A);
read(A);	
A:= A - N;	
write(A);	
read(B);	
B:= B +N;	
write(B);	

Schedule B

2. Non-serial Schedule

If interleaving of operations is allowed, then there will be non-serial schedule. It contains many possible orders in which the system can execute the individual operations of the transactions.

(c)



T1	T2

read(A);
A:= A − N;	

	read(A);
	A:= A + M;

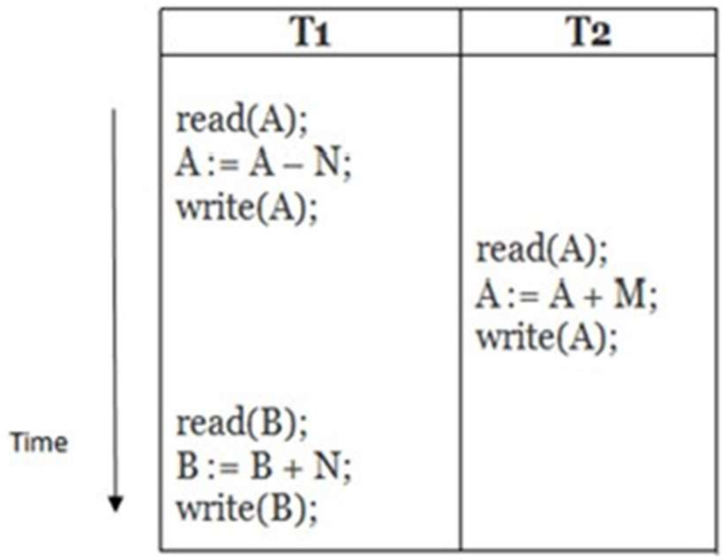
write(A);
read(B);	

	write(A);

B := B + N;
write(B);	

Schedule C

(d)



T1	T2

read(A);
A:= A − N;
write(A);	

	read(A);
	A:= A + M;
	write(A);

read(B);
B := B + N;
write(B);	

Schedule D

Concurrent Execution:

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database.
- It means that the same database is executed simultaneously on a multi-user system by different users.
- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database.
- In concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution

In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems (W - W Conflict)

The problem occurs *when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent.*

For example:

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is \$300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

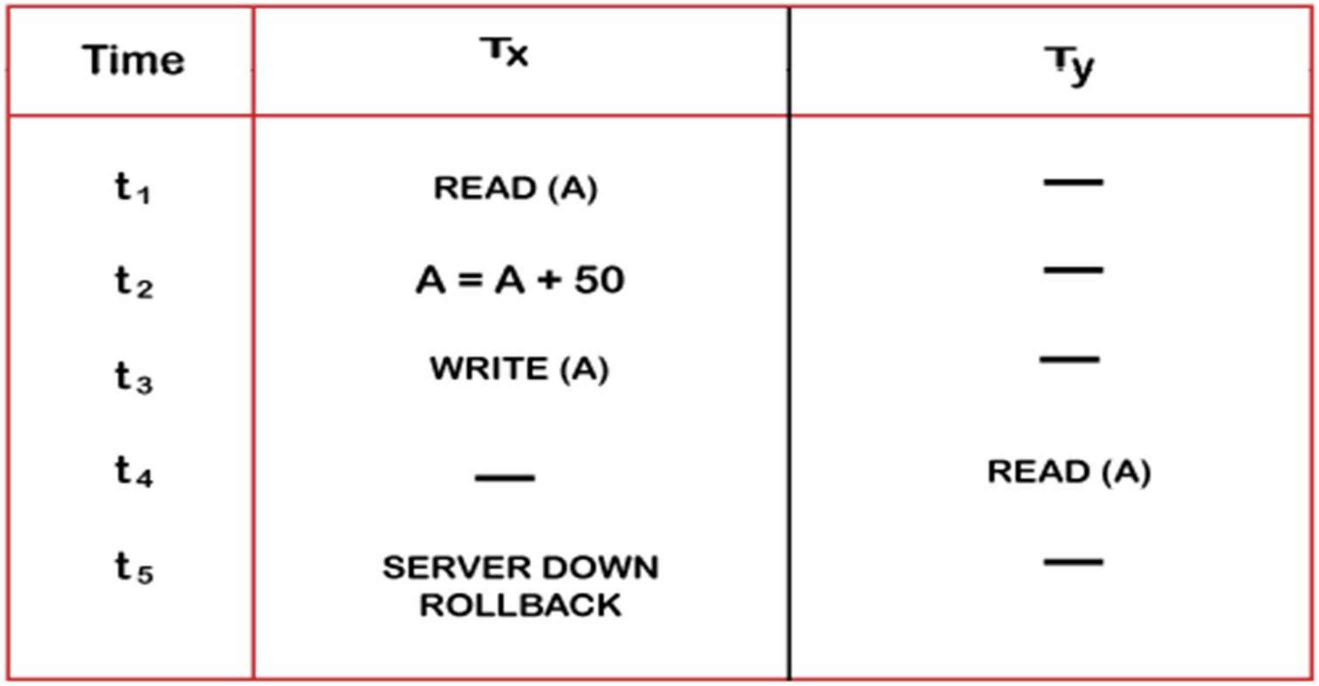
LOST UPDATE PROBLEM

Dirty Read Problems (W-R Conflict)

The dirty read problem occurs *when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.*

For example:

Consider two transactions T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:



Time	Tx	Ty
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

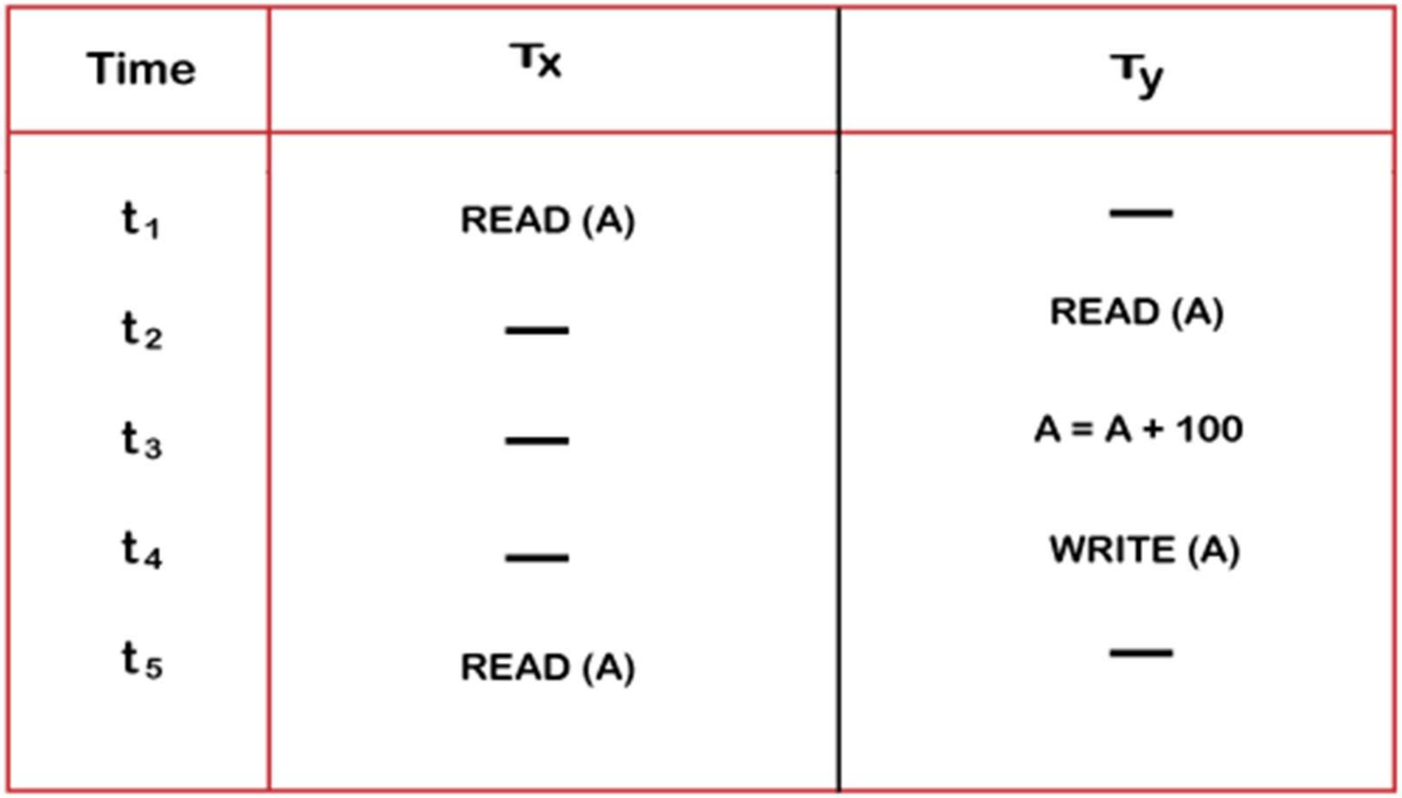
DIRTY READ PROBLEM

Unrepeatable Read Problem (W-R Conflict)

Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

For example:

Consider two transactions, T_X and T_Y, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:



Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

Conflict Serializable Schedule

1. A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
2. The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

Conflicting Operations

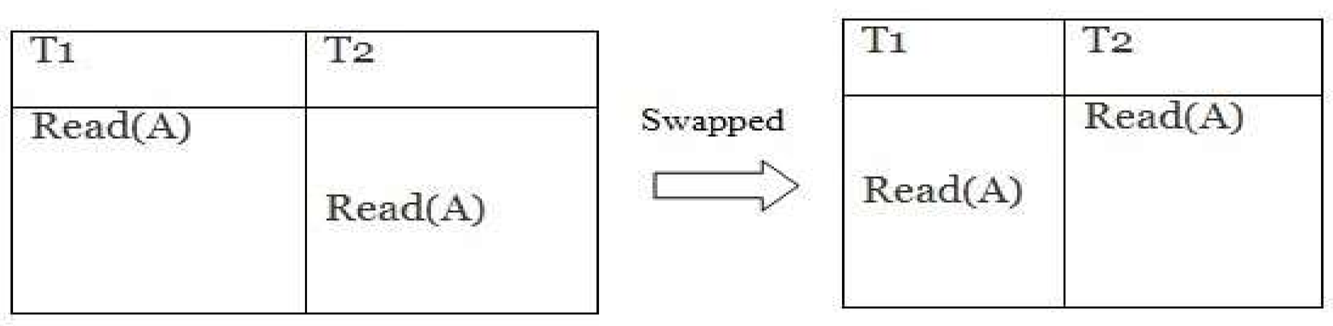
The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Example:

Swapping is possible only if S1 and S2 are logically equal.

1. T1: Read(A)   T2: Read(A)



T1	T2
Read(A)	
	Read(A)

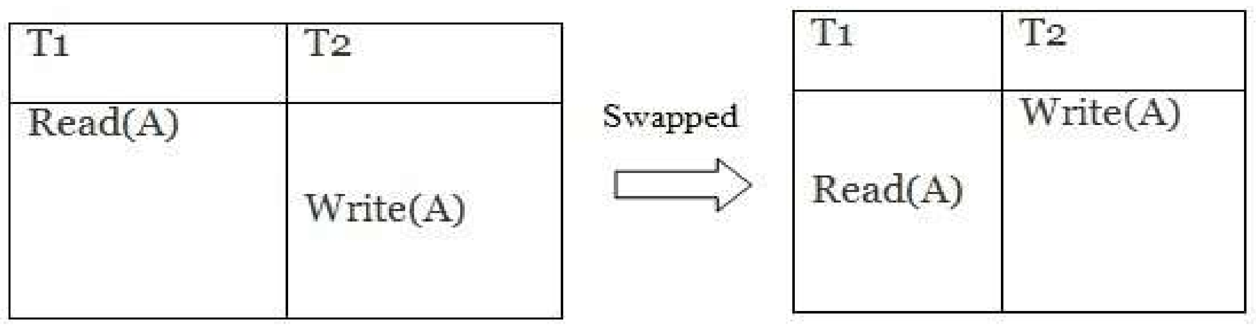
Schedule S1

T1	T2
	Read(A)
Read(A)	

Schedule S2

Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)



T1	T2
Read(A)	
	Write(A)

Swapped

T1	T2
	Write(A)
Read(A)	

Here, S1 ≠ S2. That means it is conflict.

Schedule S1 **Schedule S2**

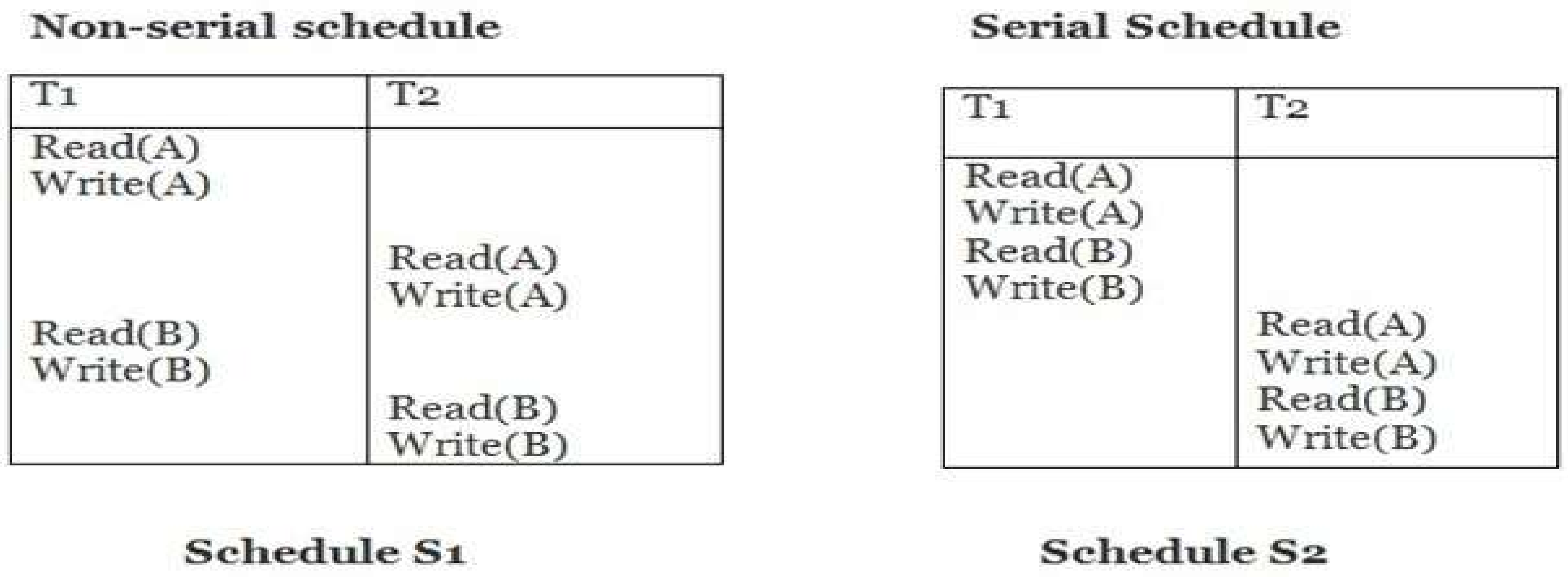
Conflict Equivalent

In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations.

In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations).

Two schedules are said to be conflict equivalent if and only if:

1. They contain the same set of the transaction.
2. If each pair of conflict operations are ordered in the same way.



Non-serial schedule

T1	T2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Schedule S1

Serial Schedule

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S2

After swapping of non-conflict operations, the schedule S1 becomes:

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Since, S1 is conflict serializable.

View Serializability

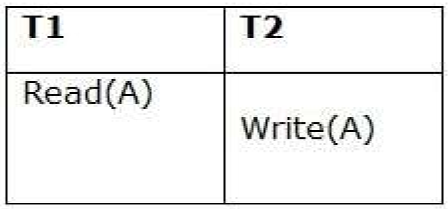
1. A schedule will view serializable if it is view equivalent to a serial schedule.
2. If a schedule is conflict serializable, then it will be view serializable.
3. The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

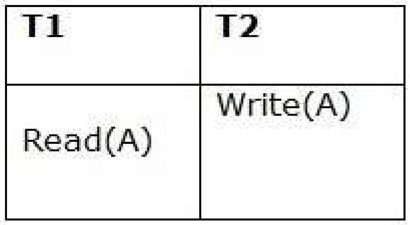
1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.



T1	T2
Read(A)	
	Write(A)

Schedule S1



T1	T2
	Write(A)
Read(A)	

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)

Schedule S1

T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

Schedule S2

two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)		
	Read(A)	
		Write(A)

Schedule S1

T1	T2	T3
	Read(A)	
Write(A)		
		Write(A)

Schedule S2

two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

T1	T2	T3
READ(A)		
	WRITE(A)	
WRITE(A)		
		WRITE(A)

Schedule S With 3 transactions, the total number of possible schedule = 3! = 6

S1 = <T1 T2 T3> S2 = <T1 T3 T2> S3 = <T2 T3 T1>
S4 = <T2 T1 T3> S5 = <T3 T1 T2> S6 = <T3 T2 T1>

T1	T2	T3
READ(A)		
WRITE(A)		
	WRITE(A)	
		WRITE(A)

Step 1: final updation on data items
In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read
The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write
The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.
The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

view equivalent serial schedule is: T1 → T2 → T3

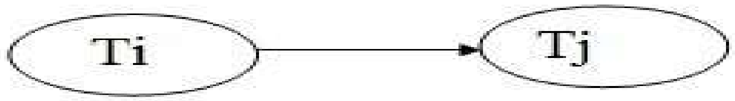
Testing of Serializability

a) Serialization Graph is used to test the Serializability of a schedule.

b) Assume a schedule S. For S, we construct a graph known as precedence graph. This graph has a pair G = (V, E), where V consists a set of vertices, and E consists a set of edges. The set of vertices is used to contain all the transactions participating in the schedule. The set of edges is used to contain all edges Ti ->Tj for which one of the three conditions holds:

Create a node Ti → Tj if Ti executes write (Q) before Tj executes read (Q).
Create a node Ti → Tj if Ti executes read (Q) before Tj executes write (Q).
Create a node Ti → Tj if Ti executes write (Q) before Tj executes write (Q).

Precedence graph for Schedule S



- If a precedence graph contains a single edge Ti → Tj, then all the instructions of Ti are executed before the first instruction of Tj is executed.
- If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

Time

T1	T2	T3
Read(A)		
	Read(B)	
A:= f_1(A)		
		Read(C)
	B:= f_2(B)	
	Write(B)	
		C:= f_3(C)
		Write(C)
Write(A)		
		Read(B)
	Read(A)	
	A:= f_4(A)	
Read(C)		
	Write(A)	
C:= f_5(C)		
Write(C)		
		B:= f_6(B)
		Write(B)

Schedule S1

$$\begin{tikzcd}
 & T1 \arrow[dr] & \\
T2 \arrow[ur] & & T3 \arrow[ll]
\end{tikzcd}$$

Read(A): In T1, no subsequent writes to A, so no new edges **Read(B):** In T2, no subsequent writes to B, so no new edges **Read(C):** In T3, no subsequent writes to C, so no new edges

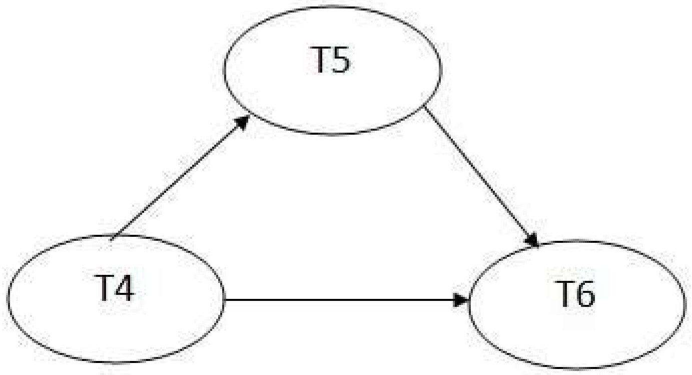
Write(B): B is subsequently read by T3, so add edge T2 → T3 **Write(C):** C is subsequently read by T1, so add edge T3 → T1 **Write(A):** A is subsequently read by T2, so add edge T1 → T2

Write(A): In T2, no subsequent reads to A, so no new edges **Write(C):** In T1, no subsequent reads to C, so no new edges **Write(B):** In T3, no subsequent reads to B, so no new edges

Time

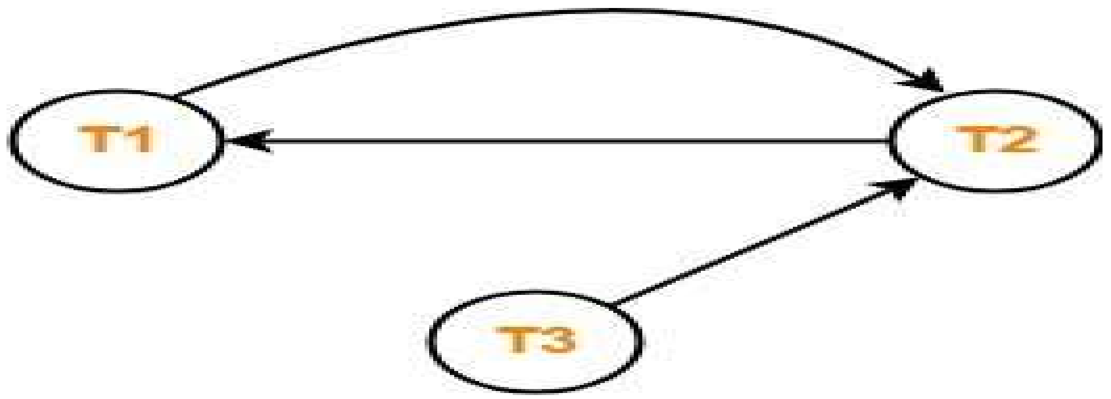
T4	T5	T6
Read(A)		
A:= f1(A)		
Read(C)		
Write(A)		
A:= f2(C)		
	Read(B)	
Write(C)		
	Read(A)	
		Read(C)
	B:= f3(B)	
	Write(B)	
		C:= f4(C)
		Read(B)
		Write(C)
	A:=f5(A)	
	Write(A)	
		B:= f6(B)
		Write(B)

Schedule S2



Check whether the given schedule S is conflict serializable or not-

S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$



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

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

T1	T2	T3	T4
	R(X)		
		W(X)	
		Commit	
W(X)			
Commit			
	W(Y)		
	R(Z)		
	Commit		
			R(X)
			R(Y)
			Commit

Recoverability of Schedule

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

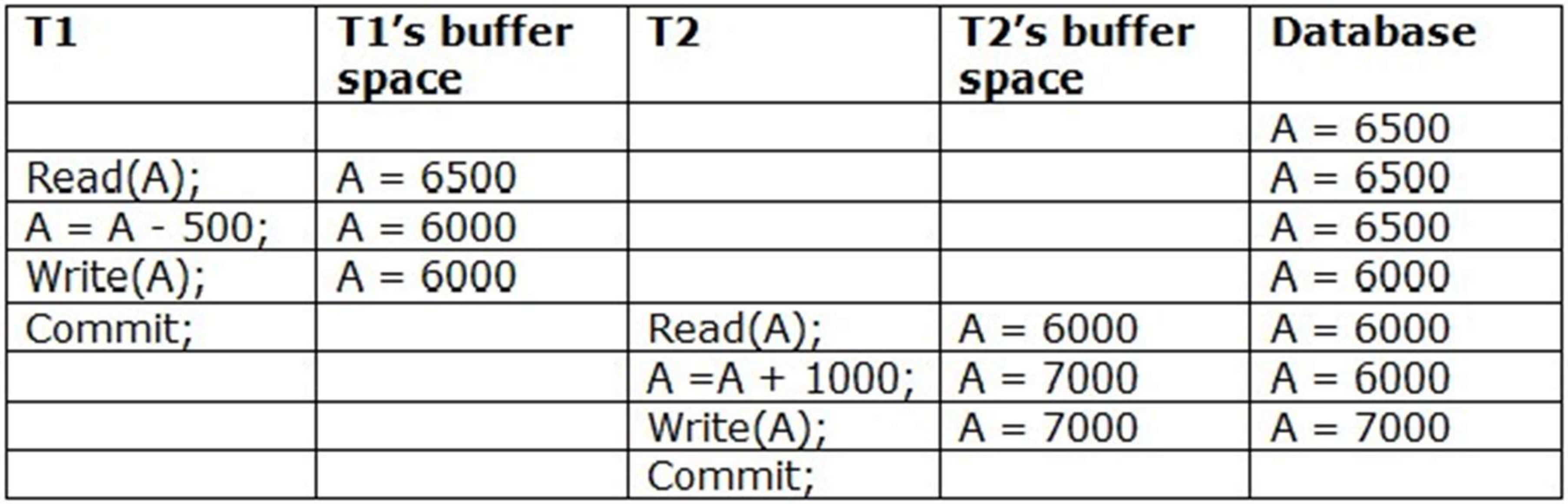
The above table shows a schedule which has two transactions. T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1. T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule.

Irrecoverable schedule: The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

T1	**T1's buffer space**	**T2**	**T2's buffer space**	**Database**
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

The above table shows a schedule with two transactions. Transaction T1 reads and writes A, and that value is read and written by transaction T2. But later on, T1 fails. Due to this, we have to rollback T1. T2 should be rollback because T2 has read the value written by T1. As it has not committed before T1 commits so we can rollback transaction T2 as well. So it is recoverable with cascade rollback.

Recoverable with cascading rollback: The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.



T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

The above Table shows a schedule with two transactions. Transaction T1 reads and write A and commits, and that value is read and written by T2. So this is a cascade less recoverable schedule

Types of Recoverable Schedules-

A recoverable schedule may be any one of these kinds-

1. Cascading Schedule
2. Cascade less Schedule
3. Strict Schedule

Cascading Schedule-

If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Schedule or Cascading Rollback or Cascading Abort.

It simply leads to the wastage of CPU time.

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

Cascadeless Schedule-

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

In other words,

Cascadeless schedule allows only committed read operations.

Therefore, it avoids cascading roll back and thus saves CPU time.

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

T1	T2
R(A)	
W(A)	
	W(A) // uncommitted
commit	

Strict Schedule-

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

In other words,

Strict schedule allows only committed read and write operations.

Clearly, strict schedule implements more restrictions than cascadeless schedule.

T1	T2
W(A)	
Commit/rollback	
	R(A) / W(A)

Failure Classification

To see where the problem has occurred, we generalize a failure into various categories, as follows −

1. **Transaction failure**

A transaction has to abort when it fails to execute or when it reaches a point from where it can’t go any further. This is called transaction failure where only a few transactions or processes are hurt.

Reasons for a transaction failure could be −

Logical errors − Where a transaction cannot complete because it has some code error or any internal error condition.

System errors − Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

2. System Crash

There are problems − external to the system − that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure.
Examples may include operating system errors.

3. Disk Failure

In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.
Disk failures include formation of bad sectors, un reachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

The recovery procedures in DBMS ensure the database's atomicity and durability. If a system crashes in the middle of a transaction and all of its data is lost, it is not regarded as durable. If just a portion of the data is updated during the transaction, it is not considered atomic. Data recovery procedures in DBMS make sure that the data is always recoverable to protect the durability property and that its state is retained to protect the atomic property. The procedures listed below are used to recover data from a DBMS,

1. Recovery based on logs.
2. Recovery through Deferred Update
3. Immediate Recovery via Immediate Update

Log-Based Recovery

1. The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
2. If any operation is performed on the database, then it will be recorded in the log.
3. But the process of storing the logs should be done before the actual transaction is applied in the database.
4. Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.
5. When the transaction is initiated, then it writes 'start' log.
   <Tn, Start>
6. When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.
   <Tn, City, 'Noida', 'Delhi' >
7. When the transaction is finished, then it writes another log to indicate the end of the transaction.
   <Tn, Commit>
8. There are two approaches to modify the database:

1. Deferred database modification:

1. The deferred modification technique occurs if the transaction does not modify the database until it has committed.
2. In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

2. Immediate database modification:

1. The Immediate modification technique occurs if database modification occurs while the transaction is still active.
2. In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Recovery using Log records

When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

1. If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then the Transaction Ti needs to be redone.
2. If log contains record<T_n, Start> but does not contain the record either <Ti, commit> or <Ti, abort>, then the Transaction Ti needs to be undone.

Deferred Update	Immediate Update
In a deferred update, the changes are not applied immediately to the database.	In an immediate update, the changes are applied directly to the database.
The log file contains all the changes that are to be applied to the database.	The log file contains both old as well as new values.
In this method once rollback is done all the records of log file are discarded and no changes are applied to the database.	In this method once rollback is done the old values are restored to the database using the records of the log file.
Concepts of buffering and caching are used in deferred update method.	Concept of shadow paging is used in immediate update method.
The major disadvantage of this method is that it requires a lot of time for recovery in case of system failure.	The major disadvantage of this method is that there are frequent I/O operations while the transaction is active.
In this method of recovery, firstly the changes carried out by a transaction on the data are done in the log file and then applied to the database on commit. Here, the maintained record gets discarded **on rollback** and thus, not applied to the database.	In this method of recovery, the database gets directly updated after the changes made by the transaction and the log file keeps the old and new values. In the case of **rollback**, these records are used to restore old values.

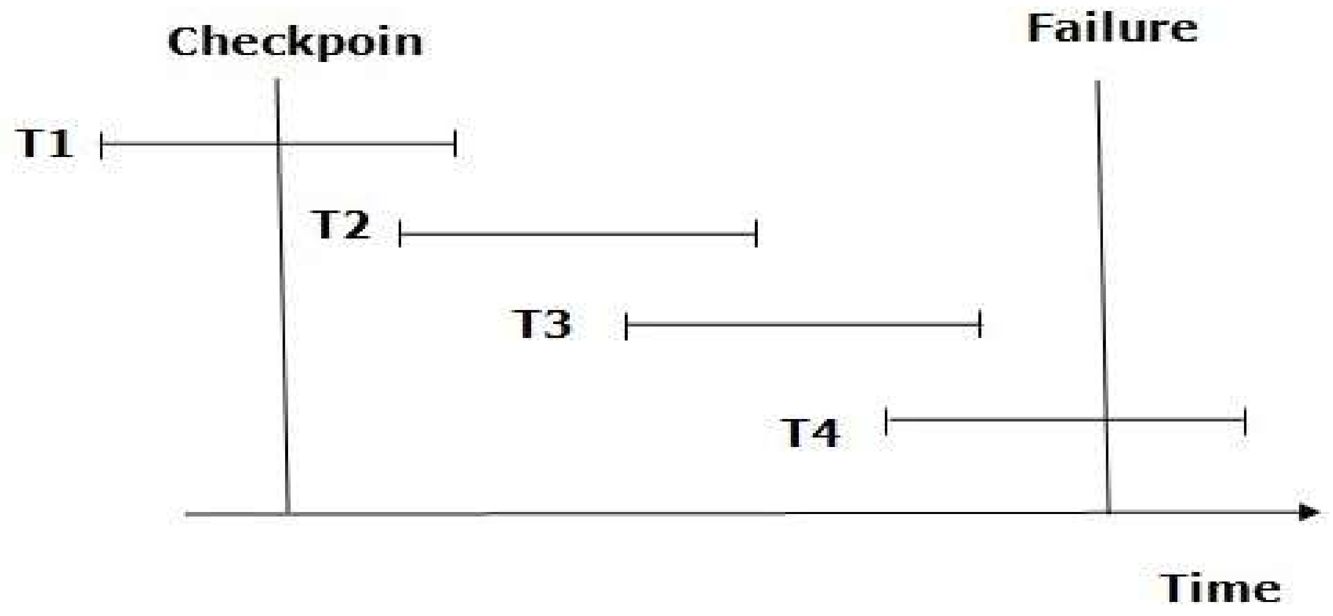
Feature	**Deferred Update**	**Immediate Update**
Update timing	Updates occur after instruction execution	Updates occur during instruction execution
Processor speed	May be faster: update occurs after instruction execution, allowing for multiple updates to be performed at once	May be slower: update occurs during instruction execution, potentially causing the processor to stall
Complexity	More complex: requires additional instructions or mechanisms to handle updates after instruction execution	Less complex: updates occur immediately during instruction execution
Consistency	May result in temporary inconsistency between data in registers and memory	Data in registers and memory are always consistent
Flexibility	May be more flexible: allows for more complex data manipulations and algorithms	Less flexible: immediate updates can limit the range of data manipulations and algorithms that can be performed

Checkpoint

1. The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk.
2. 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.
3. 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.
4. 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

In the following manner, a recovery system recovers the database from this failure:



1. The recovery system reads log files from the end to start. It reads log files from T4 to T1.

2. Recovery system maintains two lists, a redo-list, and an undo-list.

3. The transaction is put into redo state if the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>.

4. 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.

Concurrency Control

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

The concurrency control protocols ensure the *atomicity, consistency, isolation, durability* and *serializability* of the concurrent execution of the database transactions. Therefore, these protocols are categorized as:

1. Lock Based Concurrency Control Protocol
2. Time Stamp Concurrency Control Protocol
3. Validation Based Concurrency Control Protocol

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.

It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive lock:

In the exclusive lock, the data item can be both reads as well as written by the transaction.

This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

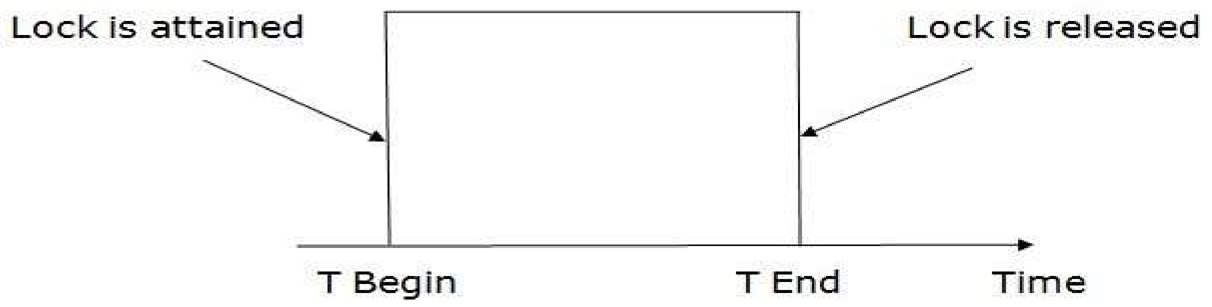
There are four types of lock protocols available:

1. Simplistic lock protocol

It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

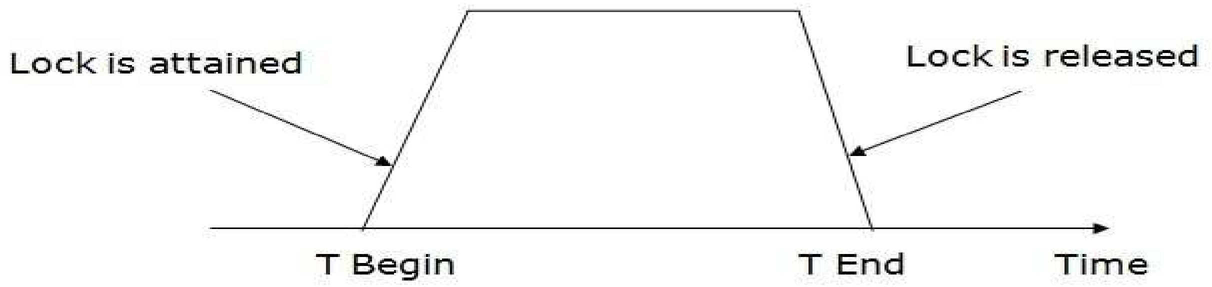
2. Pre-claiming Lock Protocol

1. Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
2. Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
3. If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
4. If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



3. Two-phase locking (2PL)

1. The two-phase locking protocol divides the execution phase of the transaction into three parts.
2. In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
3. In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
4. In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



There are two phases of 2PL:

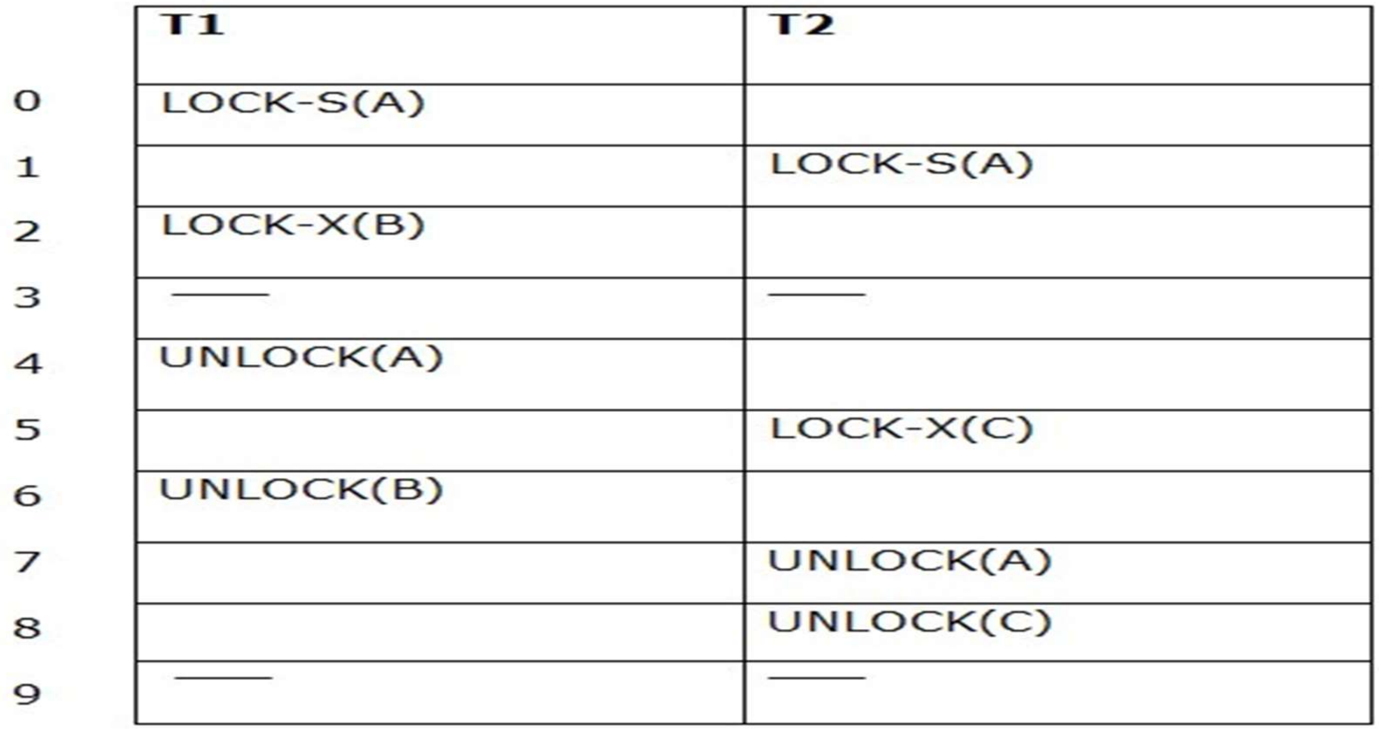
Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.

Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.



	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

The following way shows how unlocking and locking work with 2-PL.

Transaction T1:

Growing phase: from step 1-3

Shrinking phase: from step 5-7

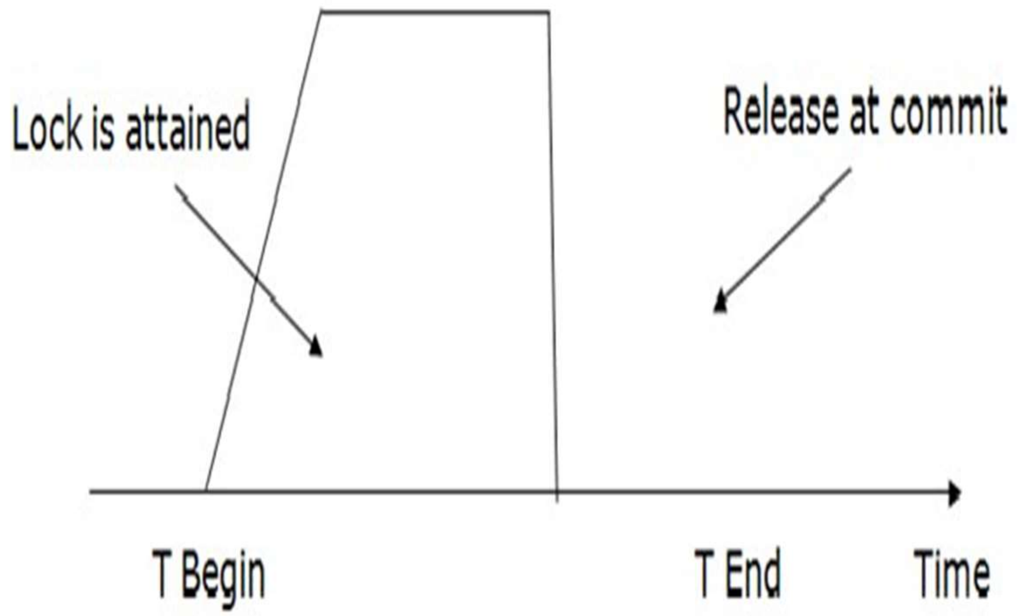
Lock point: at 3

Transaction T2:

Growing phase: from step 2-6

Shrinking phase: from step 8-9

Lock point: at 6



4. Strict Two-phase locking (Strict-2PL)

1. The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
2. The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
3. Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
4. Strict-2PL protocol does not have shrinking phase of lock release.

Timestamp Ordering Protocol

1. The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
2. The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
3. The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
4. Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
5. The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a **Read (X)** operation:

    If W_TS(X) >TS(Ti) then the operation is rejected.
    If W_TS(X) <= TS(Ti) then the operation is executed.
    Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:

    If TS(Ti) < R_TS(X) then the operation is rejected.
    If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where,

TS(TI) denotes the timestamp of the transaction Ti.
R_TS(X) denotes the Read time-stamp of data-item X.
W_TS(X) denotes the Write time-stamp of data-item X.

Advantages and Disadvantages of TO protocol:

- TO protocol ensures serializability since the precedence graph is as follows:

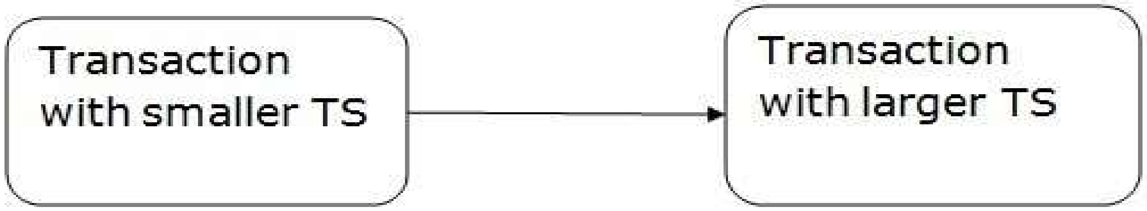


Image: Precedence Graph for TS ordering

- TS protocol ensures freedom from deadlock that means no transaction ever waits.
- But the schedule may not be recoverable and may not even be cascade-free.

Validation Based Protocol

Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases:

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.

Validation phase: In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.

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.

Here each phase has the following different timestamps:

Start(Ti): It contains the time when Ti started its execution.

Validation (T_i): It contains the time when Ti finishes its read phase and starts its validation phase.

Finish(Ti): It contains the time when Ti finishes its write phase.

This protocol is used to determine the time stamp for the transaction for serialization using the time stamp of the validation phase, as it is the actual phase which determines if the transaction will commit or rollback.

Hence TS(T) = validation(T).

The serializability is determined during the validation process. It can't be decided in advance.

While executing the transaction, it ensures a greater degree of concurrency and also less number of conflicts.

Thus it contains transactions which have less number of rollbacks.

Thomas write Rule

Thomas Write Rule provides the guarantee of serializability order for the protocol. It improves the Basic Timestamp Ordering Algorithm.
The basic Thomas write rules are as follows:

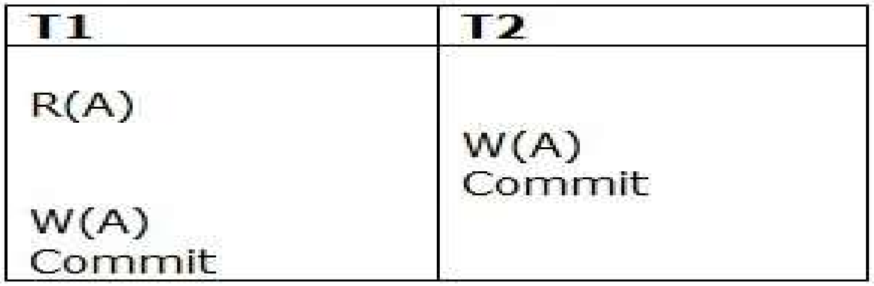
1. If TS(T) < R_TS(X) then transaction T is aborted and rolled back, and operation is rejected.

2. If TS(T) < W_TS(X) then don't execute the W_item(X) operation of the transaction and continue processing.

3. If neither condition 1 nor condition 2 occurs, then allowed to execute the WRITE operation by transaction Ti and set W_TS(X) to TS(T).

4. If we use the Thomas write rule then some serializable schedule can be permitted that does not conflict serializable as illustrate by the schedule in a given figure:

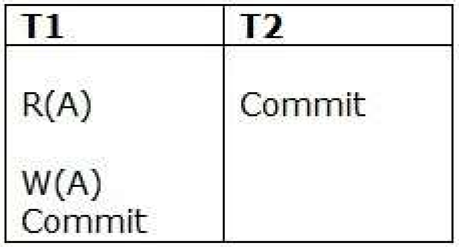
If we use the Thomas write rule then some serializable schedule can be permitted that does not conflict serializable as illustrate by the schedule in a given figure:



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

In the above figure, T1's read and precedes T1's write of the same data item. This schedule does not conflict serializable.

Thomas write rule checks that T2's write is never seen by any transaction. If we delete the write operation in transaction T2, then conflict serializable schedule can be obtained which is shown in below figure.



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

Multiple Granularity:

Let's start by understanding the meaning of granularity.

Granularity: It is the size of data item allowed to lock.

Multiple Granularity:

1. It can be defined as hierarchically breaking up the database into blocks which can be locked.
2. The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
3. It maintains the track of what to lock and how to lock.
4. It makes easy to decide either to lock a data item or to unlock a data item. This type of hierarchy can be graphically represented as a tree.

For example: Consider a tree which has four levels of nodes.

- The first level or higher level shows the entire database.
- The second level represents a node of type area. The higher level database consists of exactly these areas.
- The area consists of children nodes which are known as files. No file can be present in more than one area.
- Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.

Hence, the levels of the tree starting from the top level are as follows:

1. Database
2. Area
3. File
4. Record

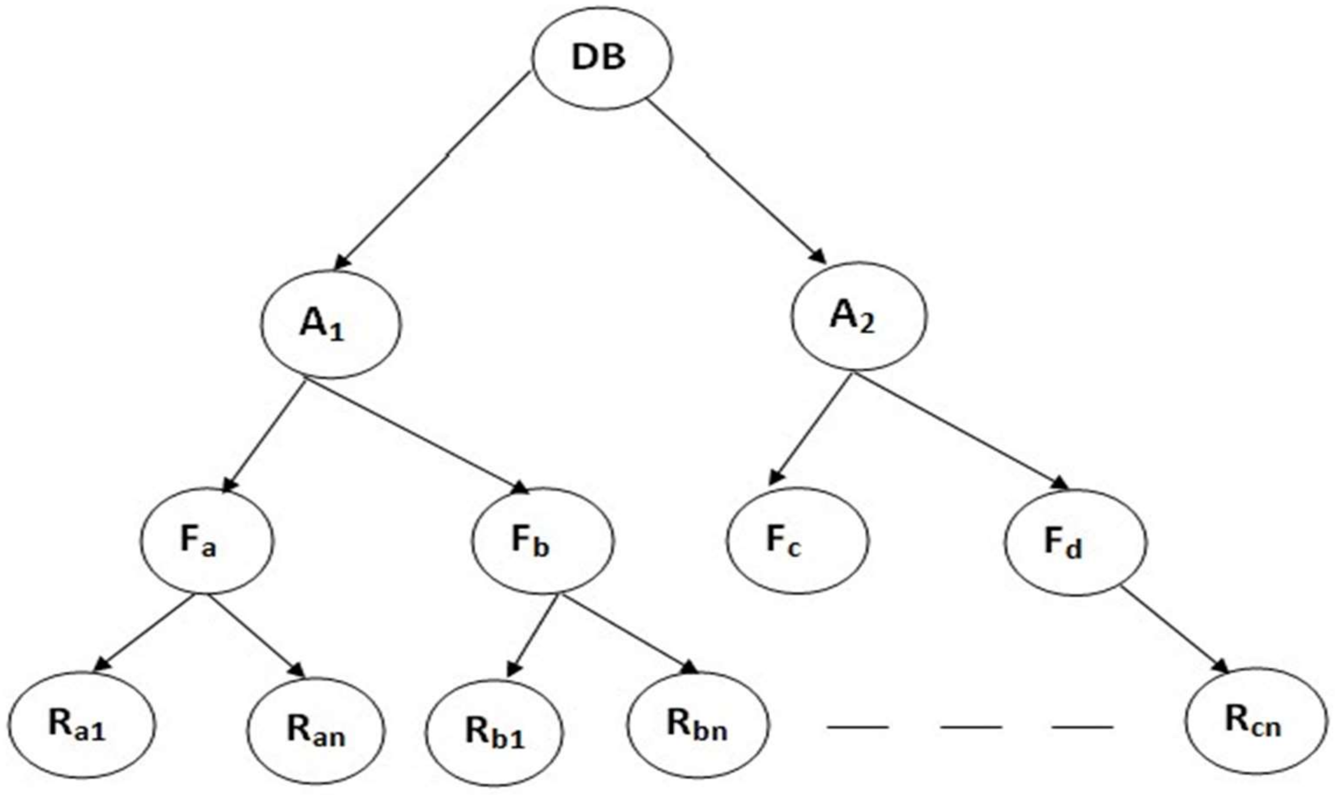


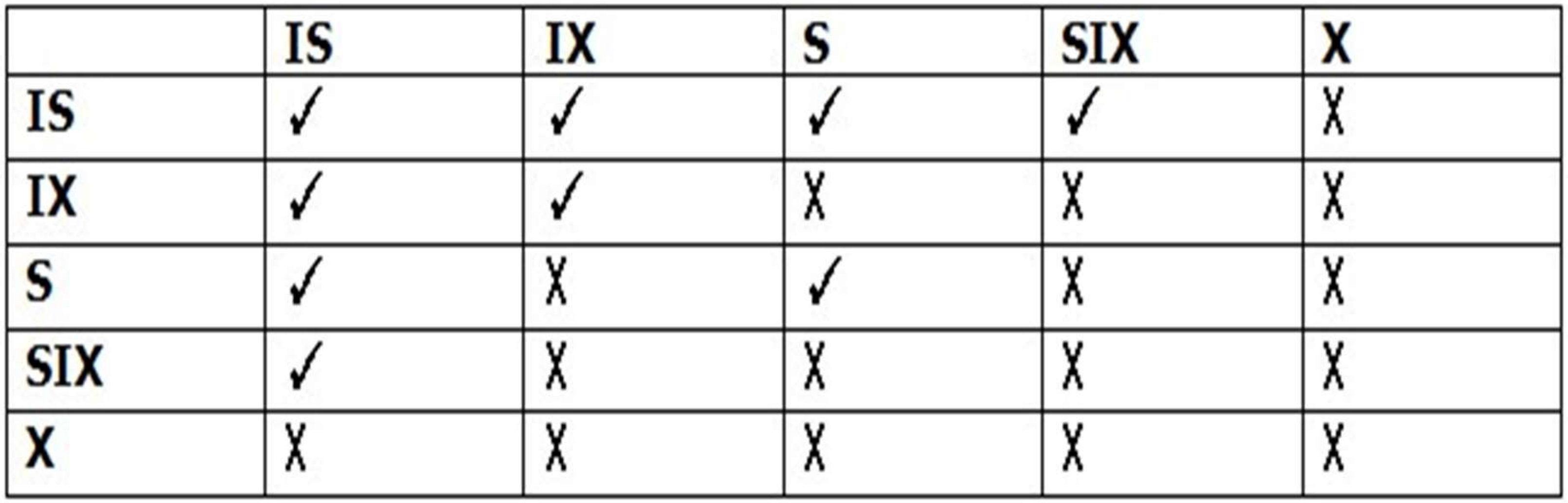
Figure: Multi Granularity tree Hierarchy

There are three additional lock modes with multiple granularity:

Intention-shared (IS): It contains explicit locking at a lower level of the tree but only with shared locks.

Intention-Exclusive (IX): It contains explicit locking at a lower level with exclusive or shared locks.

Shared & Intention-Exclusive (SIX): In this lock, the node is locked in shared mode, and some node is locked in exclusive mode by the same transaction.



	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	X
IX	✓	✓	X	X	X
S	✓	X	✓	X	X
SIX	✓	X	X	X	X
X	X	X	X	X	X

In multiple-granularity, the locks are acquired in top-down order, and locks must be released in bottom-up order.

Recovery with Concurrent Transaction:

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.

In this situation, **'checkpoint'** concept is used by most DBMS.

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.

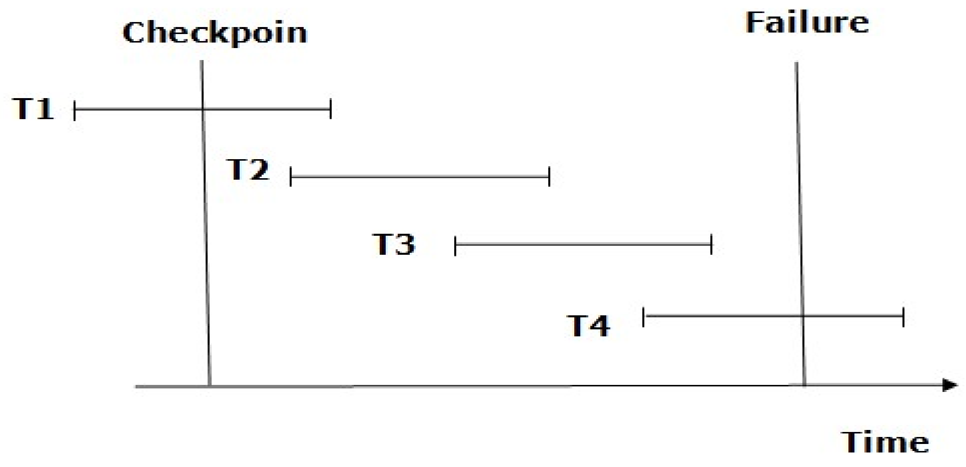
- The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log files will be created.

- 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

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 in DBMS

A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.

For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

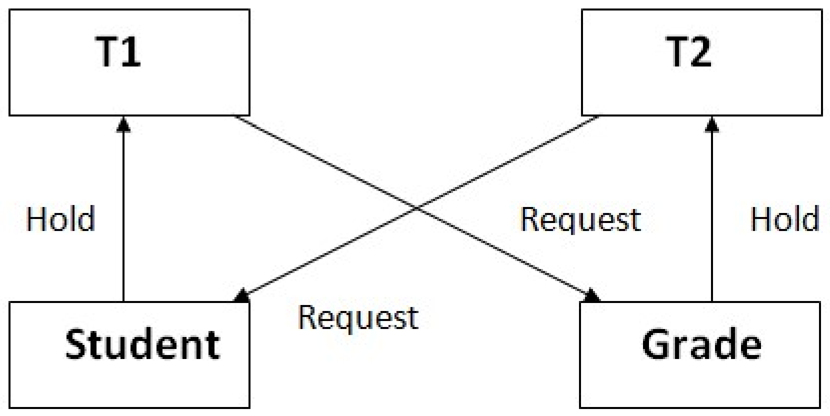


Figure: Deadlock in DBMS

Deadlock Avoidance

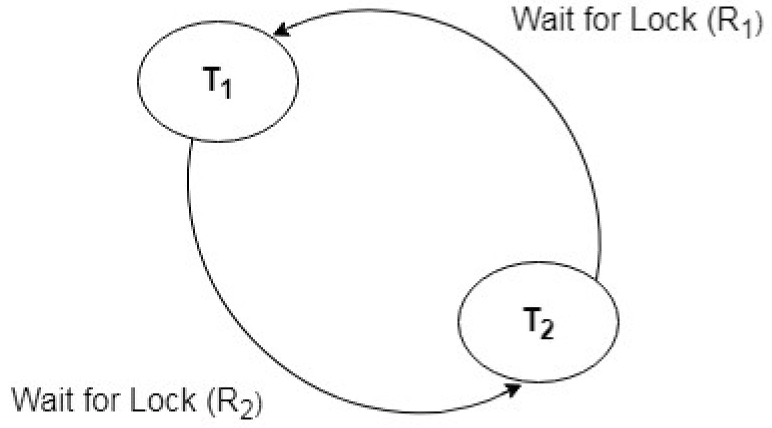
- When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resource.
- Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database. For the larger database, deadlock prevention method can be used.

Deadlock Detection

In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

Wait for Graph

- This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.
- The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.
- The wait for a graph for the above scenario is shown below:



Deadlock Prevention

•Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.

•The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.

Wait-Die scheme

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

Let's assume there are two transactions Ti and Tj and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:

1.Check if TS(Ti) < TS(Tj) - If Ti is the older transaction and Tj has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.

2.Check if TS(Ti) < TS(Tj) - If Ti is older transaction and has held some resource and if Tj is waiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.

Wound wait scheme

•In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted but with the same timestamp.

•If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.