FUCTIONAL DEPENDENCY

Page No.
Date / /

In a relational database management, functional dependency is a concept that specifies the relationship between two sets of attributes where one attribute determines the value of another attribute. It is denoted as $X \rightarrow Y$, where the attribute set of the left side of the arrow, X is called Determinant and Y is called the dependent.

roll_no	name	dep_name	dep_building
42	X	CO	A4
43	Y	IT	A3
44	Z	CO	A4
45	Z	IT	A3

roll_no → {name, dep_name, dep_building},
it can determine its subset dep_name also like roll_no → {name, dep_name},
dep_name → dep_building.

Invalid functional dependency: name → roll_no.

⇨ Armstrong's axioms/properties of F.D:

1. Reflexivity: If Y is a subset of X, then $X \rightarrow Y$ holds by reflexivity rule.
eg: roll_no, name → name is valid.

2. Augmentation: If $X \rightarrow Y$ is valid dependency, then $XZ \rightarrow YZ$ is also valid by the augmentation rule.

3. Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$ are both valid dependencies, then $X \rightarrow Z$ is also valid by transitivity rule.
eg: roll_no → dep_name & dep_name → dep_building
then roll_no → dep_building is also valid.

⇨ Types of F.D in DBMS

1. Trivial F.D: In this a dependent is always a subset of the determinant.
i.e if $X \rightarrow Y$ and Y is the subset of X then it called trivial F.D.

eg:

roll_no	name	age
42	X	17
43	Y	18
44	Z	18

Here {roll_no, name} → name is a trivial F.D since the dependent name is a subset of determinant set {roll_no, name}.
Similarly, roll_no → roll_no is also eg. trivial F.D.

2. Non-Trivial F.D: In this the dependent is strictly not a subset of determinant i.e if $X \rightarrow Y$ and Y is not a subset of X, then it is called Non-Trivial.

eg

roll_no	name	age
42	X	17
43	Y	18
44	Z	18

Here roll_no → name is a Non-Trivial F.D since the dependend name is not a subset of determinant roll_no.
Similarly {roll_no, name} → age is also a non-trivial F.D, since age is not a subset of {roll_no, name}.

3. Multivalued F.D: In this, entity of the dependent set are not dependent on each other. i.e if a → {b, c} and there exist no F.D between b and c, then it is called multivalued F.D.

eg

roll_no	name	age
48	X	17
43	Y	18
44	X	19
45	Z	18

Here roll_no → {name, age} is multivalue F.D since the dependents name & age are not dependent on each other (i.e name → age, age → name does not exist).

4. Transitive F.D: In this, dependent is indirectly dependent on determinant. ie if a → b & b → c then acc. to axiom of transitivity, a → c. This is transitive F.D.

eg.

Roll_no	name	dep	building_no
42	X	CO	4
43	Y	EC	2
44	Z	IT	1
45	X	EC	2

Here roll_no → dep & dep → building_no.
Hence acc. to axiom of transitivity roll_no → building_no is a valid F.D.
This is an indirect F.D.

5. Fully F.D: In Full F.D an attribute or set of attributes uniquely determines another attribute or set of attributes. If a relation R has attributes X, Y, Z with the dependencies X → Y and X → Z which states that those dependencies are fully functional

6. Partial F.D: In this a non key attribute dependends on a part of the composite key, rather than the whole key. If a relation R has attributes X, Y, Z where X and Y are the composite key and Z is non key attribute. Then X → Z is a partial F.D in RBDMS.

⇨ Lossy and Lossless Decomposition
A relation schema R is decomposed/divided into two or more than two relations if decomposition is loseless join.

Following conditions must hold: To check for lossless join decomposition using F.D set.

1. On taking union of attributes of R1 and relation R2 must be equal to the attribute of relation R.

   Attribute (R1) ∪ Attribute (R2) = Attribute (R)

2. On doing intersection of attributs of relation R1 and relation R2 must not be null.

   Attribute (R1) ∩ Attribute (R2) ≠ ϕ

3. The common attribute must be a key for at least one relation (R1 or R2). i.e suppose an attribute "A" is common in both R1 and R2 then either A should be Key in R1 or A should be Key in R2.

   Attribute (R1) ∩ Attribute (R2) ⟶ Attribute (R1)

   Or Attribute (R1) ∩ Attribute (R2) ⟶ Attribute (R2)

Table : R

X	Y	Z	W	P
1	244	A	I	W
2	231	B	J	X
1	534	A	K	Y
3	988	C	I	Z

Ques1 Consider a relation schema R(XYZWP) is decomposed into R1(XYZ) and R2(ZWP). Determine whether the above R1 & R2 are loseless or lossy?

Date / /

Sol Cond 1: Satisfied as attribute (R1) ∪ attribute (R2)
= Attribute (R) = (XYZWP)

Cond 2: Satisfied as Attribute (R1) ∩ attribute (R2) ≠ ∅
= (Z)

Cond 3: Not satisfied as common attribute Z is not key in any relation R1 or R2

⇨ What is dependency preserving decomposition?
It is a database design technique that focuses on breaking down a relation database into smaller components while ensuring that function dependencies within the data are preserved.
Advantages of dependency preserving decomposition
Normalization
Data Integrity
Reduction of Redundancy
Improved Query Performance
Easier Maintenance

Ques Consider a schema R(A, B, C, D) & F.D A → B & C → D. Then the decomposition of R into R1(AB) and R2 (CD) is

(a) Dependency Preserving & loseless join.
(b) loseless join but not dependency preserving
(c) Dependency Preserving but not loseless join
(d) Not dependency preserving & not lossless join

Answer C Because

$A^+ = A, B$

$B^+ = B$

$C^+ = C, D$

$D^+ = D$

$R1(AB) = A \rightarrow B$

$R2(CD) = C \rightarrow D$

NORMALIZATION

It is a process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, delation and update anomalies. So, it helps to minimize the redundancy in relations. Normals forms are used to eliminate or reduce redundancy in database tables.

Normalization of DBMS:

In DBMS, normal forms are a series of guidelines that help to ensure that the design of a database is efficient, organized and free from data anomalies. There are several levels of normalization, each with its own set of guidelines, known as normal forms.

- First Normal Form (1NF): This is the most basic level of normalization. In 1NF, each table cell should contain only

a single value, and each column should have a unique name. The first normal form helps to eliminate duplicate data & simplify queries.

Note: If a relation contain composite or multivalued attribute, it violates first N.F.

Stu-No	Stu_Name	Stu_Phone	Stu_State	Stu_country
1	Ram	123456789, 012345678	Haryana	India
2	Ram	987654321	Punjab	India
3	Suresh		Punjab	India

↓ conversion to 1NF

Stu-No	Stu_Name	Stu_Phone	Stu_State	Stu_country
1	Ram	123456789	Haryana	India
1	Ram	012345678	Haryana	India
2	Ram	987654321	Punjab	India
3	Suresh		Punjab	India

Second Normal Form: To be in 2NF, a relation must be in 1NF and relation must not contain any partial dependency. A relation is in 2NF if it has No Partial dependency, i.e no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

DOMS | Page No.
Date / /

Partial Dependency : If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

Example1 - Consider following functional dependencies in relation R(A,B, C, D)
AB → C [A & B together determine C]
BC → D [B & C together determine D]
In the above relation, AB is the only candidate key & there is no partial dependency, i.e, any proper subset of AB doesn't determine any non-prime attribute.
Prime attributes in DBMS
It is one of the attributes that make up the candidate key. For eg A & B are the prime attributes from above example.
Non-Prime attributes in DBMS
These are those attributes of the relationship that do not present in any of the possible candidate keys of the relation. For eg C & D are non-prime attributes.

Example 2 - Find the highest N.F of a relation R(A,B,C,D,E) with FD set as {BC → D, AC → BE, B → E}

Step I As we can see, (AC)+ = {A, C, B, E, D} but none of its subsets can determine all attributes of relation, so AC will be candidate key. A & C

can't be derived from any other attribute of the relation, so there will be only 1 candidate key {A C}.

Step II Prime attributes are those attributes that are part of candidate key {A, C} in this eg. and others will be non-prime {B, D, E} in this eg.

Step III The relation R is in 1NF as a relational DBMS does not allow multi-valued or composite attribute. The relation is in 2NF because BC → D is in 2NF (BC is not a proper subset of candidate key AC) & AC → BE is in 2NF (AC is candidate key) and B → E is in 2NF (B is not a proper subset of candidate key AC).

- **Third Normal Form:**

- A relational will be in 3NF if it is in 2NF & not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.

A relation is in 3NF if it holds atleast one of the following conditions for every non-trivial functional dependency X → Y.

L·H·S
Candidate Key

R·H·S
Prime attribute

} For 3NF

1. X is a super key.
2. Y is a prime attribute, i.e each element of Y is part of some candidate key.
3. Consider relation R(A, B, C, D, E)

example $A \rightarrow BC$, $CD \rightarrow E$, $B \rightarrow D$, $E \rightarrow A$

All possible candidate keys in above relation are {A, E, CD, BC}. All attributes are on right sides of all functional dependencies are prime. So it is in 3NF.

For eg. In relation R is A, since its closure determines all the other attributes. This relationship is in 3NF if and only if all the determinants of the functional dependencies of the canonical cover are superkeys, or a determinate is a prime attribute. So the schema is not in 3NF since there is the functional dependency $B \rightarrow C$ in which the determinant B is not a key, & C is not a prime attribute.

- BCNF (Boycee – Codd Normal Form (BCNF))

Rules for BCNF

1. The table should be in 3NF.
2. X should be a Super key for every functional dependency (FD) $X \rightarrow Y$ in the given relation.

Date / /

For example:

Stu_id → Stu_branch

Stu_course → {Branch_no, Stu_courseno}

Candidate Keys of the above table are {Stu_id, Stu_courseno}

Why this table is not in BCNF?

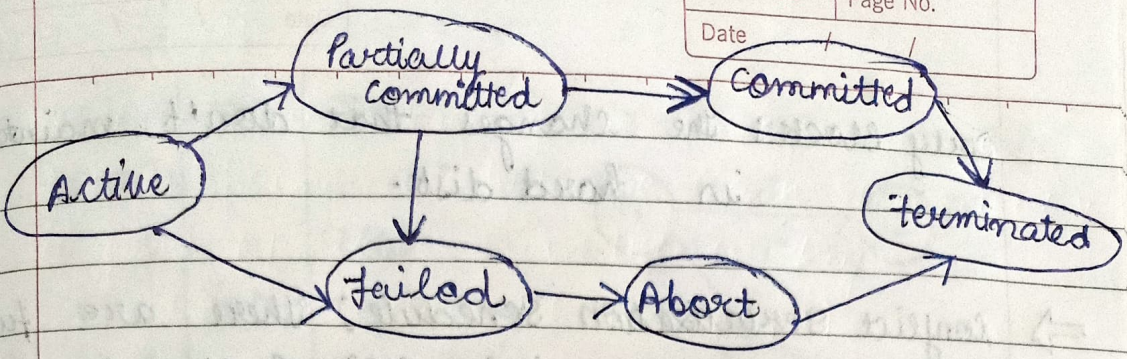
Because as we can see that neither Stu_id nor Stucourse is a superkey. As the result mentioned above clearly tell that for a table to be in BCNF, it must follow the property that for functional dependency X → Y, ~~or~~ X must be in super key and here this property fails, that's why this table is not in BCNF.

Transaction

- It is a set of operations used to perform a logical unit of work.
- A transaction generally represent change in database
- A transaction perform read & write operation.

States of Transaction:

1. Active State
2. Partially Committed State
3. Commited State
4. Abort / Roll Back
5. Terminated



Schedule: It is chronological execution sequence of multiple transactions.

Types of Schedules:

1. Serial Schedule: In this firstly one transaction get fully completed and then other will start. The 2nd transaction will not interfere until to 1st completed.

T_1	T_2
R(A)	
W(A)	
	W(B)
	R(B)

2. Non Serial Schedule: Multiple transactions executed at a time. It does not wait for one transaction.

T_1	T_2
R(A)	
	W(A)
R(B)	
	W(B)
	R(B)

DOMS | Page No.
Date / /

Dirty Block: The changes that don't maintain in hard disk.

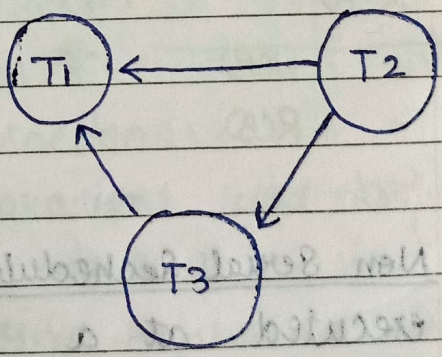
⇨ Conflict Serialization Schedule: There are two pairs Conflicting pairs (CP) & Non-Conflicting pairs (NCP)

Conditions for conflicting pairs:
1. Data Variable must be same.
2. Transactions must be different.
3. One task must be write operation.

Check whether the given schedule is conflict serialization or not.

1.

T_1	T_2	T_3
R(A)		
		R(B)
		R(A)
	R(B)	
	R(C)	
		W(B)
	W(C)	
R(C)		
W(A)		
W(C)		

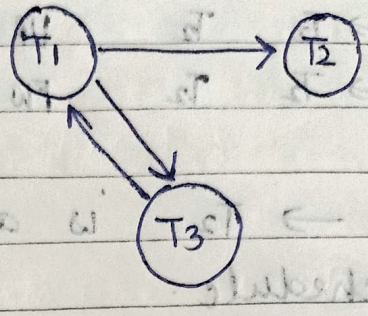


Yes, it is conflict serialization schedule.
∴ it is consistent also.

DOMS | Page No.
Date / /

2.

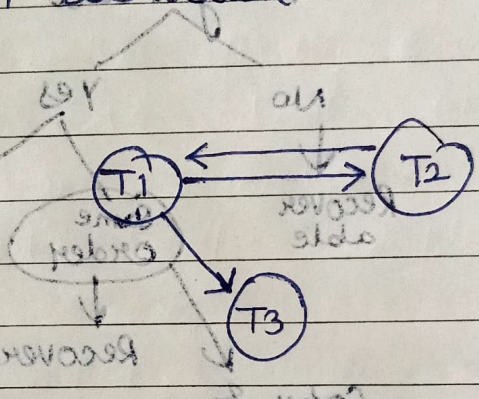
T_1	T_2	T_3
R(A)		
		R(B)
	W(C)	
W(B)		
		W(B)
	W(A)	



It is not conflict serialization schedule.

3.

T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)



It is not conflict serializable schedule.

⇨ View Serializable Schedule :

Rules : 1. Initial Read
2. Updated Read
3. Final Write

1.

T_1	T_2
R(X)	
	R(X)
W(X)	
R(Y)	
W(Y)	
	R(Y)
	W(Y)

T_1	T_2
	R(X)
	R(Y)
	W(X)
R(X)	
R(Y)	
W(X)	
	R(Y)
	W(Y)

T_2	T_1
	R(X)
R(X)	
W(X)	
	R(Y)
	W(X)
R(Y)	
W(X)	

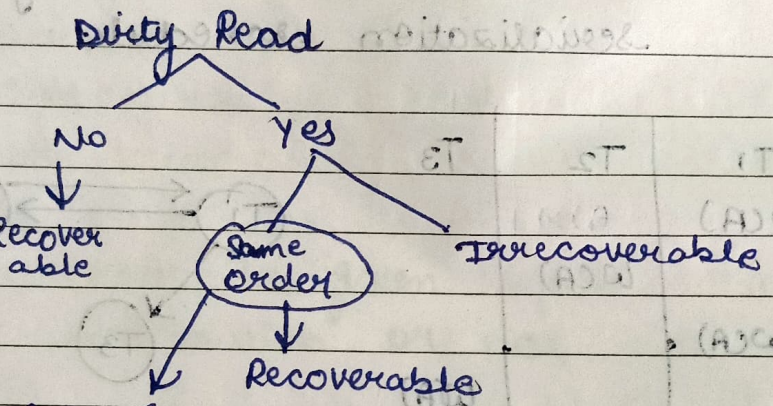
T_2	T_1
	R(X)
	W(X)
	R(Y)
R(X)	
W(X)	
R(Y)	
W(Y)	

	X	Y		X	Y		
IR →	X	T_1	IR	X	T_1	IR	T_2
UR →	T_2	T_2	UR	T_2	T_2		
FW →	T_2	T_2	FW	T_2	T_2		

$T_1 \rightarrow T_2$ ✗ $T_2 \rightarrow T_1$

$T_1 \rightarrow T_2$ is a view serializable Schedule.

⇒ Recoverable Schedule:



eg

T_1	T_2
W(A)	
	W(B) → Recoverable.
	R(A) → dirty Read
Commit	
	Commit

eg

T_1	T_2
W(A)	
	W(B) → Irrecoverable.
	R(A)
	Commit
W(A)	
Abort	

DOMS | Page No.
Date / /

⇨ Cascading Schedules: The schedule which can recover but harm our system.

eg

T_1	T_2	T_3	T_4
W(A)			
	R(A)		
	W(A)		
		R(A)	
		W(A)	
			R(A)
Abort			
	Abort		
		Abort	
			Abort

⇨ Cascadeless Schedule: Commit before Updated read.

eg:

T_1	T_2
W(A)	
	W(A)
	W(B)
W(B)	
	C
C	

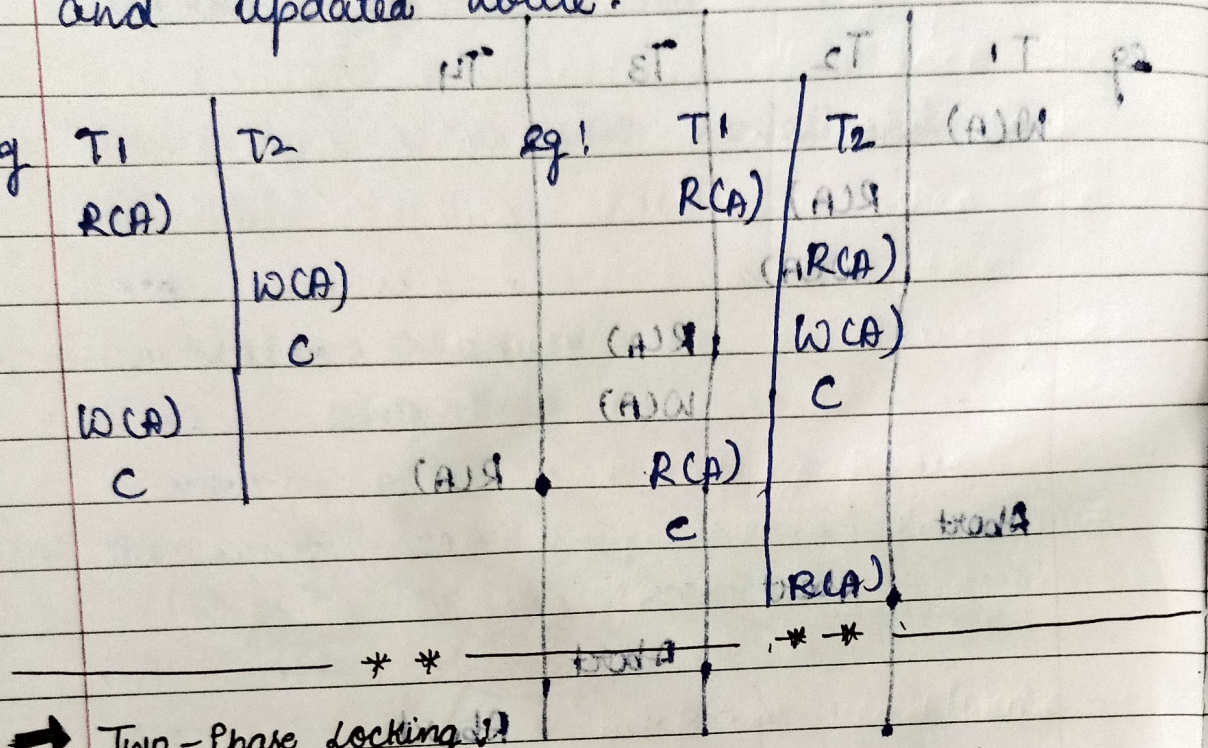
eg:

T_1	T_2
R(A)	
	R(A)
	W(A)
	C
R(A)	
C	

eg

T_1	T_2
W(A)	
C	
	R(A)

⇒ Strict Schedule: deal with read & write operation. Commit: before updated read and updated write.



➡ Two-Phase Locking:

Locking in a database management system is used for handling transactions in databases. The two phase locking protocol ensures serializable conflict schedules. A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

Types of locks used in transaction control:

- Shared lock: Data can only be read when a Shared lock is applied. Data cannot be written. It is denoted as lock-S.

- **Exclusive Lock**: Data can be read as well as written when an exclusive lock is applied. It is denoted as lock-X

Properties:

eg:

T_1	T_2
Lock-X(A)	
R(A)	
W(A)	
Unlock(A)	
	Lock-S(B)
	R(B)
	Unlock(B)
Lock-X(B)	
R(B)	
W(B)	
Unlock(B)	
	Lock-S(A)
	R(A)
	Unlock(A)

- If we do unlocking inconsistency will arise, if we don't unlock then concurrency will be poor.
- We require that transaction follow some set of rules for locking and unlocking of data item. eg: 2PL, graph based.

- We say a schedule is legal under a protocol if it can be generated using the rules of the protocol.

Date / /

Two phases of locking are:

- Growing Phase: In the growing phase, the transaction only obtains the lock. The transaction cannot release the lock in the growing phase. Only when the data changes are committed the transaction starts the shrinking phase.

- Shrinking phase: In shrinking phase transaction can only release locks but cannot obtain any lock.

- Transactions can perform read/write operation both in growing/shrinking phase.

Note: If lock conversion is allowed, then upgrading of lock (from S(a) to X(a)) is allowed in growing phase, and down-grading of lock (from X(a) to S(a)) must be done in shrinking phase.

Lock Point: The point at which the growing phase ends i.e when a transaction takes the final lock it needs to carry on its work.

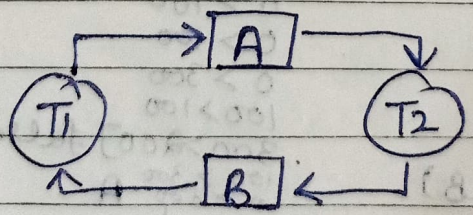
DOMS | Page No.
Date / /

- Dead lock in 2PL

eg:

T_1	T_2
Lock-X(A)	
R(A)	
W(A)	
	Lock-X(B)
	R(A)
	W(B)
Lock-X(B)	
	Lock-X(A)

- In the above given example T_1 is holding lock for A, while T_2 is holding lock for B. T_1 ~~trans~~ attempts to aquire the lock for B, while T_2 attempts to aquire the lock for A. This is a deadlock situation. T_1 has something which T_2 wants and T_1 wants something which T_2 has.



Date / /

⇒ Time Stamp Ordering Protocol

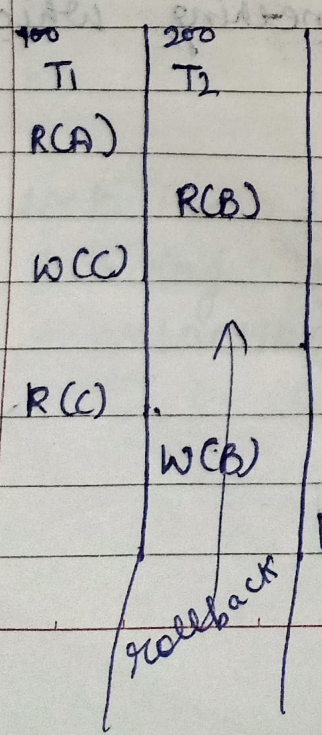
- Unique value assign to every transaction.
- Tells the order (when they enters into system)
- Read_TS (RTS) = Last (latest) transaction no. which performed read successfully.
- Write_TS (WTS) = Last (latest) transaction no. which performed write successfully.

Rules:

1. Transaction Ti issues a Read(A) operation
   (a) if WTS(A) > TS(Ti), Rollback Ti
   (b) Otherwise execute R(A) operation
   Set RTS(A) = Max{RTS(A), TS(Ti)}

2. Transaction Ti issues Write(A) operation
   (a) if RTS(A) > TS(Ti) then rollback Ti
   (b) if WTS(A) > TS(Ti) then rollback Ti
   (c) Otherwise execute Write(A) operation
   Set WTS(A) = TS(Ti)

eg:



0 > 100
0 > 200
0 > 100
0 > 100
0 > 300
100 > 100
300 > 200 false
100 > 300
0 > 300

	A	B	C
RTS	100 ∅	300 ∅	100 ∅
WTS	300 ∅	0	∅ 100