DBMS

Made with Goodnotes

Highlighting patterns:

Common highlighting of important points

Common highlight but in different color to make for color mixing

Doubtful highlighting (need to get highlighted part verified)

I tend to forget this so need to remember it better.

Note-taking patterns.

Headings

Normal text

Diagrams and visual information

Note / info panel

Doubts

Other (PYQs, Questions)

Made with Goodnotes

DBMS (+8 marks)

ER model, Relational model - 1 problem

Relational algebra, Tuple calculus - 1 problem

SQL, Integrity constraint - 1 problem

Transactional & concurrency control - 1 problem

File organization, indexing, B, B^+ trees - 1 problem

→ Time consuming

→ Multi-level indexing can also be problematic

Data

↓

Related data

for ex: s(Ram, 101, 21) ⇒ This is a record

↓

A collection of such records forms a relation / table

Eg: (Ram, 101, 21)
(Shyam, 102, 21)
(Gita, 103, 20)

↓

R_1 (Name, Rollno, Age) ⇒ This representation is called relational schema

Name of the table

↓

A collection of such relations forms a database

Eg: I can have another table with schema R_2(student, courses)

↓

The above examples look like a database for a college. We'll call it college.db

Made with Goodnotes

Note: In the above example,

These are known as attributes/columns

Name	Roll no.	Age
Ram	101	21

This is known as a tuple/row.

Also, in a relation schema, the unique attribute is underlined. In the above case, it is roll.no.

A row differs from a tuple in such a way that a row always means the entire record while a tuple can be a subset of the record as well.

The relation is a set:

$R_1 \{ (\text{Ram}, 101, 21),$
$(\text{Shyam}, 102, 21),$
$\vdots$
$\}$

Order cannot be changed

The number of elements in this set is called the quality of the set?

probably meant "cardinality"

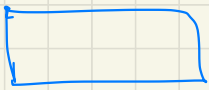
Made with Goodnotes

ER-Model

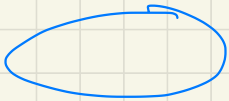
→ Diagrammatic representation of collection of objects (called entities) and relationship among the objects and their entities. → Verify if this is "other entities".

→ ER Model Components

- Entity Set
- Entity attribute
- Relationship b/w the entity



(An entity set is represented by a rectangle)



→ We represent entity sets and not single entities (although the terms are used interchangeably.)

Entity

→ Any real world object that is distinguishable from other objects based on their value (attributes) is an entity.

→ The diff. b/w an entity and an object is the distinguishability part.

→ Every entity needs to have atleast one attribute so it can be distinguished from other entities.

Made with Goodnotes

→ Entity is of 2 types

Entity

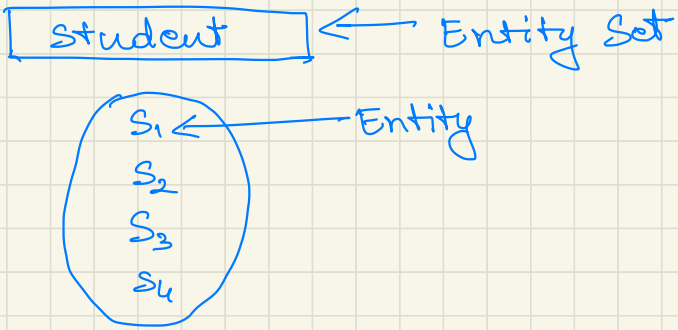
Tangible
- → Can touch
- → Physically exists
- → Eg: chalk

Intangible
- → Cannot touch
- → Logically exists i.e. logical entity.
- → Eg: Account number 1413279763

Entity Set

→ A collection of entities of the same type is called an entity set.

→ Entity sets are represented in the ER Model as:



Made with Goodnotes

Entity Attribute

→ Characteristics or properties of an entity which distinguish one entity from another.

→ Eg: Students have attributes:
- ⇒ Name
- ⇒ Roll no.
- ⇒ Mobile no.

→ Attributes are called columns in Table

→ student (Name, Roll no, Mobile no)

→ In relational schema, we call them attributes and not columns.

→ In the foll. section, we define the diff. types of attribute.

→ An attribute can be of multiple types.

For ex: An attribute can be simple, single-valued and stored. Like a roll. no.

→ But an attribute cannot belong to conflicting types.

Eg: An attribute cannot be simple and composite.

Made with Goodnotes

Attribute types

Simple attribute

→ It cannot be broken down further

→ It is also called an atomic attr.

Eg: Gender.

Represented by oval

Gender

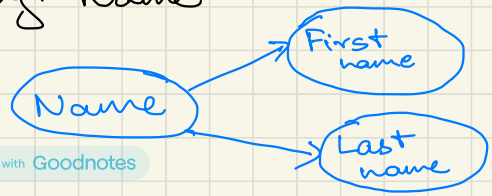
Composite attribute

→ Which can be broken down further

→ Also called inatomic

→ Represented by oval connecting with more ovals.

→ Eg: Name



Single value

→ Have only one value.

→ Represented by oval

→ Eg:

Aadhaar no.

Multivalued

→ Can have more than one value in the attribute.

→ Eg: mobile no.s, email ids

→ Represented by double ovals.

mobile no.

email

Stored value

→ Eg: DOB

→ This is value that needs to be stored.

→ Represented by oval

→ Eg: DOB

Derived value

→ Can be derived

→ Represented by dashed oval

→ Eg: Age

Age

Made with Goodnotes

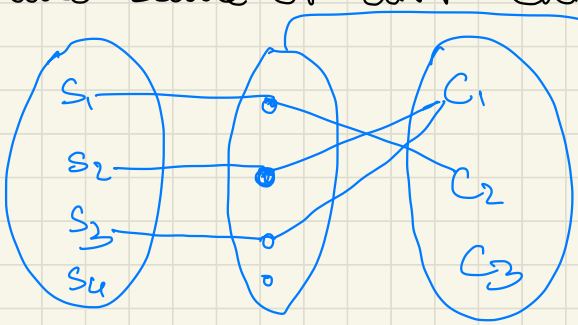
Note: Whether an attribute is simple or composite is dependant on our view of it.

Eg: Roll no. of the form 16H61A0505 can be considered atomic if we want to but if we want, we can also consider it to be composite.

Relationship b/w entities

Relation or association b/w 2 or more entities of the same or diff entity set.

Eg:



This entire thing is your relationship set while its constituent connections are your relations

Relationship set: A collection of similar type of relationships.

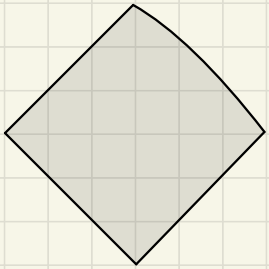
Note: A relationship represents the relation b/w 2 entities while a relationship set represents the relation b/w 2 entity sets. So, in ER model, we use relationship set.

is this actually "function"?

Unlike in a function, in a relation it is not necessary that every element should participate in the relation. Eg: S_4 in the above example

Made with Goodnotes

In E-R model, relationship set is represented by diamond/rhombus.



Example ER-model

Let's construct the example ER Model for "students enrollment in courses"

A student entity set will probably have the foll:
- → Name (composite)
- → DOB
- → Age (derived)
- → Roll no.
- → Mobile no. (multivalued)

A course entity set will probably have the foll:
- → Name
- → Course Id

The relationship b/w the 2 will be through enrollment.

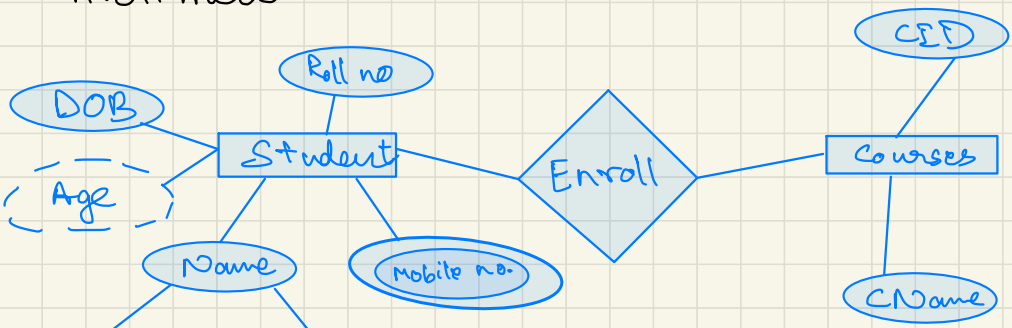
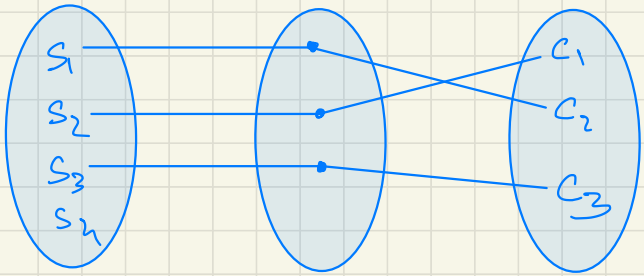


Fig: ER-model for student enrollment in courses

Made with Goodnotes

We can also create a set diagram for the above ER-model:



From the ER-model, a couple of things are not clear:

1) Is every entity in the "student" entity set supposed to enroll?

This issue is tackled by the participation constraint which is of 2 types:

a.) Total participation (all entities need to participate.)

b.) Partial participation (not all entities need to participate.)

2.) Can a single "student" entity enroll in multiple course entities?

This issue is addressed by cardinality ratios which have the foll. types:

a.) One to one
b.) One to Many
c.) Many to One
d.) Many to Many

Made with Goodnotes

Relationship set components

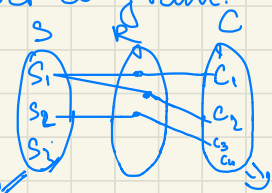
1) **Degree of relationship**: The no. of entity sets that the relation is connected to.

In the above example, the relation is connected to 2 entities. So, it's binary.

We can have unary (self join), binary, ternary . . . . n degree for a relationship.

2) **Cardinality ratio**: It show how many times (atmost) an entity can participate in a relationship.

set diagram:



ER model

atmost participation = 2 (many)

at most participation (1)

In ER model, we only deal with 2 values:

a) 1
b) Many

These values represent the atmost no. of relationships that an entity set's members can have.

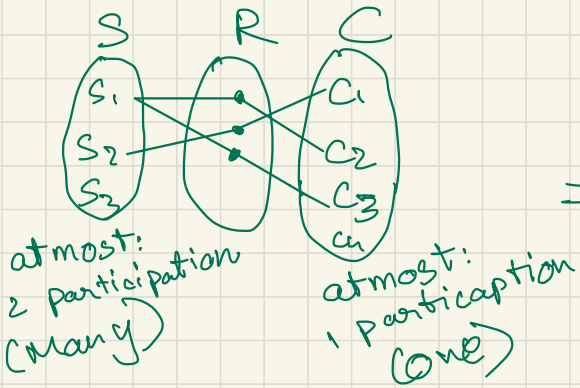
Note that the keyword here is "atmost". It represents the maximum

Made with Goodnotes

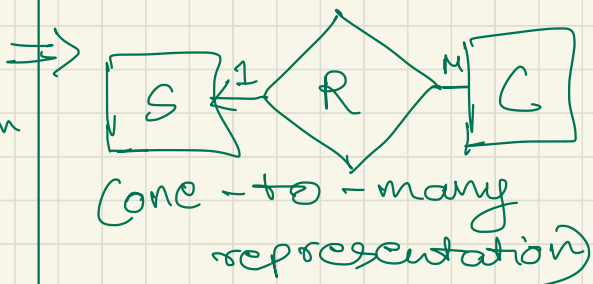
no. of relationships an entity can have. But the minimum no. of relationships is not defined. So, an entity can have 0 relationships as well.

Eg: In the above set diagram, S_3 has no relationships. Similarly, C_4 also does not have any relationships.

Note: When converting from a set diagram to an ER model, note that the representation of 1 and Many changes.



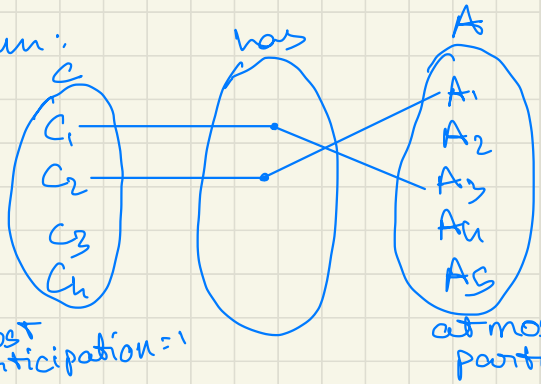
Convert to ER:



1:1 cardinality ratio:

Eg of this is a citizen-aadhar relationship

Set diagram:



Made with Goodnotes

Here, the atmost participation of every citizen is 1. Why? Since once a citizen has received an aadhar, they cannot have a 2nd aadhar.

It is valid that citizens like c_3, c_4 don't have aadhars (haven't applied for it) but cardinality only cares about atmost values.

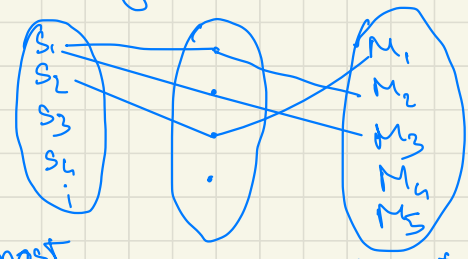
Similarly, not every aadhar that is generated needs to be assigned. But once assigned (participated), it cannot be assigned to another person as well.

ER model:

1-Many cardinality ratio:

Eg: Student have multiple mobile no.s:

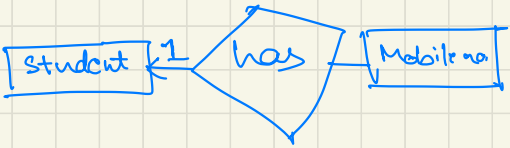
Set diagram:



atmost participation = 2

atmost participation = 1

Convert to ER-model:

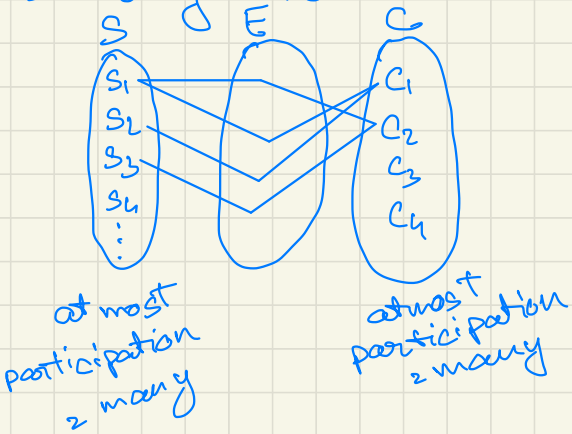


Made with Goodnotes

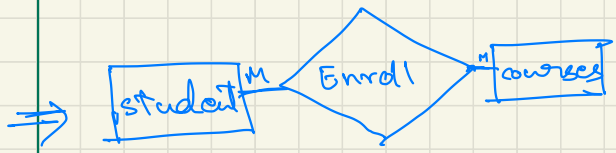
Many - Many cardinality ratio:

Eg. Student enrolled in courses

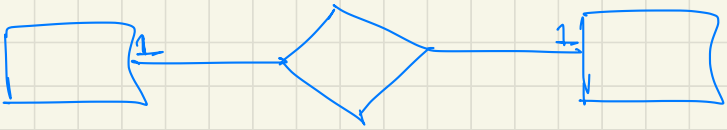
Set diagram:



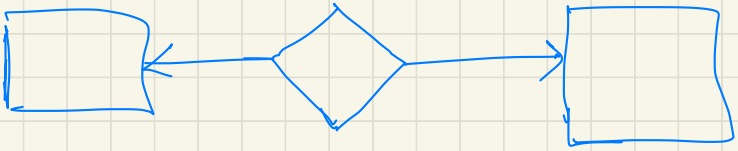
Convert to ER model:



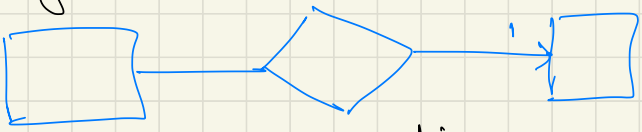
You can denote 1—1 relationship using an arrow or a 1



(or)

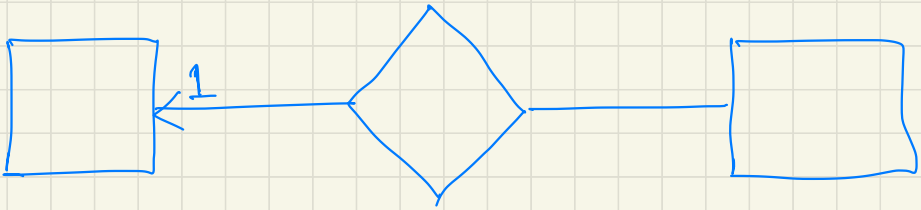


You can denote M-1 (or) 1-Many by not adding an arrow/1 on the many entity side



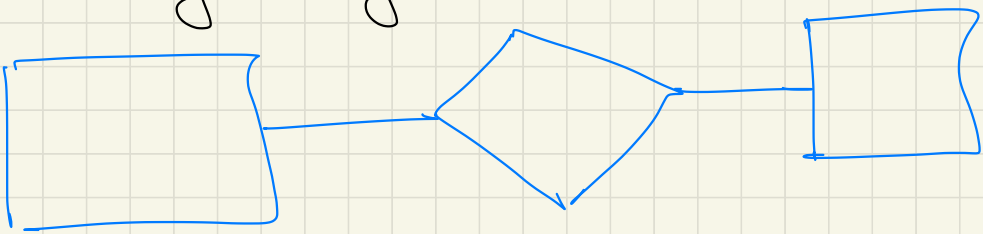
Many-1 ratio

Made with Goodnotes



1 - Many ratio

Many - Many can be denoted by not adding any arrows



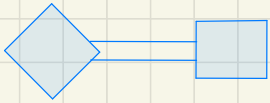
The student - courses ER model was depicting a Many - Many ratio.

Remember the notation as follows:

Many - no arrow/no.
One - add arrow/no.

3.) Participant Constraint

Total participation is denoted by double-line



Partial participation is denoted by single line

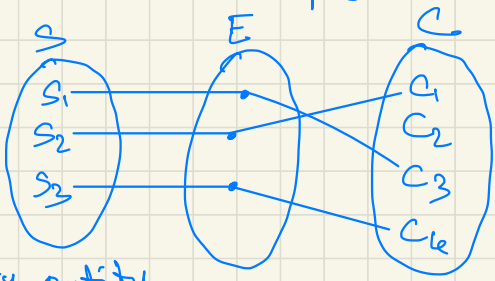
Made with Goodnotes

Participation constraint tells us if every entity of entity set participates in relationship set.

If each entity has participated, it is total participation

If some entity is not participating then it is partial participation

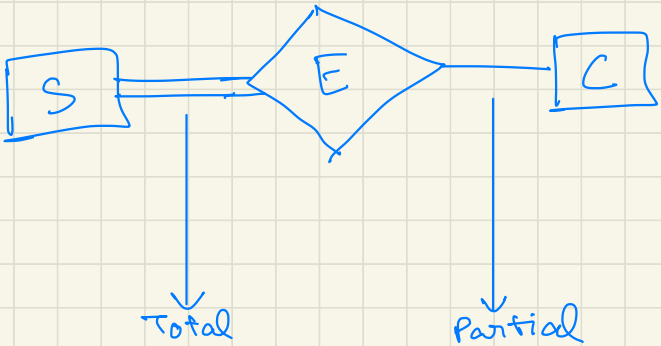
Student - course example



Every entity in S participated

Some entity in C did not participate

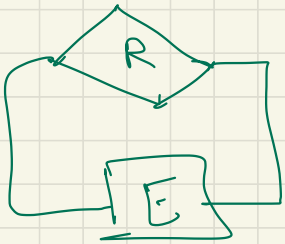
Convert to ER model



Made with Goodnotes

Note: The degree of a relationship set is different from the degree of a relation. The degree of a relation denotes the no. of columns in it.

Unary relationship set:



Also called recursive relationship

Note: A relationship set cannot exist without an entity but an entity can exist without a relationship set.

Note: All the elements in the above ER models will be converted to tables.

Eg:

In the above ex, each entity set and the relationship set will be a table.

Since the relationship set is a table, it can also have its own attributes.
For ex: Date of enrollment.

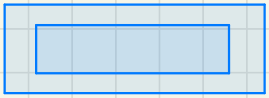
Such an attribute on a relationship is called a descriptive attribute.

Made with Goodnotes

Weak Entity → Reread/Rewatch

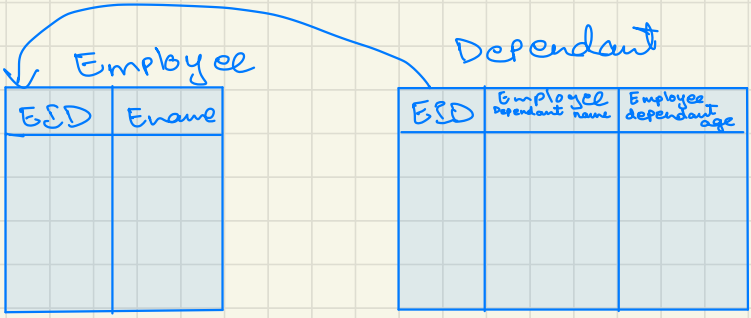
Lecture 2
21st July, 2023

→ It is dependant on strong entity.
→ It is represented by a double rectangle.



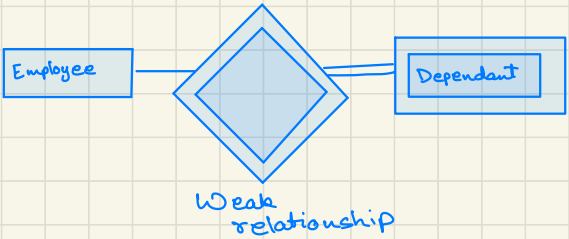
→ Primary key of strong entity is part of keys of weak entity. The key can be stored in atomic or composite form by the weak/dependant entity set but it will be stored.

Eg:



→ A strong entity can exist without a weak entity but a weak entity cannot exist without a strong entity.

→ A weak entity set always has total participation because of it's dependancy.

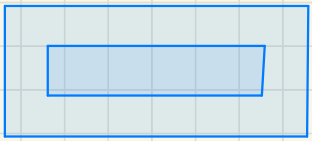


Made with Goodnotes

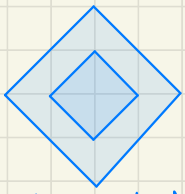
→ A relationship between a strong entity and a weak entity is called a weak relationship.

→ While we might or might not make a table for strong relationships (b/w strong entities), we DO NOT make a separate table for weak relationships since the weak table already stores all the necessary info.

→ A weak relationship is represented by a double diamond/rhombus.



Weak entity



Weak relationship

2005 - IT

Store rent payment information in Lodging.

2012 - CS

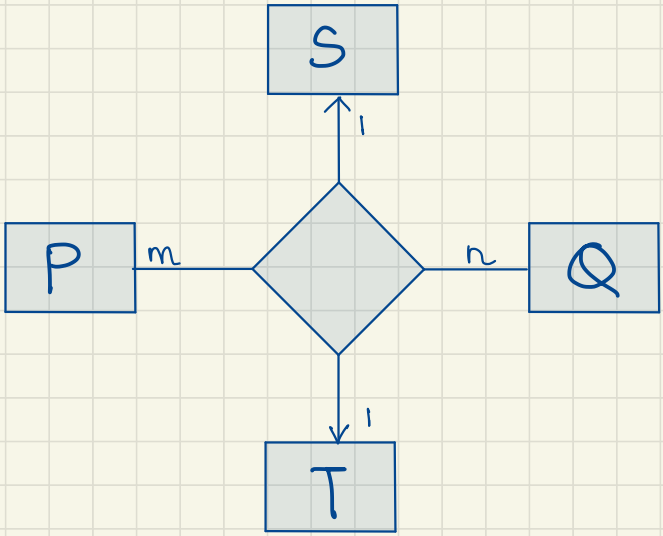
Q) A.) Attribute can have more than one value.
B.) Attribute of entity can be composite.
C.) In relational table, attribute can have multiple values.
D.) In relational table, attribute can have exactly one value or a NULL value.

Made with Goodnotes

2020-CS-2

Q.) How to represent the supporting many-one relationships of a weak entity set in ER diagram?
→ Diamond with double border.

Q.)



Which is not correct relation (table)?

I

P	Q	S	T
P_1	Q_1	S_1	T_1
P_1	Q_1	S_1	T_2

II

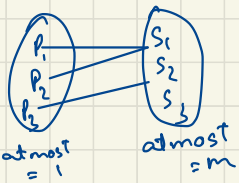
P	Q	S	T
P_1	Q_1	S_1	T_1
P_1	Q_1	S_2	T_2

III

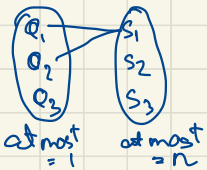
P	Q	S	T
P_1	Q_1	S_1	T_1
P_1	Q_2	S_1	T_1

Lecture 2 - 1:20:00 for explanation

P - S ⇒ m - 1 ⇒



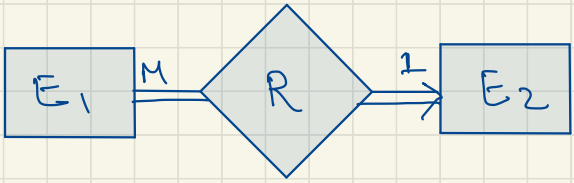
Q - S ⇒ m : 1

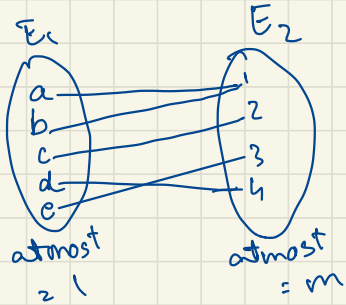


Made with Goodnotes

2018 - CS - 1

Q.)





$|E_1| > |E_2|$

(Cardinality of E_1 is greater than cardinality of E_2. So, no. of elements in E_1 > no. of elements in E_2)

Functional Dependancy (FD):

→ FD is a method which describes the relationship b/w attributes i.e. it describes attribute dependancy.

→ It is represented as:

$$A \longrightarrow B$$

This is read as

A functionally determines B

(or)

B functionally depends on A.

Made with Goodnotes

Valid / Invalid FD:

→ A functional dependancy $A \longrightarrow B$ is valid if:

(i) every element in A is distinct

A	B
1	x
2	x
3	x
4	y

(ii) for every repeated value of A, value of B is also repeated.

A	B
1	x
2	y
3	y
1	x

An example of (ii) is DOB ⟶ Age i.e. Age is functionally dependant on Date Of Birth. So, when Age is calculated for two people with the same DOB, their age will also be the same.

Note : If in an FD $A \longrightarrow B$, A has all distinct values, then it is valid.

If a functional dependancy does not satisfy the above 2 conditions, then it is invalid.

Made with Goodnotes

Shortcut to determine validity of FD:

① $A \longrightarrow B$

All values are distinct

Valid

② $A \longrightarrow B$

All values are the same

Valid

If ① & ② are not satisfied, we check for invalidity.

$A \longrightarrow B$

Check a repeated value

B returns distinct result

$\Rightarrow$ Invalid

Made with Goodnotes

2000 - 3

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	2

A.) $XY \rightarrow Z$ and $Z \rightarrow Y$

Let's take $Z \rightarrow Y$

(i) Does Z have all unique values? No (2, 3 repeated)

(ii) Does Z functionally determine the same value for Y for every repeated value in Z? No.

$Z(2) \longrightarrow Y(4)$

$Z(2) \longrightarrow Y(2)$

B.) $YZ \rightarrow X$ and $Y \longrightarrow Z$

$Y \longrightarrow Z$

(i) Y have all distinct values? Yes So, valid.

$YZ \longrightarrow X$

(i) YZ have all distinct values? Yes. So, valid.

C.) $YZ \longrightarrow X$ and $X \longrightarrow Z$

Take $X \longrightarrow Z$

(i) X all distinct? No.

(ii) X repeated give same Z value? No

$X(1) \longrightarrow Z(2)$

$X(1) \longrightarrow Z(3)$

D.) $XZ \rightarrow Y$ and $Y \longrightarrow X$

$Y \longrightarrow X$

(i) Y have all distinct values? Yes. Valid.

$XZ \rightarrow Y$

(i) XZ have distinct values? No. X(1) Z(3) repeated.

(ii) For repeated values, does XZ functionally determine same Y value? No. So, invalid.

$X(1)\,Z(3) \longrightarrow 5$

$X(1)\,Z(3) \longrightarrow 6$

Made with Goodnotes

Q.)

A	B	C
1	2	4
3	5	4
3	7	2
1	4	2

1.) $A \rightarrow B$, $BC \rightarrow A$
2.) $C \rightarrow B$, $CA \rightarrow B$
3.) $B \rightarrow C$, $AB \rightarrow C$
4.) $A \rightarrow C$, $BC \rightarrow A$

As a first step, just check if the attribute with distinct values is on left side. Since that will definitely be valid.

Q.)

A	B	C	D	E
a	2	3	4	5
2	a	3	4	5
a	2	3	6	5
2	2	3	6	6

1. $A \rightarrow BC$
2. $C \rightarrow DE$
3. $DE \rightarrow C$
4. $BC \rightarrow A$

C has all same values. So, if it's functionally dependant on any value (i.e. it is on the right side) it can be valid (if it is the only dependant, it will be definitely valid.)

Trivial and Non-trivial FD:

FD is of 2 types

FD

Trivial (always valid)

FD $X \rightarrow Y$ is called trivial if Y is a subset of X ($Y \subseteq X$)

Eg: $AB \rightarrow B$

$A \rightarrow A$

$A \subseteq A$

$B \subseteq AB$

Non-trivial (may or may not be valid)

An FD $X \rightarrow Y$ is non-trivial if $Y \not\subseteq X$.

Eg: $AB \rightarrow C$

$AB \rightarrow BC$

We deal with non-trivial functional dependancies.

Made with Goodnotes

Armstrong's Axioms of FD

Lecture-3
3rd Aug, 2023

An axiom is a self-evident principle that is accepted as true without proof.

Primary Axioms

There are 3 primary axioms, abbreviated as RAT:-

① **Reflexivity:**

If $B \subseteq A$, then $A \longrightarrow B$ is always valid.

Reflexivity is just a diff. name for a trivial FD.

② **Augmentation:**

If $A \longrightarrow B$, then $AC \longrightarrow BC$

Eg: Roll no $\longrightarrow$ name, then Roll no, Age $\rightarrow$ Name, Age

The term "augmentation" means increasing in size.

Augmentation here, basically just adds a trivial FD spin on a FD.

$A \rightarrow B$ is the normal FD $\Rightarrow$ $AC \rightarrow BC$ is the FD with a trivial spin

Made with Goodnotes

③ Transitivity:

If $A \rightarrow B$ & $B \rightarrow C$, then $A \rightarrow C$

Secondary Axioms:

① Union

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

② Decomposition

If $X \rightarrow YZ$, then $X \rightarrow Y$ & $X \rightarrow Z$

③ Pseudo Transitivity

If $X \rightarrow Y$ & $WY \rightarrow Z$, then $WX \rightarrow Z$

This can be derived from:

$X \rightarrow Y$

Using Augmentation:

$WX \rightarrow WY$

Now, we have

$WX \rightarrow WY$ & $WY \rightarrow Z$

Using transitivity:

$WX \rightarrow Z$

Made with Goodnotes

④ Composition

If $X \longrightarrow Y$ & $Z \longrightarrow W$ then, $XZ \longrightarrow YW$

This can be derived from:

$X \longrightarrow Y$ & $Z \longrightarrow W$

Using Augmentation

$XZ \longrightarrow YZ$ & $Z \rightarrow W$

Using Augmentation again:

$XZ \longrightarrow YZ$ & $YZ \longrightarrow YW$

Using transitivity:

$XZ \longrightarrow YW$

Note: Using the rules marked with *, we can determine pseudo-transitivity and composition. So, it is better to remember these 4 for exam.

Made with Goodnotes

Closure of Attribute(s)

The closure of an attribute A is the collection of attributes that can be determined by A either directly or indirectly.

Denotion:

$$\text{Closure}(A) = A^+ \quad \text{or} \quad (A)^+$$

A closure is never empty. Even if no other attributes functionally depend on A, A^+ will atleast contain A itself (reflexivity).

Given the relation schema

$$R(ABCDEF)$$

and FD

$$FD = \{ A \rightarrow B, \; C \rightarrow DE, \; AC \rightarrow F, \; D \rightarrow AF, \; E \rightarrow CF \}$$

where A, B, C, D, E & F are attributes, let's find their closures

$B^+ = \{B\}$

$B \subseteq B$

reflexivity

$A^+ = \{A, B\}$

$A \rightarrow B$

$A \subseteq A$

We cannot find any more attributes using only A & B

So, closure(A) = $\{A, B\}$

Made with Goodnotes

$C^+ = \{C, D, E, A, F, B\} = R$

$A \rightarrow B$

$C \rightarrow DE$ (using decomposition)

$C \rightarrow D$

$C \rightarrow E$

$D \rightarrow AF$ (decomposition)

$D \rightarrow A$

$D \rightarrow F$

Because C is able to determine all attributes, C is a candidate key.

$E^+ = \{E, C, F, A, B, D\} = R$

reflexivity

$E \rightarrow CF$ (decomposition)

from closure (C) i.e. C^+ because $E \rightarrow C$

Because E determines C which in turn is able to determine all other attributes, E is also a candidate key.

$D^+ = \{D, A, F, B\}$

$D \rightarrow AF$ decomposition

$A \rightarrow B$

Made with Goodnotes

$F^+ = \{F\}$

Now, let's check the closure of multiple attributes:

$(BD)^+ = \{\underbrace{B, D}, \underbrace{A, F}\}$

$D \rightarrow AF$

$BD \rightarrow BD$
(reflexivity)

$BD \rightarrow B$
(ref.)

$BD \rightarrow D$
(ref.)

$(AC)^+ = \{A, C, B, D, E, F\} = R$

AC also looks like it would be a candidate key. But it is not. Let's see why.

Key:

A key is an attribute or a collection of attributes whose value is unique & is/are capable of determining all the remaining attributes of R.

In short:

$key^+ = R$

i.e.

$closure(K) = R$

Note: FD & closures don't tell us if the values of an attr. are unique. They only help in determining closures.

Made with Goodnotes

Keys are of 3 types.

Key

Super key (SK)

A superkey is an attribute or a collection of attributes whose closure is R.

Eg: from the relation and FD in the previous example,

C, E and AC are super keys. There could be more as well.

$SK = \{C, E, AC, \ldots\}$

Candidate key (CK)

A candidate key is a superkey whose proper subset is not again a super key.

Eg: from the previous relation and FD, let's take the super key AC.

$A^+ \neq R$ ✓

$C^+ = R$

i.e. AC's proper subset C is a superkey by itself. So, AC is not a candidate key.

So,

$CK = \{C, E\}$

Candidate key is also called minimal super key.

Primary key (PK)

A candidate key selected for the database by the DB admin is called primary key.

Made with Goodnotes

Let's take a new example:

$R(ABCDEF)$

and

$A^+ = R$

$C^+ = R$

$(DE)^+ = R$

$(DC)^+ = R$

$(ADC)^+ = R$

$(BDF)^+ = R$

$(BDEF)^+ = R$

Here, the superkeys are:

$SK = \{ A, C, DE, DC, ADC, BDF, BDEF \ldots \}$

There will be many more SKs since every attribute that is grouped with an SK will form another SK.

Eg: A is already an SK.
So, AB, AC, AD, AE, AF, ABC ...
etc will all be SKs.

From the above list of SKs we can filter the candidate keys.

A $\longrightarrow$ candidate key since this is an atomic attrib. and we can't get a proper subset whose closure is R.

C $\longrightarrow$ same as above.

Made with Goodnotes

$DE \longrightarrow D^+ \neq R,\ E^+ \neq R$

So, DE is also a candidate key.

$DC \longrightarrow D^+ \neq R,\ C^+ = R$

Not a candidate key.

$ADC \longrightarrow$ Not a candidate key

$BDF \longrightarrow B^+ \neq R,\ D^+ \neq R,\ F^+ \neq R$

So, BDF is a candidate key.

$BDEF \longrightarrow (DE)^+ = R$

So, BDEF is not a candidate key.

Note: Whenever in exam, they mention just "key" without mentioning super key, candidate key or primary key, we will consider it as Candidate key only.

Candidate key (essential attribute):

→ To find the candidate key, you first need to determine the "essential attribute"

→ The essential attribute is an attribute that is not determined by any other attribute.

→ Basically it is not present on the RHS of any FD.

Made with Goodnotes

→ If the essential attribute alone turns out to be the superkey, then it WILL be the candidate key for that entity set.

→ This is because, no other attribute can determine the essential key.

→ Even if the essential key is not the candidate key by itself, it WILL be part of the candidate key since no attribute can determine it but itself.

Q.) R(ABCDE)

$CE \rightarrow D$

$D \rightarrow B$

$C \rightarrow A$

Note: See here how E is not on RHS. So, E must be part of the candidate key since no attr. functionally determine it.

Find CK

"C" is essential attribute.

$C^+ = \{C, A\}$

$D^+ = \{D, B\}$

$CE^+ = \{C, E, D, B, A\}$

Similarly, "E" is also an essential attribute.

Candidate key is CE

Q.) R(ABCDE)

$BC \rightarrow ADE$

$D \rightarrow B$

$D^+ = \{D, B\}$

$BC^+ = \{B, C, A, D, E\}$

Candidate key is BC

$BC \rightarrow ADE$ & $D \rightarrow B$

Then,

$DC \rightarrow ADE$

$DC^+ = \{D, C, A, E, B\}$

$D \rightarrow B$

Made with Goodnotes

Q.) R (ABCD)

$AB \rightarrow CD$
$C \rightarrow A$
$D \rightarrow B$

Sol: $A^+ = \{A\}$
$\Rightarrow B^+ = \{B\}$
$C^+ = \{C, A\}$
$D^+ = \{D, B\}$

$AB^+ = \{A, B, C, D\}$

Now, to find other candidate keys, let's replace attributes in AB using the axioms.

$AB \rightarrow CD$
and $C \rightarrow A$
So, using pseudo-transitivity
$CB \rightarrow CD$
$(CB^+) = \{C, B, D, A\}$

From $CB \rightarrow CD$

From $C \rightarrow A$

Similarly,
$AB \rightarrow CD$ & $D \rightarrow B$
Pseudo-transitivity
$AD \rightarrow CD$
$(AD)^+ = \{A, D, C, B\}$

$AD \rightarrow CD$ & $C \rightarrow A$

Pseudo-transitivity:

$CD \rightarrow CD$
$(CD)^+ = \{C, D, A, B\}$

$C \rightarrow A$

$D \rightarrow B$

Made with Goodnotes

Q.) R(ABCDEF)

$AB \rightarrow C$
$C \rightarrow D$
$D \rightarrow BE$
$E \rightarrow F$
$F \rightarrow A$

$\Rightarrow$

Sol:
$A^+ = \{A\}$
$B^+ = \{B\}$
$C^+ = \{C, D, B, E, F, A\} = R$
$D^+ = \{D, B, E, F, A, C\} = R$
$E^+ = \{E, F, A\}$
$F = \{F, A\}$

(Check first for essential attrib)

Also,

$AB^+ = \{A, B, C, D, E, F\} = R$

Use replacement:

$FB^+ = R$
$\downarrow$
$EB^+ = R$

$CK = \{C, D, AB, FB, EB\}$

Let's say I wanted to find the no. of superkeys possible.
How would I go about finding it?

1. Every CK is an SK. $\longrightarrow$ 5 in the above case
2. You can attach other attributes to an SK to form a CK
3. The entire attr. set ABCDEF is an SK.

For every CK, I would find the no. of SK's possible:
Let's take the CK "C".
Total no. of attr.s are 6.
Any attr.(s) added to C would form an SK.
So,

Made with Goodnotes

_ C (One space for additional attrib. to form SK
This space can contain 5 attrib.s
i.e. A, B, D, E, F)

_ _ C (Similarly 2 attribs. One can contain 5
and another 4)

_ _ _ C (5 - 4 - 3)

_ _ _ _ C (5 - 4 - 3 - 2)

I won't be doing _ _ _ _ _ C since it will only result in ABCDEF which was already counted.

The formula for combinations is

$$\frac{\frac{n!}{(n-r)!}}{r!} \Rightarrow \frac{n!}{r!\,(n-r)!}$$

where n is the no. of elements
r is the no. of places.

So, it might result in:

$$\Rightarrow \frac{5!}{1!\,4!} + \frac{5!}{2!\,3!} + \frac{5!}{3!\,2!} + \frac{5!}{4!\,1!}$$

$\Rightarrow 5 + 10 + 10 + 5$

$\Rightarrow 30 \Rightarrow$ This is not counting just C & the entire set of attrib.s ABCDEF

Including them, 32

Made with Goodnotes

Number of possible super keys when candidate key is given:

Take the following example with

R(ABCDEF)

and

CK = { C, FB }

Let's calculate the no. of superkeys for the candidate key "C":

C A B D E F

C → It is necessary in every superkey

A B D E F → Each of these may or may not be present

C	A	B	D	E	F
1	2	2	2	2	2

→ 2 values because, "A" may be present in the resulting combination of keys or may not be present.

Total = $1 * 2 * 2 * 2 * 2 * 2 = 2^5 = 32$

Made with Goodnotes

Similarly, for FB:

$$\underbrace{F \quad B}_{} \quad \underbrace{A \quad C \quad D \quad E}_{}$$

These both need to be present since they are the candidate keys

Each of these may or may not be present in the resulting combinations.

$$\begin{array}{cccccc} F & B & A & C & D & E \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 1 & 2 & 2 & 2 & 2 \end{array}$$

$$\text{Total} = 1 * 1 * 2 * 2 * 2 * 2$$
$$= 2^4$$
$$= 16$$

So, to summarise,
given a candidate key CK, the no. of superkeys that can be formed are:

$$\boxed{2^{n-k}}$$

where, n is the total no. of attribs and k is the no. of attribs. in CK

Made with Goodnotes

However, in the above example, the total no. of superkeys is not:

$$SK_C + SK_{FB}$$

where,
SK_C is no. of superkeys using C as candidate key
SK_{FB} is no. of superkeys using FB as candidate key

Since the superkeys of C & FB have overlaps.

For ex, the superkey FBC occurs in both SK_C and SK_{FB}.

The actual total no. of superkeys is:

$$\boxed{SK_C + SK_{FB} - SK_{CFB}}$$

So,

$\Rightarrow 2^5 + 2^4 - 2^3$

$\Rightarrow 32 + 16 - 8$

$\Rightarrow 40$ //

Q.) R(ABCDEF)

$$CK = \{AB, BC\}$$

No. of superkeys = ?

$n(AB) \Rightarrow AB____$ ($\downarrow 2$, $\downarrow 2$, $\downarrow 2$, $\downarrow 2$)

$\Rightarrow 2^4 \Rightarrow 16$

$n(BC) = BC____$ ($\downarrow 2$, $\downarrow 2$, $\downarrow 2$, $\downarrow 2$)

$\Rightarrow 2^4 \Rightarrow 16$

Made with Goodnotes

$n(ABC) \Rightarrow A\ B\ C$

$\underset{\downarrow}{\underset{2}{_}}\ \underset{\downarrow}{\underset{2}{_}}\ \underset{\downarrow}{\underset{2}{_}}$

$\Rightarrow 2^3 \Rightarrow 8$

Total no of superkeys:

$\Rightarrow n(AB) + n(BC) - n(ABC)$

$\Rightarrow 16 + 16 - 8$

$\Rightarrow 24$ //

Q.) 2021 - 10th question in PDF

$U(PQRST)$

$P \rightarrow QR$

$RS \rightarrow T$

Which of the foll. FDs can be inferred from the above:

1.) $PS \rightarrow T$

2.) $R \rightarrow T$

3.) $P \rightarrow R$

4.) $PS \rightarrow Q$

Sol:

1.) $PS \rightarrow T$

Using 3, $P \rightarrow R$

$P \rightarrow R$ & $RS \rightarrow T$

Then, using pseudo-transit.

$PS \rightarrow T$ ✓

2.) $R \rightarrow T$

3.) $P \rightarrow R$

$P \rightarrow QR$

Using decomposition

$P \rightarrow Q$

$P \rightarrow R$ ✓

4.) $PS \rightarrow Q$

Made with Goodnotes

Let's try using closures:

$(PS)^+ = \{P, S, Q, R, T\} \Rightarrow PS \rightarrow T$ ✓

$(R)^+ = \{R\} \Rightarrow R \rightarrow T$ ✗

$(P)^+ = \{P, Q, R\} \Rightarrow P \rightarrow R$ ✓

$(PS)^+ = \{P, S, Q, R, T\} \Rightarrow PS \rightarrow Q$ ✓

It is usually simpler to determine the closure rather than try with axioms in this case.

Q.) 1999 – 2nd question in PDF

$R(A, B, C, D, E, F)$

$C \rightarrow F$
$E \rightarrow A$
$EC \rightarrow D$
$A \rightarrow B$

Which is a key for R?

Sol:

By "key" we will always consider only "candidate key" unless mentioned otherwise.

Here, essential attributes are C & E

Let's try EC:

$C \rightarrow F$
$E \rightarrow A$
$EC \rightarrow D$
$A \rightarrow B$

$EC = \{E, C, F, A, D, B\}$

Candidate key is EC

Made with Goodnotes

Q.) 2005 CS - question 4 in PDF

$R(A, B, C, D, E, H)$

$A \rightarrow B$
$BC \rightarrow D$
$E \rightarrow G$
$D \rightarrow A$

Candidate keys of R = ?

Sol: Essential attr.s are E & H
$EH = \{E, H, C\}$

In options, only one choice has EH in all candidate keys:

AEH, BEH, DEH

Q.) 2005 IT - questions in PDF

$R(ABCDE)$
$A \rightarrow B$
$A \rightarrow C$
$CD \rightarrow E$
$B \rightarrow D$
$E \rightarrow A$

FDs not implied by above set:

1. $CD \rightarrow AC$
2. $BD \rightarrow CD$
3. $BC \rightarrow CD$
4. $AC \rightarrow BC$

Sol: $(CD)^+ = \{C, D, E, A, B\} \Rightarrow CD \rightarrow AC$ ✓

Made with Goodnotes

$(BD)^+ = \{B, D\} \Rightarrow BD \rightarrow CD$ ✗

$(BC)^+ = \{B, C, D, E, A\} \Rightarrow BC \rightarrow CD$ ✓

$(AC)^+ = \{A, C, B, D, E\} \Rightarrow AC \rightarrow BC$ ✓

Q.) 2006 CS - question 5 in PDF

$AB \rightarrow CD$
$AF \rightarrow D$
$DE \rightarrow F$
$C \rightarrow G$
$F \rightarrow E$
$G \rightarrow A$

Which option is false:

1. $\{CF\}^+ = \{ACDEFG\}$
2. $\{BG\}^+ = \{ABCDG\}$
3. $\{AF\}^+ = \{ACDEFG\}$
4. $\{AB\}^+ = \{ABCDFG\}$

Sol:

$R(ABCDEFG)$

B is essential attribute

$(CF)^+ = \{C, F, G, E, A, D\}$ ✓

$(BG)^+ = \{B, G, A, C, D\}$ ✓

$(AF)^+ = \{A, F, D, E\}$ ✗ → To get C, you need A & B. But option says $(AF)^+ = \{ACDEFG\}$

$(AB)^+ = \{A, B, C, D, G\}$ ✗

Made with Goodnotes

STARTING

DBMS

BRAHMASTRA

BATCH

Made with Goodnotes

Syllabus

- → E R model (1-2 marks)
- → Relational model
- → Integrity constraints } – 2 marks
- → Normal forms (2 marks but not by itself)
- → file org., indexing (B & B+ trees) (1-2 marks)
- → Relational algebra, tuple calculus, SQL (2-4 marks)
- → Transactions & concurrency control (2 marks)

Made with Goodnotes

Transaction

- In general terms, transactions are just a series of steps to reach a certain end goal
- In DBs, transactions are a series of operations to read/write data from the DB
- DB transactions need to be atomic i.e.

A transaction either finishes completely (all ops in the transaction) or does not complete at all.

Eg: Consider A and B being the balance in 2 accounts and both have a value of 1000 initially.

T_1
Read (A)
A = A - 100
Write (A)
Read (B)
B = B + 100
Write (B)

Transaction transferring 100 units from A to B

Made with Goodnotes

→ For net-consistency,

Value of A + B before T_1	= =	Value of A + B after T_1

Desirable properties of transaction:

A - Atomicity

C - Consistency

I - Isolation

D - Durability

Atomicity:

A transaction is an atomic unit of processing -

It should either be performed in its entirety or not performed at all.

Atomicity is the responsibility of recovery / transactional control manager of the DB.

P.T.O

Made with Goodnotes

Consistency:

Consistency varies from one system to another.

For example, in the previous bank account example, consistency meant:

Balance before transaction == Balance after transaction

Basically, a transaction that is not interrupted should take the DB from one consistent state to another.

Isolation:

A transaction should appear as though it is being executed in isolation even if multiple (tens, hundreds, thousands etc) transactions are occurring at the same time.

The isolation property of the database is the responsibility of concurrency control manager of DB.

P.T.O.

Made with Goodnotes

Result of a transaction being executed in isolation	= =	Result of a transaction being executed concurrently with other transaction

Execution of one transaction should not affect the execution of another transaction.

Durability:

Changes applied to a database must persist in the database.

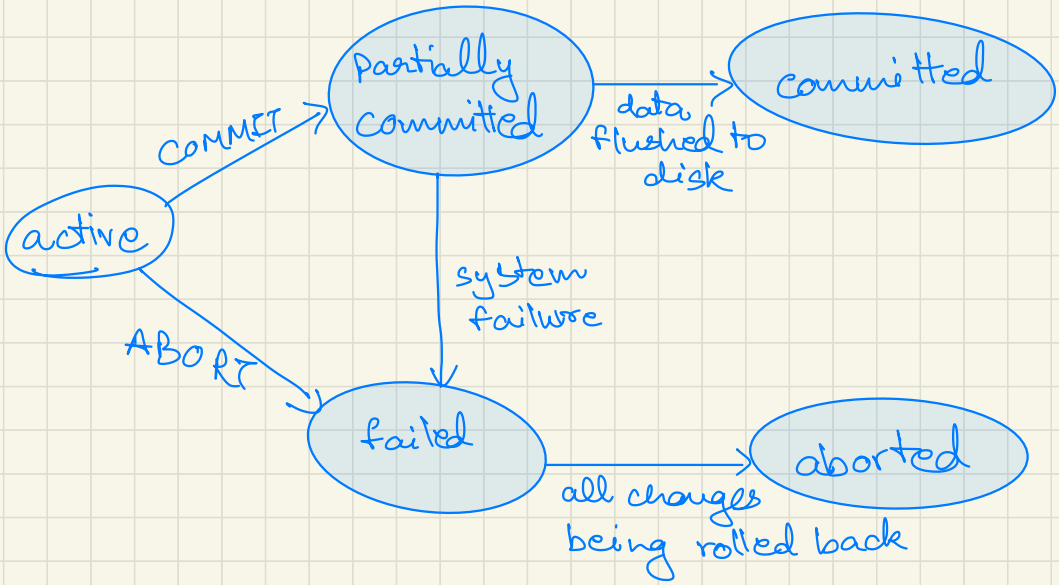
Eg: If a student's grade is recorded as 50/80 in a DB, it should remain that way until the next time the admin updates the student's scores.

Maintaining the durability of the DB is the responsibility of the Recovery Control Manager.

P.T.O

Made with Goodnotes

Transaction States



ACTIVE: Transaction remains in this step while it is executing operations

The DB is in ACTIVE state until the final statement of the transaction executes.

PARTIALLY COMMITTED: After executing all the stmts, the system is in a partially committed state before the update is flushed to disk.

Made with Goodnotes

FAILED: When it is known that the transaction cannot proceed further (due to logical or hardware errors), it is in a failed state and needs to be rolled back.

ABORTED: Transaction has been rolled back and restored to its previous state.

COMMITTED: After successful completion of writing the updates to disk, the transaction is in a committed state.

Concurrent Execution & It's Problems:

We need concurrent execution to:

1. Increase DB performance
2. Decrease wait time for transactions
3. Increase CPU throughput and response time.

Problems of concurrent execution:

Sometimes, even if the ACID properties of transactions are satisfied, we still end up in an inconsistent state. Here are a few of such problems:

1. Dirty read problem
2. Unrepeatable read problem
3. Lost update problem
4. Phantom read problem

Made with Goodnotes

All of these problems are happening in the buffer before the O/P is committed to disk.

A common pattern that can be noticed is that these issues occur when multiple transactions are reading and writing the same resource.

Dirty-Read problem:

T_1	T_2
Read (A)	
Write (A)	
	Read (A)
	Commit
Abort	

→ In this problem, T_2 reads a value that has not been committed by T_1.

→ T_2 finishes committing its value but T_1 decides to abort the transaction.

→ The value read by T_2 is invalid.

Note: Notice how there are commit and abort steps in the above transaction. This shows that the transactions have not finished executing prior to the commit/abort steps.

Made with Goodnotes

Unrepeatable-Read problem:

T_1	T_2
	Read(A)
Read(A)	
Write(A)	
	Read(A)

→ T_2 tries to read a data-item twice but between its 2 attempts to read, T_2 update the value of the data item.

→ Now, the value of the first read will differ from the value of the 2nd read.

→ This is a non-repeatable read problem.

Note: Notice how in T_1, we did a read before the write. While the read is not strictly necessary, writing without reading is called a blind-write / blind-update.

P.T.O.

Made with Goodnotes

Lost Update Problem:

T_1	T_2
Read (A)	
Write (A=20)	
	Write (A=30)
	Commit
Commit	

→ T_1 is writing a certain value to record A.

→ Before T_1 commits, T_2 updates the same recod to a diff. value and commits first.

→ Now, when T_1 commits, it is committing T_2's value again since it has no idea that the record was updated.

→ Note how there is no read o/p b/w the two writes. Due to this, neither transaction is aware of the other transaction's update.

P.T.O.

Made with Goodnotes

Phantom Read problem:

T_1	T_2
Read(A)	
	Delete(A)
Read(A)	

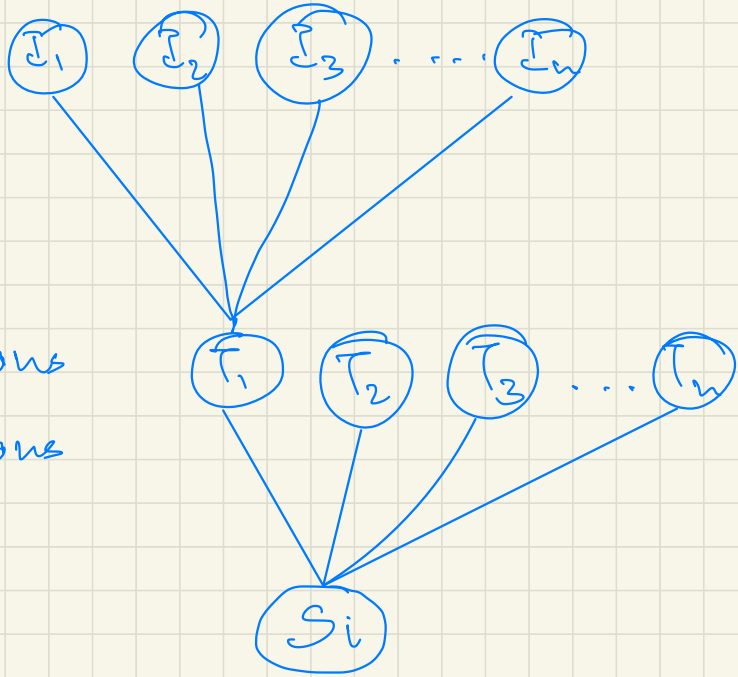
-> Here T_1 tries to read a record twice but between its attempts to read, T_2 deletes the record.

-> This causes the phantom-read issue where T_1 is trying to read a record that does not exist.

P.T.O.

Made with Goodnotes

Scheduling:



I = Instructions

T = Transactions

S = schedule

-> When multiple transactions are being executed concurrently, we can club them into a "schedule"

i.e.

A schedule is a bunch of transactions executing either concurrently or one after another.

-> If a schedule contains n transactions, each transaction will have a few operations/instructions

-> These instructions can be interleaved i.e. an instruction from T_1 can be executed followed by an inst. from T_2 and then T_1 again.

This can only be done in the same schedule.

Made with Goodnotes

→ There are 2 types of schedules:

① Serial / Linear schedule
② Non-serial schedule

Serial / Linear scheduling:

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

→ In serial scheduling, we do not perform o/ps in transactions concurrently.

→ We finish one transaction before moving on to the next.

→ This is not efficient use of CPU and decreases performance but it guarantees consistency.

→ If we have n transactions in a schedule, we can have n! valid serial schedules.

Made with Goodnotes

Non-Serial scheduling:

T_i	T_j
read (B)	
	read (B) write (B)
read (A)	
	read (A) write (A)

→ In this scheduling method, the transactions are executed concurrently.

→ Instructions from each of the transactions is interleaved thereby allowing multiple transactions to execute at once.

→ However, unless done correctly, consistency is not guaranteed with NS-scheduling.

Conversion of non-serial to serial schedule:

→ It is apparent that NS is more performant while serial is consistent.

→ The idea is to convert non-serial schedules to serial in such a way that we attain the consistancy of serial but retain the speed of non-serial.

Made with Goodnotes

Serializability: (2 marks)

→ Conversion of NS schedules to serial schedules is decided by serizability.

→ The conversion happens by swapping instructions in the non-serial schedule so that the instructions being executed are not conflicting.

T_i	T_j
Read(Q)	
	Read(Q)
Read(X)	
	Write(X)
⋮	⋮

→ In the above example Read(Q) in T_i and T_j occur. This is fine since the instructions are not conflicting. They can be swapped.

→ In the next steps, T_i reads X and T_j writes X. These instructions are conflicting.

→ Again, in order to swap instructions, we need to check if they are conflicting or not.

→ We can only swap non-conflicting instructions

Made with Goodnotes

Eg:

T_1	T_2
read A	
	read A

turned to

T_1	T_2
	read A
read A	

swaps

swap

→ Swapping is not possible if:

① Both instructions / operations are working on the same data.

② At least one o/p is a write o/p.

→ The above is also the definition for conflicting instructions.

→ 2 instructions are said to be conflicting if they belong to diff. transactions, operate on the same data and atleast one of the instructions is a WRITE O/P.

***

* Checking whether a schedule is conflict serializable

T_i	T_j
R(X)	
R(Y)	
	R(X)
	R(Y)
	W(Y)
W(X)	

Made with Goodnotes

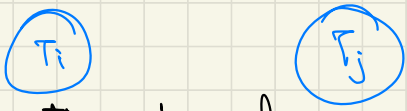
Let's take the above schedule as an example and try to figure out if the schedule is conflict serializable.

To reiterate, a schedule is not serializable if:

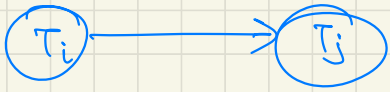
1. different transactions in the schedule are working on the same record
2. If atleast one of the transactions is performing a write o/p on the common record.

<u>Process:</u>

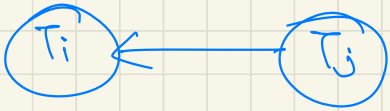
1. We always go top down on the instructions
2. We draw 2 nodes representing the transactions:



3. Based on the top-down approach, if we find that one transaction is dependant on the other to occur (i.e. there is a conflicting instruction from one transaction to the other in top-down approach), then we draw a line from the dependant transaction to the other.

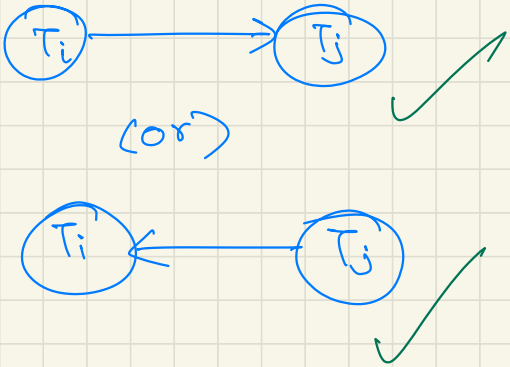


(or)

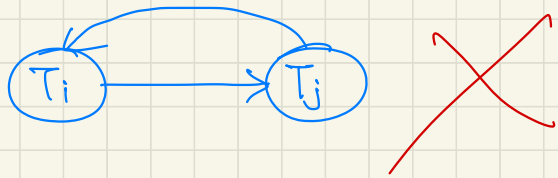


Made with Goodnotes

④ After finding all the conflicting instructions, if there is a cycle, then the transaction is not conflict serializable.



Conflict serializable



Not conflict serializable

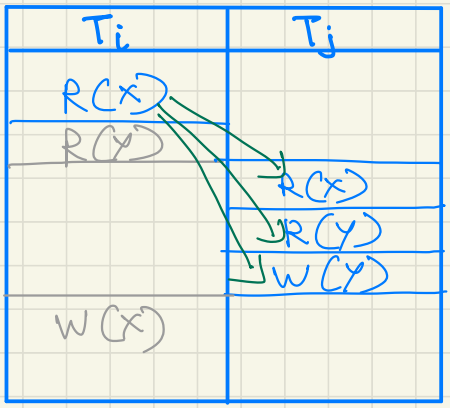
Let's use the above process to solve the earlier question:

T_i	T_j
R(X)	
R(Y)	
	R(X)
	R(Y)
	W(Y)
W(X)	

P.T.O.

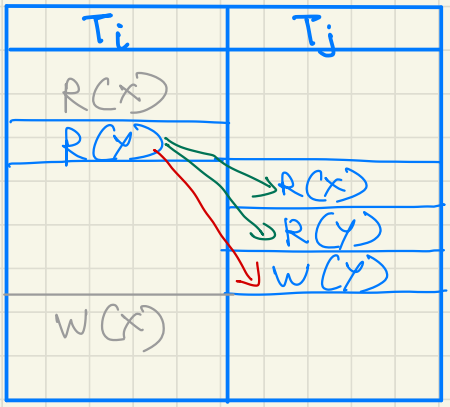
Made with Goodnotes

① $T_i \Rightarrow R(X)$:

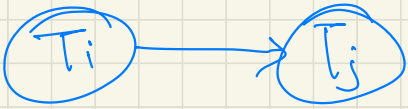


T_i's R(X) does not conflict with any o/p in T_j. No dependancies yet. $\Rightarrow$ T_i T_j

② $T_i \Rightarrow R(Y)$:



T_i's R(Y) conflicts with T_j's W(Y). Both are working on Y and one of them is a write o/p n. So, dependancy has been created.



Made with Goodnotes

The arrow goes from T_i to T_j because T_i's $R(Y)$ o/p comes first in the table.

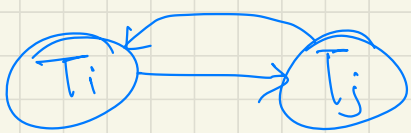
What the arrow means is that, based on the order set in the table, T_j needs T_i to finish first to be conflict serializable i.e. first T_i, then T_j.

③ T_j's $R(X)$:

T_i	T_j
$R(X)$	
$R(Y)$	
	$R(X)$
	$R(Y)$
	$W(Y)$
$W(X)$	

T_j's $R(X)$ has also run into a conflict now. Based on the table, T_j now needs T_i to finish first to be conflict serializable.

This is a new dependancy.



As can be seen, there is a cycle. So, this schedule containing T_i & T_j is not serializable.

Made with Goodnotes

Practice questions:

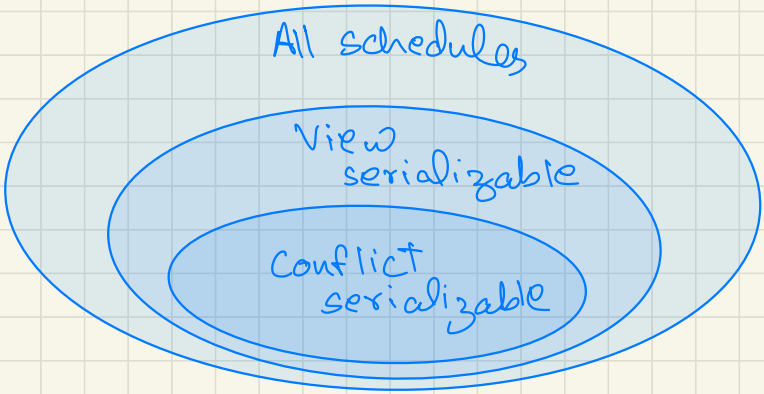
1. GATE 2021 Set 1 Q 32
2. GATE 2014 Set 3 Q65
3. GATE 2014 Set 2
4. GATE 2007 (Also NET-JAN-2017)
5. ** GATE 2010
6. ** GATE 2006

View Serializability: (skip if short on time. zero PYQs)

→ We use conflict serializability to ensure that we maintain consistency in the database.

→ Even if a schedule is not conflict serializable, it can still be consistent.

→ This is a weaker form of serializability called view serializability which states that, if a schedule is view serializable, it can be consistent.



Made with Goodnotes

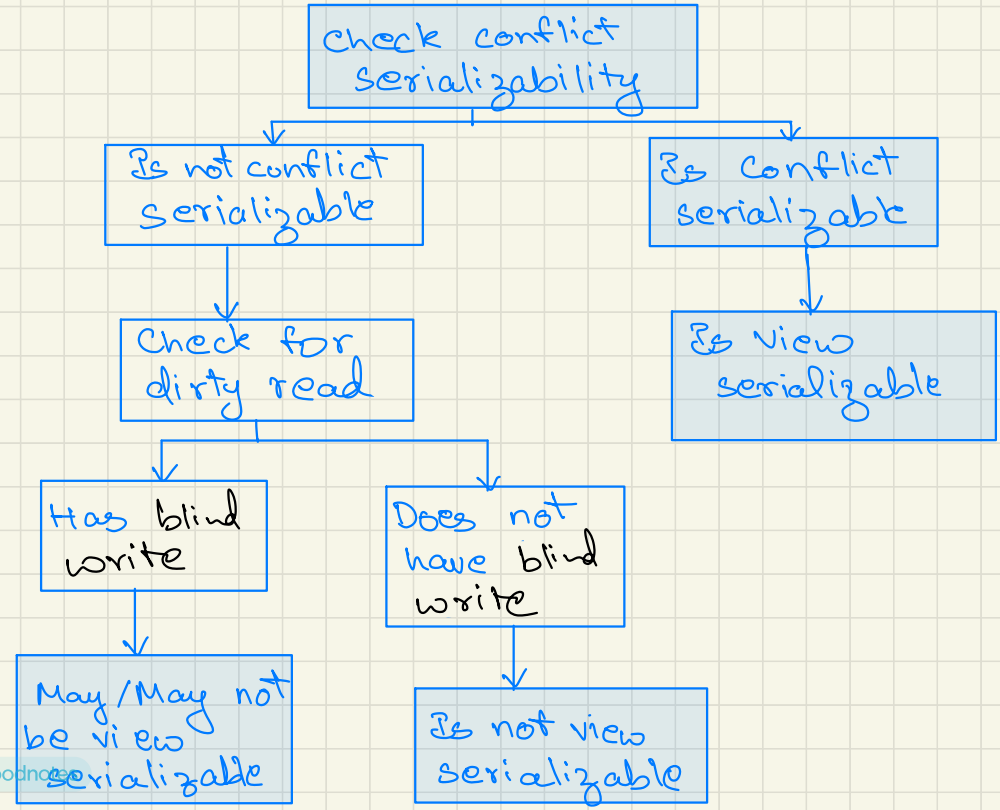
→ In terms of serializability, view serializability is a superset of conflict serializabity. i.e. any schedule that is conflict serializable is also definitely view serializable.

→ Just because a schedule is not conflict serializable does not mean it is not view serializable.

→ To reiterate, a schedule can be consistent if it is conflict serializable or view serializable.

→ The time complexity to determine if a schedule is conflict serializable is $O(n^2)$

→ Finding if a schedule is view serializable is an NPC problem i.e. very hard.



Made with Goodnotes

→ The above is a shortcut to finding if a schedule is view serializable or not.

→ The first step to finding if a schedule is view serializable is to always check if it is conflict serializable.

→ The above may not always work to find if a schedule is view serializable since a schedule could be negative on conflict serializability and have dirty writes. In this case, how do we find view serializability?

→ For this, we first need to understand view equivalency.

View Equivalency:

→ Consider there exist 2 schedules S & S' each containing transactions T_i and T_j.

→ The schedules S and S' are said to be view equivalent if:

① For each data item Q, if T_i performs the 1st read in S, then in S' as well T_i needs to read Q first.

② For each data item Q, if T_i performs the last write in S, then in S' as well T_i needs to perform the last write of Q.

③ If in S, a data item Q is first written by T_j and then read by T_i, then in S' as well, T_j must first write Q and then T_i must read Q.

Made with Goodnotes

Eg:

View Equivalent schedules

S

T_i	T_j
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

S'

T_i	T_j
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

→ The above 2 schedules are view equivalent since:

① For condition of 1st read:
- a.) A is 1st read in T_i in S which is also the case in S'.
- b.) B is 1st read in T_i in S which also happens in S'.

② For condition of last write:
- a.) A is last written by T_j in S which is also the case in S'.
- b.) B is last written by T_j in S which is also the case in S'.

③ For condition of same order write-reads:
- a.) In S, T_i writes A which is read by T_j - this also happen in S'.
- b.) In S, T_i write B which is read by T_j - this occurs in S' as well.

So, the 2 schedules are view equivalent.

(Note: Here we are talking about view equivalency & not serializability. Both schedules are conflict seri-

Made with Goodnotes

alizable so they will be view serializable as well)

Back to view Serializability:

→ Coming back to view serializability, a schedule S is view serializable if it is view equivalent to a serial schedule.

Eg: Take the foll. schedule

T_1	T_2	T_3
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

To check if the above schedule is view serializable, we need to check if it is view equivalent with any of the possible serial schedules.

T_1	T_2	T_3
read(Q)		
write(Q)		
	write(Q)	
		write(Q)

Made with Goodnotes

The above schedule is one of the possible serial schedules.

→ Note that a serial schedule is one where there is no concurrency i.e. T_2 begins only after T_1 finishes.

→ Coming back, we need to check if the original schedule is view equivalent with any of the possible serial schedules.

Since there are 3 transactions, there are 6 possible serial schedules (n!) :

T_1	T_2	T_3
T_1	T_3	T_2
T_2	T_1	T_3
T_2	T_3	T_1
T_3	T_2	T_1
T_3	T_1	T_2

→ Let's compare the original schedule with one of these.

T_1	T_2	T_3
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Serial schedule T_1 T_2 T_3

T_1	T_2	T_3
read(Q)		
write(Q)		
	write(Q)	
		write(Q)

Made with Goodnotes

→ Comparison:

① 1st read:

Q is 1st read in T_1 in S.

Same in S'

② last write:

Q is last written by T_3 in S.

Same in S'.

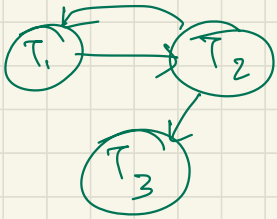
③ Write-read:

There is no instance of write-read in S.

Same in S'

→ Both schedules are view equivalent. So, they are view serializable.

Note: Before doing this lengthy method, we will first check for conflict serializability and presence of blind write in S.



(cycle so not conflict serializable)

Blind writes present in T_2 & T_3.

→ The above is a good example of a schedule that is not conflict serializable but is view serializable.

Made with Goodnotes

Q.) A schedule that is not conflict serializable and has a blind write is:

A.) Always view-serializable
B.) Not always "
C.) May be "
D) None of these

Q.) Which stmts is/are TRUE:

S_1: All view serializable schedules are conflict serializable

S_2: All conflict serializable schedules are view serializable.

S_3: If a schedule is not conflict serializable it is not view serializable.

S_4: If a schedule is not view serializable, it is not conflict serializable.

Q.) Consider schedule:

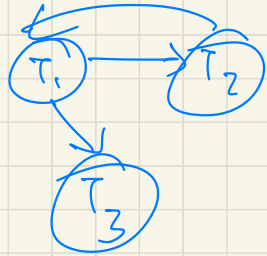
T_1	T_2	T_3
R(B)		
		R(C)
R(A)		
	W(A)	
W(A)		
	W(B)	
		W(A)
W(B)		
		W(B)
		W(C)

Is above conflict and/or view serializable?

P.T.O.

Made with Goodnotes

R(B)		R(C)
	W(A)	
W(A)		
	W(B)	
W(B)		
		W(A)
		W(B)
		W(C)



Not conflict serializable

For view serializability check against serial schedule: $T_1 T_2 T_3$

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

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

1st reads: A, B $\Rightarrow T_1$ ✓ & C $\Rightarrow T_3$ ✓

last writes: A, B, C $\Rightarrow T_3$ ✓

intermediate write-read: None ✓

So, they are view equivalent & hence the schedule is view serializable.

Made with Goodnotes

*Recoverable & Non-recoverable schedule:

→ Up until this point, we studied about conflict and view serializability because we want our system to be in a **consistent** state even when we are executing a non-serial schedule.

→ However, we have been taking a few things for granted. We have always considered that our transactions will not abort and also we have not considered system failures.

→ Take the foll. ex:

T_i	T_j
R(X)	
W(X)	
	R(X)
	Commit
Commit	

The above schedule is conflict serializable so the system should not go into an inconsistent state.

But, what if there is some sort of system failure befor T_i commits (as shown by the squigly line)?

T_j's R(X) had read the updated value of X i.e. it had read the value of X updated by T_i.

Made with Goodnotes

But now, T_i did not have a chance to commit but T_j already committed.

As can be seen, our system is in an inconsistent state.

This is a non-recoverable schedule.

Non-recoverable schedule:

A schedule in which if transaction T_j reads a value previously updated by transaction T_i such that T_j's commit occurs befor T_i's commit is a non-recoverable schedule.

T_i	T_j
R(X)	
W(X)	
	R(X)
	Commit
Commit	

As shown previously, there is a chance of the system going into an inconsistent state if the system was interrupted before T_i's commit.

The above is an example of dirty read.

Made with Goodnotes

Recoverable schedule:

Take the same example as above but just reverse the order of commits.

T_i	T_j
R(X)	
W(X)	
	R(X)
Commit	
	Commit

In this case, even if the system is interrupted before the last commit (T_j's commit), the system will not go into an inconsistent state.

If the transaction has not committed, we can always rollback to a previous state.

In the current example, the system can be interrupted at any point and it will still be able to rollback to a previously consistent state.

Note how here also there is a dirty read.

A schedule in which if transaction T_j reads a value previously written by transaction T_i such that T_i's commit occurs before T_j's commit is a recoverable schedule.

Made with Goodnotes

Summary:

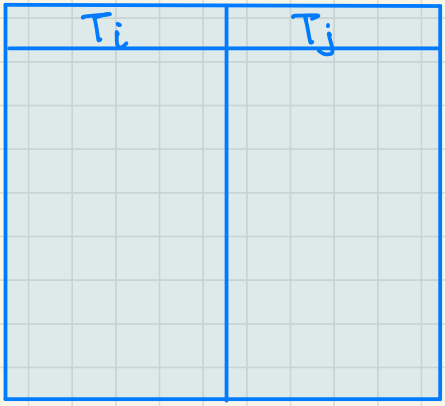
→ If a schedule has been established as conflict serializable and we want to check if it is recoverable,

check for dirty reads

→ If the transaction that performs the dirty read commits **AFTER** the transaction that writes the dirty value, the schedule is recoverable.

→ A schedule maintains consistency property in ACID if it is conflict/view serializable and is also recoverable.

Made with Goodnotes



Made with Goodnotes