Assignment Operation :

The assignment operation (←) provides a convinient way to express complex queries.

- Assignment must always be made to a temporary relation variable.

Example: Find the names of all customers who have a loan, an account, or both from bank.

temp1 ← $\Pi_{customer\_name}$ (borrower)

temp2 ← $\Pi_{customer\_name}$ (depositor)

result = temp1 ∪ temp2 ;

- Result is always ~~Right~~ Left side of (←) symbol.
- Query is always ~~Left~~ Right side of (←) symbol.

Extended Relational Algebra Operation:

- Generalized Projection.
- Aggregate Functions
- Group by Functions.

- **Generalized Projection:**

Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

- E is any relational algebra Expression.
- $F_1, F_2 \ldots F_n$ are arithmetic expressions.

Ex: Credit_info (customer_name, limit, money_spent)
Find how much more each person can spend

$$\Pi_{customer\_name,\ limit - money\_spent}(credit\ info)$$

- **Aggregation Function:**

Takes a collection of values and returns a single value as a result.

- Avg — average value
- min — minimum value
- max — max value
- sum — sum of value
- count — number of values.

Aggregate operation:

$$\mathcal{G}_{F(balance)}(Account)$$

- F is aggregate function.
- balance is Attribute name.
- $\mathcal{G}$ is Aggregate function Symbol.
- Account is table (schema name)

Ex: Student

sid	name	CGPA	Grade
101	Roy	8.5	A
102	Ivan	9.0	A
103	Joy	7.0	B
104	Jen	6.8	C
105	John	7.1	B

8.5 ≤	A
7.0 - 8.4	B
7.0 >	C

① $\mathcal{G}_{sum(CGPA)}(student)$ =

sum(CGPA)
38.4

② $\mathcal{G}_{count(CGPA)}(student)$ =

count(CGPA)
5

- in this exercise result of aggregation does not have a name.

option:1 → Can use rename operation to give it a name

option:2 → For convenience, we permit renaming as a part of aggregate operation

Ex: $\mathcal{G}_{sum(GPA)}$ as sum-balance (student).

↑ rename of sum (GPA)

sum-balance
38.4

Group by Function:

The Group by clause is another section of the select statement. This optional function tells oracle to groups rows based on distinct values that exist for specified column.

Ex-1

Sales

Company_id	Product_id	Price
C101	22	2000
C102	23	4000
C103	25	5000
C102	23	4000
C103	25	5000
C104	26	8000
C104	27	10,000

Query-1 Select Company_id, Product_id, sum(Price) from Sales Group by (Company_id, Product_id)

Ans:

Company_id	Product_id	sum(Price)
C101	22	2000
C102	23	8000
C103	25	10,000
C104	26	8,000
C104	27	10,000

Query-2 Select Company_id, sum(Price) from Sales Group by Company_id

Company_id	sum(Price)
C101	2000
C102	8,000
C103	10,000
C104	18,000

Scanned by CamScanner

Ex:2

stud

E.No	Grade
1	AA
2	AB
3	BC
4	AB
5	AA
6	BC
7	AA
8	BC

Q: Find the Number of student having different Grade

Query:1

Ans Select Grade, count(Grade) from stud Group by Grade;

o/p

Grade	count(Grade)
AA	3
AB	2
BC	3

⇒ Condition in Group by:

Find the different Grade Count and the Count is Greater than 2.

Q:2

Ans Select Grade, count(Grade) from stud Group by Grade having count(Grade)>2;

Grade	Count(Grade)
AA	3
BC	3

- Group by Syntax in Relational Algebra:

$_{company\_id,\ product\_id}\mathcal{G}_{company\_id,\ product\_id,\ sum(price)}(Sales)$

Answer of Example-1 - Query-1.

← Aggregate function simply ignore null Values.

Principles of NULL Values:

① Setting NULL value is appropriate when the actual value is Unknown. & when a value would not be meaningful.

② NULL Value is Not equivalent to zero.

③ Oracle has changes it's rules about empty string and Null Value in new Version.

④ Null Multiplied by 10 is NULL.

Ex: stud

sid	name
101	null
102	-

① insert into stud values ('101', null);

② insert into stud values (102, '');

Ans:1 Select * from stud where Name = '';

Ans: Not retrive any Value.

Ans:2 Select * from stud where Name IS NULL;

Ans: Both row retrived.

* Modification of the Database:

- The Content of the database may be modified Using the following operations
  - insertion
  - deletion
  - Updating
- All these operations are Expressed Using the assignment operator.

* insertion:

- specify a tuple to be inserted.
- write a query whose result is set of inserted tuples

→ $r \leftarrow r \cup E$

r: table, ∪: Union, E: data need to insert

Ex insert information in student John has 120 c-i from surat.

student ← Student ∪ {('John', 120, 'surat')}

Student ← student ∪ {("Ajay', Null, 'Surat')}

* Deletion:

- Can delete only whole tuple, cannot delete value of particular attribute.
- $r \leftarrow r - E$

r: table, E: condition

Ex student ← Student − $\sigma_{student\_city='surat'}$ (student);

student ← student − $\sigma_{sid \geq 10 \text{ and } sid \leq 20}$ (student);

* Updating:

- A mechanism to change a value in a tuple without changing all value in tuple.

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(E)$$

Ex ~~Student ←~~

Account ← $\Pi_{account\_no,\ branch\_name,\ balance * 0.05}$ (account)

Ex 2:

Account ← $\Pi_{account\_no,\ branch\_name,\ balance * 0.06}(\sigma_{branch\_name='Surat'}(account))$

Unit: 3

Relational Database Design

The goal of relational database design is to generate a set of relation schemas that allows us to store information without unnecessary redundancy and also allows us to retrive information easily.

This is done by designing schemas is in one of the desirable normal forms.

* Features of Good relational Designs: / Design alternatives

① Larger Schemas: It should not be so large that it has repetation of information.
- When we combine two small schema it doesn't have inconsistency.
- primary key concept is satisfied.

Ex ① instructor (id, name, dept-name, salary)
② department (dept-name, building, budget).

inst_dept (id, name, salary, dept_name, building, budget)

- In this example table 1 & table 2 join, but whenever we add instructor we need to add department name with building & budgent info. Here this building & budget information is repeating.

- Suppose we creating new department in the University, but we can not represent department with building & budget info Unless & Until the department has atleast one instructor. in means first hire instructor then start new department.

② Smaller Schema: When we divide any large schema in small schemas the selection of primary key should be proper. & when we apply joins to create the one table primary key should proper. - Combined table we get exact answer.

* Rules

① Design a relation schema in such a way that it is easy to explain its meaning

② Design the database table's in such a way that there is no insertion, deletion or Update anomalies.

③ Avoid multiple value in one attribute, and null value should be avoided.

* Anomalies in DBMS:

There are three types of anomalies that occur when the database is not normalized. There are insertion, Update and deletion anomalies.

Employee

eid	name	address	department
101	Rick	Delhi	CE
101	Rick	Delhi	IT
123	Maggie	Agra	MECH
166	John	Surat	EC
166	John	Surat	Civil

Update Anomaly: in the above table we have two rows for employee RICK as he belongs to two departments of the company. if we want to update the address of RICK then we have to update the same in two rows, otherwise data will become inconsistent.

if we not update data (address) in both row then same person have two different address which is incorrect.

Insert Anomaly: Suppose new employee Jake joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp-department field doesn't allow nulls.

Delete Anomaly: Suppose, if at a point of time, company closes the department MECH then deleting the rows that are having department as MECH would also delete the information of employee Maggie since she is assigned only to this department.

⇒ To overcome these anomalies we need to normalize the data.

- Relational database design should be in some manner that it will redundancy as well as these anomalies don't occure in it. & if it is there then we can apply Normalization on it which means decomposition of database.

* Functional dependencies (FD):-

- The single most important concept in relational schema design theory is that of a functional dependency.

- A FD is a relationship between or among attributes such that the values of one attribute depend on or determined by the value of other attributes.

- A FD is a type of constraint that is generalization of the notion of key.

def: Consider a relational schema R & let $a \subseteq R$, $b \subseteq R$, The FD

$a \longrightarrow b$.

a determined of b.

b depend on a.

- An FD cannot be inferred automatically from a given relation extention r but must be defined explicitly by someone who knows the semantics of the attributes of R.

Ex:1

Student

sid	name	city
1	John	Surat
2	Rey	Valsad
3	Maggie	Surat

⇒ FD:

sid → name

~~sid, name → city~~

~~name → city~~

Ex: sid → name because whenever sid = 1 then Name = John only.
sid = 2 then Rey only
sid = 3 then Maggie only.

Functional Dependencies: if R is

A → B

- Where A and B are sets of attribute of R
- A is called left hand side (LHS) & B is right hand side (RHS).
- The meaning of FD is that for every value of A, there is Unique value of B.
- We say FD holds for R if, for any instance of R, whenever two tuples agree in value for all attribute of A, they also agree in value for all attributes of B.

Ex: Student (sid, name, dob, Phone-number).

sid → {name, dob}

sid → {Phone_number} — X

in that case assuming student have multiple phone number. So sid → Phone-number does not hold.

sid → name
sid → dob

Ex:2

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

⇒ Valid FD

B → C
C → B
{A, B} → C
{A, B} → D
{C, D} → B

Invalid FD

A → B (tuple 1 & 2 violate)
B → A (tuple 2 & 3 violate)
D → C (tuple 3 & 4 violate).

* Closure of FD : if F is a set of FD then the closure of F, denoted as F^+, is the set of all Functional dependecies logically implied by F.

- Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of Functional dependencies.

[1] Reflexive Rule : if A is set of attr. & $B \subseteq A$ then

$$A \rightarrow B$$

[2] Augmentation Rule : if $A \rightarrow B$ holds then

$$CA \rightarrow CB$$

[3] Transitivity Rule : if $A \rightarrow B$ & $B \rightarrow C$ then

$$A \rightarrow C$$

* Some Inference Rule

[4] Union : if $A \rightarrow B$ & $A \rightarrow C$ then

$$A \rightarrow BC$$

[5] Decomposition : if $A \rightarrow BC$

$$A \rightarrow B$$
$$A \rightarrow C$$

[6] Pseudo Transitivity : if $A \rightarrow B$ holds & $CB \rightarrow D$ then

$$AC \rightarrow D$$

ex:1 R(A, B, C, D, E)

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

} compute closure F^+.

Ans:

$A \rightarrow BC$
↓
$A \rightarrow B, A \rightarrow C$ (Decomposition)
↓
$A \rightarrow D$ ($B \rightarrow D$ Transitivity)
↓
$AC \rightarrow E$ ($CD \rightarrow E$ Pseudo Transitivity)
↓
$A \rightarrow CD$ ($A \rightarrow C$, $A \rightarrow D$ Union)
↓
$A \rightarrow E$ ($CD \rightarrow E$ Transitivity)
↓
$A \rightarrow A$ (Reflexivity).
↓
$A \rightarrow ABCDE$ (Union)
↓
$E \rightarrow ABCDE$ (Transitivity)
↓

$CD \rightarrow ABCDE$

↓

$B \rightarrow D$

↓

$BC \rightarrow CD$ (augmentation)

↓

$BC \rightarrow ABCDE$ (Transitivity)

etc.....

* Closure of attribute Set :

- Given R a set of FD's, F that holds in R, & set of attributes $\geq$ in R, the closure of $\geq$ w.r.t $F(\geq^+)$ is the set of attribute.

- Use to Find Candidate Key, Superkey in Relation

Ex:1

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

$A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

$A^+ \rightarrow ABC$

$\rightarrow ABCD$

$\rightarrow ABCDE$ ← CK

$B^+ \rightarrow BD$ ← X

$E^+ \rightarrow EABCD$ ← CK

$C^+ \rightarrow C$

$D^+ \rightarrow D$

Note:
Right Hand side attribute is always part of Candidate key.

Ex:2

R(A, B, C, D)

$A \rightarrow BC$

$C \rightarrow D$

$A^+ \Rightarrow \{ABCD\}$ ✓

$B^+ \Rightarrow \{B\}$ ×

$C^+ \Rightarrow \{CD\}$ ×

R.H.S
$A^+ \rightarrow ABC$

Ex:3

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

FD: $A \rightarrow BC$

$B \rightarrow E$

$CD \rightarrow EF$

$A^+ = \{ABCE\}$

$B^+ = \{BE\}$

$C^+ = \{C\}$

$D^+ = \{D\}$

$E^+ = \{E\}$

$F^+ = \{F\}$

$AF^+ = ABCEF$

$ACD^+ = ACDBEF$ → CK.

R.H.S
A, D
$AD^+ = ABCDEF$
→ S.K

Ex:4

R(A B C D E H)

$A \rightarrow BC$

$CD \rightarrow E$

$E \rightarrow C$

$D \rightarrow AEH$

$ABH \rightarrow BD$

$DH \rightarrow BC$

$A^+ = ABC$ ×

$E^+ = EC$ ×

$D^+ = DAEHBC$ ✓ → CK.

R.H.S
×

So Here
AD
CD
BD
← Superkeys & Union of L.H.S Attrib.

Ex:5

R(A B C D E)

$A \rightarrow B$

$BC \rightarrow E$

$ED \rightarrow A$

$A^+ \Rightarrow AB$ ×

$BC^+ \Rightarrow BCE$ ×

$ED^+ \Rightarrow EDAB$ ×

$AB^+ \Rightarrow AB$ ×

$BD^+ = BD$ ×

$AD^+ = ADB$ ×

$BCD^+ = BCDEA$ ✓ Key

$ACD^+ = ACDBE$ ✓ Key

R.H.S
C, D
$CD^+ = CD$
$ACD^+ = ABCDE$
$BCD^+ = BCDEA$

Ex:6 R(ABCDE)

A → BD E
C → E
D → BC

$A^+ = ABDEC$ ✓ key
$C^+ = CE$ ×
$D^+ = DBCE$ ×
$AC^+ = ABCDE$ ✓
$CD^+ = CEBD$ ×
$AD^+ = ABDEC$ ✓
$ACD^+ = ABCDE$ ✓

R.H.S
A
$A^+ = ABDEC$ ✓

Ex:7 R(ABCDEFGHIJ)

AB D → E
AB → G
B → F
C → J
CJ → I
G → H

- First take L.H.S Single attribute

$B^+ = BF$
$C^+ = CJI$
$G^+ = GH$

- Second take L.H.S Double attribute.

$AB^+ = ABGFH$
$CJ^+ = CJI$

- Three variable & L.H.S

$ABD^+ = ABDEGF$
$ABC^+ = ABGFHCJI$ ✓
$ABCD^+ = ABCGHFEDI$ ✓ G

$ABC^+ = ABGJHF$

A.H.S A, B, C, D
$ABCD^+ = ABCDEFGHJ$

Ex:8 R(ABCDEFGHIJ)

AB → C
A → DE
B → F
F → GH
D → IJ

$A^+ = ADEIJ$
$B^+ = BFGH$
$F^+ = FGH$
$D^+ = DIJ$
$AB^+ = ABCDEFGHIJ$ ✓
ABD^+ } Superkey
ABF^+ }

R.H.S = A, B, D
$AB^+ = ABCDEFGHIJ$

Ex:9 R(ABCDE)

A → B
B → A
B → C
D → A

R.H.S D
$D^+ = ABCD$ — Candidate key

Ex:10 R(ABCDEFG)

A → B
B → ACE
C → BFD
F → DG

R.H.S
×

$A^+ = ABCDEFG$ ✓
$B^+ = ABCDEFG$ ✓
$C^+ = ABCDEFG$ ✓
$F^+ = FDG$ ×

Functional Dependency:

Trivial Attribute: y is subset of x

$X \to Y$ exists and $Y \subseteq X$

Ex $ABC \to BC$ $\{BC \subseteq ABC\}$ ← TFD

↳ BC is subset of ABC

Non-trivial FD: y is Not subset of x.

$X \to Y$ exists and $Y \not\subseteq X$

Ex $AB \to C$

$ABC \to BD$

Completely Non-trivial FD:

$X \to Y$ and X intersect $Y = \phi$

Ex $A \to D$

☆ Prime & Non-prime attribute:

Prime Att: it is a part of key

Non-Prime Att: Not part of key

Ex: R(A,B,C)

$A \to B$

$B \to C$

$A^+ = ABC \longrightarrow$ Key

Here A is part of key → Prime

BC is not part of key → Non prime

Ex:2 R(ABCDE)

$A \to D$ $A^+ \to ADBC$

$D \to B$ $B^+ \to BC$

$B \to C$ $C^+ \to C$

$E \to B$ $D^+ \to DBC$

$E^+ \to EBC$

R.H.S A & E

$AE^+ \to ADBCE \longrightarrow$ key.

A & E is prime attributes.

B, C & D is Non-prime attributes

✱ Canonical Cover / irreducible Set of FD:

- An Attribute of FD is said to be extraneous if we can remove it without changing the closure of the set of FD.

- For make irreducible set of FD we Need to

Step:1 ① Redundant FD
Step:2 ② Redundant left-hand Attribute
Step:3 ③ Redundant Right-hand Attribute.

$A \to D$

Ex:1 R(ABCDE)

$A \to D$
$BC \to AD$
$C \to B$
$E \to A$
$E \to D$

Step:1 Make Singl Attribute in R.H.S.

$A \to D$
$BC \to A$
$BC \to D$
$C \to B$
$E \to A$
$E \to D$

Step:2 See Whether any Extraneous attribute on L.H.S

Note
↳ If any FD have only one attribute L.H.S then there is Not any Extraneous Attribute.

↳ we make FD at a time.

$BC \to A$	$BC \to D$
B^+ C^+	B^+ C^+
B CBAD	B CABD
the important point is C^+ find B. So B can eleminate L.H.S. So Now FD is	↳ B can remove L.H.S.
$C \to A$	$C \to D$

Step:3 There is no FD which have more than one Attribute in L.H.S.
Whether we have any redundent FD in step:1.

✓ 1. $A \to D$ ① - Find A^+ for remove 1 FD.
✓ 2. $C \to A$ $A^+ = A$ they cannot find D. So we
× 3. $C \to D$ can not remove this FD.
4. $C \to B$ ② - $C^+ \to CDB$ ×
5. $E \to A$ ③ - $C^+ \to CABD$ ✓
6. $E \to D$ ④ - $C^+ \to CAD$ ×
⑤ - $E^+ \to EAD$ × ← Trick only A Set from B, only one for A
⑥ - $E^+ \to EAD$ ✓

∴ irreducible set is :

$A \to D$
$C \to A$
$C \to B$
$E \to A$

Ex:2 R(ABC)

$A \to B$
$B \to C$
$A \to C$
$AB \to A$

Ans

Step:1 singe Attribute in R.H.S which is already

Step:2 find extraneous Attribute in L.H.S

$AB \to A$

A^+ B^+

ABC BC

Step:3

$A \to B$	① $A^+ \to AC$	×
$B \to C$	② $B^+ \to B$	×
$A \to C$	③ $A^+ = ABC$	✓

Ans

$A \to B$
$B \to C$

Ex:3 R(ABCD)

$A \to BC$
$B \to C$
$A \to B$
$AB \to C$

Step:1 singe Att on R.H.S

$A \to B$
$A \to C$
$B \to C$
× $A \to B$
$AB \to C$

Step:2 remove extra Att. on L.H.S

$AB \to C$

A^+ B^+

ABC BC

↳ here B is extra so $A \to C$ which is already so we get

Step:3

$A \to B$	① $A^+ \to AC$	×
$A \to C$	① $A^+ \to ABC$	✓
$B \to C$	③ $B^+ \to BC$	

$A \to B$
$B \to C$

Ex:4 R(ABCDE)

$A \to BC$

$CD \to E$

$B \to D$

$E \to A$

Step:1 Single (decompose) in R.H.S

$A \to B$

$A \to C$

$CD \to E$

$B \to D$

$E \to A$

Step:2 Extra. on L.H.S

$CD \to E$

C^+	D^+
C	D

Try to replace this attribute with single.

$A \to B$

$B \to D$

$A \to C$

We can not remove any attribute.

Step:3

(1)	$A \to B$	$\to A^+ \to AC$
(2)	$A \to C$	$\to A^+ \to ABD$
(3)	$CD \to E$	$\to CD^+ \to CD$
(4)	$B \to D$	$\to B^+ \to B$
(5)	$E \to A$	$\to E^+ \to E$

Ex:5 R(ABCD)

$A \to B$

$AB \to C$

$D \to AC$

$D \to E$

Step:1 single Attribute on R.H.S

$A \to B$

$AB \to C$

$D \to A$

$D \to C$

$D \to E$

Step:2 Check Extraneous attribute L.H.S

$AB \to C$

A^+	B^+
AB	B

↳ Here i find A to B. So B is Extraneous.

Step:3

(1) $A \to B \to A^+ \to AC$ ✗

(2) $A \to C \to A^+ \to AB$ ✗

(3) $D \to A \to D^+ \to DCE$ ✗

(4) $D \to C \to D^+ \to DEABC$ ✓

(5) $D \to E \to D^+ \to DABC$ ✗

$\Rightarrow$

$A \to B$

$A \to C$

$D \to A$

$D \to E$

Ex:6 R(ABCD)
A → BC
D → AB

Step:1 A → B
A → C
D → A
D → B

Step:2 there is Non double attribute l.H.S.

Step:3 ① A → B
② A → C
③ D → A
④ D → B

A+ → AC ✗
A+ → AB ✗
D+ → DB ✗
D+ → DABC ✓

A → B
A → C
D → A

Ex:7 R(ABCD)
A → B
C → B
D → ABC
AC → D

Step:1 A → B
C → B
D → A
D → B, D → C, AC → D

Step:2 AC → D

A+ C+

AB CB

↳ we can not find A → C or C → A so Non Extra Att.

Step:3

A → B	A+ → A ✗
C → B	C+ → C ✗
D → A	D+ → DBC ✗
D → B	D+ → DCAB ✓
D → C	D+ → DAB ✗
AC → D	AC+ → ABCD ✗

A → B
C → B
D → A
D → B
AC → D

Q:8 R(ABC)
AB → C
C → B
A → B

Ans Step:1 already all R.H.S in simple.

Step:2 AB → C

A^+	B^+
AB	B

↳ Here A finds B so B is extra.

(A → C)

Step:3.

A → C	A^+ → AB	✗
C → B	C^+ → CB	✗
A → B	A^+ → ACB	✓

A → C
C → B

Decomposition:

R

sid	name	city
1	Joe	Surat
2	Alice	Bardoli
3	Alice	Surat

R_1 (sid, name)

sid	name
1 ~~Joe~~	Joe
2 ~~Alice~~	Alice
3 ~~Alice~~	Alice

R_2 (name, city)

name	city
Joe	Surat
Alice	Bardoli
Alice	Surat

$R_1 \bowtie R_2$

sid	name	city
1	Joe	Surat
2	Alice	Bardoli
2	Alice	Surat
3	Alice	Bardoli
3	Alice	Surat

↳ This is Not original R, so this decom. known as Lossy Decomposition.

R_1 (sid, name) R_2 (sid, city)

Codd Classification of FD's:

1. Partial FD's
2. Transitive FD's
3. Full FD's

Classification of dependencies

- FD
  - 1NF
  - 2NF
  - 3NF
  - BCNF
- MVD
  - 4NF
- JD
  - 5NF
- ID
- TD

[1] First Normal Form (1NF):

- Attribute of table cannot hold multiple values.
- It should hold Atomic Value.

Ex:1

Student	Age	Subject
Adam	15	Biology, Maths
Alex	14	Maths
Stuart	17	Maths

(C)

Adam	15	Biology	(candidate key: {Student, Subject})
Adam	15	Maths	
Alex	14	Maths	
Stuart	17	Maths	

[2] Second Normal Form (2NF):

- If it is already in 1NF
- It must be free from partial FD.

Table 1

Student	Age
Adam	15
Alex	14
Stuart	17

Table 2

Student	Subject
Adam	Biology
Adam	Maths
Alex	Maths
Stuart	Maths

[3] Third Normal Form (3NF):

- It must be 2NF.
- It must be free from transitive dependency.

Ex:1 R(ABCDEFGH)
A → BC
ABE → CDGH
C → GD
D → G
E → F

} Find this FD in which NF & decompose it to 3NF.

Ans:

Step:1 Find Candidate key. AE^+
$AE^+ = ABCDEFGH$

Step:2 Convert it to 2NF. Find A^+ & E^+

$A^+ = ABCDG$
$E^+ = EF$

} H is remaining.

$R_1 = ABCDG$
$R_2 = EF$
$R_3 = AEH$

2NF

Step:3 3NF Find Transitive relation.

$R_1 = ABC$
$R_2 = DG$
$R_3 = CGD$
$R_4 = EF$
$R_5 = AEH$

3NF

Ex:2 R(ABCDE)
AB → CDE
A → C
D → E

Ans:

Step:1 Find Candidate Key AB^+

Step:2 Remove Partial Dependency

$A^+ = AC$
$B^+ = B$

2NF
$R_1 = AC$
$R_2 = ABDE$

} ABCDE

Step:3 Remove Transitive Dependency.

$R_1 = AC$
$R_2 = ABD$
$R_3 = DE$

Ex:3 $R(ABCDE)$

$AB \rightarrow CDE$

$A \rightarrow C$

$C \rightarrow D$

Ans Steps: Find Candidate key = AB

Step:1 FD.

$A^+ = AC$

$B^+ = B$

$R_1 = ACD$

$R_2 = ABE$

2NF

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

$AB \rightarrow C$

$AB \rightarrow D$

$AB \rightarrow E$ FFD

Step:2 TD

$R_1 = AC$

$R_2 = CD$

$R_3 = ABE$

3NF

$A \rightarrow C$

$C \rightarrow D$

$AB \rightarrow E$

Ex:4 $R(ABCDEF)$

$A \rightarrow FC$

$C \rightarrow D$

$B \rightarrow E$

Ans: Step:1 Find CK.

$AB^+ = AFCEDB$ — B — A

$AB^+ = ABCDEF$ ✓

Step:2 Remove FD.

$A^+ = AFCD$

$B^+ = BE$

$R_1 = AFCD$

$R_2 = BE$

$R_3 = AB$

2NF

Step:3

$R_1 = AFC$

$R_2 = CD$

$R_3 = BE$

$R_4 = AB$

3NF

Ex:5 R(A B C D E F G H I J)

AB → C
BD → EF
AD → GH
A → I
H → J

Ans:

Step:1 Find Key = ABD

Step:2

$AB^+ = ABCI$
$BD^+ = BDEF$
$AD^+ = ADGHIJ$

$R_1 = ABCG$
$R_2 = BDEF$
$R_3 = ADGHIJ$
$R_4 = ABD$

$A^+ = AI$
$B^+ = B$
$D^+ = D$

$A^+ = AI$
$D^+ = D$

Ex:6 Student (sid, name, age, Subject, Address).

sid	Sname	Age	Subject	Address
1	Parth	15	Bio, Maths	Surat
2	Rui	14	Maths	Vapi
3	Riya	15	Chemistry	Surat

Ans:

⇒ Step:1 Check Given schema is in INF. → NO

↳ Bcs multiple subject in single cell.

Sid	Sname	Age	Subject	Address
1	Parth	15	Bio	Surat
1	Parth	15	Maths	Surat
2	Rui	14	Maths	Vapi
3	Riya	15	Chemistry	Surat.

⇒ Step:2 Find Candidate key.

P.K & C.K is {sid, Subject}.

⇒ Step:3 Check Whether have P.D or Remove P.D

sid → Sname, Age, Address.

R_1

sid	Sname	Age	Address
1	Parth	15	Surat
2	Rui	14	Vapi
3	Riya	15	Surat

R_2

sid	Subject
1	Bio
1	Maths
2	Maths
3	Chemistry

Transaction Management

"A transaction is a program Unit whose execution may change the contents of a database."

database (before) → transaction → database (after) (consistant state).

"Transaction is used to represent a logical Unit of database processing that must be completed in its entirety to ensure correctness".

Ex:1 Transfer 100 rupees Account A to Account B.
A = 1000, B = 2000

1. read (A)
2. A := A - 100;
3. write (A)
4. read (B)
5. B := B + 100;
6. write (B)

Commit

Power Failure

Scenario: 1
Before transaction
A+B = 3000
1000 + 2000 = 3000 ✓

After transaction
A+B = 3000
900 + 2100 = 3000 ✓

Scenario: 2
A+B = 3000 ← X
After transaction
A = 900 | A+B = 2900
B = 2000 |
Both are not same

* ACID Properties of Transaction:

[1] Atomicity: Ensures that a transaction will run to completion as an individual Unit, at the end of which either no change has occurred to the database, or database has been changed in a consistent manner. (All or none)

- See Example 1

[2] Consistency: Correctness or ensures that if the database was in a consistent state before the state of a transaction, then on termination, the database will also be in a consistent state.
- see Example 1

[3] Isolation: indicates that the actions performed by a transaction will be isolated or hidden from outside the transaction until it terminates

A=1000
B=2000

T_1	T_2
read (A)	
A := A - 100	
W (A)	
	read (A)
	read (B)
	print "A+B".
read (B)	
B := B + 100	
W (B)	

A + B = 900 + 2000 = 2900 ← X

Concurrent Execution of Transaction:

It implies interleaving execution of operations of transaction.

Benefits:
1. it helps in reducing waiting time.
2. improved throughput and resource utilization

→ But every good option will have some series drawback.

Schedule: it represents the order in which instructions of a transactions are executed.

$T_1 \rightarrow T_2 \rightarrow T_3$ ← schedule:

Problem with Concurrent Execution:

[1] Lost Update Problem [W-W conflict]:

occurs when two transaction that accesses the same db items have their operations interleaved in a way that makes the value of the db item incorrect.

A = 1000	T_1	T_2
1000	R(A)	
950	A = A − 50	
		R(A) 1000
		A = A + 100 1100
950	W(A)	
		W(A) − 1100

Lost Update

1000 − 50 = 950
950 + 100 = 1050

[2] Uncommitted dependency / Temporary Update / dirty read [WR conflict].

	T_1	T_2
100	R(A)	
120	A = A + 20	
120	W(A)	→ Update value.
		R(A) 40
		A = A + 10 130 | 110
		W(A) 130
	R(B) →	
		commit 130

Rollback

Power Failure.

A = 100

Ans = 110

[3] ~~Inconsistent Analysis~~ / Unrepeatable read
[WR conflict].

A=1000

	T1	T2	
1000	R(A)		
		R(A)	1000
		A = A+2000	3000
		W(A)	3000
3000	R(A)		

[4] Inconsistent Analysis :
Incorrect Summary Problem : } A=100 B=200

	T1	T2	
		sum=0	
		R(A)	100
		sum=sum+A	sum=100
		R(B)	200
		sum=sum+B	sum=300
200	R(B)		
B=300	B = B+100		
	W(B)		

All Problem in Read/write form.

1. RR : T1 & T2 same time read.
2. RW : ~~T1~~ want to read A & T2 want W.
(3) non-repeatable read.
3. WR : WR conflict (dirty read.
4. WW : WW conflict (dirty write.

☆ Concurrency Control : It is the process of managing simultaneously execution of transaction in a shared database, to ensure the serializability of transaction.

Purpose of Concurrency Control:

1. to enforce Isolation.
2. To preserve database consistency.
3. To resolve RW conflict & WW conflict (conflict operation.

→ Many concurrency Technique for control.

(1) Lock-based Protocol:
lock acquire → to access data item
Release lock → after completion of transaction)

All data items must be accessed in mutually exclusive manner.

Types of locks

Shared lock
↳ lock-S
- Not Update
- only read.

Exclusive lock
↳ lock-X
↳ both Read & Write

* Compatibility of lock modes.

	T_2 S	T_2 X
T_1 S	✓	X
T_1 X	X	X

Ex:

T_1	T_2
Lock-X(A)	Exclusive lock
R(A)	
A := A − 50	
W(A)	
UnLock(A)	
	Lock-S(B) — Shared lock
	R(B)
	UnLock(B)

Note: Any number of transaction hold Shared lock.
But Exclusive Lock only one trans.

* Conversion of Locks:

Upgrading — R lock — write lock
Downgrading — write lock — read lock.

(2PL) Variation of lock-base protocol

* Two phase Locking Protocol:
requires both locks and Unlock being done in two phase.

Phases

Growing phase (Expanding) — Shrinking phase.

- new locks on items can be acquired — Existing locks, but no new lock can be acquired.

Locking:

⇒ Locking point: — upwards

Locks are acquired — Releasing locks.

Locked phase.

9

transaction works / operation start at a time must in one phase. if 4 data items then it acquire lock all 4 data items. Release also same.

* 2PL protocol enforce serializability but may reduce concurrency due to the following reasons.

① Holding lock un-necessary

T1 — 1, 2, 3, ④ ← T2.

② Locking to early.

③ Penalty to other transaction

---

* Evolution of 2PL locking protocol.

2PL

Conservative 2PL (static)	Strict 2PL	Rigorous 2PL
- Acquire all lock before it starts - Release all lock after commit	- Exclusive lock can't be released until commit transaction not release any of exclusive lock until transaction commit.	transaction can't be any of shared or exclusive lock until transaction committed.
⇒ Avoid cascading rollback. ⇒ deadlock free	deadlock may occur. help in cascadeless rollback.	avoid cascading rollback.

✡ Solution of transaction problem

[1] Lost Update Problem :-

T1	T2
lock-S (A)	
R(A)	
	lock-S (A)
	R(A)

T1	T2
request (X-lock)	
acquire (X-lock)	
R(A)	
	request (lock-X) (A)
	wait
W(A)	wait
release X-lock	wait
	acquire (lock-X) (A)
	W(A)
	.
	.
	.

[2]

T1	T2
Acquire (lock-X)	
R(A)	
A = A+20	
W(A)	
	request (lock-X)
Rollback	wait
Rollback	.
R(B)	.
	wait for lock.
	acquire (lock-X)

[3]

T1	T2
acquire (S)	
R(A)	
	request (lock-X)
	wait
	.
	.
	.
	.
R(A)	
release (S)	
	acquire (lock-X)

[4]

T1	T2
	sum=0
	acquire (s-lock)
	R(A)
	sum=sum+A ;
	acquire (s-lock)
	R(B)
	sum=sum+B ;

request(x-lock)
R(B)
wait
.
.
.

Deadlock: A system is in a deadlock state if there exists a set of transaction such that every transaction in the set is waiting for another transaction in the set.

T1	T2
lock(x)	
R(A)	
W(A)	
	lock(x)
	R(B)
	W(B)
lock(x)(B)	
wait	
wait	
	lock (x).(A)
	wait

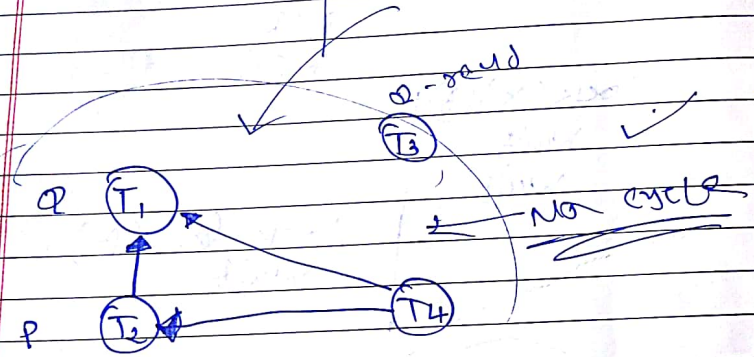
[5]

T1	T2
(x,y)	(x,y)
lock(x)	lock(y)
(x+y)	x-y

(wait for graph)

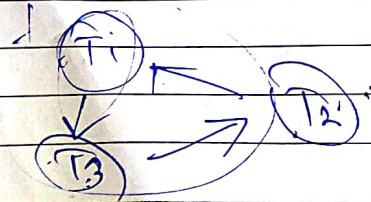
[1] Deadlock detection: T_j wait for resource held by T_2

Transaction	Datatitem	Lockmode
T_1	Q	shared / Excuy
T_2	P, Q	Excuy
T_3	Q	shared
T_4	P	Exclusive
T_4	Q	Excu.



→ Not to find whether a deadlock or not
↳ If cycle Exist then deadlock.

Wait for-Graph



Avoidance

[2] Deadlock prevention: system will never enter

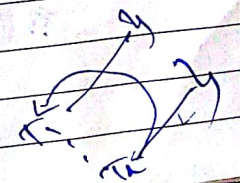
1. mutual Exclusion
2. Hold & wait
3. No preemption
4. circular wait.

① Use of time stamp: Assume T_i request datatitem currently held by T_j.

① Wait die scheme: if $T_s(T_i) < T_s(T_j)$
(T_i is older the T_j)
T_i die or wait

② Wound-wait Scheme:
if $T_s(T_i) < T_s(T_j)$

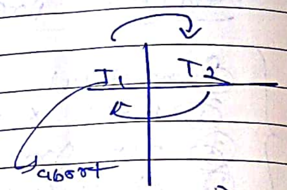
② Time out Based scheme.



classmate
Date ____
Page ____

[3] Deadlock recovery:

① Selection of victim.

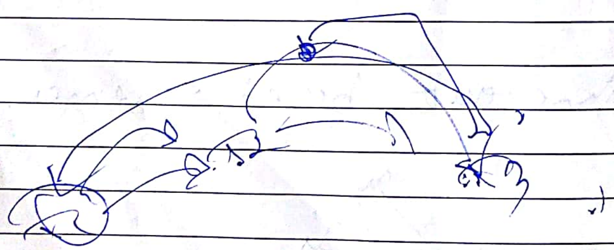


↳ minimum cost
① ↳ Length of transaction
↳ younger trans.
② ↳ Less data item used
③ ↳ data item that are to be locked.
(more data items to lock)
④ How many transaction to be roll back.
(minimum cascading rollback)

T1 — T2 — T3 — T4

② Rollback : ① Full rollback.
② Partial rollback.
(save point, checkpoint)

③ Starvation ! take care while selecting a victim does not of



classmate
Date ____
Page ____

5.6 Recovery and Atomicity

- Recovery algorithms are technique to ensure db consistency and transaction atomicity and durability despite failure.

- Recovery algo have two part
① Action taken during normal transaction processing to ensure enough info exists to recover failure
② Action taken after failure

* Type of Storage :
① Volatile storage : main memory, cache (does not survive system crash)
② Nonvolatile storage : disk, tape, (survive system crash)
③ Stable storage : mythical form of storage that survive all failure. by maintaining multiple copies on distinct nonvolatile media.

- Recovery & System Approaches :
① Log based recovery
② Shadow copy. (Paging)

[1] Log-Based Recovery :

- Log is kept on stable storage. The log is sequence of log records.
- When transaction start it write <Ti start> log record. before Ti execute write(x) a log record <Ti, X, Vi, Vj> <Ti, commit>
- Two approach using logs
① deferred database modification
② immediate database modification.

A=1000, B=2000

① Deferred Database Modification: Scheme records all modifications to the log, but defers all the writes to after Partial commit.

<T0, start>	<T0, start>	<T0, start>
<T0, A, 950>	<T0, A, 950>	<T0, A, 950>
<T0, B, 2050>	<T0, B, 2050>	<T0, B, 2050>
	<T0, commit>	<T0, commites>
	<T1, start>	<T1, start>
	<T1, C, 600>	<T1, C, 600>
		<T1, commites>
(a)	(b)	(c)

if log on stable storage at time of crash is as in case.

(a) No redo action
(b) redo (T0)
(c) redo (T0), redo (T1)

② Immediate Database modification: Allows database updates of an uncommitted transaction to be made as the writes are issued.

Scheme

<T0, start>	<T0, start>	<T0, start>
<T0, A, 1000, 950>	<T0, A, 1000, 950>	<T0, A, 1000, 950>
<T0, B, 2000, 2050>	<T0, B, 2000, 2050>	<T0, B, 2000, 2050>
	<T0, commites>	<T0, commit>
	<T1, start>	<T1, start>
	<T1, C, 700, 600>	<T1, C, 750, 600>
		<T1, commites>
(a)	(b)	(c)

(a) Undo (T0): B is restored to 2000 and A to 1000
(b) Undo (T1) and redo (T0): C is restored to 700 and then A and B are set to 950 and 2050 respectively.
(c) redo (T0) and redo (T1): A and B are set to 950 and 2050 respectively. Then C is set to 600.

[2] Shadow Copy:

[3] Checkpoints:

☆ Schedule: A sequence of instructions that specify the chronological order in which instruction of concurrent transaction executed.

* Schedule 1

T1	T2
read(A)	
A := A-50	
write(A)	
read(B)	
B := B+50	
write(B)	
commite	
	read(A)
	temp := A*0.1
	A := A-temp
	write(A)
	read(B)
	B := B+temp
	write(B)
	commite

* Schedule 2

T1	T2
	read(A)
	temp := A*0.1
	A := A-temp
	write(A)
	read(B)
	B := B+temp
	write(B)
	commite
read(A)	
A := A-50	
write(A)	
read(B)	
B := B+50	
write(B)	
commite	

* Schedule 3

T_1	T_2
read(A)	
A := A − 50	
write(A)	
	read(A)
	temp := A * 0.1
	A := A − temp
	write(A)
read(B)	
B := B + 50	
write(B)	
commit	
	read(B)
	B := B + temp
	write(B)
	commit

↳ The following schedule is not a serial schedule, but it is equivalent to schedule 1.

* Schedule 4

T_1	T_2
read(A)	
A := A − 50	
	read(A)
	temp := A * 0.1
	A := A − temp
	write(A)
write(A)	
read(B)	
B := B + 50	
write(B)	
commit	
	read(B)
	B := B + temp
	write(B)
	commit

↳ The following concurrent schedule does not preserve the sum of "A + B".

→ Schedule 1, 2, 3 Preserves A + B.

* Serializability:

A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule.

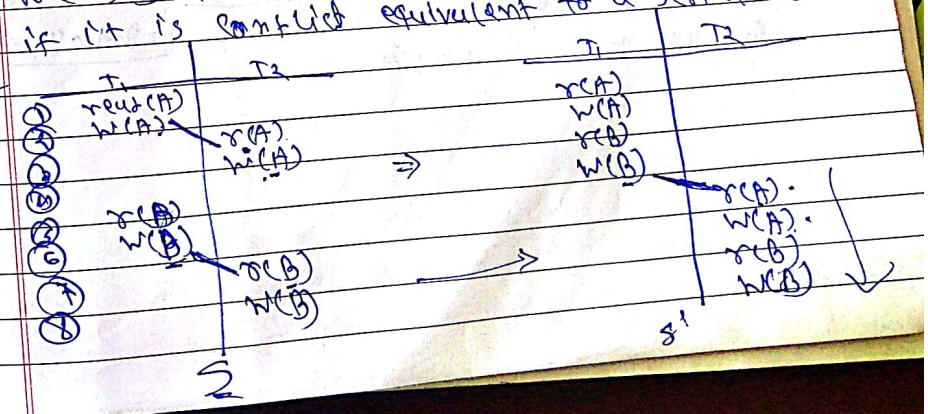
- different forms of schedule equivalence give rise to the notion of:
  1. Conflict Serializability
  2. View Serializability.

[1] Conflict Serializability:

Instruction T_1 & T_2 conflict if and only if there exists some item X accessed by both T_1 and T_2, and atleast one of these instruction wrote (write) X.

1. T_1 = read(X), T_2 = read(X) — Not conflict
2. T_1 = read(X), T_2 = write(X) — conflict
3. T_1 = write(X), T_2 = read(X) — conflict
4. T_1 = write(X), T_2 = write(X) — conflict.

- If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instruction, we say that S and S' are conflict ~~serializable~~ equivalent.
- We say that a schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

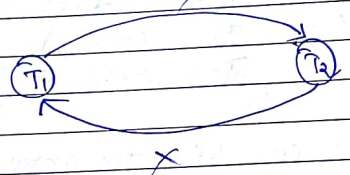


Ex:3 Example of a schedule, that is not conflict serializable

T_1	T_2
$r(x)$	
	$w(x)$
$w(x)$	

We are unable to swap instruction in the above schedule to obtain either the serial schedule $\langle T_1, T_2 \rangle$ or the serial schedule $\langle T_2, T_1 \rangle$.

- Precedence Graph: a directed graph where the vertices are the transactions. We draw an arc from T_i to T_j if the two transaction conflict. Used for conflict serializability.

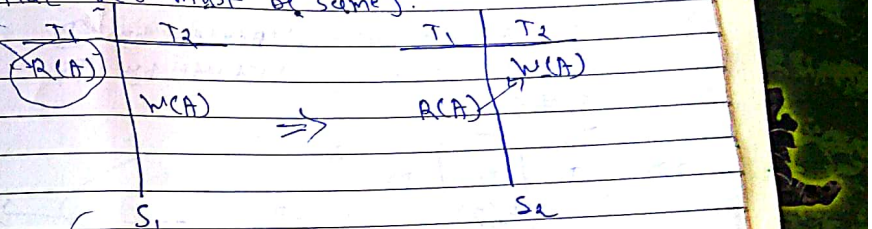


- A schedule is conflict serializable if and only if its precedence graph is acyclic.

- If Precedence graph is acyclic, the serializability order can be obtained by a topological sorting of the graph.

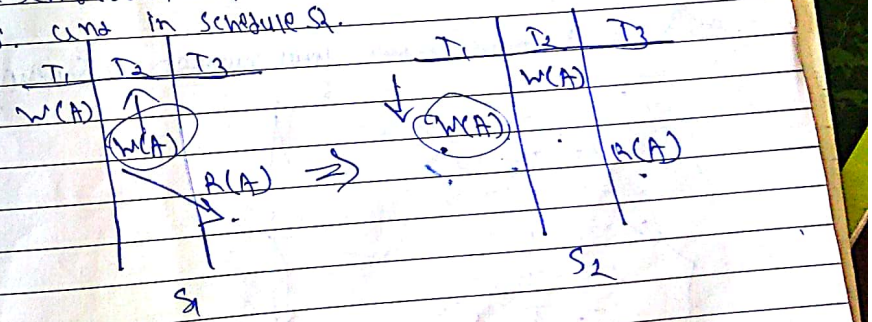
(2) View Serializability: Let S and S' be two schedules with the same set of transaction. S and S' are view equivalent if the following three condition are met, for each data item X.

① → If in schedule S, transaction T_i reads the initial value of X, then in schedule S' also transaction T_i must read the initial value of X. (Initial read must be same).



S_1: T_1 reads A from database.
S_2: T_1 reads A from T_2.
$S_1 \neq S_2$

② → WR Sequence: the S_1 & S_2 are view equivalent if in schedule S_1, T_i read A that has been updated by T_j, also in schedule S_2.



S_1: T_3 reads value of A from T_2
S_2: T_3 reads value of A from T_1
$S_1 \neq S_2$ (write-read sequence is not same in S_1 & S_2)

③ → Final Write Operation should be same:

T_1	T_2	T_3
$W_1(A)$		
	$R_2(A)$	
		$W_3(A)$

S_1

T_1	T_2	T_3
	(W(A))	$W_3(A)$
	$R_2(A)$	
$W_1(A)$		

S_2

S_1 : A is finally Updated by T_3
S_2 : A is finally Updated by T_1
$S_1 \neq S_2$.

Ex

T_1	T_2	T_3
	$R_2(B)$	
	$R_2(A)$	
$R_1(A)$		
		$R_3(A)$
$W_1(B)$		
	$W_2(B)$	
		$W_3(B)$

⇒ $R_2(B), R_2(A), R_1(A), R_3(A), W_1(B), W_2(B), W_3(B)$.

Solution: With 3 transactions, total number of Schedule possible
$= 3! = 6$

- $\langle T_1\ T_2\ T_3 \rangle$ ✓
- $\langle T_1\ T_3\ T_2 \rangle$
- $\langle T_2\ T_3\ T_1 \rangle$
- $\langle T_2\ T_1\ T_3 \rangle$ ✓
- $\langle T_3\ T_1\ T_2 \rangle$
- $\langle T_3\ T_2\ T_1 \rangle$

Check 1: Final Update on T_3.
$\langle T_1\ T_2\ T_3 \rangle$
$\langle T_2\ T_1\ T_3 \rangle$

Check 2: initial read + Write transaction Update after read.
T_2

↳ So remaining schedule $\langle T_2\ T_1\ T_3 \rangle$

Check 3: Write-read sequence (WR)
↳ no need to check here:

⇒ Hence view equivalent Serial Schedule is $T_2 \to T_1 \to T_3$.

Cascading Schedule / Rollback :

T_1	T_2	T_3
R(X)		
R(Y)		
W(X)		
	R(X)	
	W(X)	
fail		R(X)

- if here T_1 fail then it rollback. but T_2 is depend on T_1 so it also rollback and T_3 also depend on T_2 so T_3 is also rollback. So all T_3, T_2 and T_1 rollback, So here single transaction failure leads to a series of transaction rollback is called Cascading rollback.
- it leads significant amount of work.

- it is desirable to restrict the schedules to those where cascading rollback cannot occure, such schedules are called Cascadeless Schedules.

- Formally, a Cascadeless Schedule is one where for each pair of transaction T_i and T_j such that T_j read data item, previously written by T_i the commit operation of T_i appears before the read operation of T_j.

every Cascadeless Schedule is also recoverable schedule.

T_1	T_2
r(X)	
W(X)	
	R(Y)
+ commit	
	R(X)