→ Introduction / Fundamentals of DBMS

→ E-R Modellings & Integrity Constraints

→ Query Languages
- Relational Algebra
- Relational Calculus
  - TRC
  - DRC
- SQL

→ Schema Refinement : Normalisation

→ Transaction Management & Concurrency Control

→ File Organisation & Indexing.

DATA BASE MANAGEMENT SYSTEM

DATA BASE :-

→ It is a collection of interrelated data belonging to an organization (or an enterprise.

Database + DBMS → Data Base System

DBMS :-

→ It is a software (set of programs) used to manage and access the database in an efficient manner.

DBS :-

→ It is the combinational of database and DBMS.

Comparision of File System with DBMS :-

File System	DBMS
→ It difficult to access a data in a file system as the records should be search manually.	→ Data can be easily retrived or access in DBMS using a search key.
→ There is no concept of Indexing in a file system.	→ Indexing is used for faster & easier retrival of data.

File System	DBMS
→ In a file system there is no data abstraction (hiding low-level data).	→ Data abstraction is used to hide the details from the user.
→ A file systems cannot gaurantee consistency i.e., data may be in an inconsistant state.	→ In DBMS consistency of data is preserved.
→ Integrity constraints cannot be specified in a file system.	→ In DBMS, Integrity constraints can be specified at the time of defining the schema.
→ In a filesystem data may be retendant i.e., duplication of data is possible.	→ In DBMS Normalization can be used to remove the retendency in a relation. [TABLE]
→ I/O cost is more in a file system as all the Blocks of the file should be loaded from Secondary memory to main memory.	→ In DBMS, I/O cost is less as only the required Block is loaded from Secondary memory to main memory.

→ A file system exhibits less concurrency because the kind of locking supported by OS is file locking. which means the complete file should be locked even to modify one record.

→ In a file system the data is not organised.

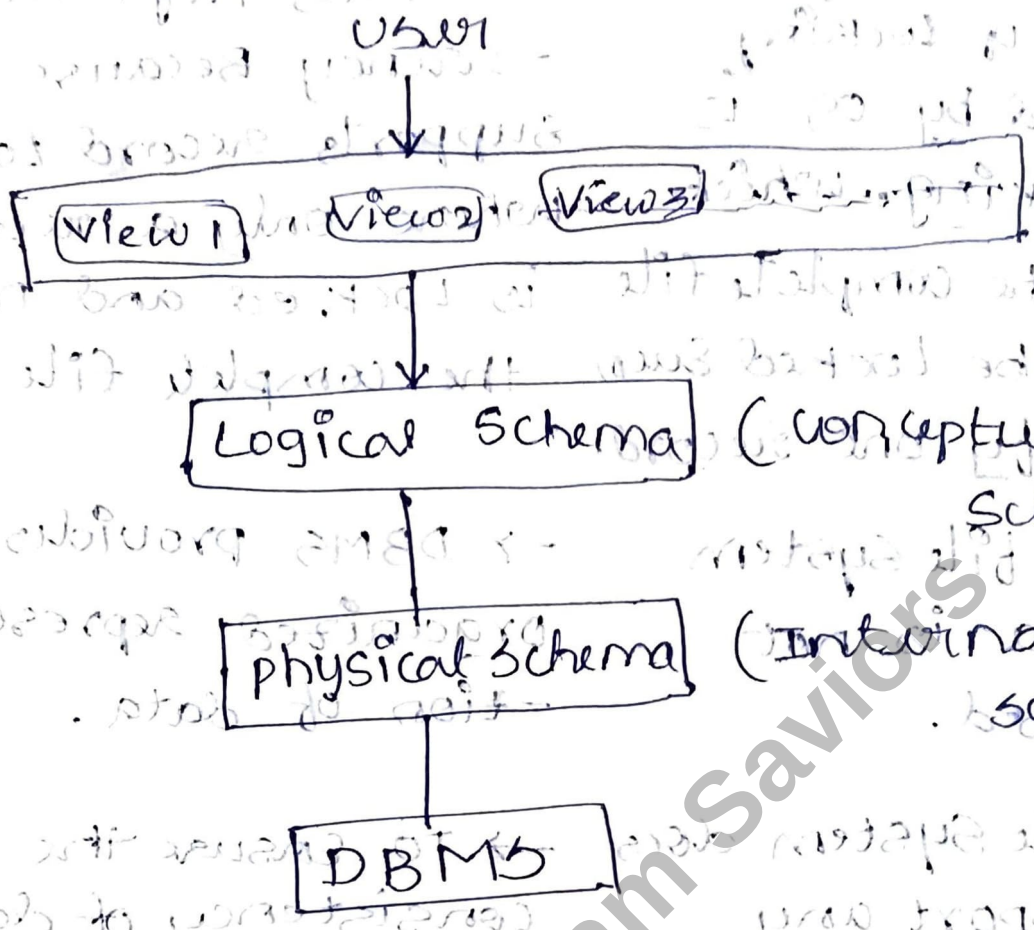
→ A file system does not support any locking leading to inconsistency of data.

→ In DBMS which exhibits High concurrency Because it supports record locking where only a record is locked and not the complete file.

→ DBMS provides an organized representation of data.

→ To ensure the consistency of data DBMS provides locking mechanisms.

LEVELS OF ABSTRACTION in DBMS :-



→ In Between user and Data there are 3 levels (3 layers of abstraction

1. physical schema (Internal)
2. Logical schema (conceptual)
3. View schema (External)

1. physical schema:-

→ It discribes the low level (or) the storage details of the database.

2. Logical schema:-

→ It provides a discribtion of the database in the form of tables.

3. View schema:-

→ It shows only a portion of the database which the user is interested and the rest of the details are abstracted or hidden from the user.

→ AS per the requirement of the user only a portion of the database is shown to the user.

DATA independence:-

→ It is the ability to modified the data at one level (or) schema without affecting the higher schemas (or) levels.

→ There are 2 types of data independence:

1. physical data independence.

2. Logical data independence:

1. physical data independence :-

→ It is the ability to modify the physical schema without affecting the Logical and view schema.

2. Logical data independence :-

→ It is the ability to modify the Logical schema without affecting the view schema.

Schema :-

→ It defines name of a Relation (Table), attributes (columns), type of attributes and constraints on the attributes.

Instance :-

→ It is the snapshot of the database i.e, it describes the data present in the database at a given moment.

Data model :-

→ A representation of data (or) database.

There are 2 types of data model :

1. ER Model (Entity Relationship)
2. Relational Model

1. ER MODEL :-

→ It is a Diagramatic Representation of the database which shows the different component that are part of the database.

→ The 3 major components of ER MODEL are :

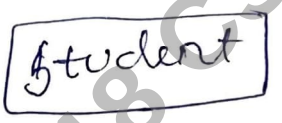
1) Entity (or) Entity set
2) Attribute
3) Relationship

1) Entity :-

→ It is a Real time object that can be distin-guish from other objects.

Ex :- Student, customer, course etc.

→ An Entity or Entity set is represented using a Rectangle.

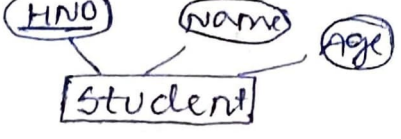
Ex :- 

→ There are 2 types of Entities :

i) Strong Entity
ii) Weak Entity

i) Strong Entity :-

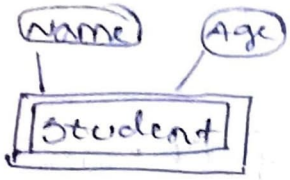
→ ANY entity that has key attribute or attributes is called Strong Entity.

Ex :- 

ii) Weak Entity:-

→ Any Entity which doesn't have any key attribute attributes is called weak Entity.

Ex:



2) Attribute:-

→ It is a characteristic (or) description of an Entity.

→ It represented using 'oval' symbol.

Ex:- (HNO)

→ The different types of attributes are:

i) Simple vs composite attribute:-

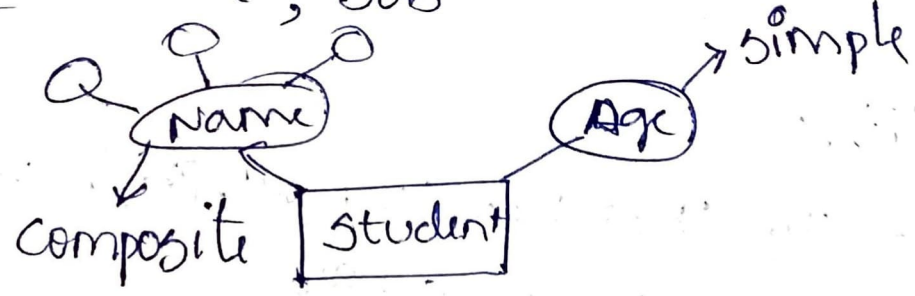
→ If an attribute cannot be divided into sum parts then it is called Simple Attribute

Ex:- Age, Gender

→ If an attribute can be divided into sum parts then it is composite attribute.

Ex:- Name, DOB

Ex:



ii) Single and multivalued attribute :-

→ If an attribute can be given only one value for a tuple or record then it is called (rows)
a single valued attribute.

Ex :- (Age), (Gender)

→ If an attribute can be given more than one value for a tuple or record then it is called Multivalued attribute;

Ex: (Name) (phNO) (Email) (Address)

iii) Stored vs Derived attribute :-

→ The value of an attribute is obtained from other attribute which is already present in the database (stored) then it is called a derived attribute.

Ex :- (DOB) → Stored (Age) → Derived

iv) Complex attribute :-

→ If an attribute is Both composite and multi-valued then it is called a complex attribute.

Ex:- (Address)

v) Key attribute (s) :-

→ If either a single attribute or a set of attributes are used for uniquely identifying an Entity belonging to an Entity set then the attribute (s) are called Key attribute (s).

Ex:- (HNO)

vi) NULL attribute :-

→ The value of the attribute is null then it is called Null attribute.

Ex:- (Landline NO)

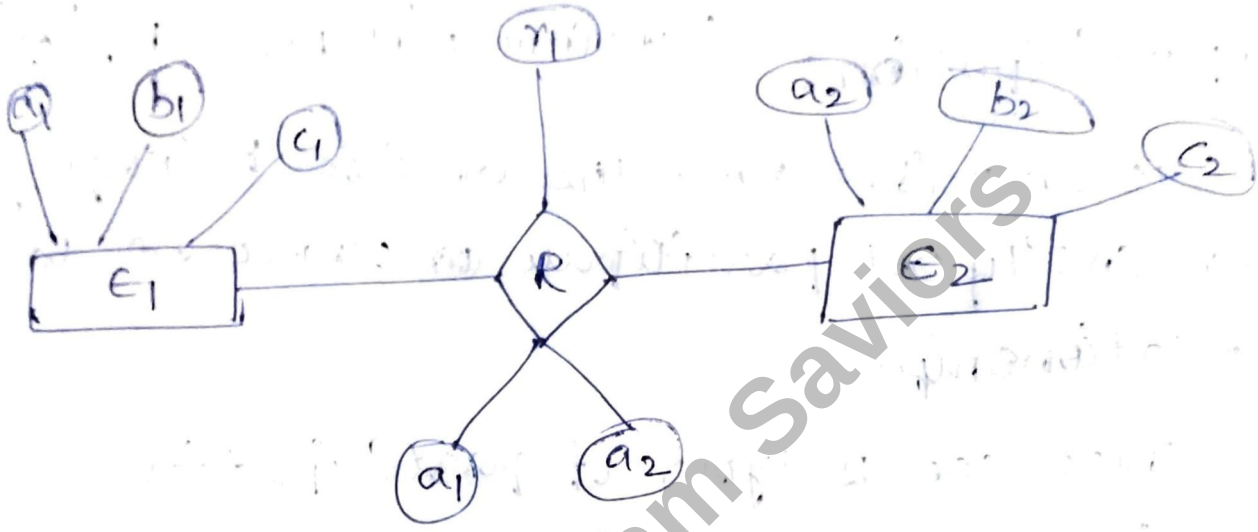
3) Relationship

→ It is used to combine (or) associate (or) relate the Entities.

→ A relationship in an E-R Model is represented using ◇ Symbol.

→ Any attribute that describe the relationship is called Descriptive attribute.

→ In addition to the descriptive attribute a relationship will have the key attributes of the connected Entities as its attributes i.e., if a relationship R connects two Entities E_1 & E_2 where a_1 and a_2 are the key attributes of E_1 & E_2 then R contains a_1 & a_2 as its attributes



★MAPPING Cardinalities :—

→ It describes how the Entities of an Entity set are mapped to the Entities of the other Entity set.

→ There are four types of mapping Cardinalities :

1) One to one
2) One to many
3) Many to one
4) Many to many

i) one-to-one relationship / cardinality :-

→ If an Entity belonging to an Entity set is mapped to Exactly one Entity of other Entity set, then it is called one-to-one relationship.

(or)

→ continue Next page

*participation :-

→ It describes how the Entities belonging to an Entity set participate. (or) connected to a relationship.

→ There are 2 types of participation

i) Total participation

ii) Partial participation

i) Total participation :-

→ If Each and Every Entity Belonging to an Entity set participate in a relationship then it is called total participation.

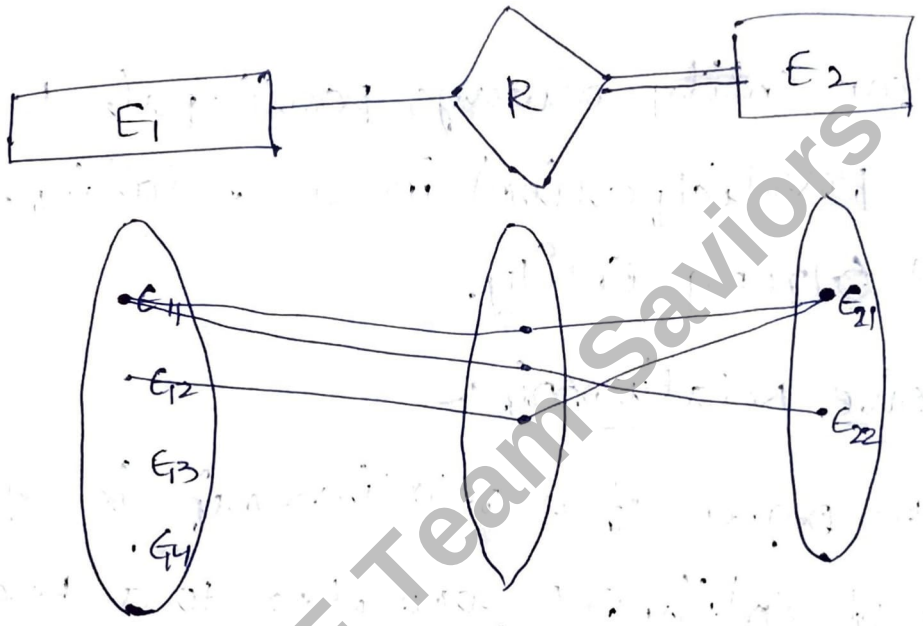
→ It represented as double line (═) in a ER diagram.

ii) partial participation:-

→ there atleast one Entity Belonging to an Entity set not connected (or) participating in a relationship then the participation of that Entity is called partial participation.

→ It is represented as single line (——).

Ex:-



E_1 — partial participation

E_2 — Total participation

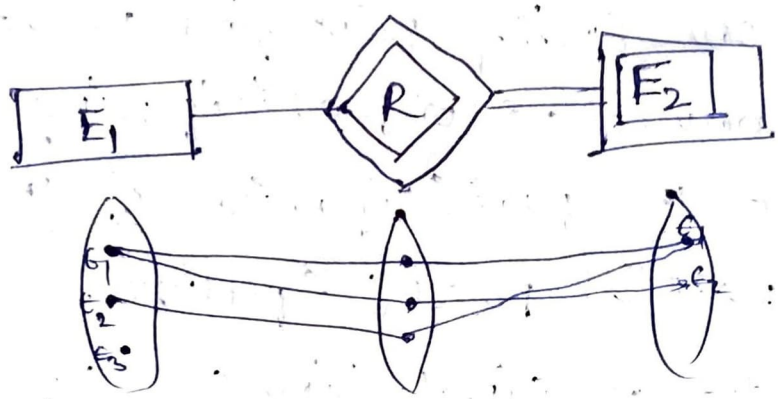
R — Relationship

*Identifying a relationship:-

→ A relationship that combines a weak Entity to a strong Entity (or) vice versa is called Identifying relationship.

→ It is represented using double rhombus.

Ex:-



NOTE:-

→ A weak Entity always participate totally (total participation) in a relationship with a strong Entity.

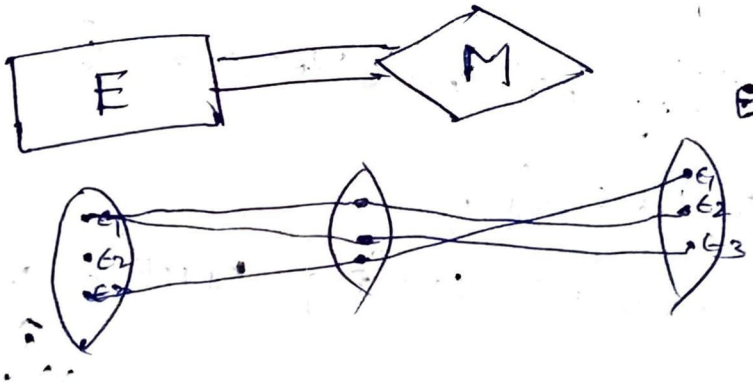
Recursive Relationship:-

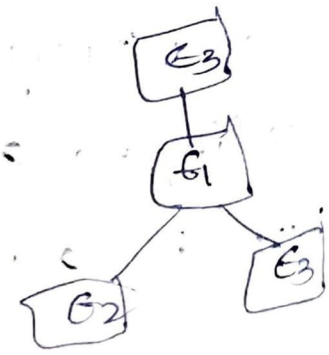
→ If the degree of a relationship is 1 i.e., the no of Entity set connected to a Relationship is 1 then it is called an Recursive relationship.

NOTE:-

→ The no of Entity sets connected to a relationship is called Degree of relationship

Ex:-



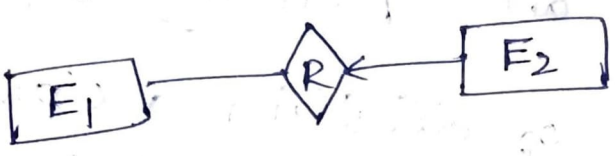


continue:-

MAPPING CARDINALITIES:-

2) one to many relationship/cardinalities:-

→ If an Entity belonging to Entity set mapped to more than 1 Entity of the other Entity set.



3) Many to one relationship/cardinalities:-

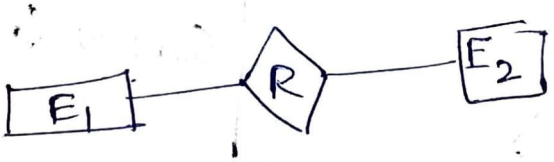
→ If multiple Entity of Entity set E_1 are mapped to single Entity of Entity set E_2 is called many to one cardinalities.

4) Many to many relationship/cardinalities:-

→ If multiple Entity of Entity set E_1 are mapped to multiple Entity of Entity set E_2 is called many to many cardinalities

Ex:-

3) 

4) 

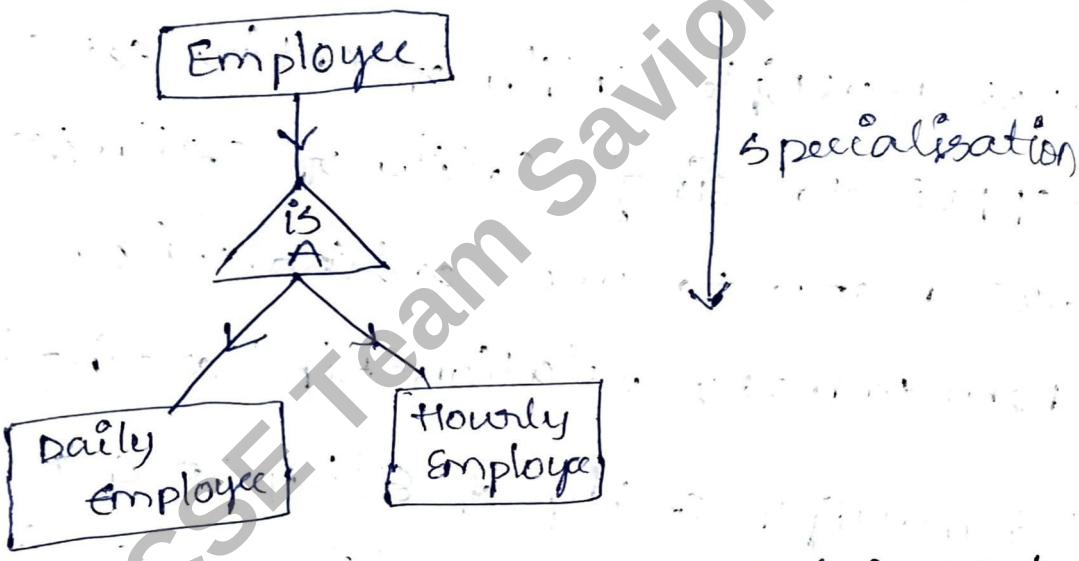
Extended features of ER Model :-

1) Subclass and Super class :-

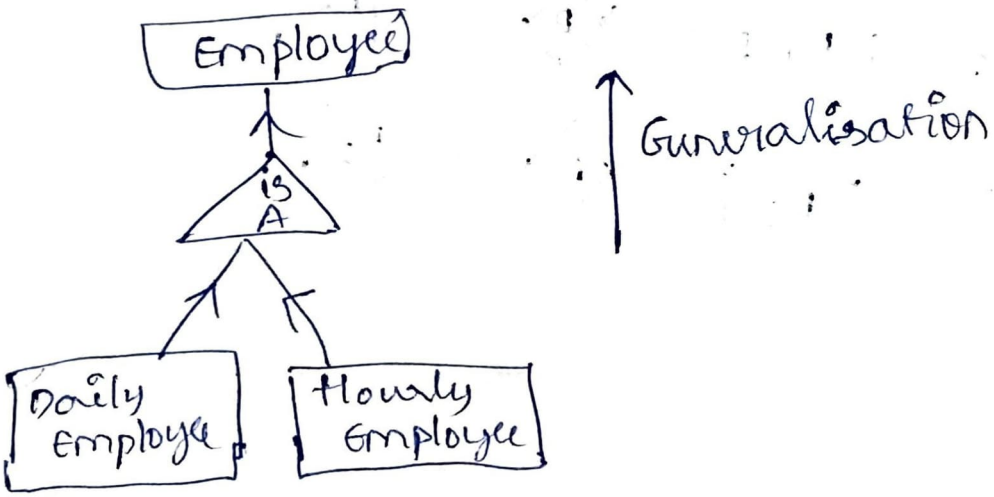
→ An Entity set is said to be a subclass of other Entity set if it shares [is-A] Relationship

2) Generalisation and Specialisation :-

→ The process of dividing a class into subclasses is called as specialisation.



→ The process of combining the subclasses to form a base class or main class is called generalisation.



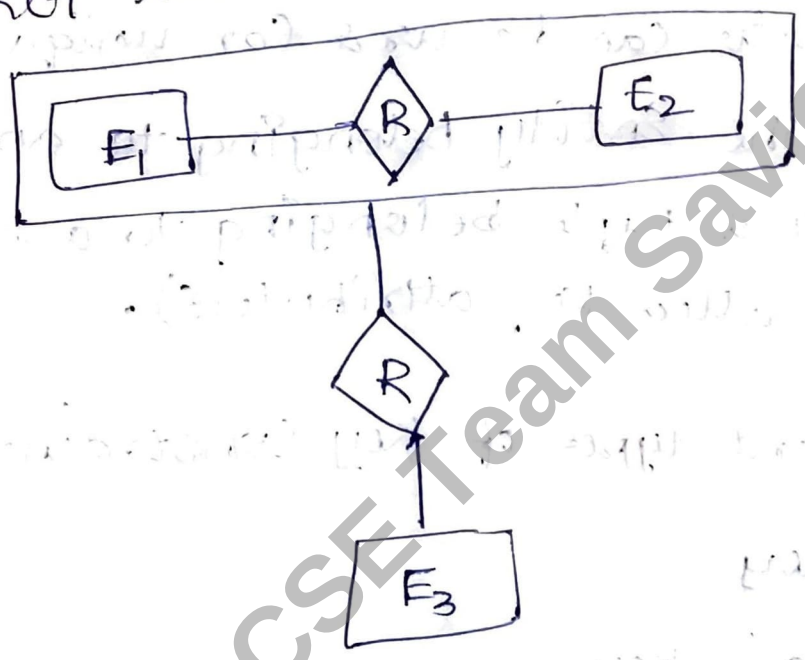
3) Aggregation :-

→ Aggregation is a process where a relation connecting to entities is treated as single entity or entityset.

(or)

→ A relationship participating in another relationship is called aggregation.

Ex :-



Integrity Constrain :-

→ A constrain is a condition. and a/dd

→ There are difference types of integrity constraints are :

1) Domain constrains
2) Key constrains
3) Semantic integrity constrains
4) Referenshal integrity constrains

1) Domain Constraints:-

→ A set (or) Range of values that can be given for an attribute.

Ex:- age, number etc.

2) Key Constraints:-

→ Either a single attribute can be use or set of attributes can be used for uniquely identifying the Entity belonging to an Entity set or a tuple belonging to a relation then it is called key attribute(s).

→ The different types of key constrains are:

1) Super key
2) Candidate key
3) primary key
4) unique key
5) foreign key

1) Super key:-

→ Either a single attribute or a set of attributes used for uniquely identifying the Entity then the attribute or attributes is

called Super key.

Ex:-

Sid	Name	Branch

{Sid}.
{Sid, name}
{Sid, Branch}
{Sid, name, Branch}

→ A relation can have any no of Super keys.

2) Candidate key:-

→ A minimal super key that can be uniquely identified the tuples of a relations is called candidate keys.

→ A relation can contains any no of candidate key.

Ex:- {Sid}, {Sid, name}, {Sid, Branch}

NOTE:-

→ Every candidate key is a Super key But Not Vicevirsa.

→ Given a relation $R(A_1, A_2, A_3, \ldots, A_n)$. find the maximum no of Super keys possibles.

Ans: $2^N - 1$

⇒ Given a relation $R(A_1, A_2, \ldots, A_n)$ with A_1 has a Candidate find the max no of super keys possible.

Ans:- 2^{N-1}

⇒ Given a relation $R(A_1, A_2, \ldots, A_n)$ with the Candidate keys has $\{A_1\}$ and $\{A_2\}$

Ans:- $2^{N-1} + 2^{N-1} - 2^{N-2}$

$2^N - 2^{N-2}$

⇒ The candidate keys are $\{A_1\}, \{A_2\}, \{A_3$

Ans:- $A \cup B \cup C = 3 \times 2^{n-1} - 3 \times 2^{n-2} + 2^{n-3}$

⇒ The candidate keys are $\{A_1, A_2\}$

Ans:- 2^{N-2}

⇒ The candidate keys are $\{A_1\}$ $\{A_2, A_3\}$

Ans:- $2^{n-1} + 2^{n-2} - 2^{n-3}$

⇒ The candidate keys are $\{A_1, A_2\}$ & $\{A_3, A$

Ans:- $2^{n-2} + 2^{N-2} - 2^{N-4}$

3) primary Key :-

→ One of the Candidate Key is used as a primary Key.

→ The primary Key is one which can be uniquely identifying the tuples of a relations.

→ A relation can contain only one primary Key.

→ A primary Keys cannot accept null values.

Not NULL constrains
↳ It can be duplicate but cannot be NULL

4) Unique Key :-

→ It is same as primary Key i.e., the values should be unique but can accept null values.

5) foreign Key :-

→ A foreign Key is the attribute of the relation that refers to the primary Key of another relation.

→ A Constraint on a foreign Key is called a referencial integrity constraint or foreign Key constraint.

* Referential Integrity (or) foreign key constraint:

→ The two constraints of foreign key are

1) The domains (type) of the foreign key should match with the domains of the primary key to which it refers to.

2) The values of the foreign key should either refers to the values of the primary key of relation to which it references or it can contains Null

UNIT-II

Relational algebra

→A set of opurators are used for Expressing Quereies in relational algebra and this opurators are Basically classified into two types:

1) Basic opurators
   - selection (σ)
   - projection (π)
   - Union ($\cup$)
   - set difference ($-$)
   - cross product ($\times$)
   - Rename (ρ)

2) Durive opurators
   - Intersection ($\cap$)
   - Division ($/$)
   - Join ($\bowtie$)

① Basic :-

Projection opurator:-

→ It is used to retreive the columns of a relations.

Ex :- $\Pi_{S\,name}$ (sailors)

Selection operator :-

→ It is used to retrieve rows (or) records (or) tuple of a relation satisfying some condition.

$$Ex :- \Pi_{sname}\left(\sigma_{rating>10}(sailor)\right)$$

union operator :-

→ It displays the distinct records of both the relations.

Ex:-

S1

SID	Sname
101	Ra
102	Je
103	Sh

∪

S2

SID	Sname
104	IC
105	UM
106	AR

S table

SID	Sname
101	Ra
102	Je
103	Sh
104	IC
105	UM
106	AR

NOTE

→ TC
oper
be
the
mus

Set

→
in Bo
the s
rela

Exr

Cross

→ If
attri
m+n

NOTE :-

→ To perform union, intersect and set difference operations the schema of 2 relations should be same. i.e., the no of attributes, name of the attribute and the domain of the attribute must be same

Set difference :-

→ It eliminates the common records present in both the relations or table and displays the rest of the records present in the first relation.

Ex

S_1

SID	Sname
101	Ram
102	Jeet

S_2

SID	Sname
101	Ravi
103	Kiran

$S_1 - S_2$ =

SID	Name
102	Jeet

Cross Product :-

→ If R and S are relations containing m, n attributes and p, q tuples, then R×S contains m+n attributes and p×q tuples

Ex

R

A	B
1	2

S

B	C
3	4

R × S

A	B	B	C
1	2	3	4

RENAME operator

→ It is used to change the name of the relation or attribute

Ex: ρ_S (sailor)

② Derived operators

→ These are obtained from the basic operators.

Intersection operator:-

→ It displays the common record present in both the relations i.e.;

$$R \cap S = R - (R - S)$$

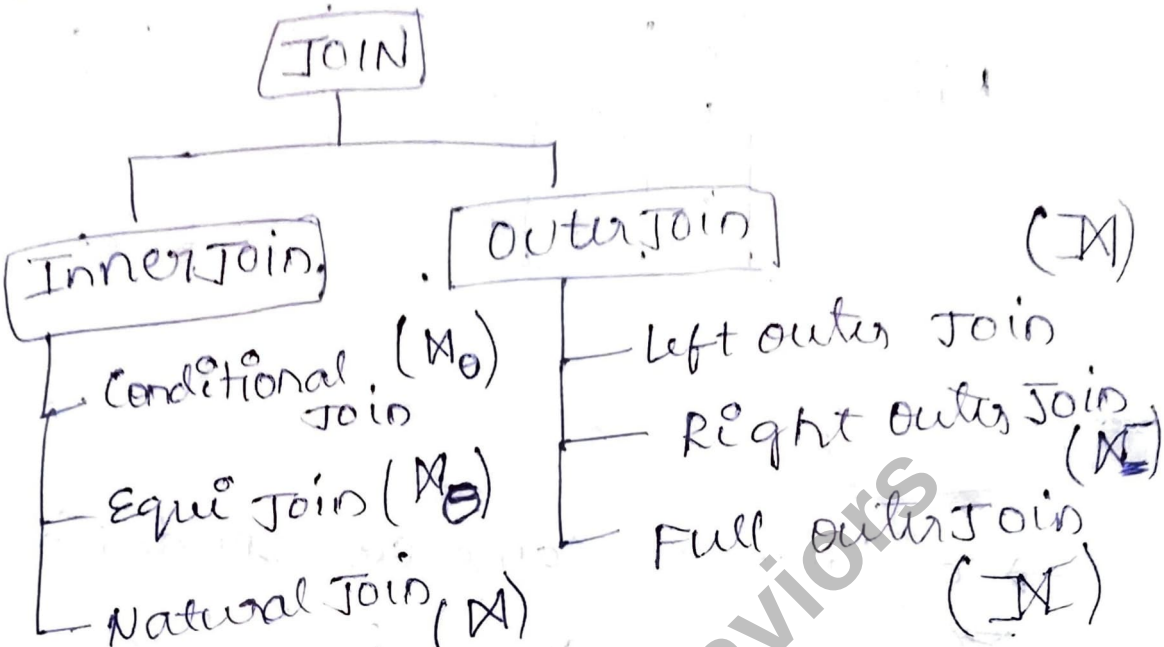
Join operator:-

$$R \bowtie S = \pi(\sigma(R \times S))$$

→ A Join operation is defined as cross product followed by selection & projection of two relations:

→ The different types of Joins are:

1) Inner Join
2) Outer Join



⇒ Conditional Join :-

→ It projects all the columns of both the relations but displays only those records that satisfy the condition, which is method.

Ex:-

R

A	B	C
1	2	3
4	5	6

S

B	C	D
2	3	4
7	8	9

R × S

A	B	C	B	C	D
1	2	3	2	3	4
1	2	3	7	8	9
4	5	6	2	3	4
				8	9

Ex: $R \bowtie_{R.B<S.B} S = \Pi_{ABCBCD}(\sigma_{R.B<S.B}(R \times S))$

$R \bowtie S =$

A	B	C	B	C	D
1	2	3	7	8	9
4	5	6	7	8	9

⇒ Equi Join :-

→ Equi Join is same as conditional Join, But the condition is '=' to.

Ex: $R \bowtie_{(R.B=S.B)} S = \Pi_{ABCBCD}(\sigma_{R.B=S.B}(R \times S))$

$R \bowtie S =$

A	B	C	B	C	D
1	2	3	2	3	4

⇒ Natural Join :-

→ It projects only the distinct columns of Both the relations and displays only those records with matching attribute values.

NOTE :- To perform a natural join atleast one column should be common in both the relations.

Ex:

$$R \bowtie S = \Pi_{ABCD}\left(\sigma_{\substack{R.B=S.B \\ R.C=S.C}}(R \times S)\right)$$

A	B	C	D
1	2	3	4

⇒ Left outer Join:-

→ It is R ⋈ S and the tuples in R that failed the Join condition.

R ⟕ S =

A	B	C	D
1	2	3	4
4	5	6	null

⇒ Right outer Join:-

→ It is R ⋈ S and the tuples in S that failed the Join condition

R ⟖ S =

A	B	C	D
1	2	3	4
null	7	8	9

⇒ Full outer JOIN :-

→ It is the union of left outer JOIN & Right outer JOIN.

$R \bowtie S =$

A	B	C	D
1	2	3	4
4	5	6	null
null	7	8	9

⇒ write a Query in Relational algebra to find the names of the sailors whose age > 30.

Ans:- $\Pi_{sname}(\sigma_{age>30}(sailor))$

⇒ find the SID's of the sailor to reserved Boat no 101.

Ans:- $\Pi_{SID}(\sigma_{BID=101}(reserves))$

⇒ find names of the sailors who reserved boat no 101.

Ans:- $\Pi_{sname}(\sigma_{BID=101}(sailor \bowtie reserves))$

$\Pi_{sname}(\sigma_{BID=101}(sailor \bowtie reserves))$

⇒ find SID of the sailors who reserved Red colored code.

Ans:- $\pi_{SID}(\sigma_{color='red'}(reserves \bowtie Boat \bowtie Sailor))$

⇒ find the names of the sailors who reserved Red coloured Boat.

Ans $\pi_{Sname}(\sigma_{color='red'}(Sailor \bowtie reserves \bowtie Boat))$

⇒ find the names of the sailors who reserved red or green coloured core.

Ans $\pi_{Sname}(\sigma_{color=red \vee color=green}(Sailor \bowtie reserves \bowtie Boat))$

(or)

$\pi_{Sname}(\sigma_{color=red}(Sailor \bowtie reserves \bowtie Boat)) \cup \pi_{Sname}(\sigma_{color=green}(Sailor \bowtie reserves \bowtie Boat))$

(or)

$\pi_{Sname}((\sigma_{color=red}^{Boat}) \cup (\sigma_{color=green}^{Boat}) \bowtie reserves \bowtie Sailor)$.

$[\because U = V]$

⇒ find the names of the sailors whose reserved red & green colored boat.

Ans:-

$$\Pi_{sname}\left(\left(\sigma_{color='red'}^{Boat}\right) \cap \left(\sigma_{color='green'}^{Boat}\right) \bowtie reserve \bowtie Sail\right.$$

(or)

$$\Pi_{sname}\left(\sigma_{color='red}(Boat \bowtie reserve \bowtie sailor)\right) \cap$$

$$\Pi_{sname}\left(\sigma_{color='green'}(Boat \bowtie reserve \bowtie sailor)\right.$$

⇒ find the names of the sailors whose age is >20 but did not reserved red coloured Boat.

Ans:-

$$\Pi_{sname}\left(\sigma_{age>20}^{sailor}\right) -$$

$$\Pi_{sname}\left(\sigma_{color='red'}(Boat \bowtie reserves \bowtie sailor)\right)$$

⇒ find the names of the sailors who reserved atleast one Boat.

Ans:- Π_{sname} (sailor ⋈ Reserves)

⇒ find the names of sailors who reserved two different Boats on the same Day.

Ans:- $\Pi_{sname}(\sigma$ (sailor ⋈ Reserves)

SQL Queries Answers for before questions

1) select sname from sailor where age > 30;

2) select sid from reserves where BID = 101;

3) select s.sname from sailor s, reserves r
where s.sid = R.sid and
R.BID = 101;

4) select R.sid from reserves R, Boat B
where R.bid = B.bid and B.color = 'red';

5) select s.sname from sailor s, reserves R,
Boat B where
R.sid = B.bid and s.sid = R.sid and
B.color = 'red';

6) select s.name from sailor s, reserve R,
Boat B where
R.bid = B.bid and s.sid = R.sid and
(B.color = 'red' or B.color = 'green');

(x)
7) select s.name from sailors, reserve R,
Boat B where
R.bid = B.bid and S.sid = R.sid and
(B.bcolor = 'red') intersect B.bcolor = 'green');

8) Select S.sname

7) SELECT S.sname from sailor s, Reserves r, Boat b where s.sid = r.sid and r.bid = b.bid and b.colour = 'red'

INTERSECT

SELECT S.sname from sailor S, Reserves r, Boat b where S.sid = r.sid and r.bid = b.bid and b.colour = 'green';

8) select sname from sailor where age > 20 MINUS select S.sname from sailor S, Reserves r, boat b where S.sid = r.sid and r.bid = b.bid and b.colour = 'red';

9) Select S.sname from sailor S, Reserves R where S.sid = R.sid;

10) SELECT S.sname from sailor S, Reserves r_1, Reserves r_2, WHERE S.sid = r_1.sid and S.sid = r_2.sid and r_1.day = r_2.day and r_1.bid <> r_2.bid;

Tuple Relational Calculus:

→ It is expressed (or) defined as $\{T / P(T)\}$ where T is a output variable and P(T) is a condition.

⇒ Express the queries in T.R.C.

Q) Find names of all sailors:

Ans: $\{T / \exists S \in Sailor (T[Sname] = S[Sname])\}$

1) $\{T / \exists S \in Sailor (T[Sname] = S[Sname] \wedge S[age] > 30)\}$

2) $\{T / \exists R \in Reserves [T[sid] = R[sid] \wedge R[Bid] = 101)\}$

3) $\{T / \exists S \in Sailor \exists R \in Reserves (T[Sname] = S[Sname] \wedge S[Sid] = R[Sid] \wedge R[Bid] = 101)\}$

4) $\{T / \exists R \in Reserves \exists B \in Boat (T[Sid] = R[Sid] \wedge R[Bid] = B[Bid] \wedge B[colour] = 'red')\}$

5) $\{T / \exists S \in Sailor \exists R \in Reserves \exists B \in Boat (T[Sname] = S[Sname] \wedge S[Sid] = R[Sid] \wedge B[Bid] = R[Bid] \wedge B[colours] = 'red')\}$

6) DRC (Domain Relational characteristics)

→ It is expressed as

$$\{\langle x_1, x_2, \ldots, x_n\rangle / p\langle x_1, x_2, \ldots, x_n\rangle\}$$

$\langle x_1, x_2, \ldots, x_n\rangle$ → Domain/output variables

$p\langle x_1, x_2, \ldots, x_n\rangle$ → Domain function

⇒ find names of sailors whose age > 30.

Ans:

$$\{\langle b\rangle / \exists\langle a, c, d\rangle (\langle a, b, c, d\rangle \in Sailor \wedge c > 30)\}$$

⇒ find Sid's of sailors whose BID = 101

Ans:

$$\{\langle a\rangle / \exists\langle b, c, d\rangle (\langle e, f, g\rangle \in reserves \wedge g = 101)\}$$

Sailor

Sid	Name	Age	R
a	b	c	d

Reserves

Sid	Day	Bid
e	f	g

Boat

Bid	bname	Bcolor
h	i	j

⇒ find names of sailors who reserve Bid=101

Ans:-

$$\{\langle b\rangle / \exists\langle a,c,d\rangle(\langle a,b,c,d\rangle \in \text{sailors}) \wedge$$

$$\exists\langle e,f,g\rangle(\langle e,f,g\rangle \in \text{Reserves} \wedge a=e \wedge g=101)\}$$

⇒ find bid of the sailors who reserved Red color Boat

Ans:-

$$\{\langle e\rangle / \exists\langle e,f,g\rangle(\langle e,f,g\rangle \in \text{Reserves} \wedge \exists\langle h,i,j\rangle(\langle h,i,j\rangle \in \text{Boat} \wedge g=h \wedge j='\text{red}'))\}$$

⇒ find the names of the sailor who reserved red coloured Boat.

Ans:-

$$\{\langle a\rangle / \exists\langle a,c,d\rangle(\langle a,b,c,d\rangle \in \text{sailor} \wedge \exists\langle h,i,j\rangle(\langle h,i,j\rangle \in \text{Boat} \wedge \exists\langle e,f,g\rangle(\langle e,b,g\rangle \in \text{Reserves} \wedge a=e \wedge g=h \wedge j='\text{red}')))\}$$

⇒ find the colors of the Boat reserved by Dustin.

Ans:- DRC:-

$$\{\langle j\rangle / \exists \langle h,i,j\rangle (\langle h,i,j\rangle \in Boat \wedge$$
$$\exists \langle a,b,c,d\rangle (\langle a,b,c,d\rangle \in Sailor \wedge$$
$$\exists \langle e,f,g\rangle (\langle e,f,g\rangle \in Reserves \wedge$$
$$a=e \wedge g=h \wedge b='Dustin'))\}$$

TRC:-

$$\{T / \exists \Phi \in Sailor . \exists R \in Reserves \exists B \in Boat$$
$$(T[bcolor] = B[bcolor] \wedge S[sid] = R[sid]$$
$$\wedge R[Bid] = B[Bid] \wedge S[name]$$
$$= 'Dustin')\}$$

SQL:-

```
SELECT b.bcolor from Sailor S, Reserves r,
       Boat b where S.sid = r.sid
       and r.bid = b.bid and
           S.Sname = 'Dustin'
```

Relational algebra

$\Pi_{sname}(\sigma_{bcolor=}$

$\Pi_{bcolor}(\sigma_{sname='Dustin'}$ (Sailor ⋈ Reserves ⋈ Boat))

Nested SQL Query :-

→ A Query in another Query where the result of the Inner Query is used by outer Query is called a nested Query.

→ The most widely used operation in nested Query is IN.

⇒ find the names of sailor whose age is greater the Average age of sailors.

Ans:- SELECT Sname from Sailor where age > (SELECT AVG(age) from Sailor)

⇒ find the names of the sailor whose reserved boat no = 101

Ans:- SELECT S.Sname from Sailor S, where Reserves R

S.Sid IN (SELECT R.Sid from Reserves r where r.Bid = 101)

⇒ find SID's of the sailors who reserved red coloured boat.

Ans:- SELECT S.SID from Sailors S where S.SID IN (SELECT R.SID from reserves R where R.SID IN (SELECT B.BID from Boats B where B.bcolor = 'red'));

⇒ find the names of sailors who reserved red coloured Boat.

Ans:- SELECT S.Sname from Sailors S where S.SID IN (SELECT R.SID from Reserves R where R.BID IN (SELECT B.BID from Boats B where B.Bcolor = 'red'));

Corelated Queries:-

→ It is also a form of nested Query But here Inner Query is dependent on the outer Query.

→ The most widely used operator for writing a corelated Query is 'EXISTS'

⇒ find the names of the sailors who reserved Boat no = 101.

Ans:-

```
SELECT S.Sname from sailors where
Exists (SELECT * from Reserves r where
        S.SID = r.SID and r.bid = 101);
```

⇒ find the ~~names~~ SID's of sailors who reserved red coloured Boat.

Ans:-

```
SELECT R.SID from Reserves R where
exists (SELECT * from Boats b where
        R.bid = b.bid and b.bcolor = 'red');
```

⇒ find the names of the sailors who reserved red color boat.

Ans:-

```
SELECT S.Sname from sailors where
Exists (SELECT * from Reserves r where
        S.Sid = r.sid
exists (SELECT * from boats b where
        S.Sid = r.sid and r.bid = b.bid
        and b.bcolor = 'red'));
```

Set comparison operators (Any, all) :-

⇒ find the name of the sailor whose rating is greater than some sailor called Horatio?

Ans:-

```
SELECT S.Sname from sailors S where
S.rating > Any (SELECT S1.rating from
                sailor S1 where
                S1.Sname = Horatio
```

⇒ find the names of the sailors whose rating is BETTER than Every sailor called Horatio.

Ans:-

```
SELECT S.Sname from sailors S where
S.rating > all (SELECT S1.rating from
                sailor S1 where
                S1.Sname = Horatio);
```

Aggregate operators and function :-

→ There are 5 aggregate operators used in SQL

1. SUM
2. COUNT
3. AVG
4. MIN
5. MAX

⇒ find the name & age of the youngest sailor.

Ans :-
```
SELECT name, min(age) from sailor;
```

⇒ find the name & age of oldest sailor.

Ans :-
```
SELECT name, max(age) from sailor;
```

⇒ find the average age of the sailor.

Ans :-
```
SELECT Avg(age) from sailor;
```

⇒ find the no of distinct sailors?

Ans :-
```
SELECT count(distinct(sname)) from
```

⇒ Group by and having clause :-

⇒ find the age of youngest sailors for each rating.

Ans:

```
SELECT min(age) from sailor Group by
rating;
```

⇒ find the age of the youngest sailor who is eligible to over (age > 18) for each rating with atleast 2 such sailor?

Ans:

```
SELECT min(age) from sailor where
age > 18 Group by rating having count(*) > 1;
```

UNIT – III

NORMALIZATION :-

→ It is a step by step process of removing redundancy in relations.

→ It is done in phases called normal forms.

→ Redundancy is due to functional dependencies

→ Normalisation is achieved through decomposition, and it has to satisfy 2 properties.

1) Lossless decomposition
2) Dependency preserving decomposition

1) Lossless decomposition :-

→ If a relation R is decomposed into smaller relations say $R_1, R_2, \ldots, R_n$ then by combining all the smaller relations $R_1, R_2, \ldots, R_n$.

→ If the original relation R is obtained without any loss of data then the decomposition is said to be loss-less decomposition.

2) Dependence preserving decomposition

→ Even after decomposing original relation R into smaller relations R1, R2, ... Rn. if the original set of dependencies are preserved then it is said to be a dependency preserving decomposition

Problems with redundancy

→ The problems with redundancy of data are:

1) Insertion anomalys
2) updation anomalys
3) deletion anomalys

Functional dependency:-

→ The value of one attribute determines the value of another attribute then it is called Functional dependency

→ It is basically represented as

$X \rightarrow Y$ where

X is determinant

Y is dependent.

→ It can be understood as X determine Y (or) Y is dependent on X.

properties of Functional Dependency:-

1) Reflexive:-

→ For a Functional dependency $X \to Y$ where $Y \subseteq X$ then $X \to Y$ is a valid functional dependency.

Eg:- $AB \to A$

$AB \to B$

2) Transitive:-

→ If $X \to Y$ is valid and $Y \to Z$ is valid then $X \to Z$ is a valid FD.

3) Augmentation:-

→ If $X \to Y$ is valid FD then $XZ \to YZ$ is also a valid FD.

4) Union :-

→ If $X \to Y$ is valid FD and $X \to Z$ valid FD then $X \to YZ$ is also a valid FD.

5) Decomposition :-

→ If $X \to YZ$ is valid FD then $X \to Y$ is valid and $X \to Z$ is also valid.

Attribute closure :- (A^+)

→ The closure of an attribute is the set of the attributes that can be obtained from an attribute.

→ If the closure of an attribute determines all the other attributes of the relation then the attribute(s) is called a Candidate Key (or) Super key.

⇒ Given: a relation R(A, B, C) where functional dependency {A → B and A → C}.
Find A^+, B^+, C^+.

Ans:-

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

$B^+ = \{B\}$

$C^+ = \{C\}$

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

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

⇒ Given R(P, Q, R, S, T)

{

P → Q

Q → R

R → S

S → T

T → P

}

find the candidate key. ⇒ P, Q, R, S, T

Ans:- $P^+ = \{P, Q, R, S, T\}$

$Q^+ = \{Q, R, S, T, P\}$

$R^+ = \{R, S, T, P, Q\}$

$S^+ = \{S, T, P, Q, R\}$

$T^+ = \{T, P, Q, R, S\}$

⇒ Given R(A, B, C, D, E) the Dependency are {AB → C, BC → AD, D → E, CE → B}. find AB^+, BC^+, D^+, CE^+.

BC → A BC → D

Ans:- $AB^+ = \{A, B, C, E, D\}$

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

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

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

→ AB, BC, CE are Candidate key

⇒ Given R(A, B, C, D, E, F, G) the dependency are {AB → CD, AF → D, DE → F, C → G, F → E, G → A}. find CF^+, BG^+, AF^+, AB^+ as well as candidate key.

Ans:- $CF^+ = \{C, F, G, E, A, D\}$

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

$AF^+ = \{A, F, D, E\}$

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

The candidate keys are BCF

⇒ Given $R(A, B, C, D, E, F, G, H)$ f.D are $\{AB \rightarrow CD, D \rightarrow E, E \rightarrow F, B \rightarrow A, F \rightarrow G, G \rightarrow H\}$ find candidate key.

Ans:-

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

$D^+ = \{E, F, G, H, D\}$

$E^+ = \{E, F, G, H\}$

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

$F^+ = \{F, G, H\}$

$G^+ = \{G, H\}$

$H^+ = \{H\}$

∴ B is a Candidate Key

→ combinations of B is a Super key

Application of Attribute closure :-

→ To determine the candidate key as well as to list prime and non prime attributes.

→ To find additional functional dependencies

→ To find the equivalence of functional dependencies.

→ To find minimal cover and Canonical cover

→ Find candidate keys and super keys.

→ the closure of an attribute gives all the attribute of a relation then the attribute is called a candidate key or a super key.

→ If it is a minimal set it is called candidate key else called super key.

Eg:- Given $R(A, B, C)\{$

$$\begin{cases} A \rightarrow B \\ B \rightarrow C \end{cases}$$

$\}$

find candidate keys & super keys as well as count.

Ansr

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

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

$C^+ = \{C\}$

A is a candidate key

Super keys = A, AB, AC, ABC

Eg:- Given R(A, B, C, D) {

$AB \rightarrow C$

$B \rightarrow D$

}

Ansr

$A^+ = \{A\}$

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

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

$C^+ = \{C\}$

$D^+ = \{D\}$

∴ AB is a candidate key

Super key = AB, ABC, ABD, ABCD

Eg: Given $R(A, B, C, D)$ {

$AB \rightarrow CD$

$CD \rightarrow AB$

}

Ans:

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

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

∴ AB, CD are candidate keys

Super keys = ABCD, ABC, ABD, ACD, BCD, AB, CD

Eg: Given $R(A, B, C)$ {

$A \rightarrow B$

$B \rightarrow C$

$C \rightarrow A$

}

Ans:

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

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

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

A, B, C → candidate keys

Super keys = A, B, C, AC, AB, BC, ABC

Eg:- Given $R(A,B,C,D)\{$

$AB \rightarrow CD$

$A \rightarrow B$

$\}$

Ans find whether AB is a candidate key or super key.

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

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

Eg:- Given $R(A,B,C,D)\{$

$A \rightarrow B$

$B \rightarrow C$

$\}$

Ans

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

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

AB is a candidate key

⇒ Given $R(A,B,C)\{AB \to C, C \to A\}$

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

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

candidate keys = $\{AB, BC\}$

⇒ Given $R(A,B,C,D)\{AB \to C, B \to D, C \to B, D \to B\}$

Ans:

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

$A^+ = \{A\}$

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

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

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

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

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

candidate keys = $\{AB, AC, AD\}$

⇒ Given $R(A,B,C,D,E)\{A \to B, BC \to D, D \to AE\}$

Ans

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

$B^+ = \{B\}$

$C^+ = \{C\}$

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

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

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

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

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

Candidate keys = $\{BC, AC, CD\}$

⇒ $R(A,B,C,D)\{AB \to C, C \to A, B \to D, D \to B\}$

Ans

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

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

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

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

$A^+ = \{A\}$ $= \{AB, BC, CD, AD\}$

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

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

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

To find additional Functional Dependencies

⇒ Given R(A, B, C, D) F.D's are {A → BC, B → CD, D → AB}. Find whether AD → C is possible.

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

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

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

AD → C is possible

⇒ Given R(A, B, C, D, E) with FD's {A → B, A → C, CD → E, B → D, E → A} which of the following FD's is not implied by the above set

a) CD → AC b) BD → CD

c) BC → CD d) AC → BC

Ans: $CD^+ = \{C, D, E, A, B\}$

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

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

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

CD → AC, BC → CD, AC → BC } ✓ BD → CD ✗

To find equivalence of FD's :-

→ Two sets of FD's, F and G are said to be equivalent if it satisfies 2 conditions

1) F covers G

2) G covers F

→ we say F covers G when the FD's of G can be obtained using the FD's of F (by for G covers F.

⇒ Consider the following 2 sets of FD's and check whether they are equivalent or not.

F : { A → C
AC → D
E → AD
E → H }

G : { A → CD
E → AH }

Ans:-

F: $A^+ = \{A, C, D\}$ ⇒ A → CD

$E^+ = \{A, D, E, H, C\}$ ⇒ E → AH

F cover G

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

$A^+ = \{A, C, D\} \Rightarrow AC \rightarrow D$

$E^+ = \{E, A, H, C, D\} \Rightarrow E \rightarrow AD$

$E \rightarrow H$

G covers F

∴ F and G are Equivalent.

$\Rightarrow F = \{A \rightarrow B$

$B \rightarrow C$

$C \rightarrow D\}$

$G = \{A \rightarrow BC$

$C \rightarrow D\}$

Ans: $F: A^+ = \{A, B, C, D\} \Rightarrow A \rightarrow BC$

$C^+ = \{C, D\} \Rightarrow C \rightarrow D$

F covers G

$G: A^+ = \{A, B, C\} \Rightarrow A \rightarrow B$

$B^+ = \{B\}$

$C^+ = \{C, D\} \Rightarrow C \rightarrow D$

G does not covers F

F & G are not Equivalent.

⇒ F: {A → B
AB → C
D → AC
D → E}

G: {A → BC
D → AB}

Ans: F: $A^+ = \{A, B, C\} \Rightarrow A \rightarrow BC$

$D^+ = \{D, A, C, E\} \Rightarrow D \rightarrow AB$

F covers G

G: $A^+ = \{A, B, C\} \Rightarrow A \rightarrow B$

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

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

G does not cover F

F, G are not equivalent.

→ A minimal cover or set of FD's is a minimal set of FD's which is equivalent to the original set of FD's. The objective of finding the minimal cover is to eliminate the redundant

⇒ find the minimal cover and canonical cover for the following set of FDs

$\{ A \to B$
$C \to B$
$D \to ABC$
$AC \to D \}$

Ans:

$\{A \to B, C \to B, D \to A, D \to B, D \to C, AC \to D\}$
1 2 3 4 5 6

$A^+ = \{A\}$ ✓, Retain using 2,3,4,5,6

$C^+ = \{C\}$ ✓, Retain using 1,3,4,5,6

$D^+ = \{D, B, C\} \Rightarrow D \to ABC$, So eliminate using 1,2,3,5,6

$D^+ = \{D, A, B\}$ ✓, Retain using 1,2,3,4

$AC^+ = \{A, C, B\}$ ✓, Retain using 1,2,3,5

Minimal cover
$= \{A \to B,$
$C \to B$
$D \to A$
$D \to C$
$AC \to D\}$

Canonical cover
$= \{A \to B$
$C \to B$
$D \to AC$
$AC \to D\}$

$\Rightarrow R(A, C, D, E, H)$

$\{A \to C,\ AC \to D,\ E \to AD,\ E \to H\}$

Find Minimal cover and canonical cover

Ans:

$\{A \to C,\ AC \to D,\ E \to A,\ E \to D,\ E \to H\}$

$A^+ = \{A\}$ ✓ using 2, 3, 4, 5

$AC^+ = \{A, C\}$ ✓ using 1, 3, 4, 5

$E^+ = \{E, H, D\}$ ✓ using 1, 2, 4, 5

$E^+ = \{E, H, A, C, D\}$ Eliminate using 1, 2, 3, 5

$E^+ = \{E, A, C, D\}$ ✓ using 1, 2, 3

Minimal cover:

$A \to C$

$AC \to D$

$E \to A$

$E \to H$

Canonical cover:

$A \to C$

$AC \to D$

$E \to AH$

Further

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

$C^+ = \{C\}$

Minimal cover:

$A \to C$

$A \to D$

$E \to A$

$E \to H$

Canonical cover:

$A \to CD$

$E \to AH$

$\Rightarrow R(A,B,C)\{AB \to C, C \to B, A \to B\}$

Ans:

$\{AB \to C, \quad C \to B, \quad A \to B\}$

1 2 3

$AB^+ = \{A, B\}$ ✓ using 2, 3

$C^+ = \{C\}$ ✓ using 1, 3

$A^+ = \{A\}$ ✓ using 1, 2

Now, using 2, 3

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

$B^+ = \{B\}$

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

Minimal cover:

$A \to C$

$C \to B$

~~$A \to B$~~

Canonical cover:

$A \to B$

⇒ R(A, B, C, D, E, F)

{ABD → AC, C → BE, AD → BF, B → C}

Ans:- {ABD → A 1

ABD → C 2

C → B 3

C → E 4

AD → B 5

AD → F 6

B → C 7 }

ABD^+ = {A, B, D, C, B, F, E} × eliminate

ABD^+ = {A, B, D, B, F, C, E} × eliminate

C^+ = {C, E} ✓

C^+ = {C, B} ✓

AD^+ = {A, D, F} ✓

AD^+ = {A, D, B, C, E} ✓.

B^+ = {B} ✓

Minimal cover:

C → B

C → E

AD → B

AD → F

B → C

NOW, A^+ = {A}

D^+ = {D}

cannot reduced, the redundancy in left side i.e., AD

Canonical cover:

$C \rightarrow BE$

$AD \rightarrow BF$

$B \rightarrow C$

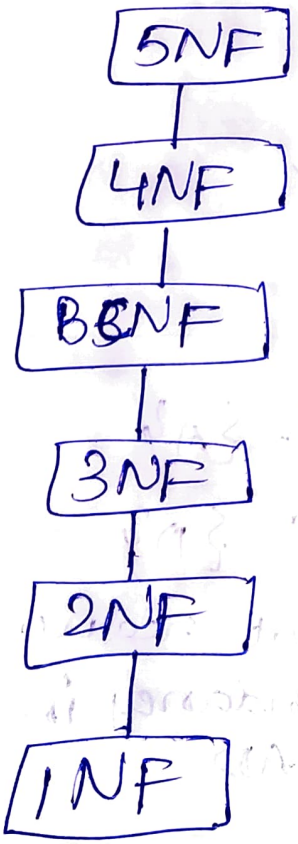
Normalization:-

→ A step by step process of removing the redundancy in a relation which is done in phases called normal forms.

→ It is achieved through decomposition. and it has to guarantee 2 properties.

i) Lossless decomposition

ii) Dependency preserving decomposition



* 1st Normal Form: (1NF)

→ A relation is said to 1st normal form if the values of the attribute are atomic (singular).

→ By default every relation is said to be in 1NF.

Second Normal form: (2NF)

→ A relation is said to be 2NF if it satisfies 2 conditions

i) It should be 1NF

ii) No partial dependencies

Partial dependencies:-

→ If the determinant of a functional dependency is part of a candidate key or a prime attribute and the dependant is a non-prime attribute. Then such a functional dependency is called a partial functional dependency.

Third Normal Form:-

→ A relation is said to 3NF if 2 conditions are satisfied.

i) It should be in 2NF.

ii) No transitive dependencies.

Transitive dependency:-

→ If the determinant of a FD is a non-prime attribute or combination of part of a candidate key with non-prime attribute and the dependent is a non-prime attribute then such a FD is called a Transitive dependency.

Non prime → Non prime

Prime + Non-prime → Non-prime

Boyce codd normal form:-

→ A relation is said to be BCNF if 2 conditions are satisfied.

i) It should be in 3NF.

ii) No Overlapping candidate key dependencies (OCD).

(OR)

ii) Each and every dependency in a relation is a full dependency.

OCD:-

→ If the determinant of a FD is not a candidate key and the dependent is a prime attribute then such a FD is called OCD.

Full FD:-

→ If the determinant of a FD is a candidate key or super key then irrespective of a dependent such a FD is called Full FD.

⇒ Identify the different types of dependencies and also the highest NF of the given relations.

→ Given $R(P, Q, R, S, T)$ {

$P \to Q$ — PD

$Q \to R$

$R \to S$

$S \to T$ }

Ans: $P^+ = \{P, Q, R, S, T\}$

$Q^+ = \{Q, R, S, T\}$

$R^+ = \{R, S, T\}$

$S^+ = \{S, T\}$

$T^+ = \{T\}$

P is a candidate key Q, R, S, T are Non-prim

$P \to Q$ {Full Dependency}

$Q \to R$ {TD}

$R \to S$ {TD} Highest NF = 2

$S \to T$ {TD}

→ Given $R(A, B, C, D, E)\{$

$AB \rightarrow CD$

$D \rightarrow \epsilon$

$A \rightarrow C$

$B \rightarrow D$ $\}$

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

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

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

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

AB is a candidate key D, D, ϵ are Non-prime

A — prime B — prime

$AB \rightarrow C$ {FD}

$AB \rightarrow D$ {FD}

$D \rightarrow \epsilon$ {TD} Highest NF = INF

$A \rightarrow C$ {PD}

$B \rightarrow D$ {PD}

⇒ Given $R(A, B, C, D)$ {

$A \rightarrow B$

$BA \rightarrow CD$

}

Ans :-

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

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

A is a candidate key and BA is a super key.

B, C, D — Non prime.

$A \rightarrow B$ {FD}

$BA \rightarrow C$ {FD}

$BA \rightarrow D$ {FD}

∴ Highest NF = BCNF

⇒ Given $R(A,B,C,D)$ {

$AB \rightarrow C$

$C \rightarrow AD$

}

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

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

AB, CB is a candidate key

$AB \rightarrow C$ {FD}

$C \rightarrow A$ {PD}

$C \rightarrow D$ {PD}

Highest NF = 2NF

⇒ Given $R(A,B,C,D)$ {

$A \rightarrow C$

$B \rightarrow D$ }

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

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

AB is a candidate key

$A \rightarrow C$ {PD} Highest NF = 1NF

$B \rightarrow D$ {PD}

⇒ Given R(S, T, U, V){

S → T

T → U

U → V

V → S }

Ansf All are candidate keys

Every one is F D

Highest NF = BCNF

⇒ Given R(A, B, C, D, E, F){

A → B

B → C

C → D

D → E

F → E }

Ansf AF is candidate key

A, F — prime attribute

Highest NF = 1NF

⇒ Given R(rollno, name, coursename, grade)

{

name → rollno

rollno → name

name coursename → grade

rollno coursename → grade

}

Ans)

CK − { name coursename$^+$ = {rollno, name, grade, courseName}

Rollno course name$^+$ = {rollno, name, grade, courseName}

}

{

Name → Rollno OCD

Rollno → Name OCD

name coursename → grade FD

roll course name → grade FD

}

Highest NF = 3NF

⇒ Decomposition to higher Normal Form:

Q) Given $R(A, B, C, D)\{$

$AB \rightarrow C$

$B \rightarrow D$ $\}$

Identify the highest Normal form of the given relation and decompose it to next highest normal form.

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

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

AB → candidate key

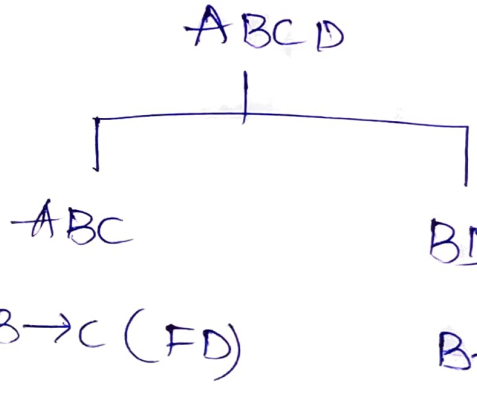
{

$AB \rightarrow C$ FD

$B \rightarrow D$ PD

}

Highest NF = 1NF



→ It

→ It is 2NF, 3NF, BCNF

⇒ Given $R(A, B, C, D, E)$ {

$A \rightarrow B$

$B \rightarrow E$

$C \rightarrow D$ }

Ans: $A^+ = \{A, B, E\}$

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

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

AC → candidate key

{

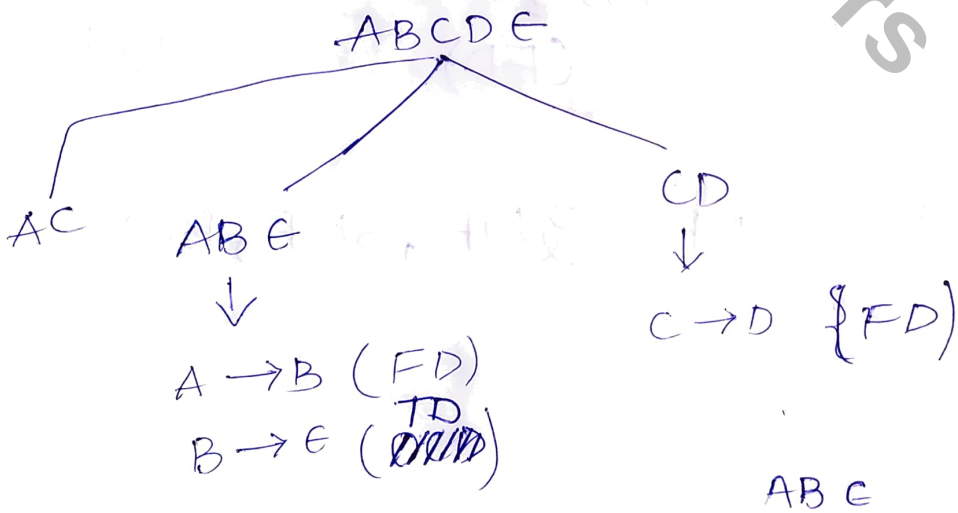
$A \rightarrow B$ PD

$B \rightarrow E$ TD

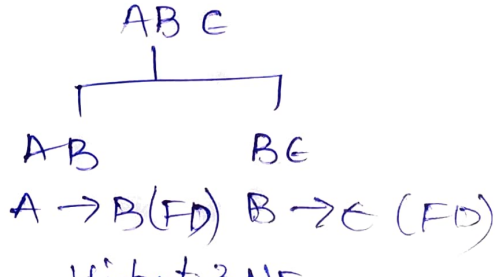
$C \rightarrow D$ PD

}

Highest NF = 1NF



Highest NF = 2NF



⇒ Given $R(A, B, C, D, E)$ {

$AB \rightarrow C$

$D \rightarrow E$ }

Ans:

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

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

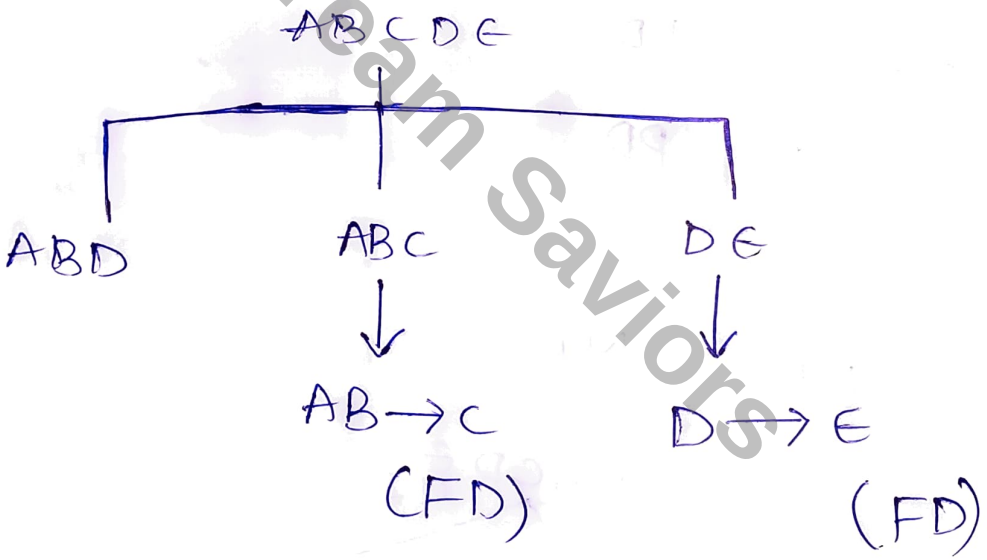
ABD^* is a candidate key

{ $AB \rightarrow C$ (PD)

$D \rightarrow E$ (PD)

}

Highest NF = 1NF



Highest NF = 2NF, 3NF, BCNF

Ex: Relation R(a, b, c, d, e, f, g, h, i, j)

$Ab \rightarrow c$

$A.f \rightarrow gh$

$bd \rightarrow ef$

$a \rightarrow i$

$h \rightarrow j$ }

Ans:

$Ab^+ = \{A, b, c, j\}$

$Ad^+ = \{A, d, g, h, j, i\}$

$bd^+ = \{b, d, e, f\}$

$a^+ = \{a, i\}$

$h^+ = \{h, j\}$

Abd is a candidate key

{

$Ab \rightarrow c$ (PD)

$Ad \rightarrow g$ (PD)

$Ad \rightarrow h$ (PD)

$bd \rightarrow e$ (PD)

$bd \rightarrow f$ (PD)

$a \rightarrow i$ (PD)

$h \rightarrow j$ (TD)

}

Highest NF = 1NF

abcdefghij

ABC → Ab → c (FD)

Adghj → Ad → gh (FD), h → j (TD)

bdef → bd → ef (FD)

ai → a → i (FD)

ABDij

~~Highest NF is BCNF, 2NF, 3NF~~

Highest NF = 2NF

⇒ Given R(a, b, c, d, e, f, g, h, i, j) {

ab → c

a → de

b → f

f → gh

d → ij }

~~Hint~~

$ab^+ = \{a, b, c, d, e, f, g, h, i, j\}$

~~ab~~ $a^+ = \{a, d, e, i, j\}$

ab is a candidate key

{

ab → c (FD)

a → d (PD)

a → e (PD)

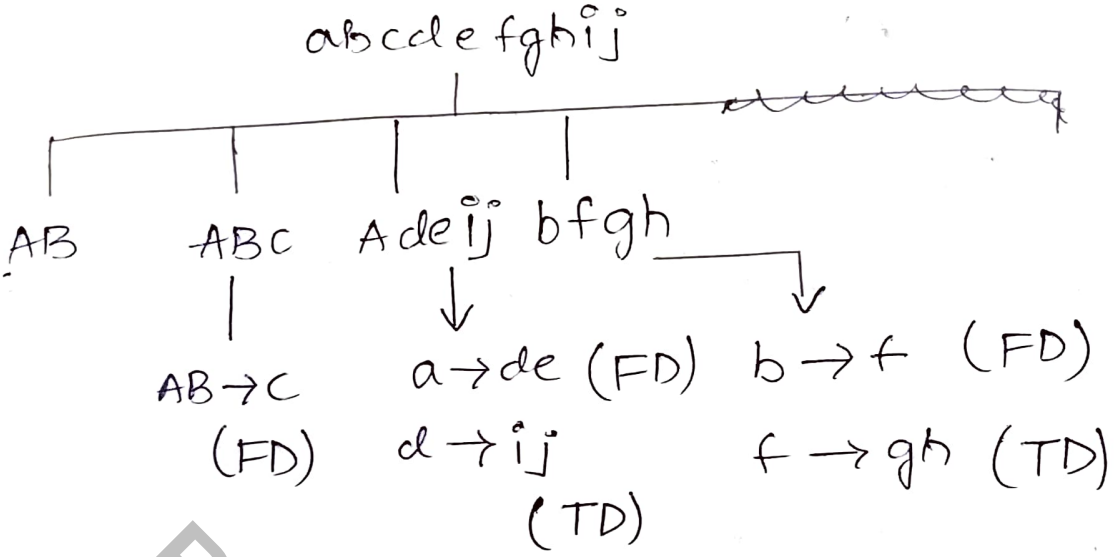
b → f (PD)

f → g (TD)

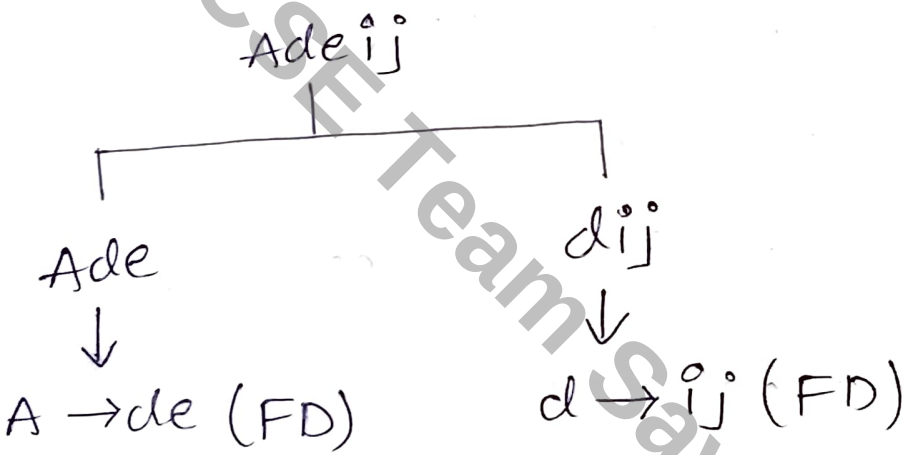
d → i (TD)

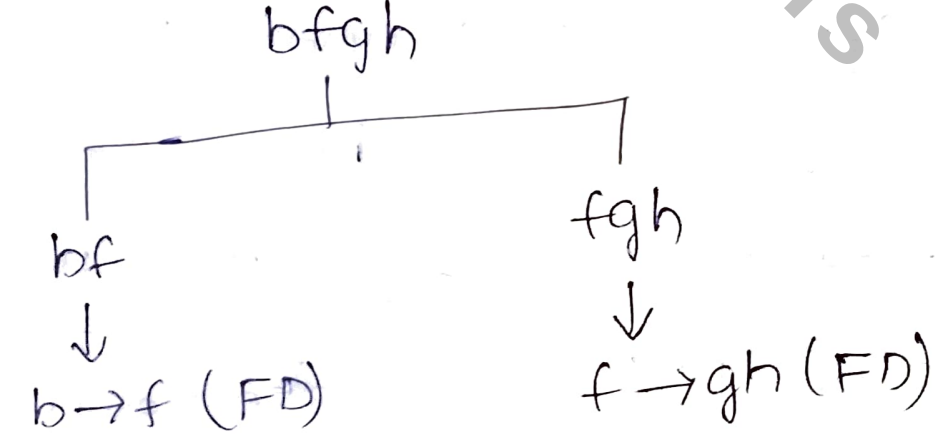
d → j (TD) }

→ Highest NF = 1NF



Highest NF = 2NF





∴ Highest NF = 3NF

→ R(A, B, C, D, E, F, G, H, I, J)

AB → C, BD → EF, AD → GH, A → I, H →

Ans: REPEATED QUESTION

4th Normal Form:-

→ The relation is said to be in 4NF. It satisfies 2 condition.

1) It should be in BCNF

2) No multivalued dependence.

Multivalued dependence:-

→ For a certain value of A there are multiple values of B and C where B & C are independent i.e., B and C are not dependent on each other then it is called multivalued dependence.

Ex:-

A	B	C
1	X	P
	Y	Q
2	Z	R
3	L	R
	M	S

5th Normal Form:-

→ A relation is said to be 5NF if it satisfies 2 conditions.

1) It should be in 4NF
2) No Join dependence (i.e., the table can be join in any order)

NOTE:-

→ A relation can be directly converted into BCNF without converting into 2NF & 3NF. But the problem is dependencies may not be preserved.

⇒ Given R(A,B,C) {

$AB \rightarrow C$

$C \rightarrow B$

}

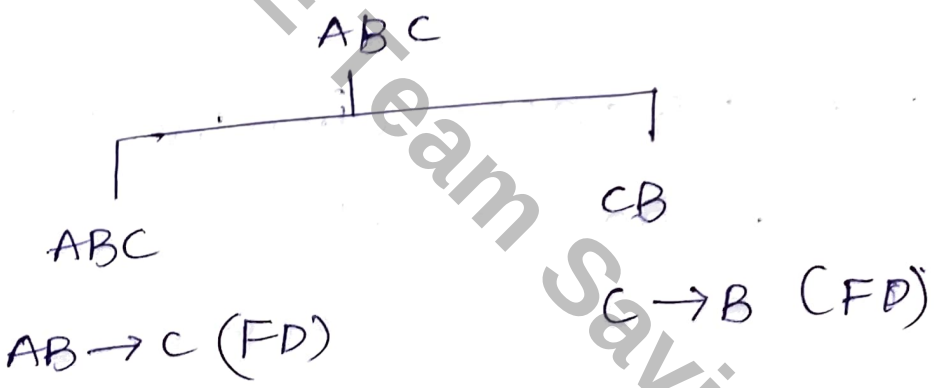
Decompose into BCNF.

Ans: AB → candidate key

{ $AB \rightarrow C$ (FD)

$C \rightarrow B$ (OCD) } 2NF

}



∴ Highest NF = BCNF

NOTE:—

if $X \rightarrow A$ is not FD no devide into XA and R − {A} in above one: ↓

$C \rightarrow B$

BC and AC

But dependency is not preserved.

UNIT – 4

Transaction Management and Concurrency Control

Transaction:–

→ The set of operations that forms a single logical unit of task (or) work.

→ Each and Every Transaction has to satisfy 4 properties called ACID Properties.

1) Atomicity
2) Consistency
3) Isolation
4) Durable

Ex:–

A = 10K B = 20K

T1

R(A)
A = A – 5K
W(A)
R(B)
B = B + 5K
W(B)

Reading Account A = 10K
withdrawl (debited) A = 10K – 5K
write A A = 5K
Reading Account B = 20K
debited B = 20K
write B B = 25K

1) Atomicity :-

→ If a transaction happen it should happen completely (or) it should not at all first.

2) Consistency :-

→ If the total sum is remains same before, and after the transaction then it means data is in consistency (or) correct stat.

3) Isolation :-

→ Each and every transaction should be independent or unaffected by other transaction that are in Progress.

4) Durability :-

→ The changes done to the database because of the transaction must persits or remain if there are system crashes or failures.

Schedule :-

→ A Schedule is a set of operations from a list of transactions

→ There are 2 types of schedule.

i) Serial schedule

ii) Inter-leaved schedule.

Ex :-

A = 10K

B = 20K.

S'

T_1	T_2
R(A)	
A = A - 5K	
W(A)	
	R(A)
	t = A * 0.1
	A = A + t
	W(A)
R(B)	
B = B + 5K	
W(B)	

S1

T_1	T_2
R(A)	
A = A - 5K	
W(A)	
R(B)	
B = B + 5K	
W(B)	
	R(A)
	t = A * 0.1
	A = A + t
	W(A)

$T_1 \rightarrow T_2$ | A = 5500, B = 25K.

$T_2 \rightarrow T_1$ | A = 6K, B = 25K

S_1 — Serial schedule.

S' — Inter-leaved schedule.

S' : A = 5500

B = 25K

Serial Schedule:-

→ If the transaction part of a schedule are executed in same sequential order than the schedule is called a serial schedule.

(or)

→ In a serial schedule all the opuriations of a transaction are executed together.

Inter-leaved Schedule:-

→ If the opurations of various transactions are executed in a inter-leaved manner i.e., If they don't occur together then it is called an Inter-leaved Schedule.

Serializability:-

→ If the outcome of the inter-leaved schedule is equivalent to the outcome of anyone of the serial schedule then such a schedule is called Serializible Schedule. And the property is called Serializability.

→ There are

i) Result

ii) Conflict

iii) View

i) Result Se

→ A Schedu

be result

equivalent

schedule.

→ Two sc

result eq

same fina

Ex:- S_1

S'

S_1 & S'

→ There are 3 ways of testing Serializability

i) Result Serializability
ii) Conflict Serializability
iii) View Serializability

i) Result Serializability:—

→ A Schedule S' (Interleaved) is said to be result serializable if it is result equivalent to any one of the Serial Schedule.

→ Two schedules S and S' are said to be result equivalent if they produce the same final database state or result.

Ex:- $S_1 \rightarrow T_1 \rightarrow T_2$ | A = 5500, B = 25K

$S' \rightarrow$ A = 5500, B = 25K

S_1 & S' are result serializability.

ii) Conflict Serializability :—

→ A schedule is said to be conflict serializable if it is conflict Equivalent to any one of the Serial schedule.

→ Two schedules S (Serial) and S' (interleaved) are said to be conflict Equivalent if they produce same set of conflicts.

→ The different types of conflicts are :

1) R — W	R — Read
2) W — R	W — write
3) W — W	

⇒ Check whether the schedule mentioned is conflict serializable.

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

Ans : (5)

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

conflict

$R_1(A) - W_2(A)$

$W_1(A) - R_2(A)$

$W_1(A) - W_2(A)$

(5)

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

$T_1 \rightarrow T_2$

conflict

$R_1(A) \rightarrow W_2(A)$

$W_1(A) \rightarrow R_2(A)$

$W_1(A) \rightarrow W_2(A)$

$\therefore S_4$ is conflict serializable.

$\Rightarrow S: R_1(x)\ R_3(x)\ W_1(x)\ R_2(x)\ W_2(x)$

Ans

T_1	T_2	T_3
R(x)		
		R(x)
W(x)		
	R(x)	
	W(x)	

conflicts :

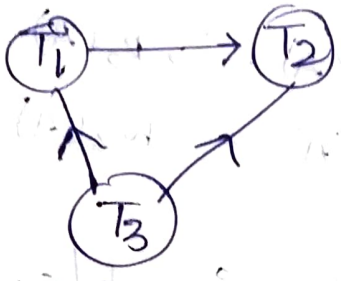
$R_1(x) - W_2(x) = (R-W)$

$W_1(x) - W_2(x) = (W-W)$

$W_1(x) - R_2(x) = (W-R)$

$R_3(x) - W_2(x) = (R-W)$

→ If atleast one conflict is present keep an edge each transaction one vertex.



[no conflict from T_2 to T_1 T_2 to T_3]

**

No cycle/loop so it is conflict serializable

→ In above graph T_3 is independent.

[T_2 is dependent on T_1 and T_3, T_1 is dependent on T_3]

→ T_1 should happen after T_3. T_2 happen after T_1 ⇒ Serial schedule = T_3 T_1 T_2

→ It is equivalent to T_3 T_1 T_2 schedule.

T_3	T_1	T_2
R(x)		
	R(x)	
	W(x)	
		R(x)
		W(x)

NOTE :-

→ A precedence graph is used in checking conflict serialisability where if a cycle is not obtained in a graph then, the schedule is said to be conflict serializable.

⇒ check following is conflict serializable

S: $R_4(A)$ $R_2(A)$ $R_3(A)$ $W_1(B)$ $W_2(A)$ $R_3(B)$ $W_2(B)$

Ans:

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

Conflicts :-

[check on same data]

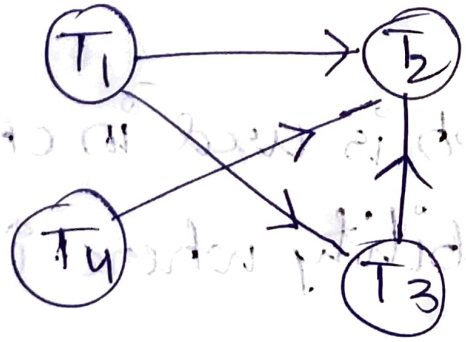
$W_1(B) - W_2(B)$

$W_1(B) - R_3(B)$

$R_3(B) - W_2(B)$

$R_4(A) - W_2(A)$

$R_3(A) - W_2(A)$



→ no-cycle is formed it is conflict Serializable.

T_1, T_4 ~~is~~ independent

T_2 is dependent on T_1, T_4, T_3

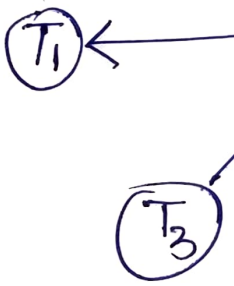
T_3 is dependent on T_1

⇒ $T_1\ T_3\ T_4\ T_2$, $T_1\ T_4\ T_3\ T_2$, $T_4\ T_1\ T_3\ T_2$ are possible serial schedule

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

Ans ⇒

T_1	T_2	T_3
$R_1(A)$		
	$R(A)$	
$R(B)$		
	$R(B)$	$R(B)$
$W(A)$		
	$W(B)$	

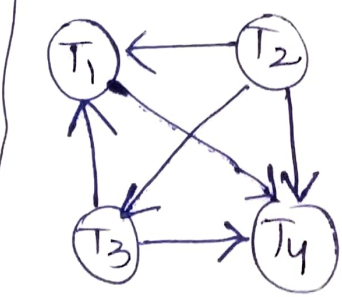


→ ~~Precedence~~ Not conflict serializable.

⇒ S : $R_2(x)$ $W_3(x)$ $W_1(x)$ $W_2(y)$ $R_2(z)$ $R_4(x)$ $R_4(y)$

Ans ⇒

T_1	T_2	T_3	T_4
	$R(x)$		
		$W(x)$	
$W(x)$			
	$W(y)$		
	$R(z)$		
			$R(x)$
			$R(y)$



→ conflict serializable

T_2 is independent

→ T_1 is dependent on T_2, T_3

T_3 is dependent on T_2

T_4 is dependent on T_1, T_2, T_3

Equivalent ~~to~~ Serial

$T_2\ T_3\ T_1\ T_4$ ⟶ ~~Conflict~~ Serializable

PRACTICE PROBLEMS BASED ON CONFLICT SERIALIZABILITY-

Problem-01:

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

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

Solution-

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

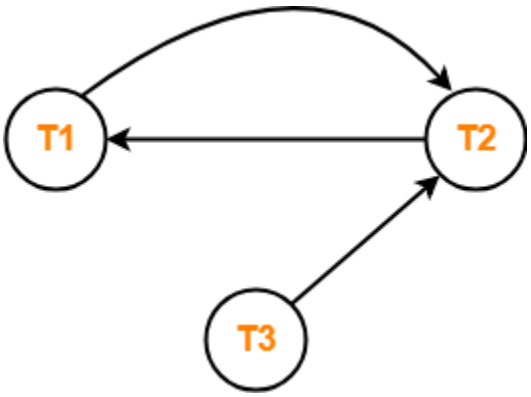
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

-
- $R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$

Step-02:

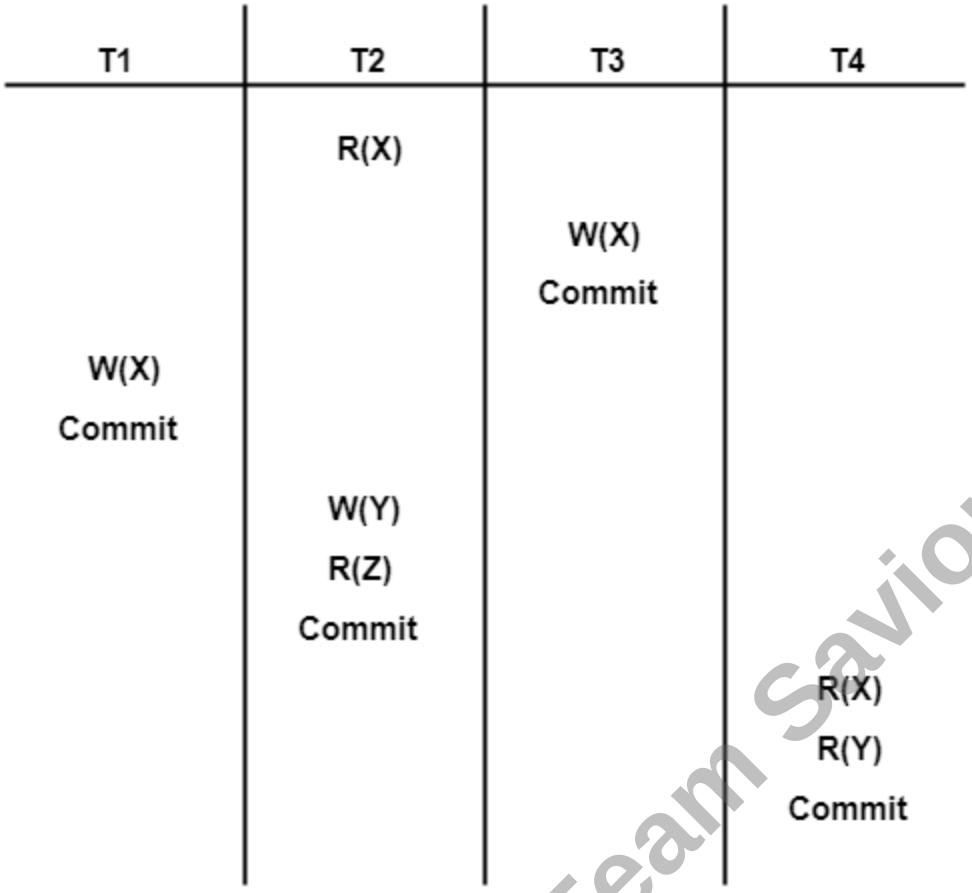
Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Problem-02:

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



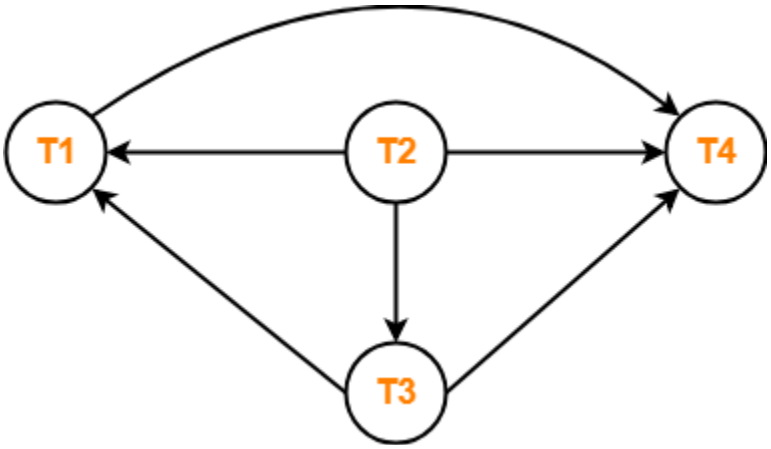
Solution- Checking Whether S is Conflict Serializable Or Not-

Step-01: List all the conflicting operations and determine the dependency between the transactions-

- $W_1(X)$, $R_4(X)$ $(T_1 \rightarrow T_4)$
- $R_2(X)$, $W_3(X)$ $(T_2 \rightarrow T_3)$
- $R_2(X)$, $W_1(X)$ $(T_2 \rightarrow T_1)$
- $W_2(Y)$, $R_4(Y)$ $(T_2 \rightarrow T_4)$
- $W_3(X)$, $W_1(X)$ $(T_3 \rightarrow T_1)$
- $W_3(X)$, $R_4(X)$ $(T_3 \rightarrow T_4)$

Step-02:

Draw the precedence graph-



- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.

Checking Whether S is Recoverable Or Not-

- Conflict serializable schedules are always recoverable.
- Therefore, the given schedule S is recoverable.

Problem-03:

Check whether the given schedule S is conflict serializable or not. If yes, then determine all the possible serialized schedules-

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

Solution- Checking Whether S is Conflict Serializable Or Not-

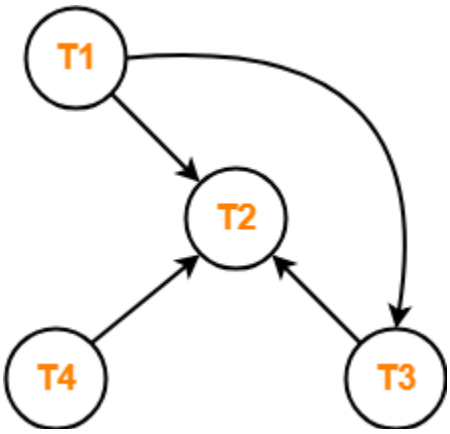
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_4(A)$, $W_2(A)$ $(T_4 \rightarrow T_2)$
- $R_3(A)$, $W_2(A)$ $(T_3 \rightarrow T_2)$
- $W_1(B)$, $R_3(B)$ $(T_1 \rightarrow T_3)$
- $W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
-
- $R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$

Step-02:

Draw the precedence graph-



- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.

Finding the Serialized Schedules-

- All the possible topological orderings of the above precedence graph will be the possible serialized schedules.
- The topological orderings can be found by performing the **Topological Sort** of the above precedence graph.

After performing the topological sort, the possible serialized schedules are-

1. $T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$
2. $T_1 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$
3. $T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$

Problem-04:

Determine all the possible serialized schedules for the given schedule-

T1	T2
R(A)	
A = A-10	
	R(A)
	Temp = 0.2 x A
	W(A)
	R(B)
W(A)	
R(B)	
B = B+10	
W(B)	
	B = B+Temp
	W(B)

Solution-

The given schedule S can be rewritten as-

T1	T2
R(A)	
	R(A)
	W(A)
	R(B)
W(A)	
R(B)	
W(B)	
	W(B)

This is because we are only concerned about the read and write operations taking place on the database.

Checking Whether S is Conflict Serializable Or Not-

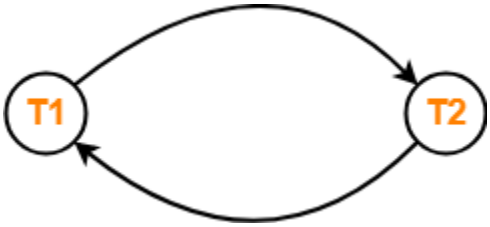
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_1(A)$, $W_2(A)$ $(T_1 \rightarrow T_2)$
- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $W_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $R_2(B)$, $W_1(B)$ $(T_2 \rightarrow T_1)$
- $R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.
- Thus, Number of possible serialized schedules = 0.

View Serializability-

If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.

View Equivalent Schedules-

Consider two schedules S1 and S2 each consisting of two transactions T1 and T2.

Schedules S1 and S2 are called view equivalent if the following three conditions hold true for them-

Condition-01:

For each data item X, if transaction T_i reads X from the database initially in schedule S1, then in schedule S2 also, T_i must perform the initial read of X from the database.

Thumb Rule

"Initial readers must be same for all the data items".

Condition-02:

If transaction T_i reads a data item that has been updated by the transaction T_j in schedule S1, then in schedule S2 also, transaction T_i must read the same data item that has been updated by the transaction T_j.

Thumb Rule

"Write-read sequence must be same.".

Condition-03:

For each data item X, if X has been updated at last by transaction T_i in schedule S1, then in schedule S2 also, X must be updated at last by transaction T_i.

Thumb Rule

"Final writers must be same for all the data items".

Checking Whether a Schedule is View Serializable Or Not-

Method-01:

Check whether the given schedule is conflict serializable or not.

- If the given schedule is conflict serializable, then it is surely view serializable.
- If the given schedule is not conflict serializable, then it may or may not be view serializable. Now check using other methods.

Thumb Rules

- All conflict serializable schedules are view serializable.
- All view serializable schedules may or may not be conflict serializable.

Method-02:

Check if there exists any blind write operation.

(Writing without reading is called as a blind write).

- If there does not exist any blind write, then the schedule is surely not view serializable.
- If there exists any blind write, then the schedule may or may not be view serializable. Check using other methods.

Thumb Rule

No blind write means not a view serializable schedule.

Method-03:

In this method, try finding a view equivalent serial schedule.

- By using the above three conditions, write all the dependencies.
- Then, draw a graph using those dependencies.
- If there exists no cycle in the graph, then the schedule is view serializable otherwise not.

PRACTICE PROBLEMS BASED ON VIEW SERIALIZABILITY-

Problem-01:

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

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

Solution-

- We know, if a schedule is conflict serializable, then it is surely view serializable.
- So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

Step-01:

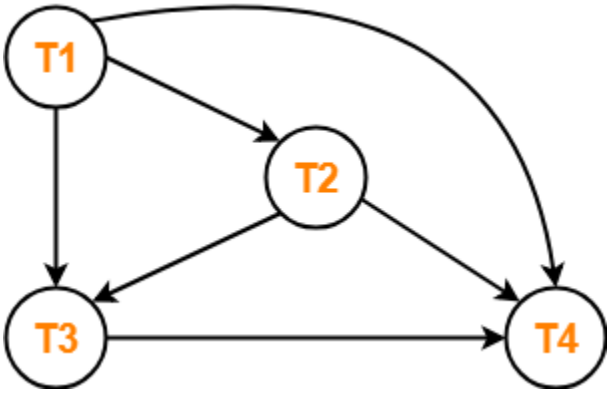
List all the conflicting operations and determine the dependency between the transactions-

- $W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $W_1(B)$, $W_3(B)$ $(T_1 \rightarrow T_3)$
- $W_1(B)$, $W_4(B)$ $(T_1 \rightarrow T_4)$
- $W_2(B)$, $W_3(B)$ $(T_2 \rightarrow T_3)$

- $W_2(B)$, $W_4(B)$ $(T_2 \rightarrow T_4)$
- $W_3(B)$, $W_4(B)$ $(T_3 \rightarrow T_4)$

Step-02:

Draw the precedence graph-



- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.
- Thus, we conclude that the given schedule is also view serializable.

Problem-02:

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

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

Solution-

- We know, if a schedule is conflict serializable, then it is surely view serializable.
- So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

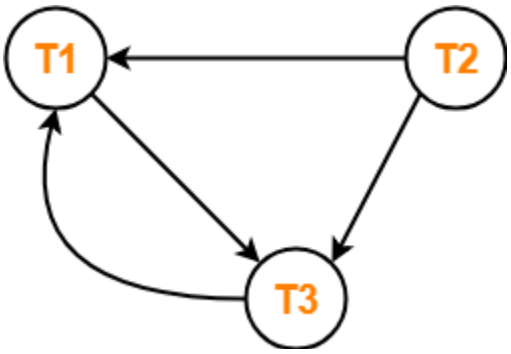
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_1(A)$, $W_3(A)$ $(T_1 \rightarrow T_3)$
- $R_2(A)$, $W_3(A)$ $(T_2 \rightarrow T_3)$
- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $W_3(A)$, $W_1(A)$ $(T_3 \rightarrow T_1)$

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
- To check whether S is view serializable or not, let us use another method.
- Let us check for blind writes.

Checking for Blind Writes-

- There exists a blind write $W_3(A)$ in the given schedule S.
- Therefore, the given schedule S may or may not be view serializable.

Now,

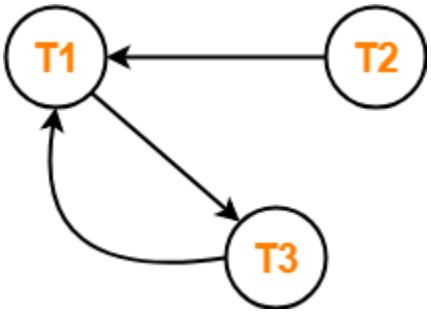
- To check whether S is view serializable or not, let us use another method.
- Let us derive the dependencies and then draw a dependency graph.

Drawing a Dependency Graph-

- T1 firstly reads A and T3 firstly updates A.
- So, T1 must execute before T3.
- Thus, we get the dependency **T1 → T3**.
- Final updation on A is made by the transaction T1.
- So, T1 must execute after all other transactions.
- Thus, we get the dependency **(T2, T3) → T1**.

- There exists no write-read sequence.

Now, let us draw a dependency graph using these dependencies-



- Clearly, there exists a cycle in the dependency graph.
- Thus, we conclude that the given schedule S is not view serializable.

Problem-03:

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

T1	T2
R (A)	
A = A + 10	
	R (A)
	A = A + 10
W (A)	
	W (A)
R (B)	
B = B + 20	
	R (B)
	B = B x 1.1
W (B)	
	W (B)

Solution-

- We know, if a schedule is conflict serializable, then it is surely view serializable.
- So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

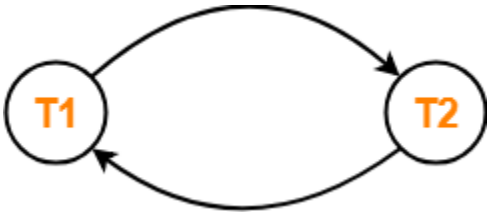
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_1(A)$, $W_2(A)$ $(T_1 \rightarrow T_2)$
- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $W_1(A)$, $W_2(A)$ $(T_1 \rightarrow T_2)$
- $R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $R_2(B)$, $W_1(B)$ $(T_2 \rightarrow T_1)$

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
- To check whether S is view serializable or not, let us use another method.
- Let us check for blind writes.

Checking for Blind Writes-

- There exists no blind write in the given schedule S.
- Therefore, it is surely not view serializable.

Alternatively,

- You could directly declare that the given schedule S is not view serializable.
- This is because there exists no blind write in the schedule.

- You need not check for conflict serializability.

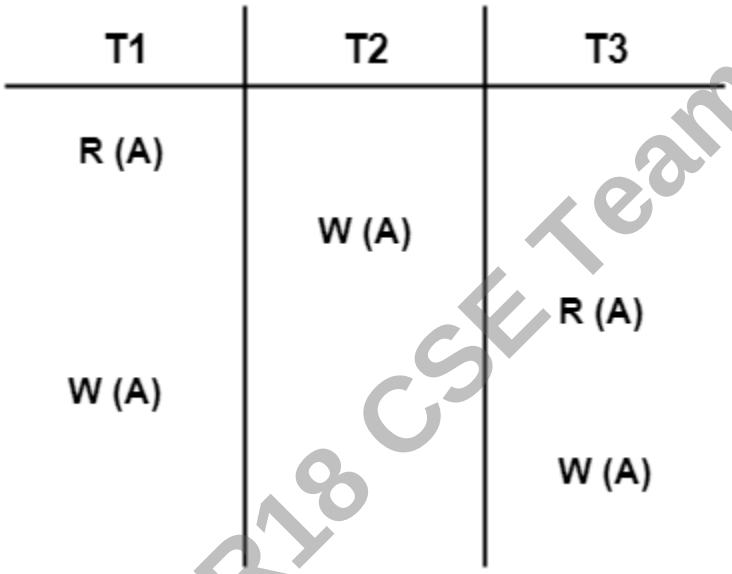
Problem-04:

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

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

Solution-

For simplicity and better understanding, we can represent the given schedule pictorially as-



- We know, if a schedule is conflict serializable, then it is surely view serializable.
- So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

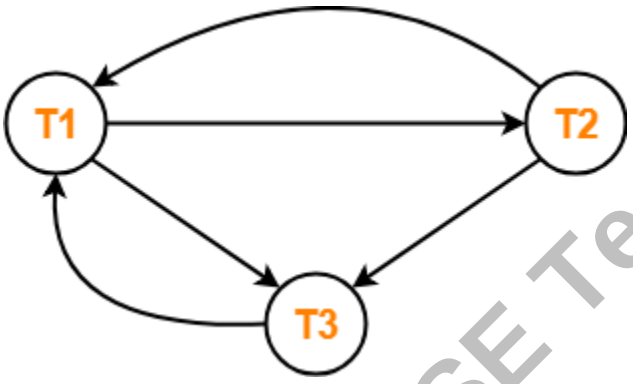
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_1(A)$, $W_2(A)$ $(T_1 \rightarrow T_2)$
- $R_1(A)$, $W_3(A)$ $(T_1 \rightarrow T_3)$
- $W_2(A)$, $R_3(A)$ $(T_2 \rightarrow T_3)$
- $W_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $W_2(A)$, $W_3(A)$ $(T_2 \rightarrow T_3)$
- $R_3(A)$, $W_1(A)$ $(T_3 \rightarrow T_1)$
- $W_1(A)$, $W_3(A)$ $(T_1 \rightarrow T_3)$

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
- To check whether S is view serializable or not, let us use another method.
- Let us check for blind writes.

Checking for Blind Writes-

- There exists a blind write $W_2(A)$ in the given schedule S.

- Therefore, the given schedule S may or may not be view serializable.

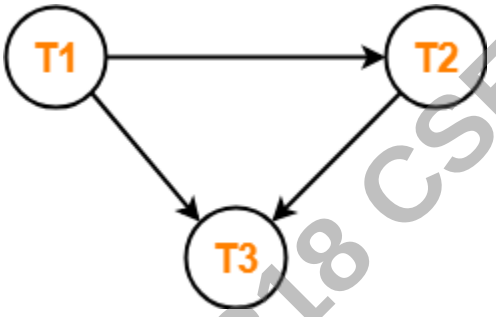
Now,

- To check whether S is view serializable or not, let us use another method.
- Let us derive the dependencies and then draw a dependency graph.

Drawing a Dependency Graph-

- T1 firstly reads A and T2 firstly updates A.
- So, T1 must execute before T2.
- Thus, we get the dependency **T1 → T2**.
- Final updation on A is made by the transaction T3.
- So, T3 must execute after all other transactions.
- Thus, we get the dependency **(T1, T2) → T3**.
- From write-read sequence, we get the dependency **T2 → T3**

Now, let us draw a dependency graph using these dependencies-



- Clearly, there exists no cycle in the dependency graph.
- Therefore, the given schedule S is view serializable.
- The serialization order T1 → T2 → T3.

RECOVERABLE SCHEDULE:

If a Transaction T2 reads a value written by another Transaction T1 and T2 commits after T1 commits, then it is called recoverable schedule because if T1 fails and rolls back , then T2 also can rollback.

Example:

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
COMMIT	
	COMMIT

IRRECOVERABLE SCHEDULE:

If a Transaction T2 reads a value written by another Transaction T1 and T2 commits before T1 commits, then it is called irrecoverable schedule because if T1 fails and rolls back , then T2 cannot rollback.

Example:

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	COMMIT
COMMIT	

COMPLETE SCHEDULE:

A Schedule is said to be complete if each and every transaction end with either commit or abort .

CASCADING ROLLBACK/ CASCADING ABORT:

If a failure of a single transaction causes multiple transactions to rollback, it is called Cascading rollback or Cascading abort.

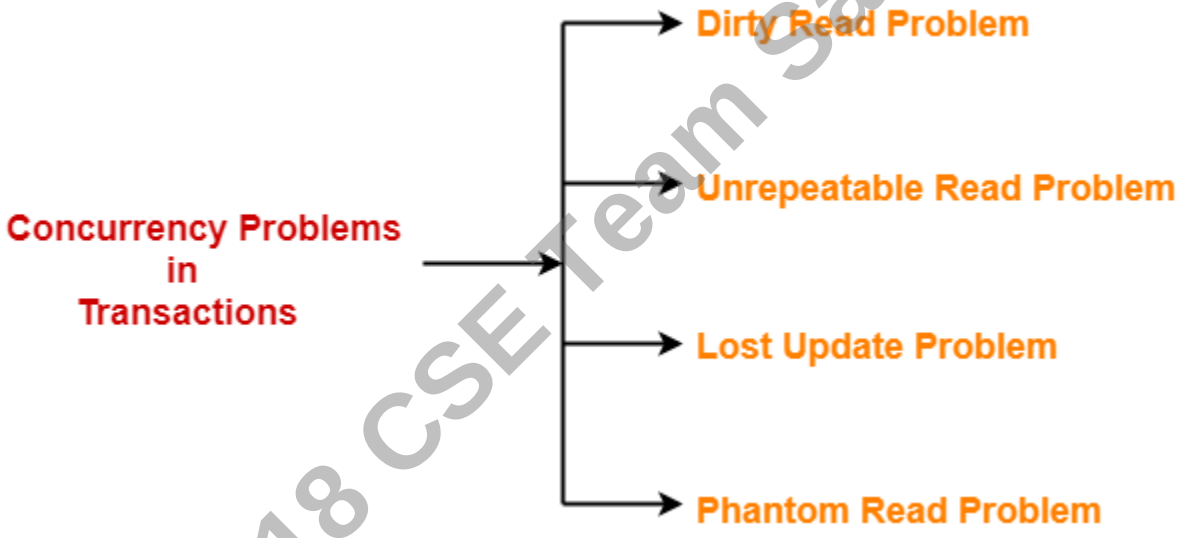
Example:

T1	**T2**	**T3**	**T4**
R(A)			
W(A)			
	R(A)		
	W(A)		
		R(A)	
		W(A)	
			R(A)
			W(A)
COMMIT			
	COMMIT		
		COMMIT	
			COMMIT

Concurrency Problems in DBMS (Different types of conflicts)-

- When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems.
- Such problems are called as **concurrency problems**.

The concurrency problems are-



1. Dirty Read Problem/ Reading uncommitted data (Write-Read Conflict)
2. Unrepeatable Read Problem/ Non repeatable read (Read-Write Conflict)
3. Lost Update Problem/ Overwriting Uncommitted data (Write-Write Conflict)
4. Phantom Read Problem

1. Dirty Read Problem-

> Reading the data written by an uncommitted transaction is called as dirty read.

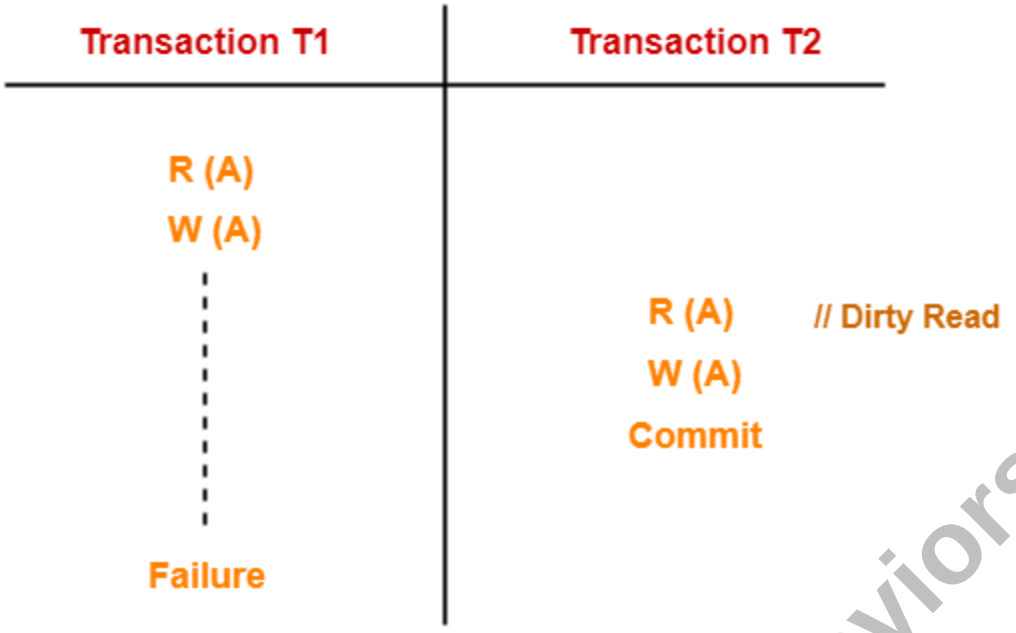
This read is called as dirty read because-

- There is always a chance that the uncommitted transaction might roll back later.
- Thus, uncommitted transaction might make other transactions read a value that does not even exist.
- This leads to inconsistency of the database.

NOTE-

- Dirty read does not lead to inconsistency always.
- It becomes problematic only when the uncommitted transaction fails and roll backs later due to some reason.

Example-



Here,

1. T1 reads the value of A.
2. T1 updates the value of A in the buffer.
3. T2 reads the value of A from the buffer.
4. T2 writes the updated the value of A.
5. T2 commits.
6. T1 fails in later stages and rolls back.

In this example,

- T2 reads the dirty value of A written by the uncommitted transaction T1.
- T1 fails in later stages and roll backs.
- Thus, the value that T2 read now stands to be incorrect.
- Therefore, database becomes inconsistent.

Example: Initially A=10,000 and B=20,000

T1	T2
R(A)	
A=A-5000	
W(A)	

	R(A)
	T=A*0.10
	A=A+T
	W(A)
R(B)	
TRANSACTION FAILURE	

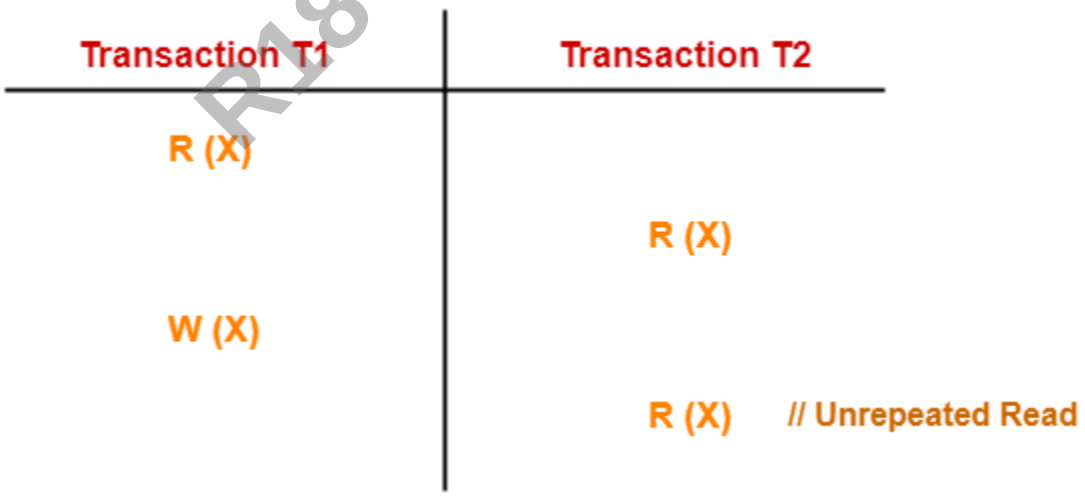
B=B+5000 // NOT ALLOWED.
ABORT AND ROLLBACK	

W(B)	

2. Unrepeatable Read Problem-

This problem occurs when a transaction gets to read unrepeated i.e. different values of the same variable in its different read operations even when it has not updated its value.

Example-



Here,

1. T1 reads the value of X (= 10 say).
2. T2 reads the value of X (= 10).
3. T1 updates the value of X (from 10 to 15 say) in the buffer.
4. T2 again reads the value of X (but = 15).

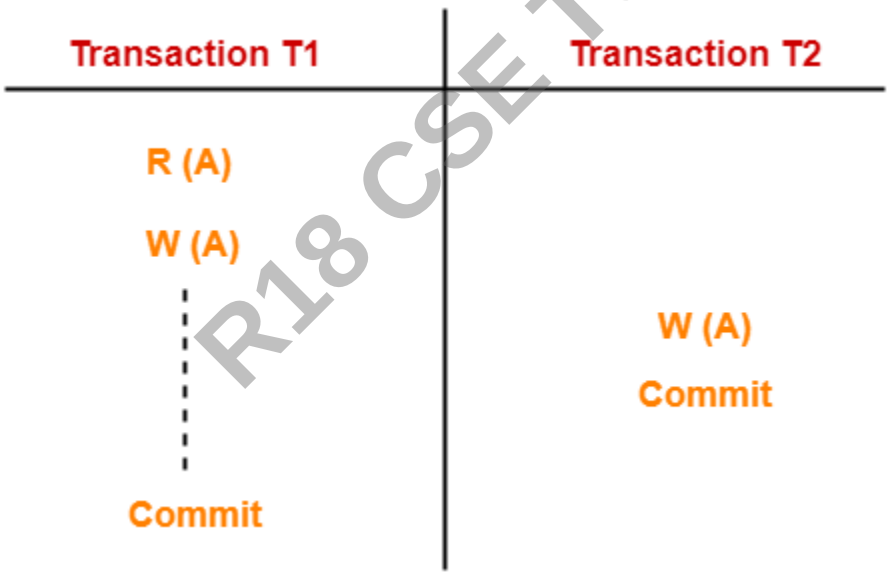
In this example,

- T2 gets to read a different value of X in its second reading.
- T2 wonders how the value of X got changed because according to it, it is running in isolation.

3. Lost Update Problem-

This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

Example-



Here,

1. T1 reads the value of A (= 10 say).
2. T1 updates the value to A (= 15 say) in the buffer.

3. T2 does blind write A = 25 (write without read) in the buffer.
4. T2 commits.
5. When T1 commits, it writes A = 25 in the database.

In this example,

- T1 writes the over written value of A in the database.
- Thus, update from T1 gets lost.

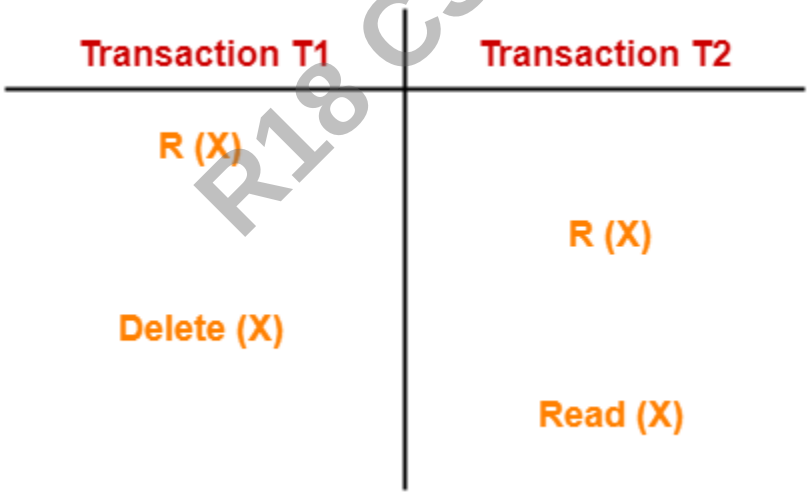
NOTE-

- This problem occurs whenever there is a write-write conflict.
- In write-write conflict, there are two writes one by each transaction on the same data item without any read in the middle.

4. Phantom Read Problem-

This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later, it finds that the variable does not exist.

Example-



Here,

1. T1 reads X.
2. T2 reads X.
3. T1 deletes X.
4. T2 tries reading X but does not find it.

In this example,

- T2 finds that there does not exist any variable X when it tries reading X again.
- T2 wonders who deleted the variable X because according to it, it is running in isolation.

Avoiding Concurrency Problems-

- To ensure consistency of the database, it is very important to prevent the occurrence of above problems.
- **Concurrency Control Protocols** help to prevent the occurrence of above problems and maintain the consistency of the database.

Concurrency control:

Concurrency Control Protocols help to prevent the concurrency problems and maintain the consistency of the database.

Concurrency control can be divided into three protocols

1. Lock Based Protocols
2. Graph Based Protocols
3. Timestamp Based Protocol

1. Lock Based Protocols in DBMS

The **protocol** utilizes locks, applied by a transaction to data item, which may block other transactions from accessing the same data during the transaction's life.

Types of locking protocols

1. Simple locking
2. 2-Phase Locking
    a. Simple 2-phase locking
    b. Strict 2-phase locking
    c. Rigorous 2-phase locking
    d. Conservative 2-Phase locking

1) Simple locking:

There are two types of lock :

1. Shared Lock
2. Exclusive Lock

Shared Locks are represented by S. The data items can only read without performing modification to it from the database. S – lock is requested using lock – s instruction.

Exclusive Locks are represented by X. The data items can be read as well as written. X – lock is requested using lock – X instruction.

T1	T2
LOCK-S(A)	
READ(A)	LOCK-S(A)
	READ(A)
UNLOCK-S(A)	
LOCK-X(B)	
READ(B)	LOCK-X(B)-NOT ALLOWED WRITE(B)-NOT ALLOWED
WRITE(B)	UNLOCK-S(A)
UNLOCK-X(B)	
	LOCK-X(B) WRITE(B)
	UNLOCK-X(B)

Lock Compatibility Matrix

- Lock Compatibility Matrix controls whether multiple transactions can acquire locks on the same resource at the same time.

	Shared	Exclusive
Shared	True	False
Exclusive	False	False

- If a resource is already locked by another transaction, then a new lock request can be granted only if the mode of the requested lock is compatible with the mode of the existing lock.
- Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive lock on item, no other transaction may hold any lock on the item.

Example:

L-S(A)
Read(A)
U-S(A)
L-X(B)
Write(B)
U-X(B)

LIMITATIONS:

1. **If the locks are released too early, it leads to Database Inconsistency.**
   Example: A=10,000, B=20,000

   A+B=30,000

   L-X(A)
   R(A)

A=A-5000
W(A)
U-X(A)

A+B=25,000 (At this point, it leads to inconsistency as the lock is released)

L-X(B)
R(B)
B=B+5000
W(B)
U-X(B)

A+B=30,000

2. It may not guarantee Serializability.

3. Dead locks are possible.

T1	T2
L-X(A)	
	L-X(B)
L-X(B)	
	L-X(A)

2) 2-Phase locking:

It is used to overcome the disadvantage of the simple lock. In the 2-phase locking, there are two phases are growing phase and shrinking phase. 2-phase ensures serializability.

Growing phase: A transaction may obtain locks but not release any locks.

Shrinking phase: A transaction may release lock but may not obtain new locks.

a) **Simple 2-phase locking:** Once a transaction releases a lock it enters in shrinking phase and in shrinking phase it cannot issue any more locks.

Example:

T1	COMMENTS
LOCK-S(A)	GROWING PHASE
READ(A)	
LOCK-X(B)	
READ(B)	
WRITE(B)	
UNLOCK-X(B)	SHRINKING PHASE
UNLOCK-S(A)	

Advantage:

- Less resource utilization.
- Guarantees serializability based on Lock points.

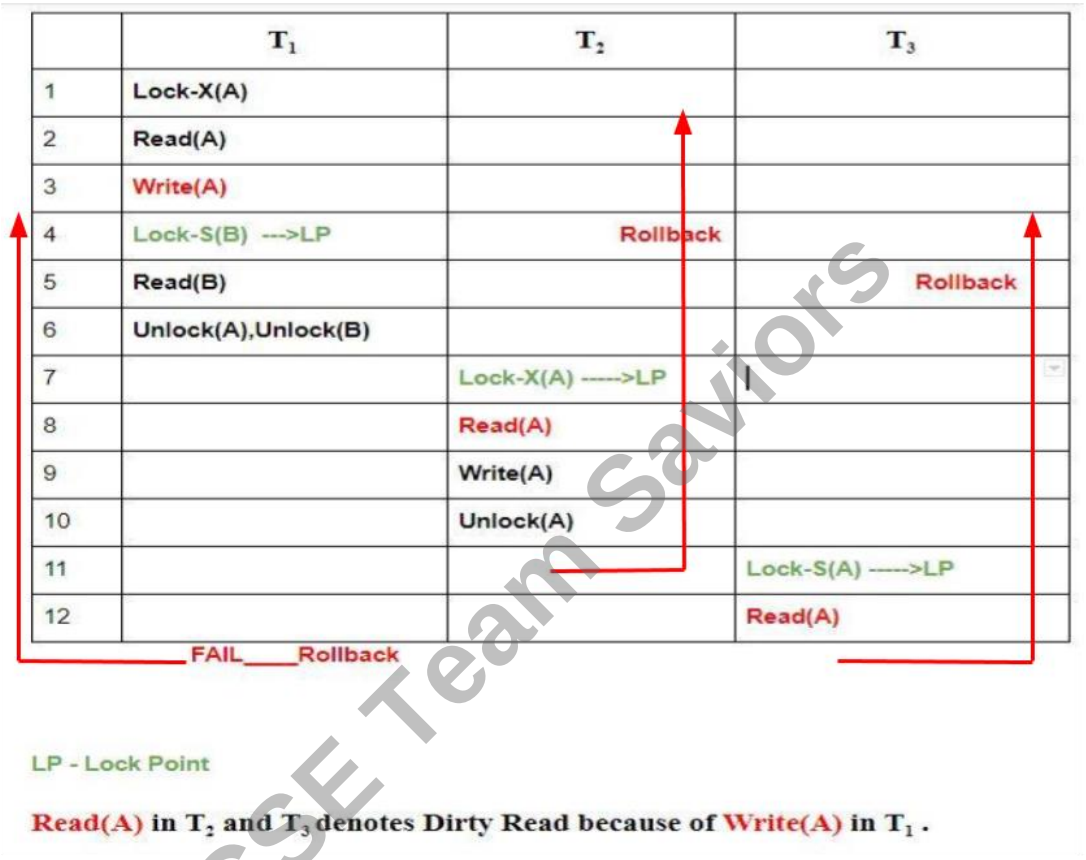
T1	T2	T3
LP		
		LP
	LP	

Disadvantage:

- Deadlock present.
- Cascading roll backs are possible

Note:

1. **Lock point**: The final point of lock in a schedule of transaction is called lock point of transaction.



	T_1	T_2	T_3
1	Lock-X(A)		
2	Read(A)		
3	Write(A)		
4	Lock-S(B) --->LP	Rollback	
5	Read(B)		Rollback
6	Unlock(A),Unlock(B)		
7		Lock-X(A) ----->LP	
8		Read(A)	
9		Write(A)	
10		Unlock(A)	
11			Lock-S(A) ----->LP
12			Read(A)

2. **Upgrade & Downgrade**: Conversion from shared to an exclusive mode is called upgrade and its opposite conversion i.e. from exclusive to shared mode is called downgrade.

Example:

Upgrade:	Downgrade:
L-S(A)	L-X(A)
R(A)	W(A)
L-X(A)	L-S(A)
W(A)	R(A)
U-(A)	U-(A)

There are three types of 2-phase locking protocols:

A) Strict 2-phase locking:
Transaction does not release any of the exclusive locks until after its commit or abort.

T1
LOCK-S(A)
READ(A)
LOCK-X(B)
READ(B)
WRITE(B)
UNLOCK-S(A)
COMMIT
UNLOCK-X(B)

Advantage:

- No dirty read problem.
- No Cascading rollbacks.

Disadvantage:

- Deadlock are possible.

B) Rigorous 2-phase locking: Transaction **"t"** does not release any of its locks until after it commits or aborts.

T1
LOCK-S(A)
READ(A)
LOCK-X(B)
READ(B)
WRITE(B)
COMMIT
UNLOCK-S(A)
UNLOCK-X(B)

Advantage:

- No Cascading rollbacks

Disadvantage:

- Deadlock can still occur.

C) Conservative 2-phase locking: The transaction will obtain all the locks at the beginning i.e. before the transaction starts and will release all the locks only after it commits.

T1
LOCK-S(A)
LOCK-X(B)
READ(A)
READ(B)
WRITE(B)
COMMIT
UNLOCK-S(A)
UNLOCK-X(B)

Advantage:

- No waiting for the data item
- No Cascading rollbacks
- No deadlocks

Disadvantage:

- Practically data implementation is difficult
- It leads to starvation.

GRAPH BASED PROTOCOLS

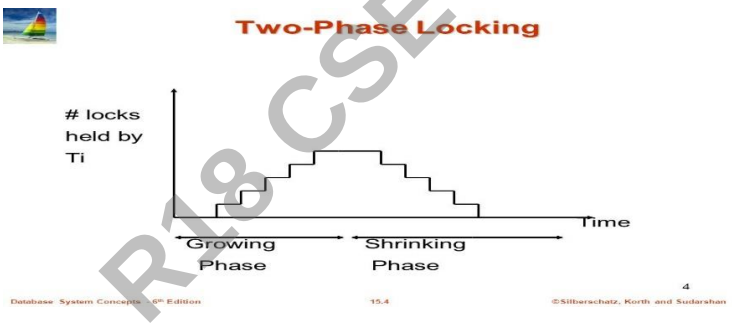
1. Graph-based protocols are an alternative to two-phase locking.

Limitations of 2PL:

a. Deadlocks can be avoided if we follow Conservative 2-PL but the problem with this protocol is it cannot be used practically.

b. It increases the waiting time and reduces the efficiency as the locks cannot be released during the growing phase

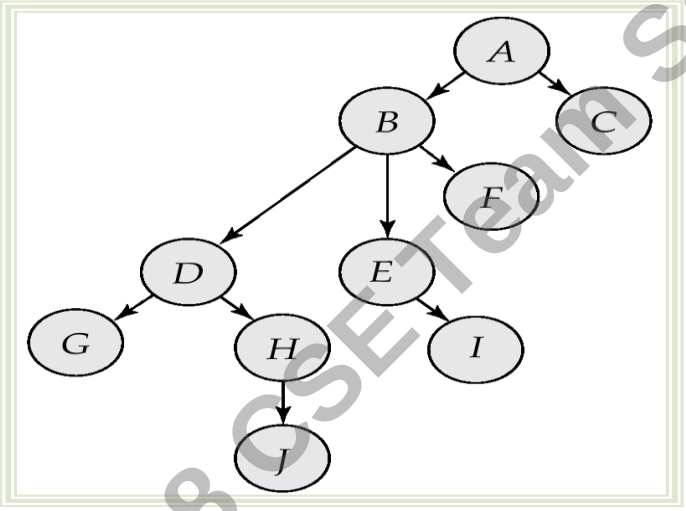
Example:



A Graph based protocol is implemented as a Tree.

- Impose a partial ordering on the set D = {d1, d2 ,..., dh} of all data items.

- If di -> dj then any transaction accessing both di and dj must access di before accessing dj.
- Implies that the set D may now be viewed as a directed acyclic graph, called a database graph.
- The tree-protocol is a simple kind of graph protocol.



The set of rules used by a Tree Protocol are:

- Only Exclusive locks are allowed.
- The first lock by Ti may be on any data item. Subsequently, a data Q can be locked by Ti only if the parent of Q is currently locked by Ti.
- Data items may be unlocked at any time.

- Data item that is locked and unlocked by Ti cannot be subsequently relocked by the same transaction Ti.

Advantage –

1. Ensures Conflict Serializable Schedule.
2. Ensures Deadlock Free Schedule
3. Shorter waiting time as unlocking can be done anytime

Disadvantage –

1. Unnecessary locking overheads may happen sometimes, like if we want both D and E, then at least we have to lock B to follow the protocol.
2. Cascading Rollbacks is still a problem. We don't follow a rule of when Unlock operation may occur so this problem persists for this protocol.
3. Prior knowledge of data access.

Timestamp-Based Protocols

Each transaction is issued a timestamp when it enters the system. If an old transaction Ti has time-stamp TS(Ti), a new transaction Tj is assigned time-stamp TS(Tj) such that

TS(Ti) <TS(Tj).

- The protocol manages concurrent execution such that the time-stamps determine the serializability order.

Timestamps can be used in two ways:

1. To determine the currency or outdatedness of a request with respect to the data object it is operating on.
2. To order events with respect to one another.

- In order to assure such behavior, the protocol maintains for each data Q two timestamp values:

- W-timestamp(A) is the largest time-stamp of any transaction that executed write(A) successfully.

T1(9)	T2(10)	T3(11)
W(A)		
		W(A)
	W(A)	

- R-timestamp(Q) is the largest time-stamp of any transaction that executed read(Q) successfully.

T1(9)	T2(10)	T3(11)
	R(A)	
		R(A)
R(A)		

There are two types of Time stamp based protocols:

1. Time stamp ordering protocol
2. Thomas Write Rule.

TIMESTAMP ORDERING PROTOCOL:

The timestamp ordering protocol provides an ordering among the transactions so that any conflicting read and write operations are executed in timestamp order.

If not such an operation is rejected and the transaction is rolled back and the rolled back transaction is given a new timestamp.

Timestamp ordering protocol works as follows −

-
-
-
-

- **If a transaction Ti issues a read(X) operation −**
  - If TS(Ti) < W-timestamp(X)
    - Operation rejected.

T1(9)	T2(10)
	W(X)
R(X)-REJECTED	

  - If TS(Ti) >= W-timestamp(X)
    - Operation executed.
  - All data-item timestamps updated.

T1(10)	T2(9)
	W(X)
R(X)-ALLOWED	

-
-

- **If a transaction Ti issues a write(X) operation −**
  - If TS(Ti) < R-timestamp(X)
    - Operation rejected.

T1(9)	T2(10)
	R(X)
W(X)-REJECT	

  - If TS(Ti) < W-timestamp(X)
    - Operation rejected and Ti rolled back.
  - Otherwise, operation executed.

T1(9)	T2(10)
	W(X)
W(X)-REJECT	

Ex:

T1(9)	T2(10)	T1(11)
R(A)		
	W(A)	
W(A)		
		R(A)
		W(A)

THOMAS WRITE RULE:

IT IGNORES OUTDATED WRITE OPERATIONS

It improves the Basic Timestamp Ordering Algorithm.

If transaction Ti wants to execute the WRITE operation on X:

The basic Thomas write rules are as follows:

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

   Example:

T1(9)	**T2(10)**
	R(X)
W(X)-REJECT	

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

T1(9)	**T2(10)**
	W(X)
	COMMIT
W(X)-OUTDATED	
COMMIT	

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

Thomas write rule ensures View Serializability but not conflict serializability.

Validation Based Protocol/ Optimistic Concurrency Control Technique:

Validation based protocols are also known as optimistic concurrency control technique or Certification based protocols

It is called optimistic concurrency control protocols because it hopes that all the operations of transaction are correct.

It is used to validate/ check the update operation (read/ write operation) on a data item by a Transaction.

In the validation based protocol, the transaction is executed in the following three phases:

1. **Read phase:** In this phase, the transaction T reads committed values of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.

2. **Validation phase:** In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability and everything is correct.
3. **Write phase:** If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back and restarted.

Here each phase has the following different timestamps:

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

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

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

- This protocol is used to determine the time stamp for the transaction for serialization using the time stamp of the validation phase, as it is the actual phase which determines if the transaction will commit or rollback.
- Hence TS(T) = validation(T).
- The serializability is determined during the validation process. It can't be decided in advance.
- While executing the transaction, it ensures a greater degree of concurrency and also less number of conflicts.
- Thus it contains transactions which have less number of rollbacks.

MULTIPLE GRANULARITY

Locking Granularity: It is the size of the data item that can be locked.

There are 2 types of locking granularity:

1. Coarse granularity: It refers to large data item sizes
   Ex: Entire relation or Table
   Adv: Low locking overhead
   Lim: Low concurrency

STUDENT

HTNO	NAME	AGE	BRANCH
501			
502			
502			

2. Fine granularity: It refers to small data item sizes
   Ex: Tuple or attribute of a relation
   Adv: High locking overhead
   Lim: High concurrency

STUDENT

HTNO	NAME	AGE	BRANCH
501			
502			
502			

Multiple Granularity: It allows the data items of any size to be locked.

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

Example: Consider a tree which has four levels of nodes.

- The first level or higher level shows the entire database.
- The second level represents a node of type area. The higher level database consists of exactly these areas.
- The area consists of children nodes which are known as files. No file can be present in more than one area.
- Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.
- Hence, the levels of the tree starting from the top level are as follows:
    1. Database
    2. Area
    3. File
    4. Record

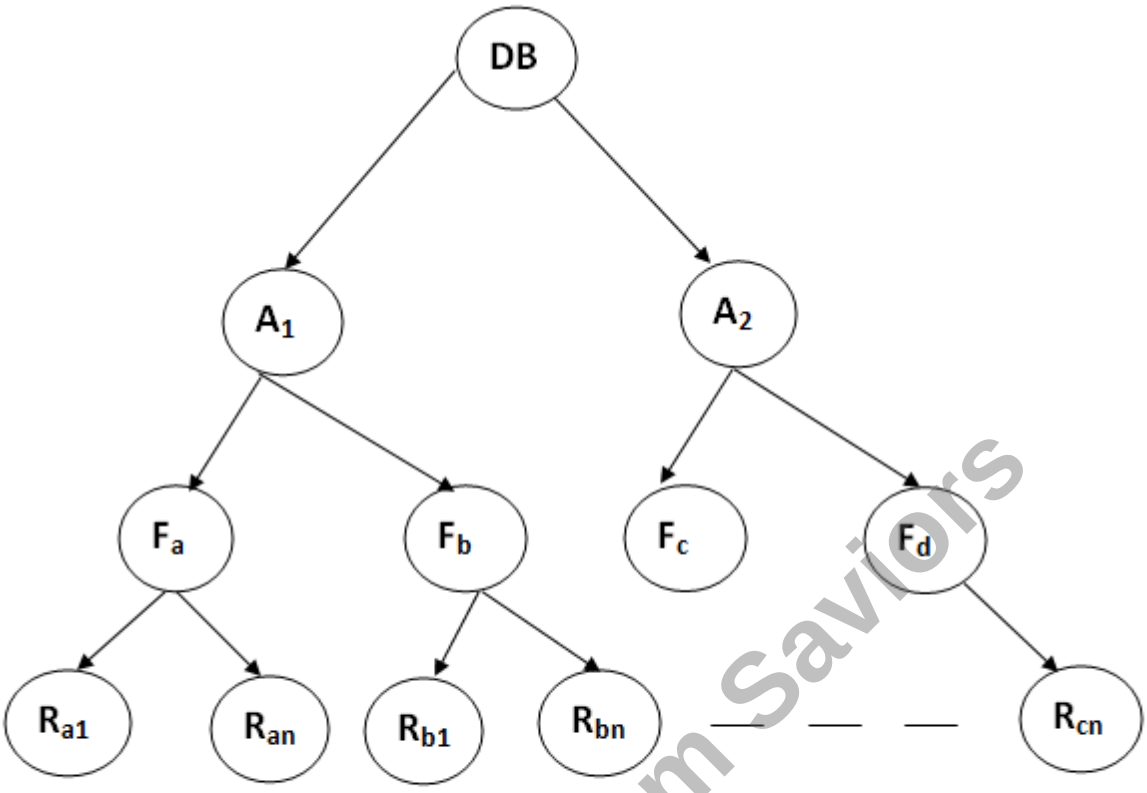
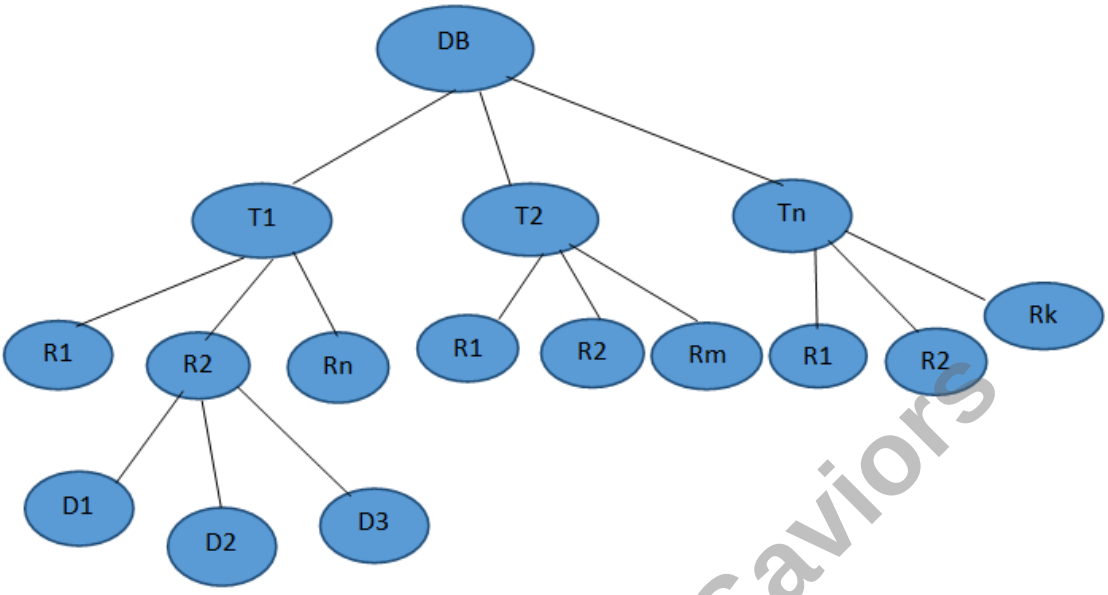


Figure: Multi Granularity tree Hierarchy

In this example, the highest level shows the entire database. The levels below are file, record, and fields.



Example: 1. A College Database.

2. It contains a set of tables like student, faculty, course etc.
3. Each table contains a set of records or rows.
4. Each record contains a set of values.

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

There are five lock modes with multiple granularity:

Locks: Shared lock, exclusive lock, intentionally shared lock (IS), intentional exclusive lock (IX), shared & Intentional executive (SIX).

Shared lock: Shared locks support read integrity. There are some circumstances for this lock:

1. A request for a shared lock must wait if any parallel task owns an exclusive lock on the resource.
2. Request for an exclusive lock must wait if other tasks paralleled individually shared locks on this resource.
3. A new request for a shared lock must wait if another task is waiting for an exclusive lock on a resource that already has a shared lock.

Exclusive lock: Exclusive locks support write integrity. They can be owned by only one transaction at a time.

1. Any transaction that requires an exclusive lock must wait if another task currently owns an exclusive lock or a shared lock against the requested resource.

	SHARED(S)	EXCLUSIVE(X)
SHARED(S)	YES	NO
EXCLUSIVE(X)	NO	NO

Intention Mode Locks

Intention-shared (IS): Intention-shared (IS) indicates that one or more shared locks will be requested on some descendant node(s) i.e. It contains explicit locking at a lower level of the tree but only with shared locks.

Intention-Exclusive (IX): Intention-exclusive (IX) indicates that one or more exclusive locks will be requested on some descendant node(s) i.e. It contains explicit locking at a lower level with exclusive or shared locks.

Shared & Intention-Exclusive (SIX): Shared-intention-exclusive (SIX) indicates that the current node is locked in shared mode but that one or more exclusive locks will be requested on some descendant node(s). i.e. In this lock, the node is locked in shared mode, and some node is locked in exclusive mode by the same transaction.

Compatibility Matrix with Intention Lock Modes: The below table describes the compatibility matrix for these lock modes:

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

It uses the intention lock modes to ensure serializability.

MULTIPLE GRANULARITY LOCKING PROTOCOL RULES:

The multiple granularity locking (MGL) protocol consists of the following rules i.e. It requires that if a transaction attempts to lock a node, then that node must follow these rules:

1. Transaction T1 should follow the lock-compatibility matrix.
2. The root of the tree must be locked first, in any mode.

3. A node N can be locked by a transaction T in S or IS mode only if the parent node N is already locked by transaction T in either IS or IX mode.

4. A node N can be locked by a transaction T in X, IX, or SIX mode only if the parent of node N is already locked by transaction T in either IX or SIX mode.

5. A transaction T can lock a node only if it has not unlocked any node (to enforce the 2PL protocol).

6. A transaction T can unlock a node, N, only if none of the children of node N are currently locked by T.

NOTE:

1. Rule 1 simply states that conflicting locks cannot be granted. Rules 2, 3, and 4 state the conditions when a transaction may lock a given node in any of the lock modes. Rules 5 and 6 of the MGL protocol enforce 2PL rules to produce serializable schedules.

- If transaction T1 wants to read a record R_{a9} in file F_a, then transaction T1 needs to lock the database, area A_1 and file F_a in IS mode. Finally, it needs to lock R_{a2} in S mode.
- If transaction T2 wants to modify record R_{a9} in file F_a, then it can do so after locking the database, area A_1 and file F_a in IX mode. Finally, it needs to lock the R_{a9} in X mode.
- If transaction T3 wants to read all the records in file F_a, then transaction T3 needs to lock the database, and area A in IS mode. At last, it needs to lock F_a in S mode.
- If transaction T4 wants to read the entire database, then T4 needs to lock the database in S mode.

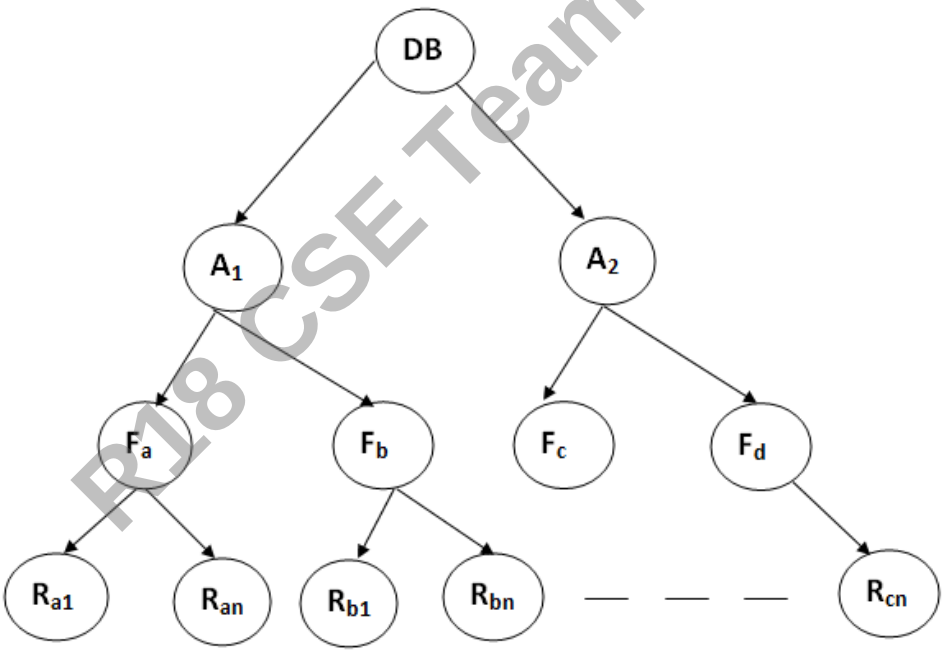


Figure: Multi Granularity tree Hierarchy

CASE STUDY:

1. T1 wants to read Rc2
2. T2 wants to write Rc2
3. T3 wants to modify Rc3
4. T4 wants to read all the records of Fd
5. T5 wants to read entire database.

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items.

Transactions are made of various operations, which are atomic in nature.

But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following −

- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −

- **Log Based Recovery**: Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- **Shadow paging**: Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

UNIT-5

LOG BASED RECOVERY

It is used to recover from a system crash or transaction failure using a log.

It is the most commonly used structure for recording database modifications.

- The log is a small file that contains a sequence of records where the records depict the operations performed by Transaction.
- Log of each transaction is maintained in some stable storage (RAM/ shared memory) So that if any failure occurs, then it can be recovered from there.
- Before any operation is performed on the database, it will be recorded in the log.
- The process of storing the logs should be done before the actual transaction is applied in the database.

A log contains:

1. Transaction Identifier
2. Data item identifier
3. Old value
4. New value

   Example: $<T_i, X_j, V_1, V_2>$

   T_i: Transaction identifier used to uniquely identify a Transaction.

   X_j: Data item on which the operations are performed.

   V_1: Old value of data item

   V_2: New value of data item

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.

  <Tn, Start>
- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

  <Tn, City, 'Noida', 'Bangalore' >
- When the transaction is finished, then it writes another log to indicate the end of the transaction.

  <Tn, Commit>

There are two approaches to modify the database using log based recovery:

1. **Deferred database modification**
2. **Immediate database modification.**

1. Deferred database modification:

In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

A Deffered database modification is also called No Undo/ Redo method.

Undo: Restores the old values

Redo: Updates the new values

In this log contains a new value of the data item.

Example: A=3000, B=2000

The deferred modification technique occurs if the transaction does not modify the database until it has committed.

TRANSACTION	LOG	DATABASE
T1		A=3000, B=2000
R(A)	<T1,START>	
A=A-500		
W(A)	<T1,A,2500>	
R(B)		
B=B+500		
W(B)	<T1,B,2500>	
COMMIT	<T1,COMMIT>	A=2500,B=2500
SYSTEM CRASH		
		REDO
		A=2500,B=2500

TRANSACTION	LOG	DATABASE
T1		A=3000, B=2000
R(A)	<T1,START>	
A=A-500		
W(A)	<T1,A,2500>	
R(B)		
B=B+500		
W(B)	<T1,B,2500>	
SYSTEM CRASH		
COMMIT		A=3000, B =2000

2. Immediate database modification:

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

TRANSACTION	LOG	DATABASE
T1		A=3000, B=2000
R(A)	<T1,START>	
A=A-500		
W(A)	<T1,A,3000,2500>	A=2500, B=2000
R(B)		
B=B+500		

W(B)	<T1,B,2000,2500>	A=2500, B=2500
COMMIT	<T1, COMMIT>	A=2500,B=2500
SYSTEM CRASH		REDO
		A=2500,B=2500

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

Immediate database modification is called Undo/ Redo strategy

Q1: Consider the following log sequence of two transactions on a bank account, with initial balance 12000, that transfer 2000 to a mortgage payment and then apply a 5% interest.

1. T1 start

TRANSACTION	LOG	DATABASE
T1		A=3000, B=2000
R(A)	<T1,START>	
A=A-500		
W(A)	<T1,A,3000,2500>	A=2500, B=2000
R(B)		
B=B+500		
W(B)	<T1,B,2000,2500>	A=2500, B=2500
SYSTEM CRASH		UNDO
COMMIT		A=3000,B=2000

2. T1 B old=12000 new=10000

3. T1 M old=0 new=2000

4. T1 commit

5. T2 start

6. T2 B old=10000 new=10500

7. T2 commit

Suppose the database system crashes just before log record 7 is written. When the system is restarted, what is the recovery procedure?

Q2.

T1	**T2**	**T3**	**T4**
R(X)			
W(X)	R(X)		W(X)
	W(X)		COMMIT
	COMMIT		
		R(X)	
		W(X)	
CRASH	CRASH	CRASH	CRASH

UNDO & REDO:

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

Shadow paging in DBMS

This is the method where all the transactions are executed in the primary memory or the shadow copy of database.

Once all the transactions completely executed, it will be updated to the database.

Hence, if there is any failure in the middle of transaction, it will not be reflected in the database.

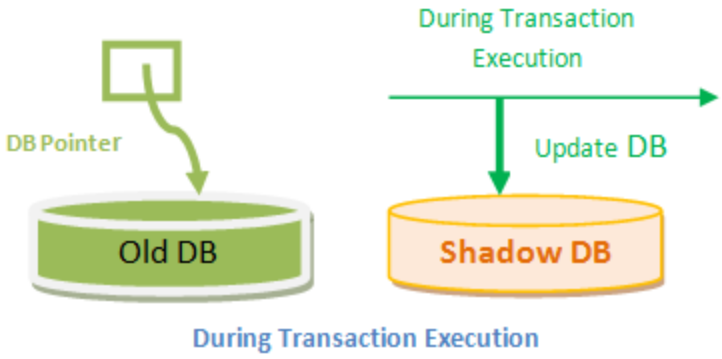
Database will be updated after all the transaction is complete.

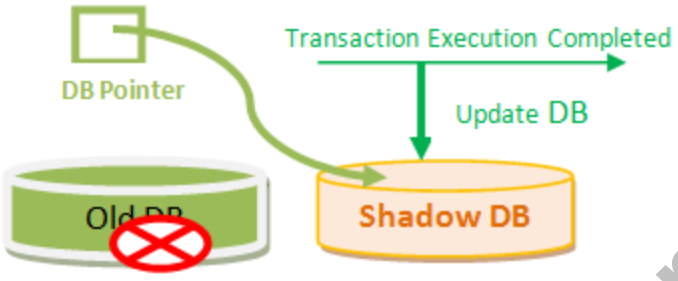
A database pointer will be always pointing to the consistent copy of the database, and copy of the database is used by transactions to update.

Once all the transactions are complete, the DB pointer is modified to point to new copy of DB, and old copy is deleted.

If there is any failure during the transaction, the pointer will be still pointing to old copy of database, and shadow database will be deleted.

If the transactions are complete then the pointer is changed to point to shadow DB, and old DB is deleted.





As we can see in above diagram, the DB pointer is always pointing to consistent and stable database.

This mechanism assumes that there will not be any disk failure and only one transaction executing at a time so that the shadow DB can hold the data for that transaction.

It is useful if the DB is comparatively small because shadow DB consumes same memory space as the actual DB.

Hence it is not efficient for huge DBs.

In addition, it cannot handle concurrent execution of transactions. It is suitable for one transaction at a time.

3. **Recovery with Concurrent Transactions**

When more than one transaction are being executed in parallel, the logs are interleaved.

At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering.

To ease this situation, most modern DBMS use the concept of 'checkpoints'.

Checkpoint

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system.

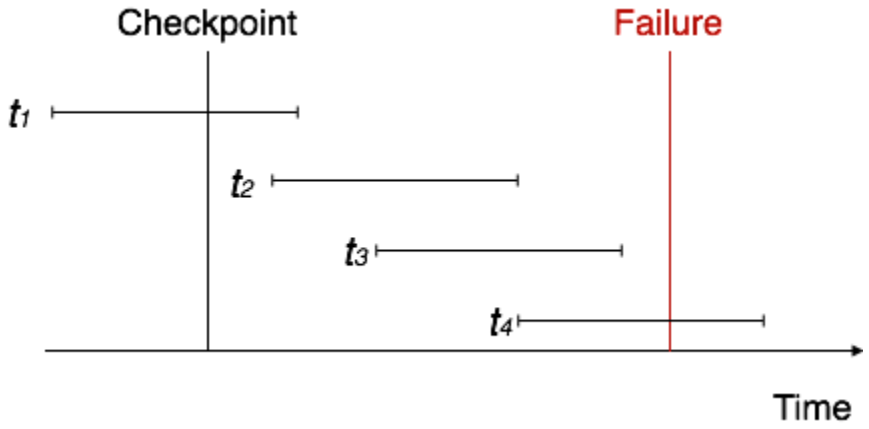
As time passes, the log file may grow too big to be handled at all.

Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk.

Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner −



- The recovery system reads log files from the end to start. It reads log files from T4 to T1.
- Recovery system maintains two lists, a redo-list, and an undo-list.
- The transaction is put into redo state if the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.
- **For example:** In the log file, transaction T2 and T3 will have <Tn, Start> and <Tn, Commit>. The T1 transaction will have only <Tn, commit> in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.
- The transaction is put into undo state if the recovery system sees a log with <Tn, Start> but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.
- **For example:** Transaction T4 will have <Tn, Start>. So T4 will be put into undo list since this transaction is not yet complete and failed amid.

Example:

<T0 start>

<T0, A, 0, 10>

<T0 commit>

<T1 start>

<T1, B, 0, 10>

<T2 start>

<T2, C, 0, 10>

<T2, C, 10, 20>

<checkpoint {T1, T2}>

<T3 start>

<T3, A, 10, 20>-------FAILURE

<T3, D, 0, 10>

<T3 commit>

Consider a simple checkpointing protocol and the following set of operations in the log.

```
(start, T4); (write, T4, y, 2, 3); (start, T1); (commit, T4); (write, T1, z, 5, 7);

(checkpoint);

(start, T2); (write, T2, x, 1, 9); (commit, T2); (start, T3); (write, T3, z, 7, 2);
```

If a crash happens now and the system tries to recover using both undo and redo operations, what are the contents of the undo list and the redo list

(A) Undo: T3, T1; Redo: T2
(B) Undo: T3, T1; Redo: T2, T4
(C) Undo: none; Redo: T2, T4, T3; T1
(D) Undo: T3, T1, T4; Redo: T2

T1	T2	T3	T4
			START
			W(Y,2,3)
START			
			COMMIT
W(Z,5,7)			
CHECKPOINT	CHECKPOINT	CHECKPOINT	CHECKPOINT
	START		
	W(X,1,9)		
	COMMIT		
		START	
		W(Z,7,2)	
CRASH	CRASH	CRASH	CRASH

TRIGGERS

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.

Trigger is a PL/SQL procedure stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).

A database definition (DDL) statement (CREATE, ALTER, or DROP).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

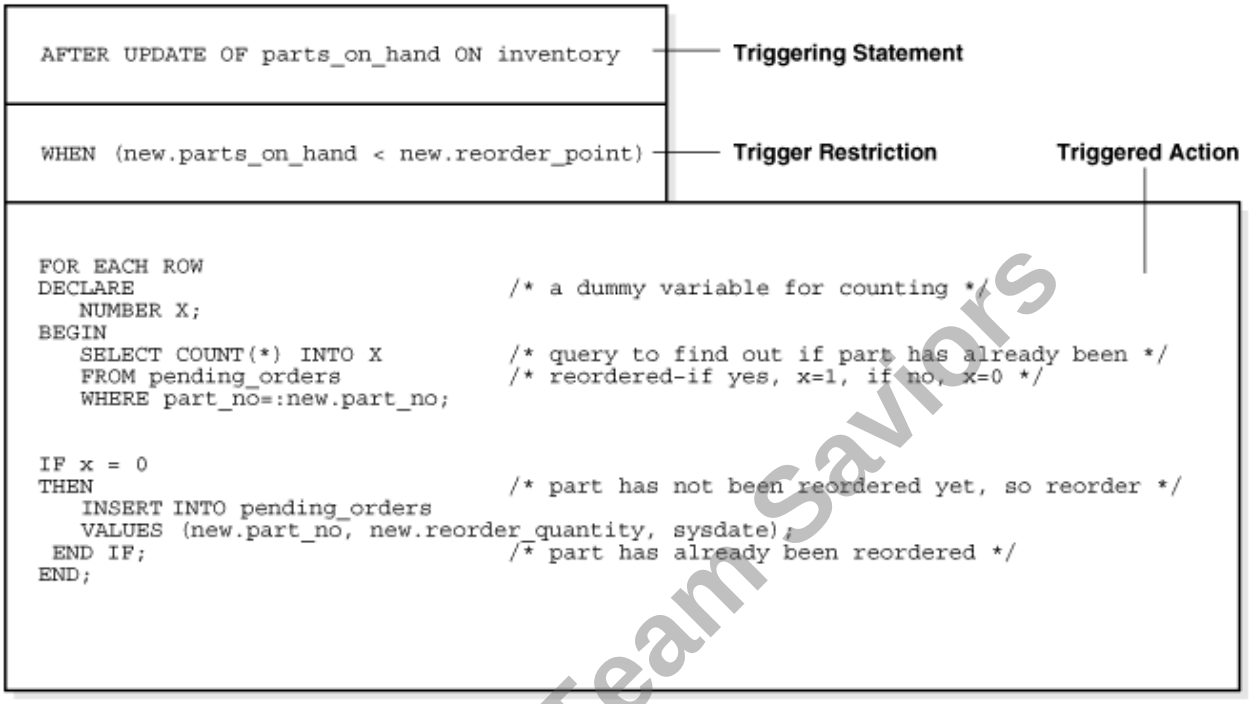
PL/SQL: Parts of a Trigger

Whenever a trigger is created, it contains the following three sequential parts:

- **Triggering Event or Statement:** The statements due to which a trigger occurs is called triggering event or statement. Such statements can be DDL statements, DML statements or any database operation, executing which gives rise to a trigger.
- **Trigger Restriction:** The condition or any limitation applied on the trigger is called trigger restriction. Thus, if such a condition is **TRUE** then trigger occurs otherwise it does not occur.
- **Trigger Action:** The body containing the executable statements that is to be executed when trigger occurs that is with the execution of Triggering statement and upon evaluation of Trigger restriction as **True** is called Trigger Action.

Example:

REORDER Trigger



Description of "Figure 22-3 The REORDER Trigger"

PL/SQL: Types of Triggers

Triggers can be classified into three categories:

1. Level Triggers
2. Event Triggers
3. Timing Triggers

which are further divided into different parts.

Level Triggers

There are 2 different types of level triggers, they are:

1. **ROW LEVEL TRIGGERS**
    - It fires for every record that got affected with the execution of DML statements like INSERT, UPDATE, DELETE etc.
    - It always use a `FOR EACH` ROW clause in a triggering statement.
2. **STATEMENT LEVEL TRIGGERS**
    - It fires once for each statement that is executed.

Event Triggers

There are 3 different types of event triggers, they are:

1. **DDL EVENT TRIGGER/ DATABASE LEVEL TRIGGERS**
    - It fires with the execution of every DDL statement(CREATE, ALTER, DROP, TRUNCATE).
2. **DML EVENT TRIGGER/ TABLE LEVEL TRIGGERS**
    - It fires with the execution of every DML statement(INSERT, UPDATE, DELETE).

3. **DATABASE EVENT TRIGGER**
    - It fires with the execution of every database operation which can be LOGON, LOGOFF, SHUTDOWN, SERVERERROR etc.

Timing Triggers

There are 2 different types of timing triggers, they are:

- **BEFORE TRIGGER**
    - It fires before executing DML statement.
    - Triggering statement may or may not executed depending upon the before condition block.
- **AFTER TRIGGER**
    - It fires after executing DML statement.

Creating a trigger:

Syntax for creating trigger:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

EXAMPLE:

```
Create or replace Trigger np before insert on account for each row
Begin
  IF :NEW.bal < 0  THEN
    DBMS_OUTPUT.PUT_LINE('BALANCE IS NEGATIVE..');
  END IF;
End;
/
```

Output:

Run SQL Command Line

```
SQL>start D://t.sql
Trigger created.

SQL>insert into account values(1,-100);
BALANCE IS NAGATIVE..
1 row created.
```

CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.

{BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.

{INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.

[OF col_name]: This specifies the column name that would be updated.

[ON table_name]: This specifies the name of the table associated with the trigger.

[REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.

[FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.

WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Example

using the CUSTOMERS table

```
Select * from customers;
```

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program creates a **row-level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values −

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
Trigger created.
```

The following points need to be considered here −

- OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
- The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.

Triggering a Trigger

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table −

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
```

When a record is created in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result −

```
Old salary:
New salary: 7500
Salary difference:
```

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The UPDATE statement will update an existing record in the table −

```
UPDATE customers
SET salary = salary + 500
WHERE id = 2;
```

When a record is updated in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result −

```
Old salary: 1500
New salary: 2000
Salary difference: 500
```

Operations in Triggers

We can perform many operations using triggers.

DROP A Trigger

```
    DROP TRIGGER trigger name;
```

Display A Trigger

The below code will display all the triggers that are present.

```
                                        SHOW TRIGGERS;
```

The below code will display all the triggers that are present in a particular database.

```
    SHOW TRIGGERS IN database_name;
```

PL / SQL Cursors

Cursor is a memory location which is used to run SQL commands.

When an SQL statement is processed, Oracle creates a memory area known as context area.

A cursor is a pointer to this context area.

It contains all information needed for processing the statement.

In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

1) Implicit Cursors: A Cursor declared by Oracle.
2) Explicit Cursors: A Cursor declared by User.

1) Implicit Cursors: These are declared after the execution of DML command.

The name of the cursor is SQL.

All the activities related to cursor like i) Opening the cursor ii) Processing the data in the cursor iii) closing the cursor
are done automatically.
Hence these cursors are called Implicit cursors.

Implicit Cursor Attributes:

There are four Implicit cursor attributes
1) SQL%ISOPEN
2) SQL%FOUND
3) SQL%NOTFOUND
4) SQL%ROWCOUNT

1) SQL%ISOPEN:

It is a boolean attribute. It always returns false. It is not used in programming as it always returns false.

2) SQL%FOUND:

It is a boolean attribute.
Returns TRUE -- if the SQL command effects the data.
Returns FALSE -- if the SQL commands do not effect the data.

3) SQL%NOTFOUND:

It is a boolean attribute
Returns TRUE -- if the SQL command do not effect the data.
Returns FALSE -- if the SQL command effects the data
Note: It is exactly negation to SQL%FOUND

4) SQL%ROWCOUNT:

Returns no of rows effected by the SQL command.

Using SQL%FOUND:

```
Begin
Update emp set sal=2000
where empno=1111;
end;
/
```

Output:

PL/SQL Procedure successfully completed.

By looking at the above message, we cannot know whether your update command is effecting the data or not.

To overcome this problem, we have SQL%FOUND attribute.
Have a look at this program

```
Begin
Update emp set sal=2000
where empno=1111;
if SQL%FOUND then
dbms_output.put_line('Update is successfull');
else
dbms_output.put_line('Update is failed');
end if;
end;
/
```

Output:

```
------------
Update is failed.

PL/SQL Procedure successfully completed.
```

Using SQL%NOTFOUND:

```
-----------------------------
```

SQL%NOTFOUND is exactly opposite to SQL%FOUND.

We rewrite the above program using SQL%NOTFOUND

```
Begin
Update emp set sal=2000
where empno=1111;
if SQL%NOTFOUND then
dbms_output.put_line('Update is failed');
else
dbms_output.put_line('Update is successful');
end if;
end;
/
```

Output:

```
------------
Update is failed.

PL/SQL Procedure successfully completed.
```

Using SQL%ROWCOUNT:

SQL%ROWCOUNT attribute is used to find the no of rows effected by SQL command.

```
begin
update emp set sal=2000
where deptno=10;
dbms_output.put_line(SQL%ROWCOUNT||' rows updated');
end;
/
```

Output:
---
3 rows updated.

Note: As a developer, we cannot control the implicit cursor.
We can you these implicit cursor attributes to know whether the command is effecting the data or not.

Explicit Cursors:

Explicit cursors are used to run select stmt which returns more than one row in a PL/SQL block

Steps to use Explicit cursors:

Step 1: Declare the cursor
Step 2: Open the cursor
Srep 3: Fetch the data from the cursor to the local variables
Step 4: close the cursor

Syntax of the above four steps:

Step 1: Declaring the cursor

cursor < cursor_name>
is < select stmt >;

Ex: cursor c1 is select * from emp;

step 2: Open the cursor : At this point data is loaded in cursor.

open < cursor_name >;

Ex: open c1;

step 3: Fetch the data(records) from the cursor to the local variables

fetch < cursor_name > into < var1 > , < var2> , ..., < varn >;;

Ex: fetch c1 into x,y,z,…;

step 4: close the cursor

close < cursor_name>;

Ex: close c1;

Explicit cursor attributes:

There are four explicit cursor attributes

1) %ISOPEN
2) %FOUND
3) %NOTFOUND
4) %ROWCOUNT

1) %ISOPEN:

It is a boolean attribute.
Returns TRUE -- if the cursor is open
Returns FALSE -- if the cursor is closed

2) %FOUND:

It is a boolean attribute
Returns TRUE -- if the fetch stmt is successfull
Returns FALSE -- if the fetch stmt fails

3) %NOTFOUND:

It is boolean attribute
Returns TRUE -- if the fetch stmt fails.
Returns FALSE -- if the fetch stmt is successfull

Note: 1) It is exactly opposite to %FOUND attribute
2) This attribute is used to break the loop of the fetch stmt.

4) %ROWCOUNT:

Returns no of rows fetched by the fetch stmt.

Example of Explicit cursor:
---

1. Write a PL/SQL block to display ename and sal of employees working in deptno 10

```
Declare
cursor c1
is select ename , sal from emp
where deptno=10;

l_ename emp.ename%type;
l_sal emp.sal%type;

begin
open c1;

loop
fetch c1 into l_ename , l_sal;
exit when c1%notfound;
dbms_output.put_line( l_ename||'....'||l_sal);
end loop;
close c1;

end;
/
```

Output:

CLARK 2450
KING 5000
MILLER 1300

Pl/SQL Proceudure successfully completed.

2: Write a PL/SQL procedure to display dname , loc from dept table

```
Declare
cursor c1
is select dname , loc from dept;

l_dname dept.dname%type;
l_loc dept.loc%type;

begin

open c1;
loop
fetch c1 into l_dname, l_loc;
exit when c1%notfound;
dbms_output.put_line(l_dname||'.....'||l_loc);

end loop;
close c1;

end;
/
```

Output:

Accounting New York
Research Dallas
Sales Chicago
Operations Boston

Pl/SQL Procedure successfully completed.

Ex:
---
3. Write a PL/SQL block which display ename and sal of employees working in deptno 10

```
Declare
cursor c1
is select ename , sal from emp
where deptno=10;
begin

for emp_rec in c1 loop
dbms_output.put_line(emp_rec.ename||'.....'||emp_rec.sal);
end loop;

end;
/
```

Output:

CLARK 2450
KING 5000
MILLER 1300

Pl/SQL Proceudure successfully completed.

Note: In the above program emp_rec in implicitly declared record variable, which is capable of storing one row of the cursor.

4.The following example uses a cursor to select the five highest paid employees from the `emp` table.

Input Table

```
SQL> SELECT ename, empno, sal FROM emp ORDER BY sal DESC;

ENAME           EMPNO       SAL
---------- --------- --------
KING             7839     5000
SCOTT            7788     3000
FORD             7902     3000
JONES            7566     2975
BLAKE            7698     2850
```

```
CLARK           7782     2450
ALLEN           7499     1600
TURNER          7844     1500
MILLER          7934     1300
WARD            7521     1250
MARTIN          7654     1250
ADAMS           7876     1100
JAMES           7900      950
SMITH           7369      800
```

PL/SQL Block

```
-- available online in file 'sample2'
DECLARE
   CURSOR c1 is
      SELECT ename, empno, sal FROM emp
         ORDER BY sal DESC;   -- start with highest paid employee
   my_ename VARCHAR2(10);
   my_empno NUMBER(4);
   my_sal   NUMBER(7,2);
BEGIN
   OPEN c1;
   FOR i IN 1..5 LOOP
      FETCH c1 INTO my_ename, my_empno, my_sal;
      EXIT WHEN c1%NOTFOUND;  /* in case the number requested */
                              /* is more than the total       */
                              /* number of employees          */
      INSERT INTO temp VALUES (my_sal, my_empno, my_ename);
      COMMIT;
   END LOOP;
   CLOSE c1;
END;
```

Output Table

```
SQL> SELECT * FROM temp ORDER BY col1 DESC;

NUM_COL1 NUM_COL2  CHAR_COL
-------- --------  --------
    5000      7839  KING
    3000      7902  FORD
    3000      7788  SCOTT
    2975      7566  JONES
    2850      7698  BLAKE
```

FOR CURSOR:

```
DECLARE
 Cursor C1 is Select * from emp;
r emp%rowtype;
BEGIN
Open C1;
Loop
Fetch C1 into r;
exit when c1%notfound;
dbms_output.put_line(r.ename||' '||r.sal);
end loop;
close C1;
end;
/
```

FOR LOOP CURSOR:

Cursor For loops:

It is shortcut way of writing explicit cursors.
When we use cursor for loops , following steps are not required.
1) Open the cursor
2) Fetch stmt
3) exit when condition
4) closing the cursor
5) declaring the local variables

```
DECLARE
 Cursor C1 is Select * from emp;
```

```
BEGIN
for r in C1;
loop
dbms_output.put_line(r.ename||' '||r.sal);
end loop;
end;
/
```


QUERY OPTIMIZATION

QUERY PROCESSING:

Query Processing would mean the entire process or activity which involves query translation into low level instructions, query optimization to save resources, cost estimation or evaluation of query, and extraction of data from the database.

Goal: To find an efficient Query Execution Plan for a given SQL query which would minimize the cost considerably, especially time.

Cost Factors: Disk accesses [which typically consumes time], read/write operations [which typically needs resources such as memory/RAM].

The major steps involved in query processing are depicted in the figure below;

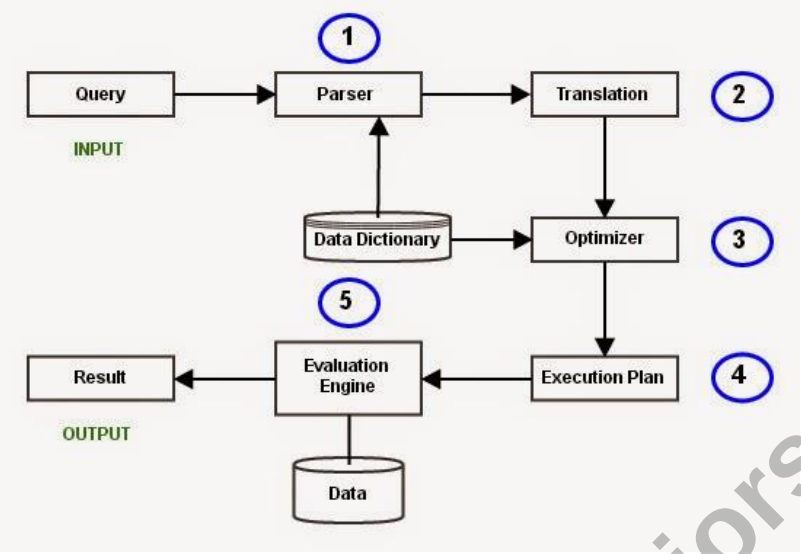


Figure 1 - Steps in Database Query Processing

Let us discuss the whole process with an example. Let us consider the following two relations as the example tables for our discussion;

Employee(Eno, Ename, Phone)

Proj_Assigned(Eno, Proj_No, Role, DOP)

where,

Eno is Employee number,

Ename is Employee name,

Proj_No is Project Number in which an employee is assigned,

Role is the role of an employee in a project,

DOP is duration of the project in months.

With this information, let us write a query to

find the list of all employees who are working in a project which is more than 10 months old.

```
SELECT Ename
FROM Employee, Proj_Assigned
WHERE Employee.Eno = Proj_Assigned.Eno AND DOP > 10;
```

Input:

A query written in SQL is given as input to the query processor. For our case, let us consider the SQL query written above.

Step 1: Parsing

In this step, the parser of the query processor module checks the syntax of the query, the user's privileges to execute the query, the table names and attribute names, etc. The correct table names, attribute names and the privilege of the users can be taken from the system catalog (data dictionary).

Step 2: Translation

If we have written a valid query, then it is converted from high level language SQL to low level instruction in Relational Algebra.

For example, our SQL query can be converted into a Relational Algebra equivalent as follows;

$$\pi_{Ename}(\sigma_{DOP>10 \wedge Employee.Eno=Proj\_Assigned.Eno}(Employee \times Proj\_Assigned))$$

Step 3: Optimizer

Optimizer uses the statistical data stored as part of data dictionary. The statistical data are information about the size of the table, the length of records, the indexes created on the table, etc. Optimizer also checks for the conditions and conditional attributes which are parts of the query.

Step 4: Execution Plan

A query can be expressed in many ways. The query processor module, at this stage, using the information collected in step 3 to find different relational algebra expressions that are equivalent and return the result of the one which we have written already.

For our example, the query written in Relational algebra can also be written as the one given below;

$$\pi_{Ename}(Employee \bowtie_{Eno} (\sigma_{DOP>10}(Proj\_Assigned)))$$

So far, we have got two execution plans. Only condition is that both plans should give the same result.

Step 5: Evaluation

Though we got many execution plans constructed through statistical data, though they return same result (obvious), they differ in terms of Time consumption to execute the query, or the Space required executing the query. Hence, it is mandatory choose one plan which obviously consumes less cost.

At this stage, we choose one execution plan of the several we have developed. This Execution plan accesses data from the database to give the final result.

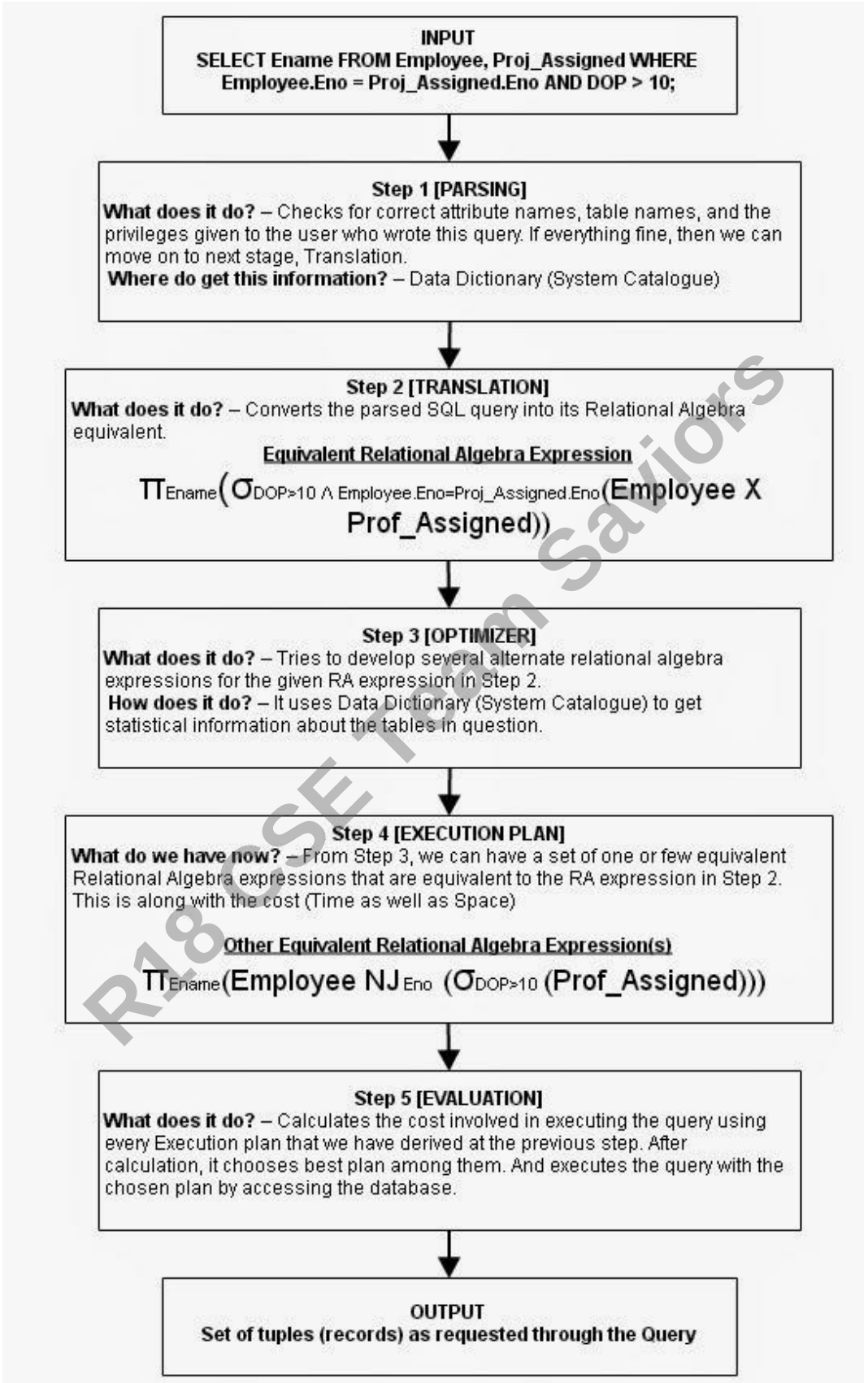
In our example, the second plan may be good. In the first plan, we join two relations (costly operation) then apply the condition (conditions are considered as filters) on the joined relation. This consumes more time as well as space.

In the second plan, we filter one of the tables (Proj_Assigned) and the result is joined with the Employee table. This join may need to compare less number of records. Hence, the second plan is the best (with the information known, not always).

Output:

The final result is shown to the user.

The overall information discussed above are depicted in Figure below;



SQL Optimization

Sql Statements are used to retrieve data from the database.

We can get same results by writing different sql queries.

But use of the best query is important when performance is considered. So we need to sql query tuning based on the requirement.

Here is the list of queries which we use requlary and how these sql queries can be optimized for better performance.

SQL Tuning/SQL Optimization Techniques:

1) The sql query becomes faster if you use the actual columns names in SELECT statement instead of than '*'.

For Example: Write the query as

```
SELECT id, first_name, last_name, age, subject FROM student_details;
```

Instead of:

```
SELECT * FROM student_details;
```

2) HAVING clause is used to filter the rows after all the rows are selected. It is just like a filter. Do not use HAVING clause for any other purposes.

For Example: Write the query as

```
SELECT subject, count(subject)
FROM student_details
WHERE subject != 'Science'
```

```
AND subject != 'Maths'
GROUP BY subject;
```

Instead of:

```
SELECT subject, count(subject)
FROM student_details
GROUP BY subject
HAVING subject!= 'Vancouver' AND subject!= 'Toronto';
```

3) Sometimes you may have more than one subqueries in your main query. Try to minimize the number of subquery block in your query.
For Example: Write the query as

```
SELECT name
FROM employee
WHERE (salary, age ) = (SELECT MAX (salary), MAX (age)
FROM employee_details)
AND dept = 'Electronics';
```

Instead of:

```
SELECT name
FROM employee
WHERE salary = (SELECT MAX(salary) FROM employee_details)
AND age = (SELECT MAX(age) FROM employee_details)
AND emp_dept = 'Electronics';
```

4) Use operator EXISTS, IN and table joins appropriately in your query.
a) Usually IN has the slowest performance.
b) IN is efficient when most of the filter criteria is in the sub-query.
c) EXISTS is efficient when most of the filter criteria is in the main query.
For Example: Write the query as

```
Select * from Sailor S
where EXISTS (select * from Reserves R
where S.sid=R.sid)
```

Instead of:

```
Select * from Sailor S
where sid IN
(select sid from Reserves)
```

5) Use EXISTS instead of DISTINCT when using joins which involves tables having one-to-many relationship.
For Example: Write the query as

```
SELECT d.dept_id, d.dept
FROM dept d
WHERE EXISTS ( SELECT 'X' FROM employee e WHERE e.dept =
d.dept);
```

Instead of:

```
SELECT DISTINCT d.dept_id, d.dept
FROM dept d,employee e
WHERE e.dept = e.dept;
```

6) Try to use UNION ALL in place of UNION.
For Example: Write the query as

```
SELECT id, first_name
FROM student_details_class10
UNION ALL
SELECT id, first_name
FROM sports_team;
```

Instead of:

```
SELECT id, first_name, subject
FROM student_details_class10
UNION
SELECT id, first_name
FROM sports_team;
```

7) Be careful while using conditions in WHERE clause.
For Example: Write the query as

Write the query as

```
SELECT id, first_name, age
FROM student_details
WHERE first_name LIKE 'Chan%';
```

Instead of:

```
SELECT id, first_name, age
FROM student_details
WHERE SUBSTR(first_name,1,3) = 'Cha';
```

8) Use DECODE to avoid the scanning of same rows or joining the same table repetitively. DECODE can also be made used in place of GROUP BY or ORDER BY clause.

For Example: Write the query as

```
SELECT id FROM employee
WHERE name LIKE 'Ramesh%'
and location = 'Bangalore';
```

Instead of:

```
SELECT DECODE(location,'Bangalore',id,NULL) id FROM employee
WHERE name LIKE 'Ramesh%';
```

There are two methods of query optimization.

1. Cost based Optimization (Physical)
This is based on the cost of the query. The query can use different paths based on indexes, constraints, sorting methods etc. This method mainly uses the statistics like record size, number of records, number of records per block, number of blocks, table size, whether whole table fits in a block, organization of tables, uniqueness of column values, size of columns etc.

Suppose, we have series of table joined in a query.

T1 ∞ T2 ∞ T3 ∞ T4∞ T5 ∞ T6

For above query we can have any order of evaluation. We can start taking any two tables in any order and start evaluating the query. Ideally, we can have join combinations in (2(n-1))! / (n-1)! ways. For example, suppose we have 5 tables involved in join, then we can have 8! / 4! = 1680 combinations. But when query optimizer runs, it does not evaluate in all these ways always. It uses dynamic programming where it generates the costs for join orders of any combination of tables. It is calculated and generated only once. This least cost for all the table combination is then stored in the database and is used for future use. i.e.; say we have a set of tables, T = { T1 , T2 , T3 .. Tn}, then it generates least cost combination for all the tables and stores it.

Dynamic Programming

As we learnt above, the least cost for the joins of any combination of table is generated here. These values are stored in the database and when those tables are used in the query, this combination is selected for evaluating the query.

While generating the cost, it follows below steps :

Suppose we have set of tables, T = {T1 , T2 , T3 .. Tn}, in a DB. It picks the first table, and computes cost for joining with rest of the tables in set T. It calculates cost for each of the tables and then chooses the best cost. It continues doing the same with rest of the tables in set T. It will generate 2n – 1 cases and it selects the lowest cost and stores it. When a query uses those tables, it checks for the costs here and that combination is used to evaluate the query. This is called dynamic programming.

In this method, time required to find optimized query is in the order of 3n, where n is the number of tables. Suppose we have 5 tables, then time required in 35 = 243, which is lesser than finding all the combination of tables and then deciding the best combination (1680). Also, the space required for computing and storing the cost is also less and is in the order of 2n. In above example, it is 25 = 32.

Left Deep Trees
This is another method of determining the cost of the joins. Here, the tables and joins are represented in the form of trees. The joins always form the root of the tree and table is kept at the right side of the root. LHS of the root always point to the next join. Hence it gets deeper and deeper on LHS. Hence it is called as left deep tree.

Here instead of calculating the best join cost for set of tables, best join cost for joining with each table is calculated. In this method, time required to find optimized query is in the order of n2n, where n is the number of tables. Suppose we have 5 tables, then time required in 5*25 =160, which is lesser than dynamic programming. Also, the space required for computing storing the cost is also less and is in the order of 2n. In above example, it is 25 = 32, same as dynamic programming.

Interesting Sort Orders
This method is an enhancement to dynamic programming. Here, while calculating the best join order costs, it also considers the sorted tables. It assumes, calculating the join orders on sorted tables would be efficient. i.e.; suppose we have unsorted tables T1 , T2 , T3 .. Tn and we have join on these tables.

(T1 ∞T2)∞ T3 ∞… ∞ Tn

This method uses hash join or merge join method to calculate the cost. Hash Join will simply join the tables. We get sorted output in merge join method, but it is costlier than hash join. Even though merge join is costlier at this stage, when it moves to join with third table, the join will have less effort to sort the tables. This is because first table is the sorted result of first two tables. Hence it will reduce the total cost of the query.

But the number of tables involved in the join would be relatively less and this cost/space difference will be hardly noticeable.

All these cost based optimizations are expensive and are suitable for large number of data. There is another method of optimization called heuristic optimization, which is better compared to cost based optimization.

2. Heuristic Optimization (Logical)
This method is also known as rule based optimization. This is based on the equivalence rule on relational expressions; hence the number of combination of queries get reduces here. Hence the cost of the query too reduces.

This method creates relational tree for the given query based on the equivalence rules. These equivalence rules by providing an alternative way of writing and evaluating the query, gives the better path to evaluate the query. This rule need not

be true in all cases. It needs to be examined after applying those rules. The most important set of rules followed in this method is listed below:

Perform all the selection operation as early as possible in the query. This should be first and foremost set of actions on the tables in the query. By performing the selection operation, we can reduce the number of records involved in the query, rather than using the whole tables throughout the query.
Suppose we have a query to retrieve the students with age 18 and studying in class DESIGN_01. We can get all the student details from STUDENT table, and class details from CLASS table. We can write this query in two different ways.

Here both the queries will return same result. But when we observe them closely we can see that first query will join the two tables first and then applies the filters. That means, it traverses whole table to join, hence the number of records involved is more. But he second query, applies the filters on each table first. This reduces the number of records on each table (in class table, the number of record reduces to one in this case!). Then it joins these intermediary tables. Hence the cost in this case is comparatively less.

Instead of writing query the optimizer creates relational algebra and tree for above case.

Perform all the projection as early as possible in the query. This is similar to selection but will reduce the number of columns in the query.
Suppose for example, we have to select only student name, address and class name of students with age 18 from STUDENT and CLASS tables.

Here again, both the queries look alike, results alike. But when we compare the number of records and attributes involved at each stage, second query uses less records and hence more efficient.

Next step is to perform most restrictive joins and selection operations. When we say most restrictive joins and selection means, select those set of tables and views which will result in comparatively less number of records. Any query will have better performance when tables with few records are joined. Hence throughout heuristic method of optimization, the rules are formed to get less number of records at each stage, so that query performance is better. So is the case here too.
Suppose we have STUDENT, CLASS and TEACHER tables. Any student can attend only one class in an academic year and only one teacher takes a class. But a class

can have more than 50 students. Now we have to retrieve STUDENT_NAME, ADDRESS, AGE, CLASS_NAME and TEACHER_NAME of each student in a school.

∏STD_NAME, ADDRESS, AGE, CLASS_NAME, TEACHER_NAME ((STUDENT ∞ CLASS_ID CLASS)∞ TECH_IDTEACHER)

Not So efficient

∏STD_NAME, ADDRESS, AGE, CLASS_NAME, TEACHER_NAME (STUDENT ∞ CLASS_ID (CLASS∞ TECH_IDTEACHER))

Efficient

In the first query, it tries to select the records of students from each class. This will result in a very huge intermediary table. This table is then joined with another small table. Hence the traversing of number of records is also more. But in the second query, CLASS and TEACHER are joined first, which has one to one relation here. Hence the number of resulting record is STUDENT table give the final result. Hence this second method is more efficient.

Sometimes we can combine above heuristic steps with cost based optimization technique to get better results.
All these methods need not be always true. It also depends on the table size, column size, type of selection, projection, join sort, constraints, indexes, statistics etc. Above optimization describes the best way of optimizing the queries.