Database Management Systems

Database Management Systems

INDEX

1 DATABASE DESIGN

1.1 Introduction

A database is an organized collection of structured data or related data.

1.2 Limitations of File System

- The physical details of the data to access the database are managed by the user.
- File system can be used to manage small database.
- When database is large, the operating system fails to control the concurrency.
- Only single user can access the whole data of the file system at a time.

1.3 Integrity Constraints Of RDBMS

- According to codd, data in database file must be stored in tabular format (set of row's and column's).
- According to codd, no two row's/records of the table should be equal.

1.3.1 Arity

Number of attributes of database table.

1.3.2 Cordinality

Number of records of database table.

1.4 Keys in database

- **Condidate Key:** A minimal set of attributes that differentiate records/row's of DB table uniquely.
- **Primary Key :** One of the candidate key whose field value cannot be null.
- **Simple Candidate Key:** A candidate key with only one attribute.
- **Compound Candidate Key:** A candidate key with atleast two attributes.
- **Overlapped Candidate Key:** Two or more candidate key with some common attribute.
- **Prime attribute:** The attribute that belongs to some candidate keys of a relation.
- **Non-prime attribute:** The attribute that does not belongs to any of the candidate keys of the relation.

Example :

Consider a relation R (ABCDE)

1. Assume candidate key : {AB, BC}

2. The above candidate keys are compound and overlapped.
3. Prime attribute {A, B, C}
4. Non-prime attribute : {D, E}

1.4.1 Difference between Primary key and Alternative key

Primary Key	Alternative Key
1. Any one candidate key whose field value can not be null. 2. Atmost one primary key allowed for any DB table.	1. All candidate key of relation except primary key, whose field value can be null. 2. Many (o or more) alternative keys are allowed for DB table.

Note:
For any RDBMS table there must be atleast one candidate key, whose field value can not be null.

(Unique + Not Null) ≠Primary key

1.4.2 Super key

The set of attributes which can differentiate records/tuples uniquely or super set of candidate key.

Example 1: R (ABCD) with CK = {A}

∴ Super key = CK · [Any subset of other attributes (BCD)]

= A · [2^3] = 8 Super key.

Example 2 : Let R be the relational schema with n-attibutes, R (A_1, A_2, ... A_n) then number of superkeys possible:

(I) With (A_1) as candidate key : 2^{n-1}

(II) With {A_1, A_2} as candidate key : $2^{n-1} - 2^{n-2} + 2^{n-1}$

(III) With {A_1A_2, A_3A_4} as candidate key: $2^{n-2} - 2^{n-4} + 2^{n-2}$

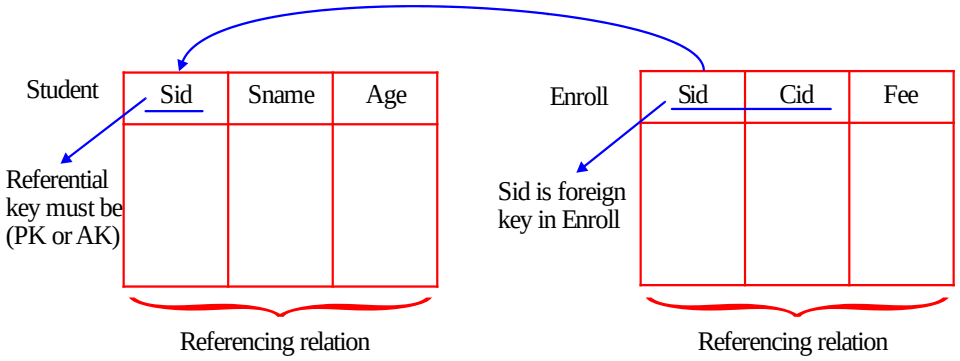
(IV) The maximum number of super keys possible when each attribute of R is candidate key : 2^{n-1}

(V) The minimum number of super key possible when all the attributes combined form single candidate key:
1 {A_1A_2 ..... A_n : candidate key}

1.5 Referential Integrity Constraints

1.5.1 Foreign key

Foreign key is a set of attributes that references primary key or alternative key of the same relation or other relation.



Referenced Relation

1. **Insertion :** No violation
2. **Deletion :** [May cause violation]
   (a) On delete no action : Means if it cause problem on delete then deletion is not allowed on table.
   (b) On delete cascade : If we want to delete primary key value from referenced table then it will delete that value from referencing table also.
   (c) On delete set null : If we want to delete primary key value from referenced table then it will try to set the null values in place of that value in referencing table.

Note:
If foreign key field is not null attribute then "On delete set null" is same as" on delete no action."

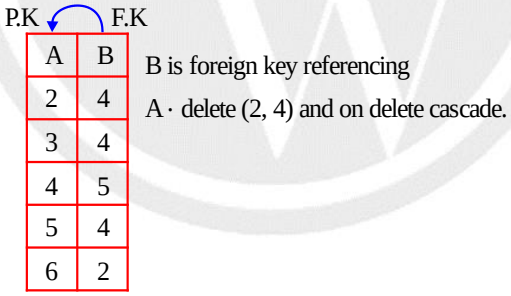
3. **Updation :** [May cause violation]
   (a) On update no action
   (b) On update cascade
   (c) On update set null

Referencing Relation

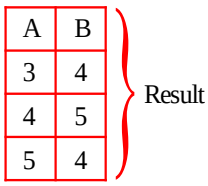
1. **Insertion :** [May cause violation]
2. **Deletion :** No violation
3. **Updation :** [May cause violation]

Note
If integrity violation occurs because of insertion or updation in referencing table then restrict insertion and updation.

Example:



So, If we delete (2, 4) then PK "2". gets deleted from the table and all the tuples in which B is referencing PK.2" also gets deleted.



1.6 Database Design Goals

1. 0% redundancy
2. Loss-less join
3. Dependency preservation

1.6.1 Functional Dependency

Let R be the relational shcema and x, y be the non-empty set of attributes. Consider t_1, t_2 are any tuples of relation then $x \to y$ (y functionally determined by x):

If $\forall t_1, t_2\ t_1 \cdot x = t_2 \cdot x$ then $t_1 \cdot y = t_2 \cdot y$

1.6.2 Types of Functional Dependency

1. Trivial Functional Dependency

Consider relational schema R(ABCD)

A FD $x \to y$ is trivial FD only if $x \supseteq y$.

Example :

$AB \to A$

$A \to A$

$B \to B$

2. Non-trivial Functional Dependency

Consider relational schema R(ABCD)

A FD $x \to y$ is non-trivial only if

$x \cap y = \phi$ means no common attributes between x and y attribute sets.

Example :

$A \to B$

$AB \to CD$

3. Semi-Non-Trivial Functional Dependency

A combination of both trivial FD and non-trivial FD.

Example :

$A \to AB \equiv \{A \to A, A \to B\}$

1.6.3 Attribute Closure (X^+)

The set of all possible attributes determined by x.

Example :

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

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

1.6.4 Armstrong's Axioms

1. **Reflexive :** If $x \supseteq y$ then $x \to y$ or $x \to x$
2. **Transitivity :** If $x \to y$ and $y \to z$ then $x \to z$
3. **Augmentation :** If $x \to y$ then $xz \to yz$
4. **Splitting :** If $x \to yz$ then $x \to y$, $x \to z$
5. **Union :** If $x \to y$ and $x \to z$ then $x \to yz$
6. **Pseudo transitivity :** If $x \to y$, $yw \to z$ then $xw \to z$

1.6.5 Finding Candidate Key [Minimal Super Key]

X is candidate key of R
If and only if

1. x must be super key of relation R
   $(x)^+$ = {determine all attributes of R}
2. No proper subset of 'x' is super key of relation R.
   $\forall\ y \subset x : (y)^+$ = {not determine all attributes of R}

- $x \rightarrow y$ is a non-trivial FD in R with y is a prime attribute then relation R has atleast two candidate key.

(x → y) : Non – trivial FD
↓
Prime-attribute

1.7 Membership of FD

A FD $x \rightarrow y$ is member (logically implied) of FD set F if and only if $(x)^+$ should determine 'y' in FD set F.

Example : Given FD Set F : { ….}
↓
x → y FD belongs to F or not ?
↓
$x^+ = \{\ldots y \ldots\}$
then x → y is member of F.

1.7.1 Testing of Two FD Sets whether they are equal or not

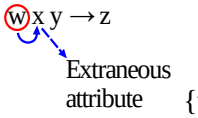
F and G FD sets equality test:
F and G FD sets logically equal if and only

1. **F cover's G :** All FD's of G set must be member's of F set.
   $F \supseteq G$
2. **G cover's F :** All FD's of F set must be member's of G set.
   $F \subseteq G$

F Cover G	G Cover F	Result
True	True	$F \equiv G$
True	False	$F \supset G$
False	True	$F \subset G$
False	False	F & G are not comparable

1.7.2 Minimal Cover or Canonical Cover

- Minimal Cover of given FD Set (F) is minimum possible FD's (Fm), which is logically equal to 'F' : (F = Fm)
- Remove extraneous attribute (useless attribute) from each determinant of FD set F.



$\{wxy \rightarrow z, w \rightarrow x\} \equiv \{wy \rightarrow z, w \rightarrow x\}$

- Every FD must be simple (RHS of any FD should have single attribute).

Example : F = {A → CD} then {A → C, A→ D}

- FD set must be non-redundant FD. (eliminating unnecessary FD's)

Example : F = {A → B, B → C, A→ C} then

F = {A → B, B→ C} because A → C is redundant.

Note:

Minimal Cover of FD Set (F) may not unique, but all minimal cover's logically equal.

$\therefore \quad \boxed{Fm_1 \equiv Fm_2 = F}$

1.8 Normalization

Normalization used to eliminate/reduce redundency in DB relations.

The normalization is especially meant to eliminate the following anomalies:

- Insertion anomaly
- Deletion anomaly
- Update anomaly
- Join anomaly

1.8.1 Normalization of DB table

Decompose relation into two or more sub-relations in-order to reduce or eliminate redundancy and DB anomalise.

1.8.2 Properties of Decomposition

- **Loss-less Join decomposition:** Relational Schema R decomposed into R1, R2, R3, ... Rn sub-relations.
- In general [R1 ⋈ R2 ⋈ R3 ..... Rn] ⊇ R

1. If [R_1 ⋈ R_2 ⋈ R_3.....⋈ R_n] ≡ R then loss-less join decomposition.
2. If [R_1 ⋈ R_2 ⋈ R_3 ..... ⋈ R_n] ⊃ R then lossy join decomposition.

Example :

Let R be the relational schema decomposed into R_1 and R_2.

Given decomposition is loss-less join if–

1. $R_1 \cup R_2 = R$ (all attributes covers)
2. $R_1 \cap R_2 \neq \phi$
3. $\underbrace{R_1 \cap R_2 \to R_1}_{R_1 \cap R_2 \text{ is SK of } R_1}$ or $\underbrace{R_1 \cap R_2 \to R_2}_{R_1 \cap R_2 \text{ is SK of } R_2}$

1.8.3 Dependency Preserving Decompostion

Relational Schema R with FD set F decomposed into R_1, R_2, ... R_n sub relations

Assume F_1, F_2, ... F_n FD sets of sub relations respectively.

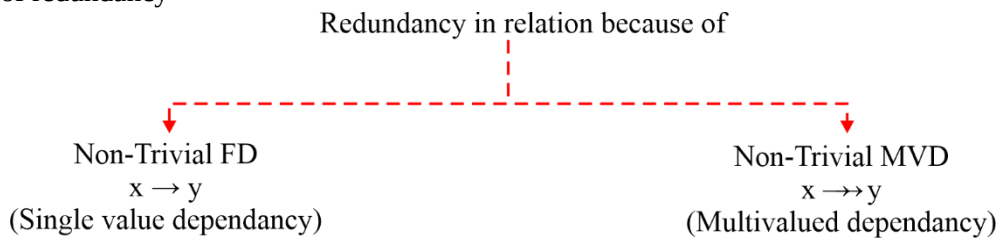
In general

[$F_1 \cup F_2 \cup$ ..... F_n] ⊆ F

1. If [$F_1 \cup F_2 \cup$ ..... F_n] ≡ F then dependency preserving decomposition.
2. If [$F_1 \cup F_2 \cup$ ..... F_n] ⊂ F then not dependency preserving decompostion.

1.9 Normal Forms

Used to find degree of redundancy



- BCNF relations have 0% redundancy over FD's where as 4NF relations have 0% redundancy over FD and MVD.
- To Eliminate redundancy over Non-Trivial FD, relation should decompose into BCNF but BCNF may not avoid the redundancy due to MVD.
- To eliminate redundancy over non-trivial MVD, relation should decompose into 4NF (4^{th} Normal Form).

1.9.1 First Normal Form

- Default NF of RDBMS relations.
- Relation (DB Table) R is in 1 NF if and only if no multivalued attributes in R. [Every attribute of R must be atomic/single valued].

Note:
Degree of redundancy is very high in 1NF relation.

Important Point 1

- $x \rightarrow y$ FD forms redundancy in relational schema R if and only if –
  (i) Non-Trivial FD
  and
  (ii) X is not super key.
- $x \rightarrow y$ FD not forms any redundancy in relation R if and only if–
  (i) Trivial FD [$x \supseteq y$] or Semi-trivial
  or
  (ii) x : super key

1.9.2 Second Normal Form (2NF)

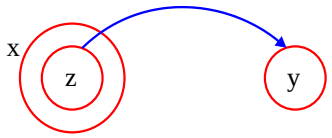
Relational Schema R is in 2 NF iff

- R should be 1NF
- R should not contain any partial dependency

Partial Dependency

Let R be the relational schema and x, y, z are non-empty set of attributes.

X : Any candidate key of R.
Y : Non-prime attribute
Z : Proper subset of candidate key



$\therefore$ $z \rightarrow y$ is said to be partial dependency iff–

- z is proper subset of candidate key
- y should be non-prime attribute.

Important Point 2

$$\underbrace{[\text{Proper subset of Candidate key}]}_{\text{Not super key}} \rightarrow [\text{Non-Prime attribute}]$$

The above FD forms redundancy in R.

1.9.3 Third Normal Form (3NF)

Relational schema R is in 3NF iff every non-trivial FD $x \rightarrow y$ in R with–

(i) x must be super key (SK)
or
(ii) y must be prime attribute.

$\therefore$ {SK $\rightarrow$ Prime/Non-Prime, (Not SK) $\rightarrow$ Prime attribute}

Important Point 3

3NF allow FD set:
[Proper subset of candidate key] $\rightarrow$ [Proper subset of other candidate key]
which forms redundancy in R.

1.9.4 Boyce codd Normal Form (BCNF)

Relational schema R is in BCNF iff every non-trivial FD "$x \rightarrow y$" with x must be super key.
$\therefore$ Prime/ Non-prime attributes must be determine by super key.

Important Point 4

If R is in 3NF but not BCNF then [Proper subset of candidate key] $\rightarrow$ [Proper subset of other candidate key] must exists in R.

Important Point 5

If relational shcema are with only simple candidate key then R always in:

I. 2NF
II. May or may not 3NF/BCNF

Reason : [Proper subset of candidate key] $\rightarrow$ [Non-prime attribute]
From the above statement, we can conclude that "partial dependency" not possible if all CK's are simple candidate key.

Important Point 6

If relational schema R with only prime attribute (No non-prime attribute in R) then R always in:

I. 3NF
II. May or may not BCNF

Important Point 7

If relational schema R with no non-trivial FD's then R always in BCNF.

1.9.5 Database design Table

DB design goal	1NF	2NF	3NF	BCNF
1. Loss-less jon decomposition	Yes	Yes	Yes	Yes
2. Dependency preserving decomposition	Yes	Yes	Yes	May not
3. O% redundency	No	No	No	Yes [Over FD]

- Every relation possible to decompose into 1NF, 2NF, 3NF, BCNF with loss-less join decomposition.
- Every relation possible to decompose into 1NF, 2NF, 3NF with Dependency preserving decomposition.
- Not every relation can decompose into BCNF and 4NF with dependency preserving decompostion.
- Most accurate normal form to design simple database is 3NF because every relation is possible to decompose into 3NF with loss-less join and dependency preserving.

❑❑❑

2 ER MODEL

2.1 Introduction

Entity relationship diagram used to represent diagrametic design (High level design) of DB.

2.2 Main components in ER Diagram

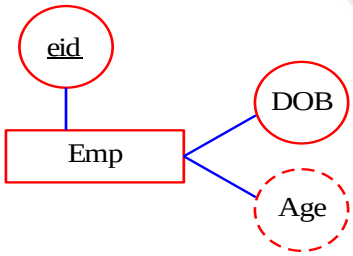
(i) Attributes
(ii) Entity sets
(iii) Relationship sets

(a) Entity sets

It is a set of entities of the same type denoted by a rectangular box in ER diagram. Entity can be identified by a list of attributes which are placed in ovals.

Represent by : 

Example :



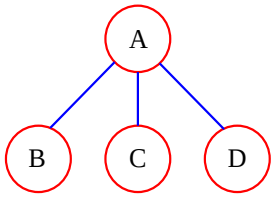
(b) Attributes

(i) Attribute : 

(ii) Key attribute : 

(iii) Derived attribute : 

(iv) Composite attribute: Attribute which can be represented as two or more attributes.

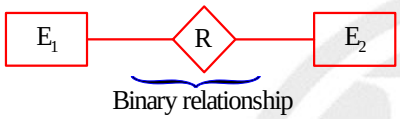


(v) Multivalued attribute: 

(c) Relationship set: It is used to relate two or more entity set.

Represented by : 

Example:



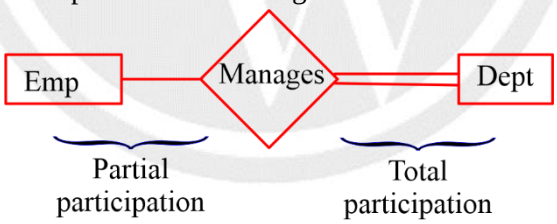
2.3 Participation

- If every entities of entity set are participated with relationship set then it is total participation (100% participation) otherwise it will be partial participation (< 100% participation)

Example :

Consider Emp and Dept entity set.

Manages relationship set such that each dept must have manager.

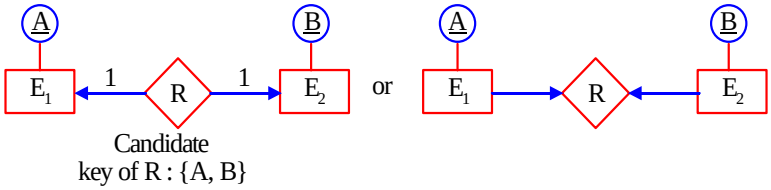


2.4 Mapping [Cardinality constraints of relationship set]

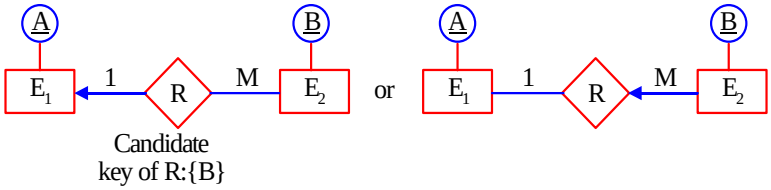
One mapping : Atmost one (0 or 1)

Many mapping : 0 or more (0 ..... *)

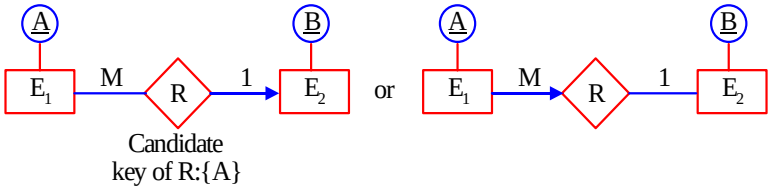
1. Binary Relationship Mapping (One : One)



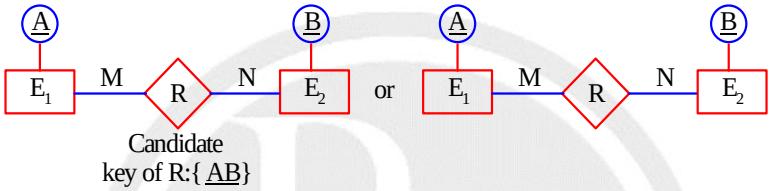
2. Binary Relationship Mapping (One : Many)



3. Binary Relationship Mapping (Many to One)



4. Binary Relationship Mapping (Many to Many)



2.5 RDBMS Design of Given ER Diagram

(For binary relationship)

1. Partial participation on both side of binary relationship

- **One to Many :** Merge relationship set towards many side. So, 2 relational tables.
- **Many to one :** Merge relationship set towards many side. So, 2 relational tables.
- **One to one :** Merge relationship set any one side. So, 2 relational tables.
- **Many to Many :** Separate table for each entity set and relationship set. so, 3 relational tables.

2. Full participation on "one" side of many to one relationship

Merge the entities and relationship set into single relational table. So, 1 table.

3. Full participation on "Many" side of Many-to-one relationship

Merge relationship set towards many side. So, 2 relational tables.

4. Full participation on any "one" side in one-to-one relationship

Merge the entity sets and relationship set into single table. So, 1 table.

5. Full participation on any "Many" side of Many-to-Many relationship

Merge relationship set towards any "Many" side of relationship. So, 2 table.

6. Full participation on both side of relationship

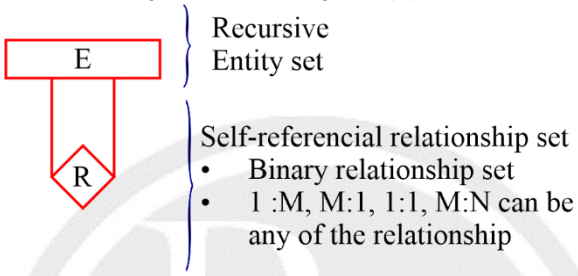
- 1:1
- 1:M
- M:1
- M:N

Merge the entity sets and relationship set into single relational table so, 1 relational table

2.6 Self-Referencial Relationship Set

(Recursive entity set)

Entities of entity set (E) related to some other entity of same entity set (E).



2.7 Weak Entity Set and Weak Relationship Set

The entity set with no key. (Attributes of weak entity sets are not sufficient to differenciate entities uniquely).

Points:

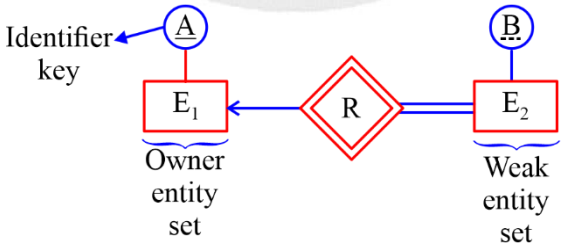
(a) For each weak entity set there must be owner entity set, which is strong entity set.

(b) Relationship set between weak entity set and identifier entity set is also "weak relationship set".

(c) The participation towards weak entity set end must be "total participation".

(d) The mapping between identifier entity set and weak entity set must be one : many (1 : M)

Example:



Note:

Weak entity set and multivalued attributes allowed to represent in ER diagram but not allowed in RDBMS table.

❑❑❑

3 RELATION MODEL AND SQL

3.1 Procedural Query Language and Non-procedural Query Language

Procedural Query Language	Non-procedural Query Language
Formulation of how to access data from the database table and what data required to retrieve from DB tables. "Relational Algebra"	Formulation of what data retrieve from DB tables. "Relational Calculus"

3.2 Relational Algebra

(Always generate distinct tuples)

Relational algebra refers to a procedural query language that takes relation instances as input and returns relation instances as output.

3.2.1 Basic operators

π : Projection operator

σ : Selection operator

× : Cross-product operator

U : Union

– : Set difference

ϱ : Rename operator

3.2.2 Derived operators

∩ : Intersection {using "_"}

⋈ : Join {using X, σ}

/ or ÷ : Division {using π, x, –}

I. π : Projection

- $\pi_{attribute\_name}$ (R): It is used to project required attribute from relation R.
- $\sigma_{Condition(P)}$ (R): It is used to select records from relation R, those satisfied the condition (P).

Example:

R

A	B	C
8	4	5
2	4	5
7	4	6
3	5	5

$\pi_{B,C}(R)$:

B	C
4	5
4	6
5	5

$\sigma_{A>S}(R)$:

A	B	C
8	4	5
7	4	6

II. Cross product (×):

- R × S : It result all attributes of R followed by all attributes of S, and each record of R paired with every record of S.
- Degree (R × S) = Degree (R) + Degree (S)
- $|(R \times S)| = |R| \times |S|$

Note:
- Relation R with n tuples and
- Relation S with 0 tuples then

number of tuples in R × S = 0 tuples

3.3 Join (⋈)

I. Natural join (⋈)

$R \bowtie S \equiv \pi_{\text{distinct attributes}}(\sigma_{\text{equality between common attributes of R and S}}(R \times S))$

Example:

- T_1 (ABC) and T_2 (BCDE)

$$\therefore \quad T_1 \bowtie T_2 = \pi_{ABCDE}\left(\begin{matrix}\sigma_{T_1.B=T_2.B}(T_1 \times T_2) \\ \cap T_1.C=T_2.C\end{matrix}\right)$$

- T_1 (AB) and T_2 (CD)

$\therefore \quad T_1 \bowtie T_2 \equiv T_1 \times T_2 = \pi_{ABCD}(T_1 \times T_2)$

Note:
Natural join equal to cross-product if join condition is empty.

II. Conditional Join ($\bowtie_c$)

- $R \bowtie_c S \equiv \sigma_c (R \times S)$

III Outer Joins:

(a) LEFT OUTER JOIN

R ⟕ S : It produces

(R ⋈ S) ∪ {Records of R those are failed join condition with remaining attributes null}

(b) RIGHT OUTER JOIN (⟖)

R ⟖ S : It produces]

(R ⋈ S) ∪ {Records of S those are failed join condition with remaining attributes null}

(C) FULL OUTER JOIN ($\bowtie$)

$R \bowtie S = (R \bowtie S) \cup (R \bowtie S)$

3.4 Rename operator (ϱ)

- It is used to rename table name and attribute names for query processing.

Example:

(I) Stud (Sid, Sname, age)

ϱ(Temp, Stud) : Temp (Sid, Sname, age)

(II) $\varrho_{I,N,A}$ (Stud) : Stud (I, N, A)

All attributes renaming

(III) $\varrho_{\substack{Sid \to I \\ age \to A}}$ (Stud) : Stud (I, Sname, A)

$_{age \to A}$

Some attribute renaming

Example:

Retrieve eicls of female employees whoose salary more than every male employee?

Result :

$$\pi_{eid}\left(\sigma_{(gen=Female)}^{(Emp)}\right) - \pi_{eid}\left(E \bowtie \delta_{I,S,G}\left(emp\right)\right)$$

$$\begin{pmatrix} gen = female \\ \cap \\ G = male \\ \cap \\ Sal \le S \end{pmatrix}$$

3.5 Division

- It is used to retrieve attribute value of R which has paired with every attribute value of other relation S.
  $\pi_{AB}(R)/\pi_{B}(S)$: It will retrieves values of attribute 'A' from R for which there must be pairing 'B' value for every 'B' of S.

Expansion of '/' by using basic operator

- Example: Retrieve sid's who enrolled every course.

Result:

π_{sidcid}(Enroll)/ π_{cid}(Course)

Step 1: Sid's not enrolled every course of course relation.

(Sid's enrolled proper subset of course)

$\pi_{sid}((\pi_{sid}(Enroll) \times \pi_{cid}(course)) - \pi_{sidcid}(Enroll))$

Step 2:

[sid's enrolled every course] = [sid's enrolled some course] – [sid's not enrolled every course]

$\therefore \quad \pi_{sidcid}(E) / \pi_{cid}(c) = \pi_{sid}(E) - \pi_{sid}((\pi_{sid}(E) \times \pi_{cid}(C) - \pi_{sidcid}(E))$

3.6 Set operator

U: Union operator

– : Except minus

$\cap$: Intersection operator

- To apply set operations relations must be union compatable.
- R and S relations union compatable
- If and only if-

  (i) Arity of R equal to Arity iof S and

  (ii) Domain of attributes of R must be same as domain of attributes of s respectively.

Example 1:

$\pi_{Sid\ Sname}(.....) \cap \pi_{Sid}(.....)$

{Arity not same so, set operation not allowed}

Example 2:

$\pi_{Sid\ S\ name}(.....) \cap \pi_{Sid\ marks}(.....)$

{Arity same but Sname domain is different from marks so, not allowed}

Example 3:

$\pi_{Sid\ Sname}(.....) \cap \pi_{Stud\ ID,\ Stud\ name}(.....)$

{Arity and domains are same so, allowed for set operation}

- 1. Set operation on relation:

R

A
2
2
2
3

S

B
2
2
4

$R \cup S : \{x / x \in R\ V\ x \in s\} \equiv$

A
2
3
4

$R - S : \{x / x \in R \wedge x \notin S\} \equiv$

A
3

$R \cap S : \{x / x \in R \wedge x \in S\} \equiv$

A
2

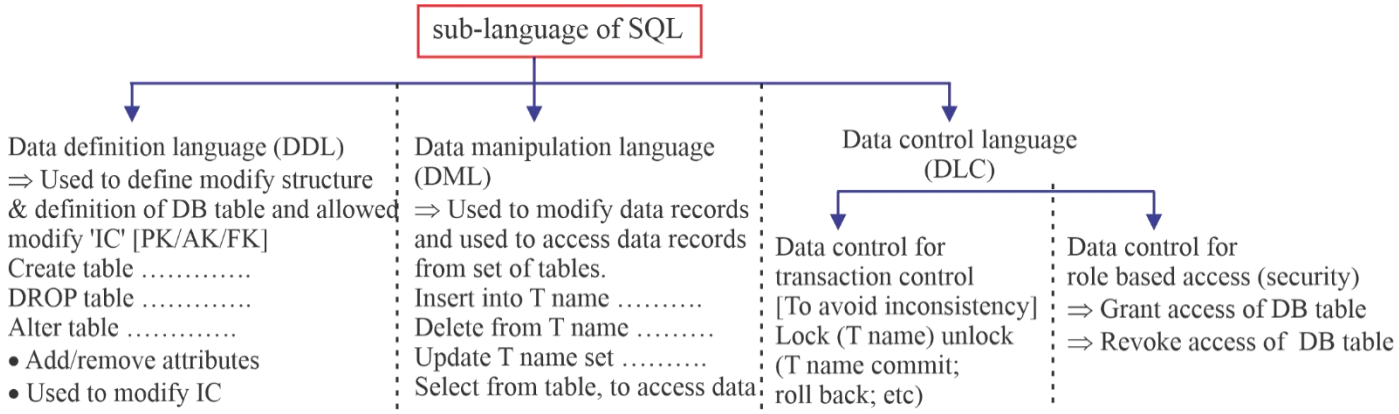
3.7 Cardinality Table

Assume relation R with n Distinct tuples and relation S with m distinct tuples:

RA Expression	Cardinality
R × S	n * m tuples
R ⋈ S	{0 to n * m tuples}
R =⋈ S	{n to n* m tuples}
R ⋈= S	{m to n*m tuples}
R =⋈= S	{max (n,m) to n*m tuples}
R∪S	{max(n,m) to n + m}
R∩ S	{0 to min (m, n}
R–S	{0 to n tuples}
R/S	{0 to [n/m] tuples}

3.8 SQL

3.8.1 [Structure Query Lang]



3.8.2 Relational Query and SQL query

- SQL : SELECT DISTINCT $A_1, A_2, \ldots\ldots A_n$

FROM $R_1, R_2, \ldots\ldots R_n$
WHERE P;

Equivalent RA : $\pi_{A_1, A_2 \ldots\ldots A_n}\left(\sigma_p\left(R_1 \times R_2 \times \ldots\ldots R_n\right)\right)$

- SELECT ≡ projection of RA (π)
  FROM ≡ Cross-product (x)
  WHERE ≡ Selection operator of RA (σ)

3.8.3 Basic SQL Clauses

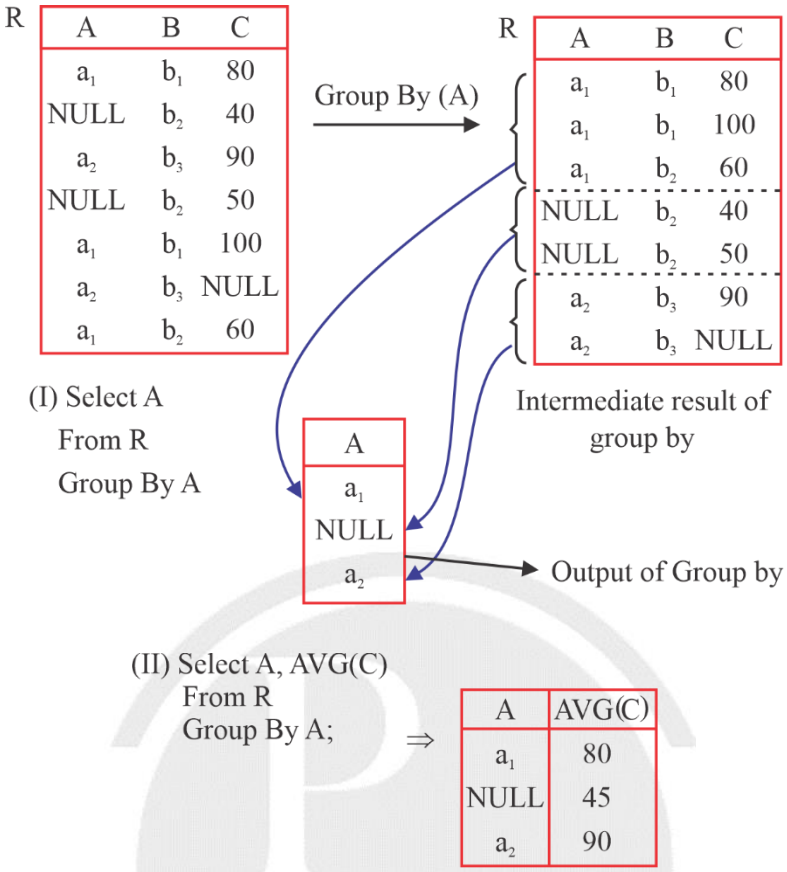
SELECT DISTINCT $A_1, A_2, \ldots\ldots A_n$
FROM $R_1, R_2, \ldots\ldots R_n$
WHERE condition
GROUP BY (attributes)
HAVING condition
ORDER BY (attributes) (DESC)

3.8.4 Execution Flow

FROM ⇒ cross-product of relations
↓
WHERE ⇒ Selection operator (σ) to apply condition for each record
↓
GROUP BY
↓
HAVING
↓
SELECT
↓
DISTINCT
↓
ORDER BY

3.8.5 GROUP BY:

- It is used to group records data a based on specific attribute.
- It GROUP BY clause used then
  (a) Every attribute of GROUP BY clause must be selected in SELECT clause.
  (b) Not allowed to select any other attribute in SELECT clause.
  Allowed to select aggregate function along with GROUP BY attribute in SELECT Clause.



3.8.6 HAVING Clause

- HAVING clause must be followed by GROUP BY clause.
- HAVING clause used to select groups those are satisfied having clause condition.
- HAVING clause condition must be over aggregation function such as some ( ), Every ( ) etc. but not allowed direct attribute comparison

Example:

1.
```
SELECT A
FROM R
GROUP BY A
HAVING AVG (c) > 60;
```

2.
```
SELECT A, AVG (c)
FROM R
GROUP BY A
HAVING some (c) > 50;
```

3.
```
SELECT A
FROM R
GROUP BY A
HAVING C> 60;
```
} Error

⇩
Not allowed

Note:
WHERE clause condition tested for each record but HAVING clause condition tested for each GROUP.

3.8.7 WITH Clause

- WITH clause is used to create sub-query result that can be re-used many times in query processing.

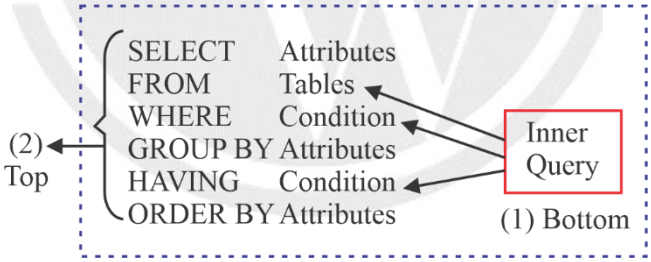
Example:

WITH Temp (A_1, A_2) as
(SELECT T_1 . A_1, T_2 . A_2
FROM Emp T_1, Emp T_2
WHERE T_1 . A_1 < T_2 . A_1)

Note:
Every Query which is written by using HAVING clause can be re-written by using WHERE clause.

3.8.8 Nested Query Without Co-relations

- Inner query independent of outer query.



- Execution flow must be Bottom to Top mean first bottom (Inner query) evaluated then outer query executed.

3.8.9 Co-related Nested Query

- In Nested Co-related query inner query uses attributes from outer query tables.
- In Co-related Nested query inner query allowed in WHERE, HAVING clause of outer query.

Example:

```
SELECT A
FROM R
WHERE (SELECT count(*)
        FROM S
        WHERE S.B < R.A) < 5;
```

Important point 1

- If co-relation in WHERE clause then inner query re-computes for each record of outer query From clause.
- If correlation in HAVING clause then inner query re-computes for each group of outer query.

3.9 Function used for nested Queries

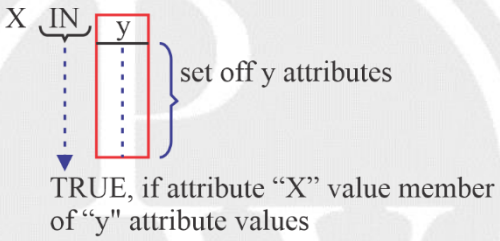
1. IN/ NOT IN
2. ANY
3. ALL

} Best suitable for non- co-related query

4. EXISTS/NOT EXISTS } Best suitable for co-related query

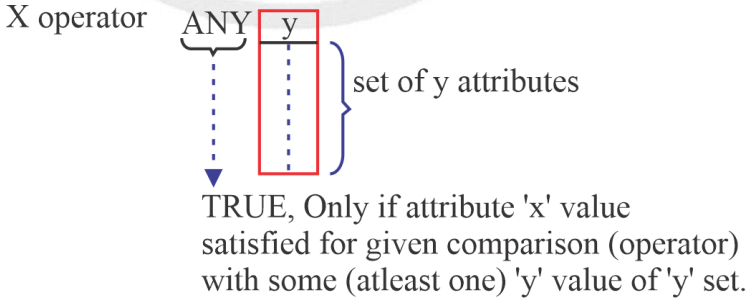
1. IN Function

It is used for membership testing.

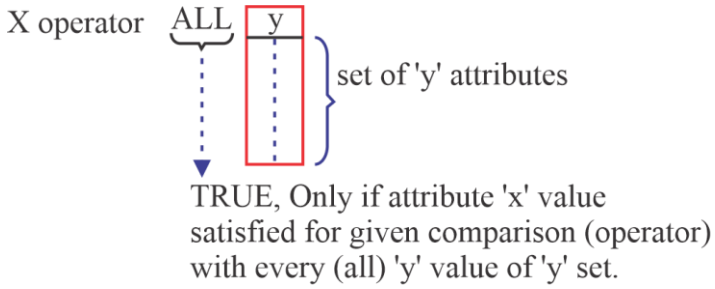


Note:
Queries of "equi Join" can implement by IN function ≡ (R ⋈ S)

2. ANY Function (operator "<, ≤, >, ≥, =, < >")



3. ALL Function



Note:
ANY function can be used as queries of conditional join query.

Example:

$$\pi_A\left(R \underset{R.A > S.B}{\bowtie} S\right) \equiv \text{SELECT A FROM R WHERE R.A > ANY (SELECT B FROM S)}$$

$$\pi_A(R) - \pi_A\left(R \underset{R.A \le S.B}{\bowtie} S\right) \equiv \text{SELECT A FROM R WHERE R.A > ALL (SELECT B FROM S)}$$

Important point 2

- IN function equal to '= ANY' function

$$X \text{ IN } Y \equiv X = \text{ANY } Y$$

- IN function not equal to '= ANY' if more than one attribute used for IN comparison.

$$(A,B) \text{ IN } (C\ D) \neq (A,B) = \underbrace{\text{ANY}}_{\text{Not allowed}} (C\ D)$$

- NOT IN function equal to "< > ALL".

$$X \text{ NOT IN } Y \equiv X <> \text{ALL } Y$$

4. **EXISTS** clause
   - It is used to test result of inner query is empty or not empty.
   -

EXISTS (Query)

TRUE, if result of inner query is non-empty that is atleast one record in result of inner query.

Example:

```
SELECT R.A
FROM R
WHERE EXISTS (SELECT *
              FROM S
              WHERE R.A > S.B);
```

Atleast one record of S relation satisfy R.A > S.B

Note:
EXISTS function can use as conditional join queries with join condition in inner query WHERE clause.

Emp (eid sal dno. Gen) Retrieve eids of femal's whose salary more than (some) male emp.

$$\text{RA: } \pi_{eid}\, \sigma_{gen\ =\ Female}(Emp) \bowtie_{Sal > S} \rho_{I,S,D,G}\left(\sigma_{gen=male}(emp)\right)$$

SQL [JOIN Query]

```
SELECT DISTINCT T1 . eid
FROM EMP T1, Emp T2
WHERE
T1.gen = Female and
T2.gen = male and
 T1.sal > T2 . sal
```

SQL [Mested Query]

```
SELECT Eid
FROM Emp
WHERE gen = Female
and sal > ANY (SELECT sal
                FROM Emp
                WHERE gen = male);
```

SQL [Co – related Nested Query]

```
SELECT Eid
FROM Emp T1
WHERE T1.gen = Female and
EXISTS (SELECT *
        FROM Emp T2
        WHERE T2.gen = male and T1. Sal > T2 . sal);
```

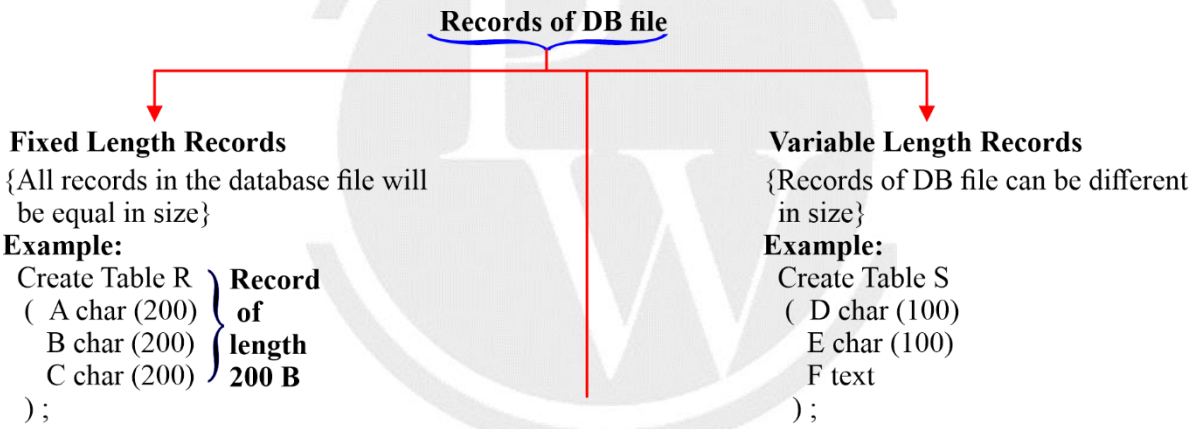
❑❑❑

4 File Organization and Indexing [B and B+ Tree]

4.1 File Organization

- Database is collection of files [Tables].
- File is collection of blocks [Pages].
- Block is collection of records.

Note:
Data access from disk (DB file) to main memory block by block.



4.2 Records organization of DB file

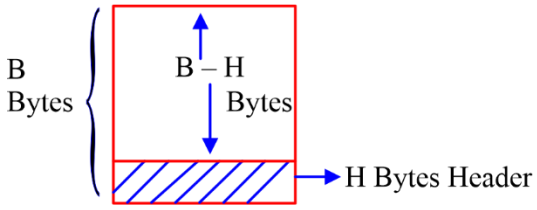
Spanned Organization	Unspanned Organization
1. Record allowed to span in more than one block.	1. Complete record must be in one block.
2. **Advantage:** Possible to allocate file without any internal fragmentation.	2. **Advantage:** less access cost and easy to organize DB file.
3. **Disadvantage:** More access cost to access spanned records.	3. **Disadvantage:** May not possible to allocate DB file without any internal fragmentation.

Note:
1. Spanned organization prefer to store database record of the file with variable length record.

2. Unspanned organization prefer to store DB record of file with fixed length record.

4.3 Block factor [BF]

The maximum possible records which can store in block (maximum record per block).



Consider record size = R bytes

1. $\therefore$ Block Factor $= \left\lfloor \frac{\text{Block size} - \text{Header size}}{\text{Record size}} \right\rfloor$

   For unspanned organization

2. $\therefore$ Block Factor $= \left\lceil \frac{\text{Block size} - \text{Header size}}{\text{Record size}} \right\rceil$

   For spanned organization

4.4 Categories of Index

4.4.1 Dense Index

For each DB record of DB file there must be index entry in index file.

4.4.2 Sparse Index

For set of DB records (blocks) of DB file there exists index entry in index file.

Important Point I

- Sparse index possible only over ordered field of DB file.
- In Sparse indexing

$$\left\lceil \begin{matrix}\text{No. of index} \\ \text{file entries}\end{matrix} \right\rceil < \left\lceil \begin{matrix}\text{No. of records} \\ \text{of DB file}\end{matrix} \right\rceil$$

4.4.3 Block Factor of Index

Assume Block size = B Bytes
Header size = H Bytes
Search key = K Bytes
Pointer size = P Bytes

$\therefore$ Block Factor of index $= \left\lfloor \frac{B-H}{K+P} \right\rfloor$

Note:
By default header size will be 0 bytes, if not given in problem.

4.5 I/O Cost [Access cost]

The number of secondary memory block (DB file block) required to transfer from disk to main memory in order to access required record.

1. I/O cost to access data record without index from DB file of n blocks

(a) Over ordered field:
Access cost = $[\log_2 n]$ block

(b) Over unordered field:
Access cost = n blocks

2. If Dense Index Used:

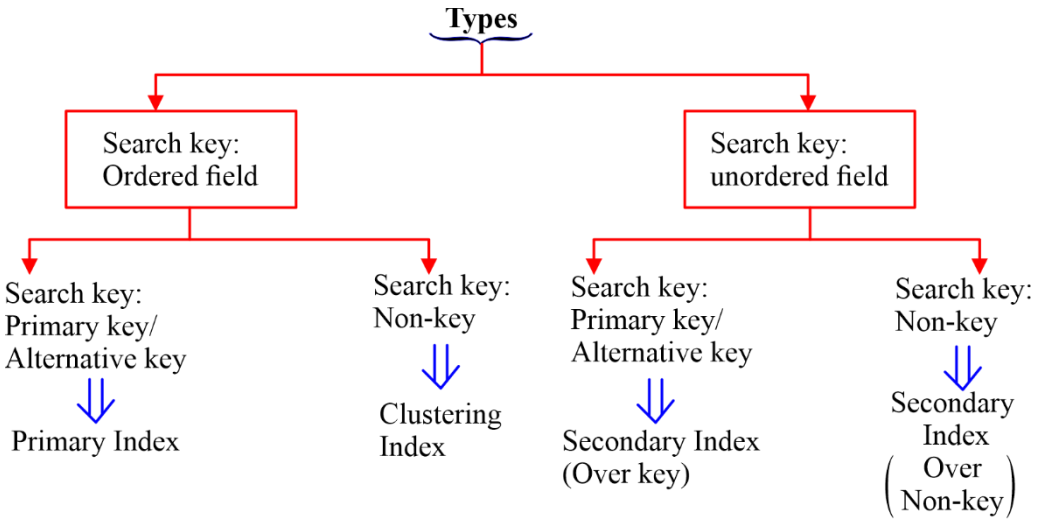
- Number of Dense Index block = $\left\lceil \frac{\text{number of records}}{\text{Block Factor of Index}} \right\rceil$
- Access cost (Number of Block access)
  $= \left\lceil \log_2 \left(\text{number of Dense index blocks at } 1^{st} \text{ level} \right) \right\rceil + 1$

3. If sparse Index Used:

- Number of DB blocks = $\left\lfloor \frac{\text{number of records}}{\text{BF of DB}} \right\rfloor$
- Number of sparse Index block (At 1^{st} level)
  $= \left\lceil \frac{\text{number of DB blocks}}{\text{BF of Index}} \right\rceil$
- Access cost (Number of block access)
  $\left\lceil \log_2 \left(\text{number of sparse index blocks at } 1^{st} \text{ level} \right) \right\rceil + 1$

4.6 Types of Index

- Search key: Fields of DB file which is used for indexing



4.6.1 Primary Index

Search Key: Ordered field and key (candidate key)

- I/O cost to access record using primary index with multilevel indexing is (k + 1) blocks.
  Where k is level of indexing.
- For any database relation at most one primary index is possible because, search key must be ordered field.
- Primary index can be either dense or sparse but sparse primary index preferred.

4.6.2 Clustered Index

Search Key: Ordered field and non-key.

- I/O cost:

  $$\text{Single level index blocks} : \left\lceil \frac{\text{No. of distinct values over non-key}}{\text{BF of Index}} \right\rceil$$

  $$\therefore \text{ Access cost} = \left\lceil \log_2(\text{single level index blocks}) \right\rceil + 1$$

- I/O cost to access cluster of record using clustered index with multilevel index is:

  $$k + \underbrace{(\text{one or more block of DB})}_{\text{Until next cluster begins}}$$

- For any DB relation at most one clustering index is possible because search key is ordered field.

> **Note:**
> For any DB relation either primary index or clustering index is possible but not both simultaneously.

4.6.3 Secondary Index [over key]

Search Key : Unordered field and key.

- Secondary index is alternative index to access data records even primary or clustering index already exists.
- Secondary index over key is always dense index.
- I/O cost to access record using secondary index over key with multilevel index: (k + 1) blocks.

4.6.4 Secondary Index [over non-key]

Search Key: Unordered field and non-key.

- I/O cost to access record using secondary index over non-key with multilevel index = [k blocks from k level of index] + [some more DB blocks]

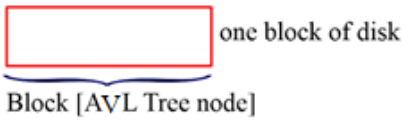
> **Note:**
> Many secondary index's possible for DB file.

4.7 Balanced Search Tree [Height Restricted Search Tree]

- The maximum height of search tree for n distinct keys should not exceed O (log n)
- Worst case search cost to search element is : O (log n)

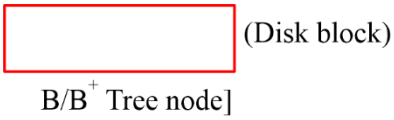
4.7.1 Why B/B+ Tree Index preferred rather than balanced BST/AVL for DB index

(a) AVL: If AVL tree (balanced BST) used for DB index:



- Disadvantage: Each index block with only one key so the number of levels of index is high. (I/O cost to access data is also very high).
- Wastage of disc space, which is allocated for index (only one key stored per block)

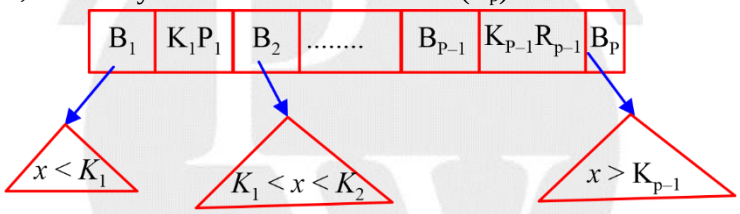
(b) B/B$^+$ Tree:



- The access cost (I/O cost) is less because many keys store in one node (number of levels of index will be less).
- The wastage of disk space is less.

4.7.2 B Tree Definition

- Order P (degree): The maximum possible child printers can be stored in B Tree node.
  Node structure: P child pointer, P – 1 Key and P – 1 Record Pointer (R_p).



$\therefore$ order of node $\Rightarrow$

$P \times$ (size of block pointer) + (P - 1) (size of key field + size of record pointer) $\leq$ Block size

1. Every internal node except root must be atleast $\left\lceil \frac{p}{2} \right\rceil$ block pointer and $\left\lceil \frac{p}{2} \right\rceil$ - 1 keys and at most P block pointer and (P – 1) keys.
2. Root can have at least 2 child/block pointer and 1 keys and at most P block pointer and (p - 1) keys.
3. Every leaf node must be at same level.

- **Advantages:**

B Tree index best suitable for random access of any one record.

Example:

```
SELECT *
FROM R
WHERE A = 15;
```

$\therefore$ Worst case access cost = (k + 1) blocks
Best case access cost = 2 blocks

- **Disadvantages:**

B Tree index not suitable for sequential access of range of records.

4.7.3 B+ Tree

Order P: The maximum possible pointers can be store in B^+ tree node.

1. Node Structure:

- **Leaf node:** (set of (key, R_P) pairs and one block pointer)

K_1R_1	K_2R_2	…….	$K_{P-1}R_{P-1}$	B_P

→Block Pointer {pointer to next leaf}

∴ Order of leaf node = $(P-1)[K + R_P] + 1 * B_P \leq$ Block size.

- **Internal Node:**
  Order of internal node = $P * B_P + (P-1) * K \leq$ Block size.
- B^+ Tree best suitable for random access of any one record and sequential access of range of records.
- I/O cost to access range of 'x' records sequential:

$$K + \frac{x}{\left\lceil \frac{P}{2} \right\rceil - 1} + x = 0(K + x)$$

To reach leaf

No. of times leaf accessed

DB block

Important point 2

If disk block allocation for B and B^+ tree nodes are equal size then

1. Order P of B tree < order P of B^+ Tree.
2. [number of index blocks of B tree index for n keys] ≥ [number of index blocks of B^+ Tree nodes for n keys]
3. I/O cost ≥ I/O cost

Important point 3

If order P of B Tree is equal to order P of B^+ Tree then

1. [number of B Tree node n distinct keys] ≤ [number of B^+ Tree nodes for n distinct keys]
2. [B Tree level for n keys] ≤ [B^+ Tree levels for n keys]

❑❑❑

5 TRANSACTIONS AND CONCURRENCY CONTROL

5.1 Transaction

A set of logically related operation to perform unit of work.

5.2 Degree of concurrency

The number of users (Transactions) using data bases simultaneously (concurrently).

5.2.1 Flat File System

In flat file system 1 file is a resource so, only one user allowed at a time.

5.2.2 DBMS File System

- In DBMS file system 1 record is a resource so, it allows as many users as the number of records.
- More degree of concurrency.

5.3 Main Operations in Transactions

5.3.1 Read (A)

Access the data item ‘A’ from DB file ‘Disk’ to programmed variable (main memory) in order to use current value of ‘A’ in transaction logic.

5.3.2 Write (A)

Modification of data item ‘A’ in DB file.

5.4 ACID Properties

To preserve integrity (correctness) each transaction must satisfy ACID properties

DBMS Software	A:Atomicity, D:Durability} Recovery management component of DBMS software take cares of atomicity and durability
	I : Isolation} Concurrency control component of DBMS software is responsible for isolation.
DBA User	C : Consistency} User (DBA or DB developer) is responsible for consistency.

5.5 Atomicity

Execute all operations of transaction including commit or execute none of the operation of transaction by the time of transaction termination.
Recovery management component should roll back, if transaction fails anywhere before commit.

5.5.1 Redo Operation

- It performs all the modification of database file because of transaction commit.
- Clean all the log entries of committed transaction.
- Redo is not performed after every commit but after every check point.

5.5.2 Undo Operation

- Undone all the write operation performed by the transaction.
- Convert dirty block (updated block) into clean block.

5.5.3 Check point

- Check point issued by DBMS software in regular interval.
- If checkpoint issued
  (I) Performs Redo operation on all the committed transaction until previous checkpoint.

Example :

9 : 00 AM	DB Started
:	
9 : 05 AM	1st checkpoint
:	
9 : 10 AM	2nd checkpoint
:	
9 : 15 AM	System Crash

5.5.4 System Crash

If System crash/failure happen, required operation to recover are
(I) All committed transaction until previous checkpoint will perform Redo.
(II) All uncommitted transaction in entire system will perform undo.
(III) Clean all log entries.

5.5.5 Roll back (Abort)

Undo modification of database file which are done by failure transaction.

5.6 Durability

- Durability maintained by Recovery management component of DBMS Software.
- The transaction should able to recover under any case of failure.
- Transaction fails because of

1. Power failure
2. Software crash
   - OS restarted
   - DBMS restarted
3. OS/DBMS concurrency controller may kill transaction.
4. Hardware Crash (Disk failure)
   RAID architecture of disk design is used to overcome the problem.

5.7 Consistency

- User responsible for consistency.
- Database should be consistent before and after the execution of the transaction.
- DB operations requested by user (SQL queries/transaction operation) must be logically correct.

5.8 Isolation

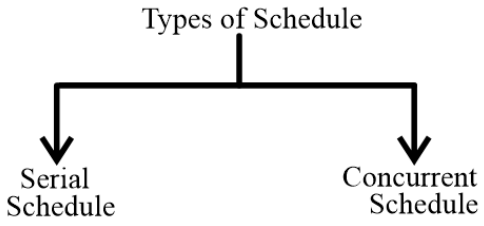
- Isolation maintained by concurrency control component.
- Isolation means concurrent execution of 2 or more transaction result must be equal to result of some serial schedule.

5.9 Schedules

Time order execution sequence of two or more transactions.

Example :

S : $R_2(A)$ $R_1(A)$ $W_3(A)$



5.9.1 Serial Schedule

- After Commit of one transaction, begins (Start) another transaction.
- Number of possible serial Schedules with 'n' transactions is "n!"
- The execution sequence of Serial Schedule always generates consistent result.

Example

S : $R_1(A)$ $W_1(A)$ Commit (T_1) $R_2(A)W_2(A)$ commit (T_2).

5.9.2 Advantage

- Serial Schedule always produce correct result (integrity guaranteed) as no resource sharing.

5.9.3 Disadvantage

- Less degree of concurrency.
- Through put of system is low.
- It allows transactions to execute one after another.

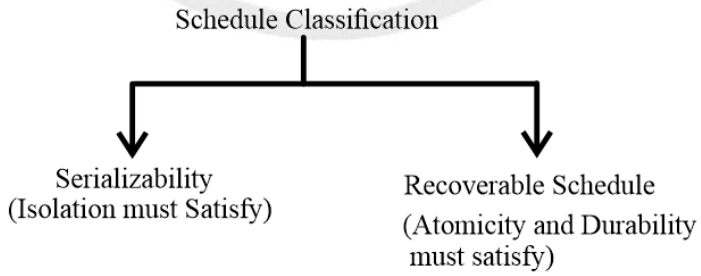
5.10 Concurrent Schedule

Transactions can execute concurrently or simultaneously

- May result inconsistency.
- Better through put and less response time.
- To maintain consistency transaction should satisfy ACID property.
- If T_1 and T_2 transaction with 'n' and 'm' operations each, then

  No. of concurrent Schedule = $\frac{(n+m)!}{n!\ m!}$

5.11 Schedule Classification

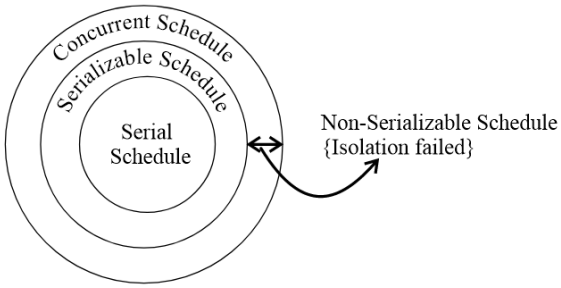


5.12 Serializable Schedule

A Schedule is serializable Schedule if it is equivalent to a Serial Schedule.

(i) Conflict Serializability

(ii) View Serializability



5.13 Conflict Serializability

5.13.1 Topological order

Graph traversal algorithm for directed graph

(i) Visit vertex (v), whose indegree is 'O' and delete 'V' from the graph.

(ii) Repeate (i) for all the vertices of graph.

5.13.2 Conflict Pairs

Two operations form Schedule (s) are conflict pair if and only if

(i) Atleast one write operation.

(ii) Operation on different data item.

(iii) Operations are from different transactions.

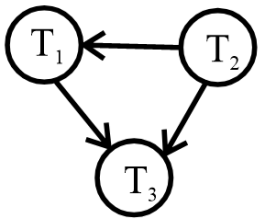
$$\left.\begin{array}{l} S: r_i(A).....w_j(A) \\ S: w_i(A).....r_j(A) \\ S: w_i(A).....w_j(A) \end{array}\right\} \text{Conflict Pairs}$$

5.13.3 Precedence Graph

A graph G in which vertex (v) represent the transaction of Schedule and edges (E) represent conflict pair precedence's.

Example:

S : $R_2(A)$ $R_2(B)$ $W_1(A)$ $W_2(B)$ $W_3(A)$ $W_3(B)$



5.13.4 Conflict Equal Schedule

Schedule S_1 and S_2 are conflict equal Schedule if and only if Schedule S_2 can derived by inter changing consecutive non-conflict pairs of Schedule S_1.

$$S_1 \xleftarrow[\text{Consecutive non-conflict pair}]{\text{Interchange}} S_2$$

Note:

If Schedule S_1 and S_2 have

(i) Same set of transactions, and

(ii) Same precedence graph and

(iii) A cyclic precedence graph then

S_1 and S_2 are conflict equal Schedule.

Important Point 1:

1. If S_1, S_2 Schedule are conflict equal then precedence graph of S_1 and S_2 must be same.
2. If S_1 and S_2 have same precedence graph then S_1 and S_2 may or may not conflict equal.

5.14 Conflict Serializable Schedule

Schedule (s) is conflict serializable Schedule if and only if some serial Schedule (S) must be conflict equal to Schedule (S).

$$\underbrace{\text{Schedule (s)}}_{\text{Conflict Serializable Scchedule}} \xleftrightarrow[\text{Equal}]{\text{Conflict}} (S') \text{Serial Schedule}$$

Important Point 2:

Schedule (s) is conflict serializable Schedule if and only if

1. Precedence graph is a cyclic and
2. Conflict equal serial schedule are to apological order of precedence graph.

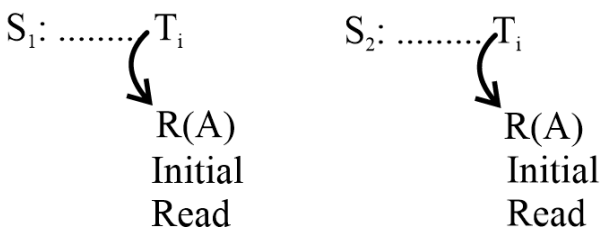
Note:

[No. of serial schedule conflict equal to schedule s] = [No. of topological order of schedules precedence graph]

5.15 View Serializable Schedule

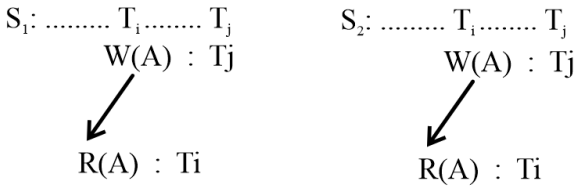
- Schedule (s) is view serializable if and only if some serial schedule (s') view equal to schedule (s).
- Let S_1 and S_2 be two schedule with the same set of transactions, S_1 and S_2 are view equal view equivalent if the following three conditions are met:

5.15.1 Initial Read



For each data item A, if transaction T_i reads the initial value of A in Schedule S_1, then transaction T_i must, in Schedule S_2, also read the initial value of A.

5.15.2 Updated Read



For each data item A if transaction T_i perform Read (A) in Schedule S_1 and that value was produced by transaction T_j, then transaction Ti must in Schedule S_2 also read the value of A that is produce by the transaction Tj.

5.15.3 Final Write

S_1 : ..... Ti .....

Final write of A : $w(A)$ (Last write)

S_2 : ..... Ti....

Final write of A : $w(A)$ (Last write)

For each data item A, the transaction that perform the final write (A) operation in schedule (S_1) must perform the final write (A) operation in schedule S_2.

Important Point 3

- If schedule S is conflict serializable schedule, then S is also view serializable schedule.
- If schedule s is not conflict serializable then it may or may not view serializable
- Every view serializable schedule that is not conflict serializable.

5.16 Classification based on Recoverability

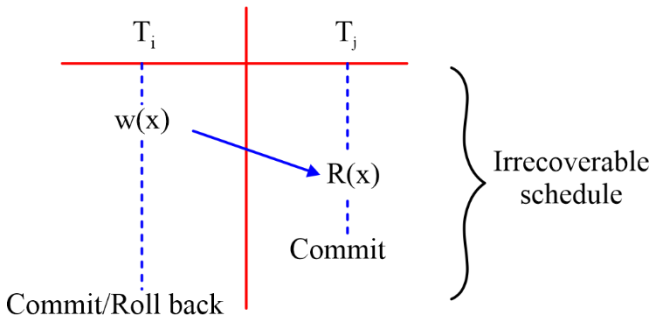
The concurrent execution may leads:

1. Irrecoverable schedule
2. Cascading rollback problem
3. Lost update problem

The above problems can occur even if the schedule is serializable (Integrity satisfied)

5.16.1 Irrecoverable Schedule

- A schedule(s) is irrecoverable if and only if transaction T_j reads data item 'x', which is updated by T_i and commit of T_j before commit/rollback of transaction T_i.
- Irrecoverable means unable to recover or rollback.



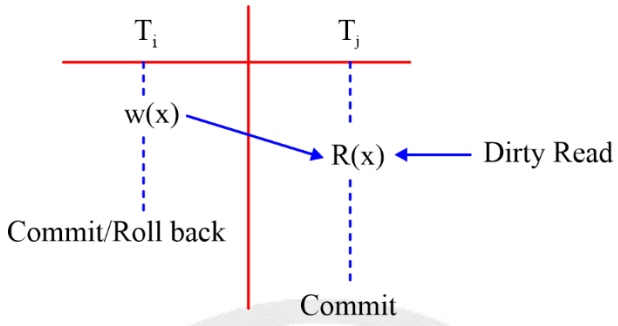
5.16.2 Recoverable Schedule

A schedule(s) is recoverable if and only if

(I) No uncommitted read (no dirty ready in schedule(s).

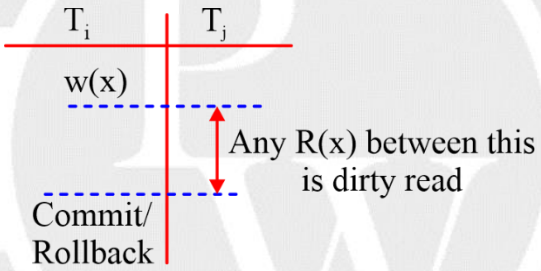
Or

(II) If transaction T_j reads data item 'x' which is updated by transaction 'T_i', then commit of T_j must be delayed until commit/rollback of T_i.



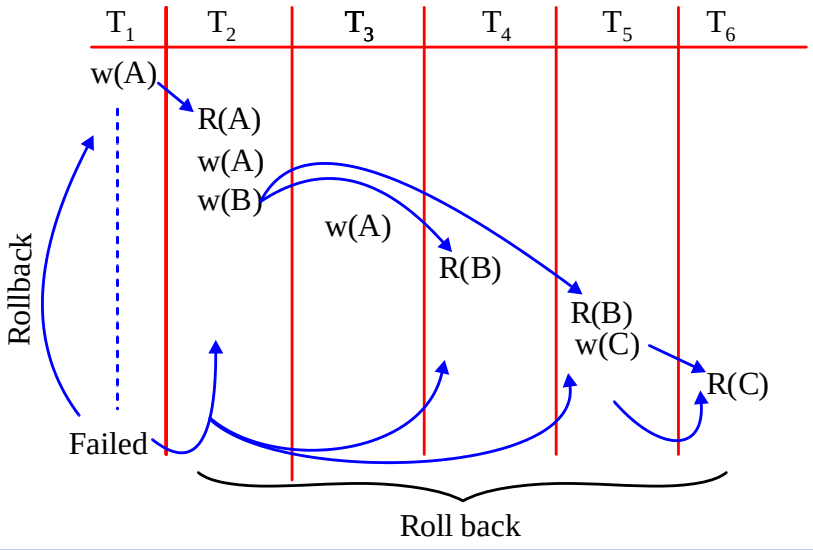
5.16.3 Uncommitted Read (Dirty Read)

Transaction T_j reads data item 'x' which is updated by uncommitted transaction. T_i



5.17 Cascading Rollback Schedule (Problem)

When some transaction reads data items which is updated by some other transaction then because of failure of the transaction that updated the data item may rollback all the dependent transaction.

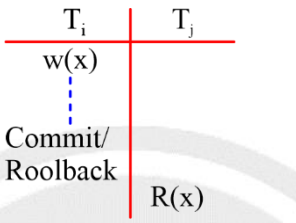


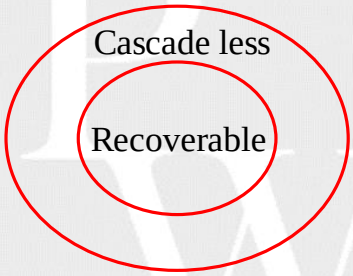
Failure of T_1 forced to rollback T_2, T_4, T_5 and T_6.

- **Disadvantage of Cascading Rollback**
  1. It waste the CPU execution time.
  2. Wastage of I/O access cost.

5.17.1 Cascadeless Rollback Recoverable Schedule

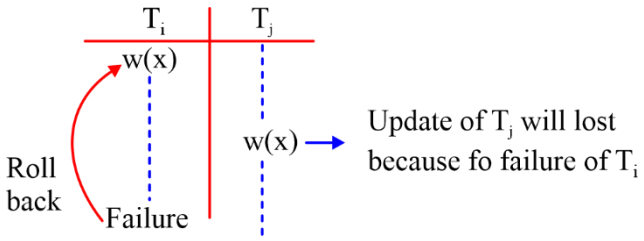
- No dirty read in the schedule.
- Transaction T_j should delay read (x) which is updated by T_i, until T_i commit or rollback.





5.18 Lost Update Problem

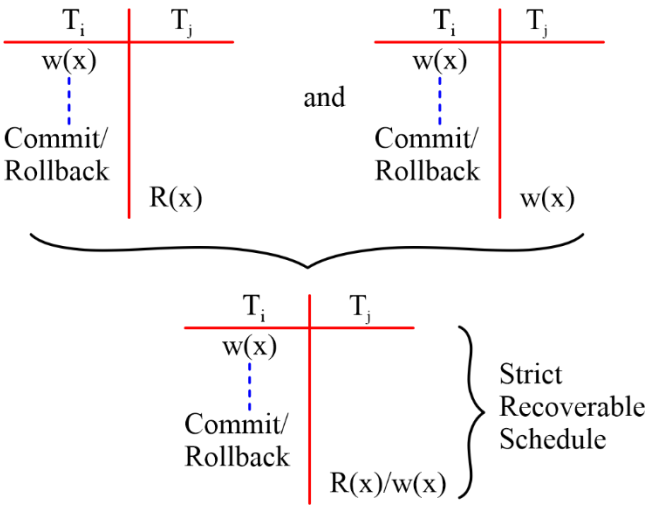
Lost update problem occur if T_j write (x) which is already written by uncommitted transaction T_i.

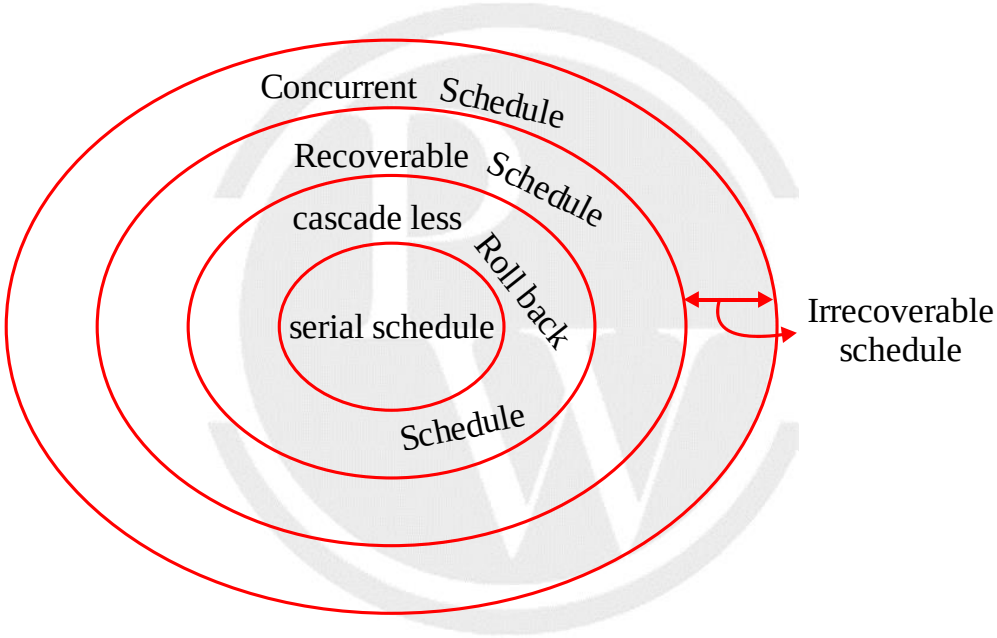


5.19 Strict Recoverable schedule

A schedule must satisfy

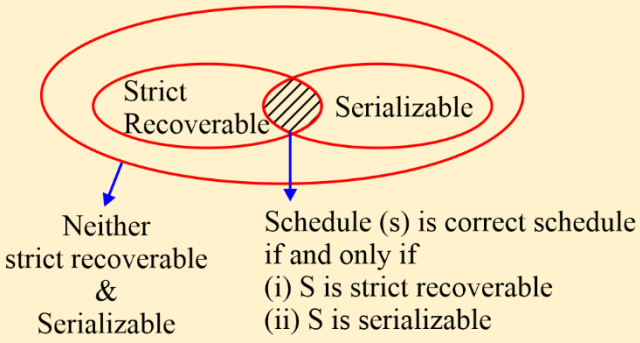
1. Cascadeless rollaback recoverable schedule and
2. If T_i writes (x) then other transaction T_j write (x) must be delayed until T_i commit or rollback.





Note:

1. Strict recoverable schedule may or may not be serializable.
2. Serializable schedule may or may not be recoverable.



5.20 Concurrency control protocol

Concurrency control protocol should not allow to execute:

1. Non-serializable schedule (Violate Isolation).
2. Non-strict recoverable schedule (Violate atomicity and Durability).

5.21 Locking Protocol

Lock is as variable used to identify the status of data item.

Transaction (T_1)

Lock (A): Granted by concurrency controller

R(A)

W(A)

Lock(B): Denied by concurrency controller

5.22 Types of Lock

1. Shared Lock(s): Read only lock.
2. Exclusive lock (x): Read/write lock.

1. Shared (s mode)

- Exists when concurrent transaction granted READ access.
- Produce no conflict for Read only transactions.
- Issued when transaction wants to read and exclusive lock not held on item.

2. Exclusive (x) mode

- Exists when access reserved for locking transaction.
- Used when potential for conflict exists.
- Issued when transaction wants to updated unlocked data.

Example:

Transaction (T_1)

S(A) : Granted shared lock

R(A) } only read allowed

X(B) : Granted exclusive lock

R(B) }
W(B) } Read and write

5.22.1 Lock compatible table

Data item A	S	X ← Holded by T_i
Requested by T_j — S	Yes	No
Requested by T_j — X	No	No

5.23 Phase Locking Protocol (2PL)

- It guaranteed serializability.
- Transaction (T) allowed to request lock on any data item in any mode (x/s) until first unlock of transaction (T).

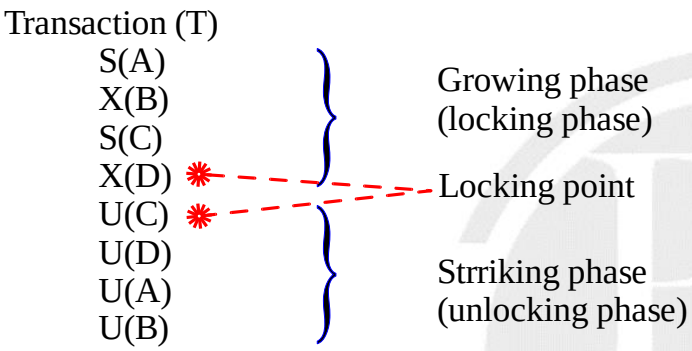
1. Growing Phase

Acquires all the required locks without unlocking any data. Once all locks have been acquired, the transaction is in it locked point.

2. Shrinking phase

Release all locks and can not obtain any new lock governing rules of 2 PL.

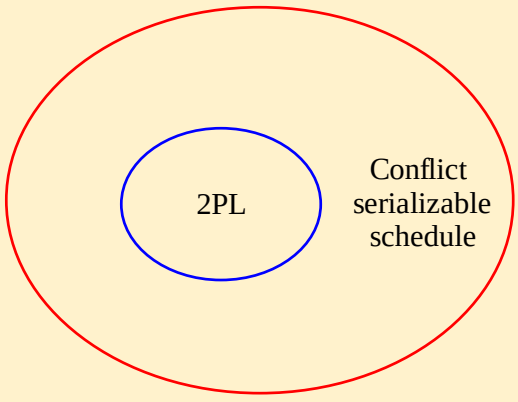
Example:



Important point 1

1. If schedule (s) is executed by 2PL then schedule guaranteed is conflict serializable schedule.
2. Conflict equal schedule is based on lock points order of the transaction.
3. If schedule is not conflict serializable schedule (cyclic precedence graph) then schedule not allowed to execute by 2 PL.

Note: Every schedule which is allowed by 2PL is always conflict serializable, but not every conflict serializable schedule is allowed by 2PL.



5.23.1 Limitations of 2PL

1. 2PL restriction may leads to deadlock.
2. 2PL restriction may leads to starvation.
3. 2PL condition not sufficient to avoid
   (I) Ir-recoverable schedules
   (II) Cascading rollback problem
   (III) lost updated problem

5.24 Strict 2PL protocol

- **Basic 2PL:** Lock request of transaction (T) not allowed in shrinking phase of transaction T

  and

  Strict recoverable: All exclusive lock of transaction must hold until commit/Rollback of transaction T.

5.24.1 Strict 2PL protocol guarantees

(a) Serializability.
(b) Strict recoverable.

Disadvantage:

It is not free form deadlock and starvation.

Example:

Transaction (T)

$$\left.\begin{array}{l} S(A) \\ X(B) \\ S(C) \\ X(D) \end{array}\right\} \text{growing phase}$$

$$\left.\begin{array}{l} U(A) \\ U(B) \\ \text{Commit} \\ U(B) \\ U(D) \end{array}\right\} \text{Strinking phase}$$

5.25 Rigorous 2PL

Basic 2 PL : A lock request allowed only in growing phase of transaction and

all locks (shared/Exclusive) of transaction T must be hold until commit or Rollback.

5.26 Conservative 2PL

Transaction (T) must lock all required data itmes in starting phase.

If locks are not available then unlock all data items and re-request all data item lock again.

	Basic 2PL	Strict 2PL	Rigorous 2PL	Conservative 2PL
Guaranteed Serializability	Yes	Yes	Yes	Yes
Guaranteed strict Recoverable	No	Yes	Yes	No
Free from dead lock	No	No	No	Yes
Free from starvation	No	No	No	NO

5.27 Time stamp ordering protocol

Assign global unique time stamp value to each transaction and produces order for transaction submission.

The time stamp values of transaction can be used for transaction identification and to set priority between the transaction.

5.27.1 Time stamp value for each data item

- Assume data item is A

I. **Read TS value (A):** It is the highest transaction time stamp value that has executed R(A) successfully.

II. **Write TS value (A):** It is the highest transaction TS value that has executed W(A) successfully.

1. Basic Time stamp ordering protocol:

a. If transaction T issues R(A) operation.

If (WTS(A)) > TS(T)

then Rollback T

else

{Allow to execute R(A) successfully

set RTS (A) = max {RTS(A), TS(T)}}

b. If transaction T issues W(A) operation
if (RTS(A)) > TS (T)
{then Rollback trans (T)}
else if (WTS(A) > TS(T))
{then Roll back T}
Else
{Allow to execute W(A) successfully
set WTS(A) = {TS(T)}}

2. Thomas write rule stamp ordering protocol

a. If transaction T issues R(A operation
if (WTS(A)) > TS(T)

{then Rollback transaction T}

else

{allow to execute R(A) successfully

set RTS (A) = max {TS(T), RTS (A)}

b. If transaction T issue W(A) operation
if (RTS (A) > TS(T))

{then roll back T}

else if (WTS(A)) > TS(T)

{then ignore w(A) of transaction T and continue.}

else

{Allow to execute w(A) successfully

set WTS(A) = TS(T)}

3. Strick time stamp ordering protocol

A transition T_2 that issues a R(A) or W(A) such that TS (T_2) > WTS (A) has it read operation delayed until the transaction T_1 that write the value x has committed or rolled back.

Basic TS ordering protocol	Thomas write TS ordering protocol	Strict TS ordering protocol
1. Ensue conflict serialization	1. Ensure view serialization	1. Ensure conflict serialization
2. Free from deadlock	2. Free from deadlock	2. Free from deadlock
3. Not free from starvation	3. Not free from starvation	3. Not free from starvation
4. Not guaranteed strict recoverable	4. Not guaranteed strict recoverable	4. Guaranteed strict recoverable

❑❑❑

GATE Exam 2025?

SHRESHTH BATCH

ME | CE | EE | EC | CS & IT

- Live Classes & Recorded Videos
- DPPs & Solutions
- Doubt Solving
- Batches are available in ***Hindi***

Weekday & Weekend Batches Available

JOIN NOW!