Some DBMS : SQl , MySQl , oracle , DB2

Father of DBMS : EF Codd

* Data : raw facts and statistics
  eg: text , image , audio , video etc.

* Information : Processed form of data , usable data

* Database : collection of related data stored together on purpose for further use

* Types of databases

Structured Database (10%)	Unstructured Database (90%)
• data which can be organised in the form of rows and columns constitutes structured database	• data which can't be organised constitutes unstructured database. nowadays it is utilised via AI
• eg : data of employes in a firm.	• eg : photos , videos , social media data.
• stored and accessed using RDMS etc.	• cloud storage etc.

* Database Management System (DBMS)
  A system (software) which faciliates storage , maintenance & utilisation of database (structured)
  eg SQL (Microsoft)
  Oracle 9i , 11 , 12c (Oracle)
  MySQL (Oracle)
  DB2 (IBM)

* Relational DBMS : DBMS which maintains data in the form of relations.

Program

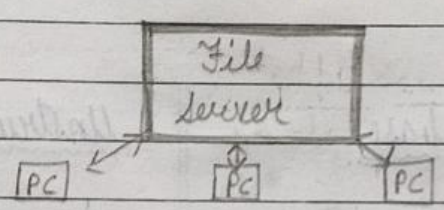
PL
Presentation Logic
input

BL
Business logic
processing

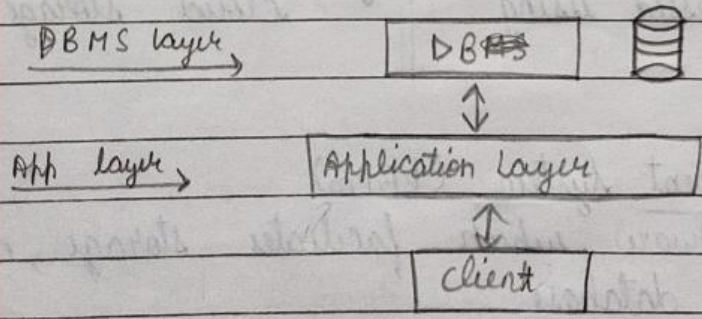
DL
Database logic
database

Types of DBMS Architecture

1) **One-Tier Architecture**:
- client and server are at same place (usually in the same computer)
  (uses local servers)
- client directly access the file server.
- includes window projects



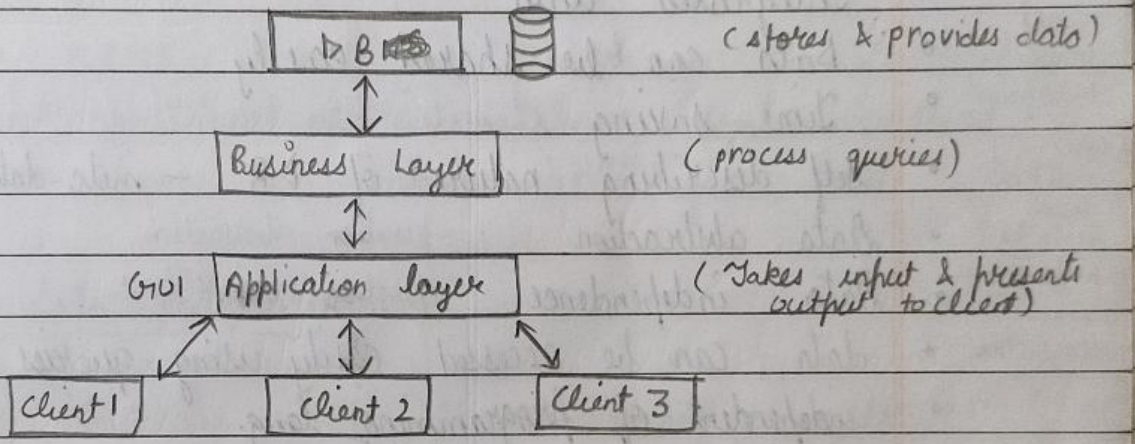
2) **Two-Tier Architecture**
- usually client and server are ~~at~~ at ~~same~~ diff places &
  (at two diff machines connected via network)
  communicate through application layer.
  (uses local servers)
- direct interaction with DBMS through application (ODBC)
  Open Database Connectivity
- window projects



Advantages	Disadvantages
- Easy to maintain & modify - communication is faster -	- limited clients - security issues (Queries are processed by database itself)

3) 3-Tier Architecture

- multiple clients are present at remote places & server too is located at a remote destination. (remote server is used)
- client doesn't interact directly, with DBMS but through business layer.
- web projects



Advantages

- Security
- Scalability (large no. of remote clients can be handled)
- Performance is better (due to presence of business layer)

- professional data

Advantages of DBMS over file system

Disadv. Require technical person

- Reduce data redundancy
- Consistency of data
- ensures back up & recovery
- More privacy & security (role based access)
- Concurrency (Multiple access at same time) Multiple views
- Integrated data
- Data can be shared easily
- Time saving
- Self describing nature of DB – meta-data (in catalog) In file pr. system str. is defined in appl. prog itself
- Data abstraction / operation abstraction
- Data independence / operation independence
- data can be accessed easily using queries
- independent of programming lang.
- Atomicity either all or none
- Integrity (provides constraints)

It ensures correct transaction is performed when multiple users tries to perform some transaction

Terms related to RDBMS

- Entity set / relation – table
- entity / tuple / record / instance – row
- attributes / fields – columns
- Degree – no. of attributes in a relation / no of entity sets (in case of relation ships)
- Cardinality – no of records

Schema : A logical representation of a database

* Abstraction
Hiding the complexities of the system from the client & providing an abstract view of only relevant info.

* Data Independence (Transparency)

Property of DBMS which helps to change the Database schema at one level of a database system without having to change the schema at next higher levels.

- Logical data independence

What is stored

Ability to change the logical schema without having to change external schema or application programs.

eg : Addition or removal of new entities in conceptual schema won't effect existing application program.

- no effect on view level
- even if changes are made in conceptual schema by one user, views of other users won't change (bcz virtual schema of only required data is presented to them)
- user won't get any idea of change in concep. schema

- Physical data independence

How data is stored

Ability to change the physical schema without having to change logical schema

Types of models

Types of lang. DDL DML DCL

File system V/S DBMS

Page No.
Date

Q. Changes include:

- using new storage devices
- using different data structures
- Modifying indexes

storage places & indices change frequently but user doesn't get to know coz applic. program remains the same.

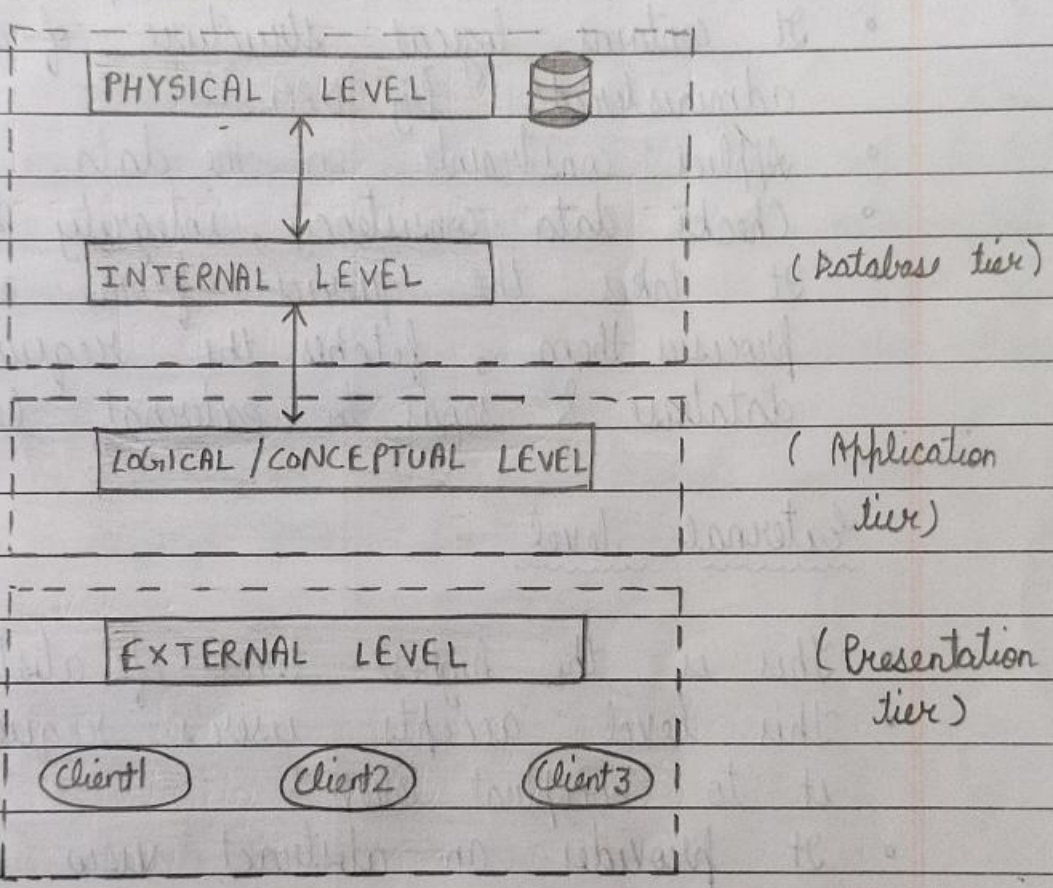
~~Eg.~~

Name : Payal
Roll No : 23609

This is a theoritical model

THREE LEVEL OF ABSTRACTION
(Three Tier Architecture)

It involves following three levels:
1) External level (Client side)
2) Logical / Conceptual / Business level
3) Internal level



Internal level

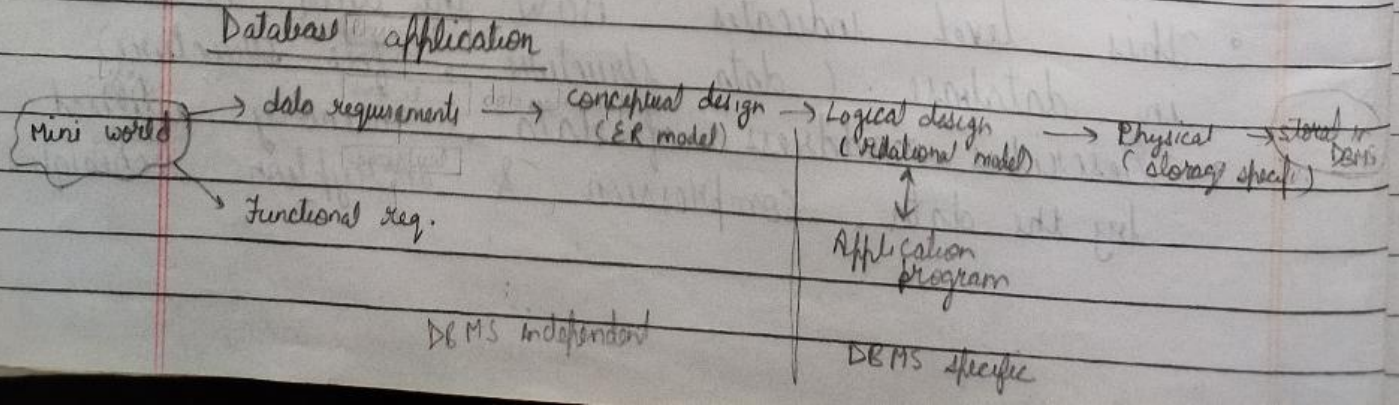
- This is the lowest level of abstraction and the one closest to the database.
- This level indicates HOW the data is stored in database (data structure, file structure)
- Describes address of data, memory utilised by the data, compression & encryption techniques.

Logical / Conceptual Level

- This is intermediate level of abstraction which faciliates communication between the client & the database.
- This level describes WHAT data is stored in the database & relationships among the data.
- It contains logical structure of database administered by DBA.
- Applies constraints on the data
- Checks data consistency, integrity & security.
- It takes the queries from external level, processes them, fetches the required data from database & sends to external level.

External level

- This is the highest level of abstraction.
- This level accepts user's request & sends it to conceptual level.
- It provides an abstract view of data to the user & gives only relevant information.
- Multiple user views may exists
- This level is closest to the user concerned with the way data is viewed by individual users.



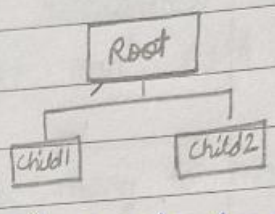
DATA MODELS

Data model is an abstract model that defines the logical structure of a database.
It is independent of hardware or software constraints.

1) Hierarchical Data Model

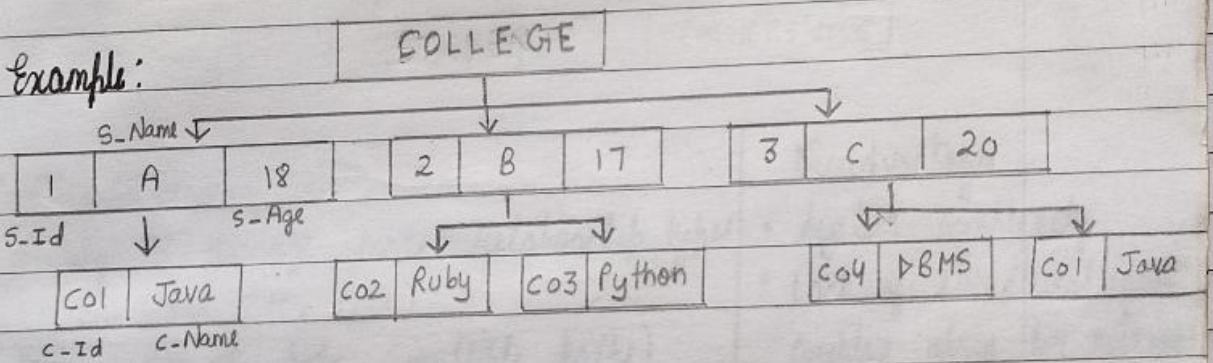
record based

oldest model (1950) (North American Rockwell/IBM)
(one → many) cardinality



- H. Data Model represents data hierarchically (top down/ bottom up) in the form of parent - child relationship (tree like structure)
- A parent may have many children but children have only one parent. (one to many) & (one to one)
- Data of entities (parent/child) is stored in records (nodes)

Example:



Disadvantages

- Data redundancy (in case of M to 1) but here
- lack of structural independence. Rigid structure. poor flexibility.
- If parent data is deleted, all child elements gets deleted.
- Difficult to design complex relationships
- Difficult to implement M:N
- programming complexity

DML, DDL ✗

• Complex algos for retrieval

Advantages

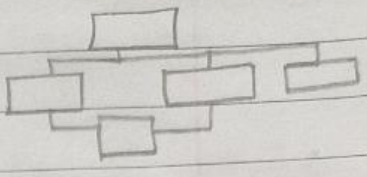
- Fast & easy retrieval
- Referential integrity
  changes in parent record is updated in child
- efficient for 1:N
- security enforced
  level of retrieval can be defined

Page No.
Date

record-based

Network Model (1969)

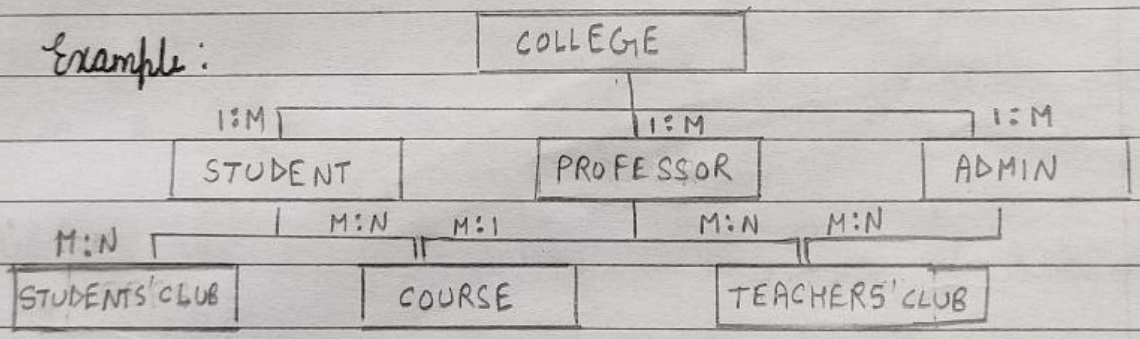
Charles Bachman



(graph structure) set form

- This is an extension in hierarchical model.
- It represents data in owner-member relationship where each owner may have multiple members and each member may have multiple owners
- It can describe one to one, M as well as 1, many to many rel.
- There can be more than one path from a previous node to successor nodes.
- connected through pointers.

Example:



Advantages

- Ability to manage more relationship types (M:N, 1:1, 1:M, M:1)
- Easy access to data (multiple paths)
- Data integrity (owner-member rel.)
- flexibility

Disadvantages

- System complexity
- Lack of structural indep.
- Complex algos for retrieval

Page No.
Date

record-based

Relational Model

Relational data model represent data by the collection of inter-related tables. Each table consists of rows and columns.

- Table / relation : It consists of rows & columns.
- Attribute / field / column : It defines the properties of the entity.
- Row / tuple / record (instance) : data is stored in records.
- Domain : set of values for a particular attribute in a relation.
- Relational schema : It contains the name of relation & all the attributes present in it.
- Relational key : It is an attribute / group of attributes which uniquely identifies a record in a relation.

Properties :
- order of columns/rows is immaterial
- A relation must not contain identical rows.
- Each attribute must & have a distinct name.
- A relation must have a unique name.
- Each record ~~must~~ attribute must have a single value corresponding to a record.
- A relation should contain key attribute.

Page No.
Date

Example :

PK | FK | FK | PK | Attribute

Record (instance) →

S_Id	S_Name	S_Age
S1	A	20
S2	B	18
S3	C	17
S4	D	19

Student

PK S_Id	B_Id
S1	B4
S2	B3
S3	B1
S4	B2

Issue

B_Id	Subject
B1	Math
B2	DBMS
B3	DSA
B4	Java

Book

Constraints

- Domain constr. : Integer , Character , Boolean , String etc.
- Key constr. :
  - Not null (*)
  - Unique
  - Primary key
  - Candidat key
  - Composite key
  - Alternat key
- Ref. integrity constr : foreign key.

Advantages : most popular
- Simple & easy to use
- hides physical storage details
- scalable
- redundancy reduced
- structural independence (no navigation)
- easy retrieval (using Sql)

Disadv.
- Hardware overhead
- Information Island phenomenon (a lot of info — redundancy, inconsist)
- failed to manage complex info. systems (multiple databases)

Some RDBMS : Mysql , Oracle , MS Access , Sql server
Language used : Sql

Languages :
ODL, SQL3

Page No.
Date

2015

object-based

Object-Oriented Model (set of objects)

- It represents data in the form of objects which contains some properties & behaviours.
- It supports OOPs concepts of classes & objects, inh., poly., encap.
- It allows storage of complex data such as images, audio & video etc.
- Data is not fragmented rather stored as a unit in objects. Each object is an independently functioning unit.
- As data is not fragmented, retrieval is easy.

instances of class

Objects are composed of:

- **Messages** : Messages acts as a communication medium b/w an object & outside world.
  - Read-only message : if invoked, doesn't change value of variables.
  - parameters Update message : if invoked, changes the value of variables.

- **Methods** : set of instructions which perform a particular task
  defines behaviour of object
  Executes when message is passed.
  - Read only method
  - update method

- **Variables** : data of an object which distinguishes it from other objects is stored in variables.
  defines state of object

Class blueprint of an object.
Consists of declarations of methods & variables.
inheritance, multiple inheritance, encaps etc

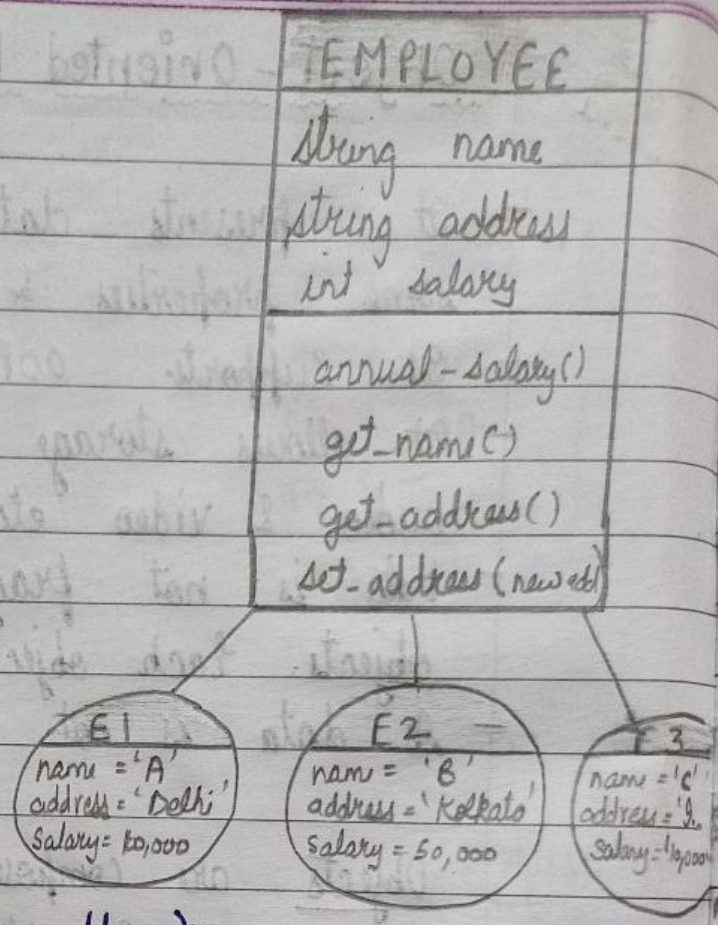
Page No.
Date

Class Definition

```
class employee {
// variables
    string name;
    string address;
    int salary;

// Messages & methods
    int annual-salary()
    string get-name();
    string get-address();
    int set-address(string new_address);

};
```

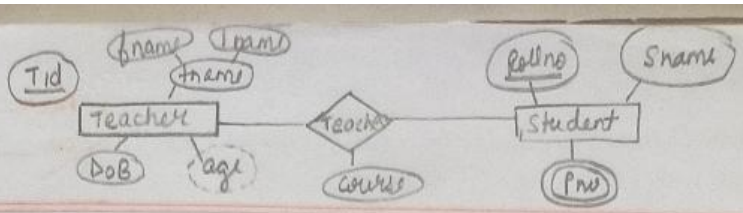


Advantages:

- allows storage of multimedia
- Query lang is not required for retrieval
- No ~~cons~~ primary keys
- Complex relationships can be easily implemented.

Disadv.

- Language dependent
- lack of structural independence
- lack of experienced programmers (new concept)



E-R MODEL (by Dr Peter Chen in 1976)

diagrammatic representation of database easy to understand even by non-techys

(Entity - Relationship Model)

- non technical design

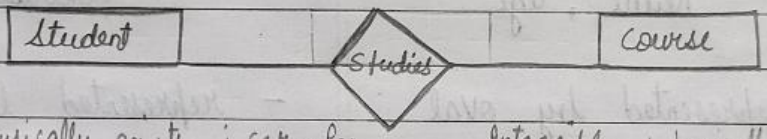
E-R model is a conceptual data model diagram which represents real-world entities and the relationships between them along with their attributes

- standard & logical way of visualising data

★ Entity : Entity is a real-world object having some properties (attributes) whose values distinguishes it from other object

eg: students, teachers, courses etc.

- It is represented by a rectangle

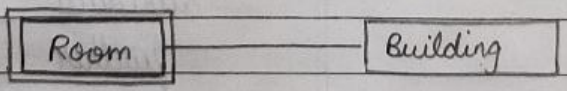


easy to convert in RDBMS

tangible : physically exists : car, Pen intangible : logically exists : Account

Weak entity : an entity which depends on another entity

- It doesn't contain any key attribute of its own
- represented by double rectangle



Entity set : set of same types of entities i.e. having same attributes of a particular entity type, int terms it is collection of all entities of a particular ent in database

Symbol representations :

entity : rectangle — Student

attributes : eclipse — name

relation : diamond — Study

★

Entity type	Entity set
No relation with time	Entity set change with change in time

Entity type is like class which decides the type & attributes of an entity

{Address_Phone}
Address {Phone}
Street add. | No | street | Ap no | city | state | zip
Area code | Phone NO

composite → ( )
multivalued → { }

eg: { Address_phone ({ Phone (Area_code, Phone_No)}, Address (Street_Address (No, Street, apartment_no), city, state, zip))}

Page No

★ Attributes

each attribute has its domain / value set (specified by datatype) (not specified in ER model)

SSN
Social Security No
In US, it is assigned to each individual

It is used to define property of an entity
eg: attributes of entity 'student': Name, Registration no. etc.

- For each attribute there is a set of permitted values called domains

Types of Attributes :

Single	Multivalued
- It can have only one value for an instance	- It can have multiple values for an instance
- eg: Registration No., Name, age	- eg: Name, age Mobile no., address (Permanent, Corresponding)
- represented by oval	- represented by double oval
age	address

multivalued attr can have constraints to restrict the no of values

a new table in RDBMS

ER model doesn't contain PK decided during modeling

Significance of comp attr
Composit attr are combo of simple ones. Their simple components are used in DBMS

Simple	Composite
- can't be further divided	- composed of many other attributes. Hence, can be further divided
- eg: age	- eg: name (first name, last name)
oval connected to entity (rect)	address (city, state, country)

Composit attr are useful in situations where a user sometimes refer to comp attr as a unit but at other times refer to its components

On RDBMS
Oval connected to oval

Stored	Derived
- Attribute which is directly taken as input and stored	- Attribute which can be derived from other attributes
- eg: date of birth	- eg: age (can be derived from DoB)
- represented by oval (DoB)	- represented by dotted oval (age)

for attributes which change cont.

In google maps, dist. stored - current loc & final loc

Relationship set: a set of similar kind of relations.

ER model represents entity set & relationship set.
& not individual entities / relationships
↓
in RDBMS

Page No
Date

Key	Non - Key
- Attribute which can't be repeated. Represents the main property of the entity - eg: Registration no., ID etc - represented by oval with text underlined Roll No	- Attribute which may or may not be repeated. - Name, age etc. - represented by oval Name

ER model doesn't contain PK included during mapping

* **Relationship** May have descriptive attributes

- It describes the relation or link (association) between entities of same / diff. entity set
- It is represented by a diamond.
- Weak relationship is represented by double diamond.

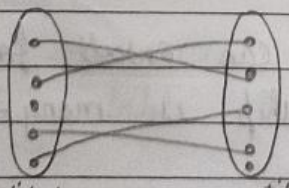
Relationship set: a set of similar kind of relations.

Types of relationships

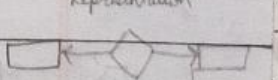
diff
Teacher — Teaches — St

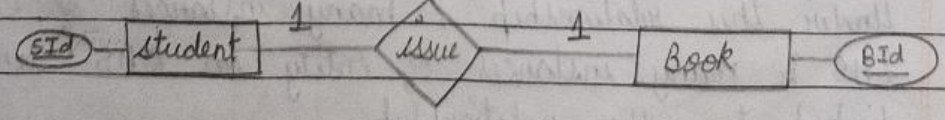
same
St — monitors — St

1) One - to - one Relationship :
- Under this relationship, only one instance of an entity can be linked with the relationship on both sides.

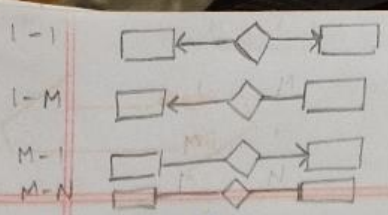


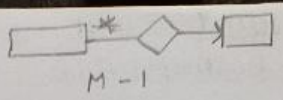
Representation



eg: 

If a student can issue only 1 book, then this will be a one - to - one relationship.

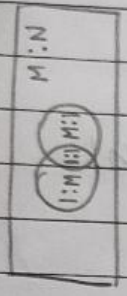


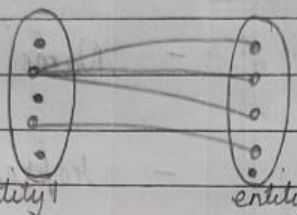


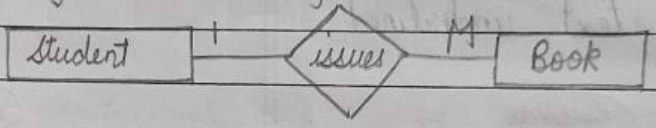
edge - one
star / null - many

2) One - to - many Relationship

Under this relationship, only one instance of ~~an~~ entity on LHS & many instances of entity on RHS can be linked to the relationship.



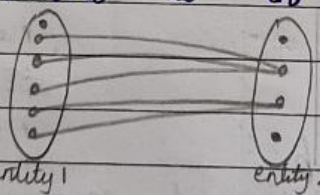


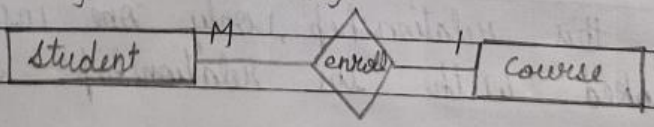
eg: 

If a student can issue multiple books at a time, then this relationship is one to many.

3) Many - to - one Relationship

Under this relationship, many instances of entity on LHS & only one instance of entity on RHS can be linked to the relationship.



eg: 

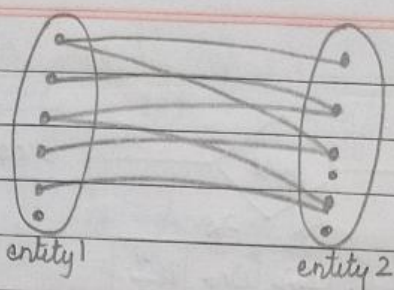
If a student can enroll for only 1 course, then this relationship is many - to - one.

4) Many - to - many Relationship

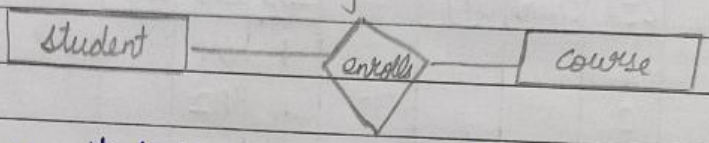
Under this relationship, many instances of entity on LHS & many instances of entity on RHS can be linked to the relationship.

* Every one - one is one + many
Every 1 - M & M - 1 is M - N
1 - many - one & many to many

Table - Entity set / schema
ea row - entity / instance



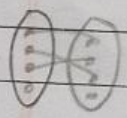
eg:

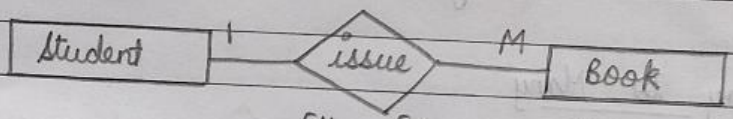


If a student can enroll for multiple courses at a time, then this relationship is many-to-many

Implementation of ER Models

→ **One-to-one**





PK

SId	SName	Class
1	-	-
2	-	-
3	-	-
4	-	-

describes student

FK FK

SId	BId
1	A2
2	A4
3	A1
4	A3

PK PK
describes issue details

PK

BId	BName	Sub
A1	-	-
A2	-	-
A3	-	-
A4	-	-

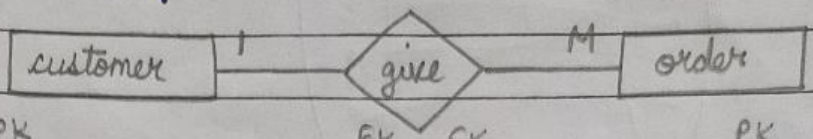
describes book

- In relationship table, SID as well as BId can be made Primary key
- We can merge any of the entity tables with relationship table to reduce them to 2 tables

SId	SName	Class	BId
1	-	-	A2
2	-	-	A4
3	-	-	A1
4	-	-	A3

BId	BName	Sub
A1	-	-
A2	-	-
A3	-	-
A4	-	-

One - to - many



PK

CId	CName	City
1	-	-
2	-	-
3	-	-
4	-	-

FK FK

CId	Ono	Date
1	A1	
1	A2	
1	A3	
2	A4	

PK

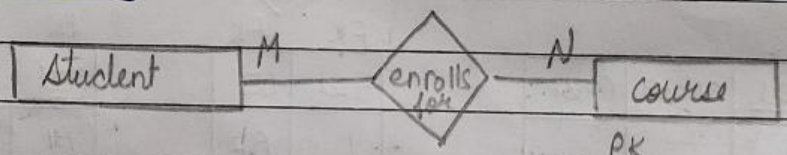
PK

Ono	OName	Cost
A1	-	-
A2	-	-
A3	-	-
A4	-	-

descriptive attribute

- ~~entity~~ Attribute of entity on 'many' side will serve as PK for relationship table
- Can be merged with table on 'many' side

Many to Many



PK

Roll	Name	Age
1	-	-
2	-	-
3	-	-
4	-	-

Roll	CId
1	C1
2	C1
1	C2
3	C4

Composit key

PK

CId	CName	Credits
C1		
C2		
C3		
C4		

- no merging

Null : no value

- Not applicable
- unknown
  - Missing eg height
  - not known whether value exists

M:N

one + many / many - one & many to many

M:N

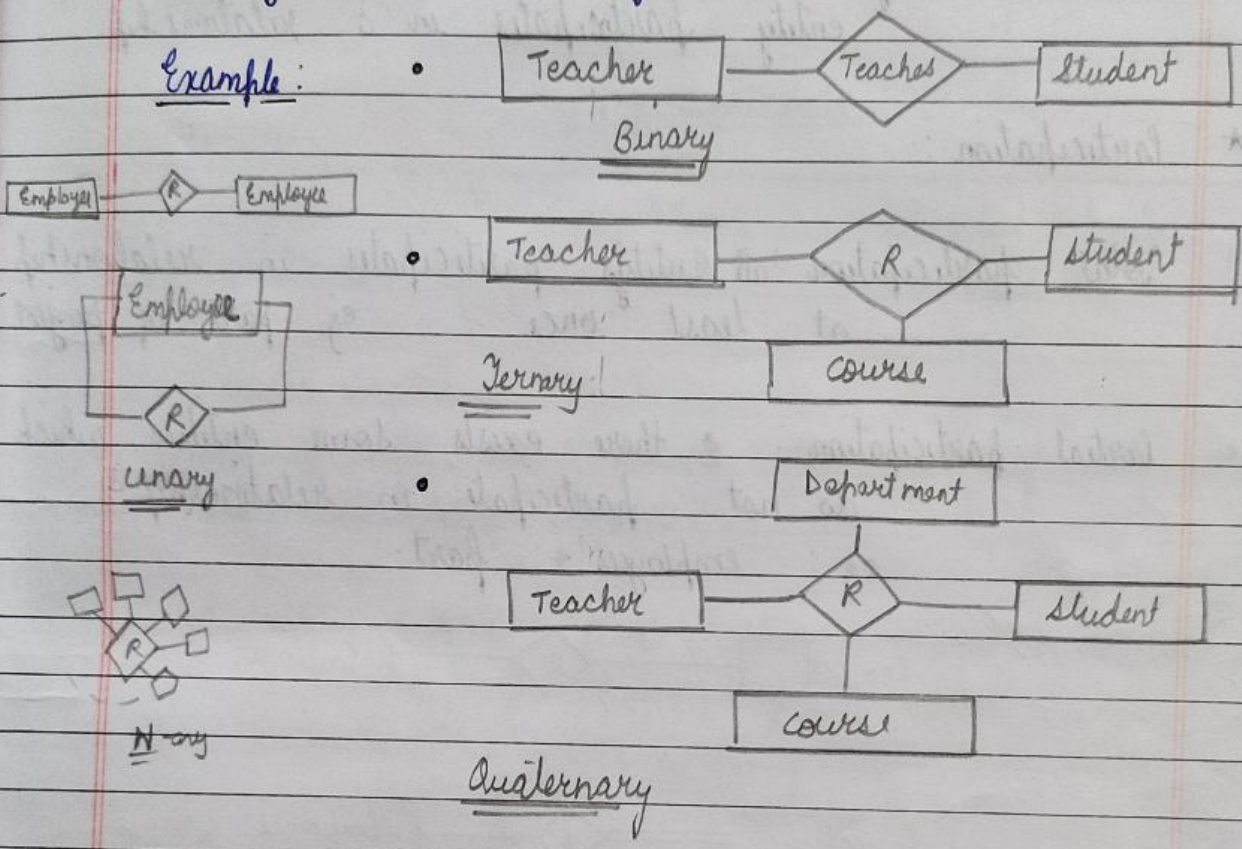
Every one - one is

Degree of a relationship

no. of entity set associated with the relationship set.

- usually it is binary 80-85%

Example:

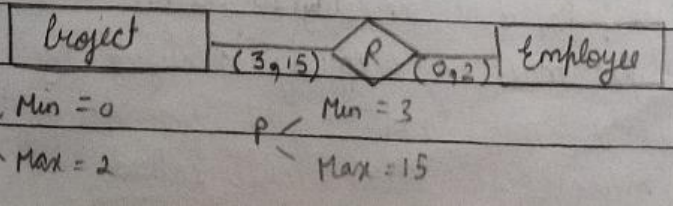


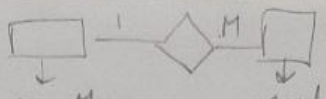
★ **Cardinality & Cardinality ratio**

Express the no. of entities to which other entity can be related via a relationship.

★ **Participation constraints**: specifies whether the existance of an entity depends on its being related to another entity via a relationship type.

- These constraints specify the max. & min. no. of relationship instance that each entity can/must participate in





(MinC, MaxC)

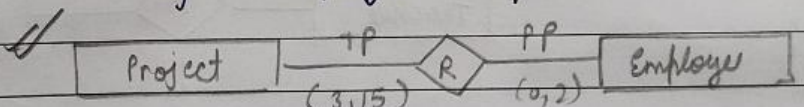
Max cardinality : It defines the max. no. of times an entity participates in a relationship

Min Cardinality : It defines the min. no. of times an entity participates in a relationship.

★ Participation :

minC ≥ 1 **Total participation** : all entities participates in relationship at least once eg part. of project

minC = 0 **Partial participation** : there exists some entities which do not participate in relationship
eg : employees' part.



or

→ In a rel. set, PKs of all the entity set are int and serve as SK for rel. set
→ Relationships must be uniquely identified by participating entities attr.
→ Relationships must not repeat for descriptive attributes
Rather make desc. attr. multi valued.

★ every one - one
many - one & many to many

INTEGRITY CONSTRAINTS

Integrity constraints are the set of rules applied on attributes of an entity.

Integrity Constraints

- Domain constraints
- Entity Integrity constraints
- Referential Integrity Constraints
- Key constraint

1) Domain Constraints
set of rules that restricts the kind of data stored in the attributes (domains).

Example : Integer, float, string, character, time, dat, timestamp, currency

	int	string	int	
	Roll	Name	Age	
	1	A	21	
	2	B	28	
not allowed ←	(B)	C	(A)	→ not allowed

Date

2) Entity integrity Constraints
It ensures the uniqueness of each record in a table.
Not Null: null values are not allowed in it.
Unique Key: It doesn't allow duplicacy
only 1 NULL value is allowed.

Primary key: uniquely identifies each record.
have distinct values + null values are not allowed.

PK unique

Id	Name	Age
1	NULL	20
2	B	21
not allowed ← NULL	(B)	17

not allowed

3) Referential Integrity Constraints
ensures that the relationship between two tables is valid.
Foreign key.

entityset → table
entity instance → row
column → Attribute / field
Row → tuple / record

keys must not change frequently

Page No.
Date

4) Constraints / Keys (basically, the rules applied on attributes)

every table should have a key

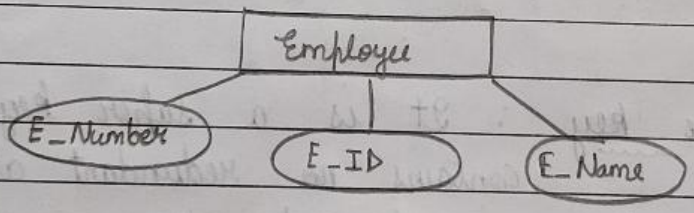
Key : Keys are ~~used~~ the set of attributes which uniquely identifies ~~two~~ each ~~by~~ tuple. $(Key)^+ = R$

- Keys are also used to establish relationship among the tables in a schema.

Types of keys

1) **Super key** : It is a single attribute or a set of attributes which uniquely identifies each tuple in a table.
(all the candidate keys and their combos)

eg:



(Name may repeat)

Possible super keys are:
- E_ID
- E_Number
- E_ID , E_Name
- E_Number , E_Name
- E_ID , E_Number
- E_ID , E_Number , E_Name

→ $R(A_1, A_2, A_3 \dots A_n)$

Imp → Find SKs from CKs
CKs from SKs

- Here A_1 is candidate key

no. of super keys = ?

A_1 — — — — — A_n $= 2^n$
2 2 2 2 $(n-1)$ 2

If a set of attr. is not a SK, none of its subsets is

Max no. of super keys = $2^n - 1$ (excluding ϕ, all the subsets)

suppose 5 attr. : ${}^5C_1 + {}^5C_2 + {}^5C_3 + {}^5C_4 + {}^5C_5 = 31$

All supersets of a CK are SK

R(A B C D)

1	1	5	1
2	1	7	1
3	1	7	1
4	2	7	1
5	2	5	1
6	2	5	2

SK = A, AB, AC, AD, ABC, ABD, ACD, ABCD

CK = A

PK = A

- A_1, A_2 are candidate keys

no. of super keys = ?

A_1 — $n-1$ — A_n $\quad 2^{n-1}$

A_2 — $n-1$ — A_n $\quad 2^{n-1}$

$A_1 A_2$ — $n-2$ — A_n $\quad 2^{n-2}$

$2^{n-1} + 2^{n-1} - 2^{n-2} = 3.2^{n-2}$

R(A B C)

1	1	1
2	1	2
3	2	1
4	2	2

SK = A, AB, AC, ABC, BC

CK = A, BC

PK = any one of CK

- $A_1 A_2$, $A_3 A_4$ are candidate keys

no. of super keys = ?

$A_1 A_2$ — $n-2$ — A_n $\quad 2^{n-2}$

$A_3 A_4$ — $n-2$ — A_n $\quad 2^{n-2}$

$A_1 A_2 A_3 A_4$ — $n-4$ — A_n $\quad 2^{n-4}$

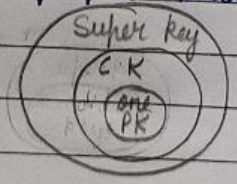
$2^{n-2} + 2^{n-2} - 2^{n-4}$

neither of a key whose proper subset is a SK

2) Candidate key : It is a super key that contains no redundant attributes.

- also called minimal super key.
- they are proper subset of set of super keys

(can have null values)



(nullx)

3) Primary key : It is a column or set of columns (attributes) in a table that uniquely identifies each tuple.

- It is subset of set of candidate keys
- It must not contain null values

PK = unique + not null

- A ~~table~~ relation can have only 1 PK.
- It is used to represent diff entity instances that is why it shouldn't contain null values.

It is generated & provided to the user

- Not null (indicated by *)
- unique (only 1 null allowed)

4) Composite key : It consists of more than one attribute to uniquely identify rows of a table.
eg: [Customer_ID , Order_ID] forms a composite key.

Referential
5) Foreign key : It is an attribute or set of attributes which points to (refers to) the primary key of same or another table.
- The table which contain the FK - referencing table
- _____ PK - referenced table
- Multiple FK can be present in a table.
- A PK in a particular referenced table can be referenced by multiple FK in referencing tables.

PK	SID	SName	SAge
	1	A	-
	2	B	-
	3	C	-
	4	D	-

referenced / base table

CID	SID	FK
C1	1	
C2	1	
C3	2	
C4	3	

referencing table

★ Foreign key maintains referential integrity (trust) checks for any ambiguity

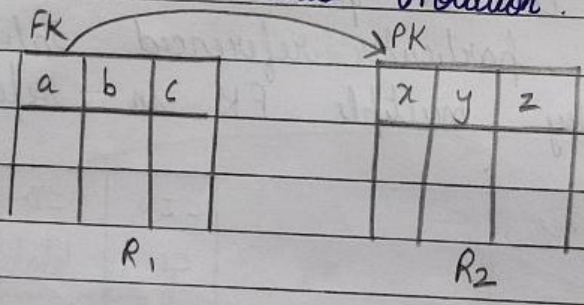
Referenced (base) table	Referencing table
- Insert	—
- Delete	must be deleted from here if not
- update PK	____ updated ____

otherwise integrity loss

- ~~Del~~ on delete cascade preferred
- on delete set Null (if FK happens to be PK → error)
- on delete no action generates error (by default)

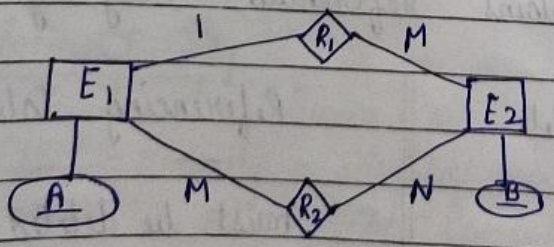
Referencing table	referenced table
- delete	—
- update	check first
- insert	check first

Ques: Let $R_1(a, b, c)$ and $R_2(x, y, z)$ be two relations in which a is FK in R_1 that refers to PK of R_2.
Consider 4 options:
a) insert into R_1 c) Delete from R_1
b) insert into R_2 d) Delete from R_2
which will cause violation.



Question:

→ What is the min. no. of tables req. to represent this ER model into relational model?
a) 2 b) 3 c) 4 d) 5



so
E_1, E_2 R_1, R_2

E_1, R_1, E_2
A A B B
E_1, R_1E_2

E_1, R_2, E_2
A A B B

$f: \alpha \rightarrow \beta$ means if α is given, u can search β
α can determine β
- follow rules of f(x)

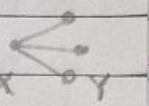
α can't repeat unique but not PK
if repeat then β also repeats

FUNCTIONAL DEPENDENCIES

If $\alpha \subseteq R$, $\beta \subseteq R$
& $\alpha \rightarrow \beta$
Then, $t_1[\alpha] = t_2[\alpha]$
$\Rightarrow t_1[\beta] = t_2[\beta]$

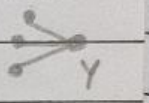
	α	β
t_1	a	b
t_2	a	b
	-	-

1 to many not allowed



α repeats

Many to 1 allowed



α	β
1	a
2	a

valid $\alpha \rightarrow \beta$

whether both values of β are same/not is determined by α

$\alpha \rightarrow \beta$ - α determines β or β is determined by α
(determinant, dependent)

FD is a relationship between two attributes, typically between PK & other non key attributes within a table.
If $X \rightarrow Y$ then X uniquely determines Y.

Types:

$X \rightarrow Y$

Trivial

no need to mention them

- Y is subset of X i.e. $Y \subseteq X$
  eg: $XY \rightarrow X$, $X \rightarrow X$
- Trivial functional dependencies are always valid / true.
- LHS $\cap$ RHS $\neq \phi$

Sid → Sid ($Sid \cap Sid = Sid$), Sid Sname → Sid ($Sid\,Sname \cap Sid = Sid$)

Non trivial

- Y is not a subset of X
  eg: $XY \rightarrow XYZ$ same trivial, $X \rightarrow Y$ non Trivial
- May or may not be valid.
- ~~LHS $\cap$ RHS $\neq \phi$~~ LHS $\cap$ RHS $= \phi$ non Trivial
  Sid → Sname

data depends on FD & not vice versa

For every X there must be only 1 Y

→ a) $A \rightarrow BC$

b) $DE \rightarrow C$

c) $C \rightarrow DE$

d) $BC \rightarrow A$

(check only repeated values of x)

R

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

If A is distinct AB is also distinct

- (If all are unique, It will be a FD certainly)
  $\propto \Rightarrow$ PK
- (If all values of B are same, It will be a FD)

→

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

a) $XY \rightarrow Z$ && $Z \rightarrow Y$

b) $YZ \rightarrow X$ && $Y \rightarrow Z$

c) $YZ \rightarrow X$ && $X \rightarrow Z$

d) $XZ \rightarrow Y$ && $Y \rightarrow Z$

→

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

a) $A \rightarrow B$ && $BC \rightarrow A$

b) $C \rightarrow B$ && $CA \rightarrow B$

c) $B \rightarrow C$ && $AB \rightarrow C$

d) $A \rightarrow C$ && $BC \rightarrow A$

CLOSURE METHOD

(Attribute closure / closure on attribute set / closure of attribute set)
Attribute closure of an attribute set A can be defined as a set of attributes which can be functionally determined by it
- Denoted by F^+

→ R(ABC)
$A \to B$, $B \to C$
$A^+ = ABC$
$B^+ = BC$

→ R(ABCDEF)
$A \to B$, $C \to DE$, $AC \to F$, $D \to AF$, $E \to CF$
$D^+ = DAFB$
$DE^+ = DAFBCE$

Armstrong Axioms

- Axiom is a statement that is taken to be true & serve as a premise or starting point for further arguments
- Armstrong axioms holds on every relational database & can be used to generate closure sets

Primary rules : (RAT rules)

- Reflexivity : If $Y \subseteq X$
  Then $X \to Y$
  eg : $ABC \to AB$
- Augmentation : If $X \to Y$
  Then, $XZ \to YZ$
  $XZ \to Y$ ✓ $XZ \to Z$ ✓
  ∴ $XZ \to YZ$
- Transitivity : If $X \to Y$ & $Y \to Z$
  Then $X \to Z$

Secondary rules :

- **Union** : If $X \to Y$ && $X \to Z$
  Then $X \to YZ$

  $AB \to CD$
  ↕
  $AB \to C$
  $AB \to D$
  but not
  ~~$A \to CD$ or $B \to CD$~~
  ↓
  $AB \to CD$ ✓
  but not required

- **Decomposition** : If $X \to YZ$
  Then $X \to Y$ && $X \to Z$

- **Pseudo trans** : If $X \to Y$ & $WY \to Z$
  Then $WX \to Z$

  $X \to Y \Rightarrow XW \to WY \Rightarrow WX \to Z$
  $X \to Y$, $Y\cancel{W} \to Z \to XW \to Z$

- **Composition** : If $X \to Y$ && $Z \to W$
  $XZ \to YW$

→ R(ABCDEFG)
$A \to B$, $BC \to DE$, $AEG \to G$.
$(AC)^+ = ACBDE$

→ R(ABCDE)
$A \to BC$, $CD \to E$, $B \to D$, $E \to A$
$B^+ = BD$

→ R(ABCDEF)
$AB \to C$, $BC \to AD$, $D \to E$, $CF \to B$
$(AB)^+ = ABCDE$

→ R(ABCDEFGH)
$A \to BC$, $CD \to E$, $E \to C$, $D \to AEH$, $ABH \to BD$, $DH \to BC$
$BCD \to H$?!
$(BCD)^+ = BCDEAH$ ✓

F & G are not on RHS so can't be obtained

Comparison of sets of FD

* Equivalence of functional Dependencies

→ R(ACDEH)

F:	G:
$A \to C$	$A \to CD$
$AC \to D$	$E \to AH$
$E \to AD$	
$E \to H$	
$A^+ = ACD$	$A^+ = ACD$
$(AC)^+ = ACD$	$E^+ = EADHC$
$E^+ = EAHCD$	
∴ $F \subseteq G$	∴ $G \subseteq F$

∴ $F = G$

a) $F \subseteq G$
b) $F \supseteq G$
c) $F = G$
d) $F \neq G$

→ R(PQRS)

Both

X:	Y:
$P \to Q$	$P \to QR$
$Q \to R$	$R \to S$
$R \to S$	
$P^+ = PQRS$	$P^+ = PQRS$
$R^+ = RS$	$Q^+ = Q$
$Y \subseteq X$	$R^+ = RS$
	$X \subseteq Y$

a) $X \subseteq Y$ ×
b) $Y \subseteq X$ ✓
c) $X = Y$ ×
d) $X \neq Y$ ×

→ R(ABC)

F:	G:
$A \to B$	$A \to BC$
$B \to C$	$B \to A$
$C \to A$	$C \to A$
$A^+ = ABC$	$A^+ = ABC$
$B^+ = BCA$	$B^+ = BAC$
$C^+ = ABC$	$C^+ = CAB$
$G \subseteq F$	$F \subseteq G$

$G = F$

Both R(VWXYZ)

F:	G:
$W \to X$	$W \to XY$
$WX \to Y$	$Z \to WX$
$Z \to WY$	
$Z \to V$	
$W^+ = WXY$	$W^+ = WXY$
$Z^+ = ZWYVX$	$(WX)^+ = WXY$
$G \subseteq F$	$Z^+ = ZWXY$
	$F \not\subseteq G$

$F \neq G$

- First remove Trivials
- Then LHS red
- RHS red
- whole

Page No.
Date

★ Minimal set / Canonical Cover / Irreducible set of FD

There can be redundancy on α side / β side / both sides
(extraneous attr) (extra FD)

First remove trivials

Remove α Red [1] Decompose all FD (to eliminate β side red.)

[2] Find closure for each time
1 - By including that FD
2 - By excluding that FD

Remove whole red [3] If closures in each case comes out to be the same
→ FD is redundant, eliminate it.
else
→ FD is essential

Remove β Red [4] Now, compute closure of LHS of FD having more than 1 attribute & find closure of attributes separately too by ignoring that & Remove acc.

Example: $R(WXYZ)$

$X \to W$

$WZ \to XY$

$Y \to WXZ$

① Decompose:

$X \to W$ ✓

~~$WZ \to X$~~ ~~X~~

$WZ \to Y$ ✓

~~$Y \to W$~~ ~~X~~

$Y \to X$ ✓

$Y \to Z$ ✓

②

$X^+ = XW$ — $X^+ = X$

$(WZ)^+ = WZXY$ — $(WZ)^+ = WZYX$

" — $(WZ)^+ = WZ$

$Y^+ = YWXZ$ — $Y^+ = YXZW$

" — $Y^+ = YZ$

" — $Y^+ = YXW$

③ So $X \to W$
$WZ \to Y$
$Y \to X$
$Y \to Z$

④ $(WZ)^+ = WXYZ$
$W^+ = W$
$Z^+ = Z$

Canonical form: $X \to W$
$WZ \to Y$
$Y \to XZ$

- There may be multiple canonical forms for the same set of FD
- Depends on order

If red. occurs due to 2 FD we remove the first one & keep 2nd one
so if order is reversed 2nd one is kept

→ R(ABC) $AB \to C$, $C \to AB$, $B \to C$, $ABC \to AC$, $A \to C$, $AC \to B$

~~$AB \to C$~~, $C \to A$, ~~$C \to B$~~, $B \to C$, ~~$ABC \to A$~~, ~~$ABC \to C$~~, $A \to C$, $AC \to B$

$AB^+ = ABC$ $C^+ = CAB$ $C^+ = CAB$ $B^+ = BC$ $A^+ = ACB$ $AC^+ = ABC$

$AB^+ = ABC$ $C^+ = CB$ $C^+ = CAB$ $B^+ = B$ $A^+ = A$ $AC^+ = AC$

$C \to A$, $B \to C$, $A \to C$, $AC \to B$

$AC^+ = ABC$ $A^+ = ACB$ $C^+ = CAB$ $\therefore A \to B$

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

→ R(ABCD)

$A \to B$ $AC \to D$

$C \to B$

$D \to ABC$

$A \to B$ ✓	$A^+ = AB$	$A^+ = A$
$C \to B$ ✓	$C^+ = CB$	$C^+ = C$
$D \to A$ ✓	$D^+ = DABC$	$D^+ = DBC$
~~$D \to B$~~ ✗	"	$D^+ = DCAB$
$D \to C$ ✓	"	$D^+ = DAB$
$AC \to D$ ✓	$(AC)^+ = ACBD$	$(AC)^+ = ACB$

$(AC)^+ = ABCD$ ✓

$A^+ = AB$ ✓

$C^+ = CB$ ✓

Canonical ~~to~~ cover (minimal set) : $A \to B$, $C \to B$, $D \to AC$, $AC \to D$

→ R(VWXYZ)

$V \to W$

$VW \to X$

$Y \to VXZ$

$V \to W$ ✓	$V^+ = VWX$	$V^+ = V$
$VW \to X$ ✓	$(VW)^+ = VWX$	$(VW)^+ = VW$
$Y \to V$ ✓	$Y^+ = YVXZW$	$Y^+ = YXZ$
~~$Y \to X$~~ ✗	"	$Y^+ = YVZWX$
$Y \to Z$ ✓	"	$Y^+ = YVXW$

$(VW)^+ = VWX$ $V^+ = VWX$ ✓ $W^+ = W$

$V \to X$

Minimal Cover : $V \to WX$ ~~$V \to X$~~ $Y \to VZ$

Superset of SK is always a SK
subset of SK can be a SK
superset of CK or subset of CK can never be a CK
Superset of CK is always a SK
Subset of CK can never be a SK

★ To find SK, CK

→ R (A B C D)

AB → C , C → BD , D → A

A, B, C, D — (C)

AB, AC, AD, BC, BD, CD — (AB) , (AC) , BC , (BD) , CD

ABC, ABD, ACD, BCD — ABC , ABD , ACD , BCD

ABCD — ABCD

→ attributes which are not there on RHS must be there in CK

→ Once u get a CK , further combos must not contain it

Procedure : ① write all the attr which are not on RHS (if not prefer alternate)
② Start making combos by inc 1 at a time
(if u get ϕ in step ① start with single attr)
③ check if its closure is entire set of attr for each combo
If yes → it is a CK
no → use it for further combos
④ In next level of combos first check whether it contains the obtained CK if yes → leave it
no → check —

→ R (A B C D)

A → B , B → C , C → A

D must be there

$D^+ = D$ X

$(AD)^+ = ADBC$ ✓

$(BD)^+ = BDCA$ ✓

$(CD)^+ = CDAB$ ✓

→ R (A B C D)

3 - 4 combos

AB → CD , D → A

B must be there

$B^+ = B$ X

$(AB)^+ = ABCD$ ✓

$(BC)^+ = BC$ X

$(BD)^+ = BDAC$ ✓

→ R(ABCDEF)

AB→C, C→DE, E→F, D→A, C→B

$A^{\times}$ $B^{\times}$ (C) = DEBFA $D^{\times}$ $E^{\times}$ $F^{\times}$

(AB) $AD^{\times}$ $AE^{\times}$ $AF^{\times}$ (BD) $BE^{\times}$ $BF^{\times}$ $DE^{\times}$ $DF^{\times}$ $EF^{\times}$

$ADE^{\times}$ $ADF^{\times}$ ADB $AEF^{\times}$ ABE ABF $BEF^{\times}$ BDE BDF $DEF^{\times}$

$ADEF^{\times}$ ADEB ADFB AEFB BDEF

10 combos E_3

5 combos E_4

→ R(ABCDEF)

AB→C, C→D, B→AE

B, F must be +nt

$(BF)^+ = FBAECD$ ✓

all other combos are red.

→ R(ABCD)

AB→CD, D→B, C→A

AB, AD, BC, CD

→ R(ABCD)

AB→CD, C→A, D→B

$A^+ = A$ × $B^+ = B$ × $C^+ = CA$ × $D^+ = DB$ ×

$(AB)^+ = ABCD$ ✓ $(AC)^+ = AC$ × $(AD)^+$ ~~✓~~ $(BC)^+ = BCAD$ ✓, $BD^{\times}$, $(CD) = CDBA$ ✓

→ R(ABCDE)

AB→CD, D→A, BC→DE

B must be +nt

single pair

$B^+ = B$ ×

$(AB)^+$ ✓ BC ✓ BD ✓ BE ×

B must be there ∴ AB_ × BC_ × BD_ × BE_ (combi)

3 = $(BEA)^+$ × $(BEC)^+$ × $(BDE)^+$ ×

B must be makes combos of 4 leaving 1 oth 1by1 except B

4 = ABCD × ABCE × ABDE × BCDE ×

∴ AB, BC, BD

→ R(WXYZ)

Z→W, Y→XZ, XW→Y

X^+ X $Y^+ = YXZW$ ✓ $W^+ = X$ $Z^+ = ZW$ ×

WX ✓ WZ × XZ = XZWY ✓

Y & XZ

→ R(ABCDEF)

AB→C, DC→AE, E→F B, D

$BD^+ = BD$ × ABD ✓ BDC ✓ BDE × BDF ×

BDEF ×

Don't make their combos further only for these

If essentials form CK no need to proceed further

→ R(A B C D E)

CE → D , D → B , C → A

C , E must

$(CE)^+$ = CEDBA ✓

AB, AC, AD

ABC, ABD

AB

→ R(AB C D E F G H I J)

AB → C , AD → GH , BD → EF , A → I , H → J

A B D must

$(ABD)^+$ = ABDC GH EF I J ✓

→ R(ABCDE)

A → B , BC → E , DE → A

C D must

$(CD)^+$ = CD ✗

$(ACD)^+$ = ACDBE ✓ $(BCD)^+$ = BCDEA ✓ $(ECD)^+$ = ECDAB ✓

→ R(ABCDE)

BC → ADE

D → B

C must

AC ✗ (BC) ✓ (ED) = CDBAE ✓ CE ✗

ABC ✗ ACD ✗ ACE ✗ CDE ✗ BCE

ABCE ✓ ACED ✗

→ R(ABCDEF)

AB → C , C → D , D → BE , E → F , F → A

A ✗ B ✗ $(C)^+$ = CDBEFA ✓ $(D)^+$ = DBEFAC ✓ E = EFA ✗ F ✗

AB ✓ AE ✗ AF ✗ BE ✓ BF ✓ EF ✗

AEF ✗

→ R(ABCDEFGH)

CH → G , A → BC , B → CFH , E → A , F → EG

D ✗ | $(AD)^+$ = ABCDFHEG ✓ $(BD)^+$ = BCFHDEGA ✓ (CD) ✗ (DE) ✓

DF ✓ DG ✗ DH ✗

CDG ✗ CDH ✗ HDG ✗ | CDGH ✗

Alternate

R(ABCD)

AB → CD , D → B , C → A

AB ∈ D

↓ discard

SK AB⁺ = ABCD

A⁺ = A ✗

B⁺ = B ✗ AB is CK

Prime attr = A, B

CK
AB

Check → if PA are on RHS

- A in C → A
- B in D → B

Replace in AB

BC
B⁺ X
C⁺ = CA ✗

BC is CK

AD
A⁺ X
D⁺ DB ✗

AD is CK

Prime attr = A, B, C, D

CK
AB
BC
AD
CD

Check →

- AB → C AB → D

Replace in BC

CD AB ✓
C⁺ X
D⁺ X

CD is CK

Replace in AD

CD ✓ , AB ✓

→ R(ABCDEF)

AB → C , C → DE , E → F , D → A , C → B

~~AB C D E F~~

AB

A^+ x

B^+ x

AB is CK

D → A

C → B

~~AB~~ ~~AC~~

D^+ x A^+ x

B^+ x C^+ ✓

BD is CK **C is CK**

AB → C C → D

~~CD~~ AB ✓

C^+ = ✓

PA = A, B, C, D

CK
AB
~~BD~~
C

multiple entity data in 1 table

Adv. remove anomalies
size ↓
speed ↑
simple queries
OLTP systems
reduce red/inco

Problem Redundancy

Idea: In the table Student_Info we tried to store entities data about student. (in 1 table)

Result: Entire branch data of a branch must be repeated for every student of the branch

unnecessary repetition

Key

Sid	Name	Age	BrCode	Branch	HOD_Name
1	A	18	101	CS	ABC
2	B	19	101	CS	ABC
3	C	18	101	CS	ABC
4	D	21	102	EC	PQR
5	E	20	102	EC	PQR
6	F	19	103	ME	XYZ

Student_info

Redundancy: When some data is stored multiple times unnecessarily in a database

Disadvantages
- Insertion, deletion & modification anomalies
- Inconsistency of data
- Inc in database size & access time (slow)

suppose we change HOD of CS but due to some issues it is not updated at all places

- Insertion anomalies: When certain data (attribute) cannot be inserted into database without the presence of other data
  eg: Branch data can't be inserted without student data i.e. if a branch doesn't contain any student, it doesn't exist in the database.

- Deletion anomalies If you want to delete some unwanted data & it causes deletion of necessary data
  eg: If we delete data of student with Sid = 6, all data of ME branch vanishes from the table bcz F was the only St. in ME branch.

- Modification anomalies (updation) If we want to update a single piece of data but to do this we need to update all its copies.

OLTP require norm. OLAP - not req.

Process of removing redundant data from your tables to improve storage efficiency, data integrity & scalability. speed too. usually involves decomposition of tables.

Decomposition

NORMALISATION

Diff. entities must be stored in diff. tables

Key

Sid	Name	Age	Br_Code
1	A	18	101
2	B	19	101
3	C	18	101
4	D	21	102
5	E	20	102
6	F	19	103

Student_Info

Key

Br_Code	Branch	HOD_Name
101	CS	ABC
102	EC	PQR
103	ME	XYZ

Branch_Info

FIRST NORMAL FORM

(when ERD is mapped into relations → It is in INF)

Multival attr → split row sometimes column

Composite attr → split column

Each multival attr will have a seperate table

Take multi columns for multival attr (not preferd)

- Values of all attributes must be atomic (single)
- Every column must have unique name. & rows → unique
- Order of rows & columns is irrelevant
- all values of a particular column must be from same domain.
- A relation must have a PK

not in INF

PK

Roll	Name	Course
101	Modi	CN/OS
102	Sonia	DBMS/CO

↓

Roll	Name	Course
101	Modi	CN
101	Modi	OS
102	Sonia	DBMS
102	Sonia	CO

PK = Roll, Course

SECOND NORMAL FORM

For a table to be in 2NF

- It must be in INF
- It must not contain partial dependencies.

eg R(A B C D)

CK = AB

prime attributes = A, B

non prime = C, D

AB → D But B → C (partial)

To find whether a table is in 2NF
when it is given that table is in 1NF

- Find all CK
- Find prime & non prime attributes
- Non prime attributes must not be involved in partial dependencies.

→ Major problem with partial dependencies

A	B
1	–
–	2
3	–
5	4

AB combined can find D
But B alone can't find C as it contains null values

→ To solve → decompose:

$R(\underbrace{ABC}_{\text{CK}}D) \rightarrow R_1(\underbrace{AB}_{\text{CK}}D)$ & $R_2(BC)$

(This must contain CK)

→ R(ABC) 1NF ✓

B → C

Candidate keys:

$(AB)^+ = ABC$

So AB is CK

Prime attributes : A, B

Non prime — : C

B → C is partial dep.

not in 2NF

R(ABC) → R_1(AB) & R_2(BC)

2NF ✓ 2NF ✓

B → C

R

A	B	C
a	1	x
b	2	y
a	3	z
c	3	z
d	3	z
e	3	z

R_1

A	B
a	1
b	2
a	3
c	3
d	3
e	3

R_2

B	C
1	x
2	y
3	z

→ R(ABCDE)

$AB \to C$, $D \to E$

- Candidate keys:

$(ABD)^+ = ABDCE$

∴ ABD is the CK

- Prime attributes : A, B, D
- Non prime — : C, E

Part of CK ← $AB \to C$ (NP) is partial dep.

Part of CK $D \to E$ (NP) is also ——

∴ It is not in 2NF

To convert in 2NF:

$R(ABCDE) \to R_1(ABC), R_2(DE), R_3(ABD)$

$AB \to C$ $D \to E$

→ R(ABCDE)

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

- Candidate keys:

$(AC)^+ = ACBDE$

∴ AC is CK

- Prime attributes : A, C
- Non prime — : B, D, E
- ~~A~~ Partial dependencies:

$A \to B$, $C \to D$

∴ It is not in 2NF

To convert into 2NF

$R(ABCDE) \to R_1(ABE), R_2(CD), R_3(AC)$

To convert into 2NF
- while decomposition
- make a table of CK & attr dep on it
- Partial dep. If NP attr determine some other attr, u may include it in the table

→ R(ABCDEFGHIJ)

$AB \to C$, $AD \to GH$, $BD \to EF$, $A \to I$, $H \to J$

CK: $(ABD)^+ = ABDCGHEFIJ$ ⇒ ABD is CK

PA = A, B, D , NPA = C, E, F, G, H, I, J

PD = $AB \to C$, $AD \to GH$, $BD \to EF$, $A \to I$,

Dec.: $R_1(ABC)$, $R_2(ADGH)$, $R_3(BDEF)$, $R_4(AI)$, $R_5(ABD)$

Partial dep. → proper subset of CK → NP
Transitive dep: → NP → NP
So α = Super Key $\beta \Rightarrow$ NP

THIRD NORMAL FORM

NP → NP

Transitive Dependency:
A FD from $\alpha - \beta$ is called transitive if $\alpha, \beta \in$ Non prime

R(ABC)
A → B, B → C
It is in 2NF

Conditions:
A relation R is in 3NF if
- It is in 2NF
- It has no transitive dependency

$R(ABC) \rightarrow R_1(BC), R_2(AB)$

Alternative:
For every dep. from $\alpha \rightarrow \beta$
- either α is super Key or
- β is a prime attribute

R

A	B	C
a	1	x
b	1	x
c	1	x
d	2	y
e	2	y
f	3	z
g	3	z

R_1

A	B
a	1
b	1
c	1
d	2
e	2
f	3
g	3

R_2

B	C
1	x
2	y
3	z

→ R(ABCDEFGH)
ABC→DE, E→GH, H→G, G→H, ABCD→EF
BCNF✓ so is in 3NF

⇒ R(ABCD) AB → CD AC → BD ~~BC → D~~ 9mp
CK = AB, AC

★ union of proper subsets of diff CKs is treated as proper subset of CK

→ R(A, B, C, D, E, F)
AB → CDEF BD → F

But if it was ABD → F it is in BCNF so it is in 3NF

CK: AB
PA = A, B NPA = C, D, E, F

2NF: AB → CDEF, BD → F
(AB: not part of CK; BD: not part of CK)

3NF: AB → CDEF, BD → F
(AB: PA; BD: NPA NPA)

→ R(ABCDE)

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

CK : $(AC)^+ = ABCDE$ ∴ AC is CK

PA = A, C NPA = B, D, E

$R_1(\underline{A}BE)$ $A \to B$, $B \to E$

↓

$R_{11}(\underline{A}B)$ $A \to B$

$R_{12}(\underline{B}E)$ $B \to E$

$R_2(\underline{C}D)$ $C \to D$

$R_3(\underline{AC})$

→ R(ABCDEFGHIJ)

$AB \to C$, $A \to DE$, $B \to F$, $F \to GH$, $D \to IJ$

CK : $(AB)^+ = ABCDEFGHIJ$ ⇒ AB is CK

PA = A, B NPA = C, D, E, F, G, H, I, J

$R_1(\underline{ADE}IJ)$ $A \to DE$, $D \to IJ$

↓

$R_{11}(\underline{AD}E)$ $A \to DE$

$R_{12}(\underline{D}IJ)$ $D \to IJ$

$R_2(\underline{BF}GH)$ $B \to F$, $F \to GH$

↓

$R_{21}(\underline{BF})$ $B \to F$

$R_{22}(\underline{F}GH)$ $F \to GH$

$R_3(\underline{ABC})$ $AB \to C$

→ R(ABCDE)

$AB \to C$, $B \to D$, $D \to E$

CK : $(AB)^+ = ABCDE$ AB is CK

PK → (B)

$R_1(\underline{B}DE)$ $B \to D$, $D \to E$

↓

$R_{11}(\underline{B}D)$ $B \to D$

$R_{12}(\underline{D}E)$ $D \to E$

$R_2(\underline{AB}C)$ $AB \to C$

→ R(ABCDEFGHIJ)

AB → C , AD → GH , BD → EF , A → I , H → J

CK: $(ABD)^+$ = ABCDEFGHIJ ABD is CK

R_1 (AB CI)
AB → C , A → I

R_{11} (ABC)
AB → C

R_{12} (AI)
A → I

R_2 (ADGHJ)
AD → GH, H → J

R_{21} (ADGH)
AD → GH

R_{22} (HJ)
H → J

R_3 (BDEF)
BD → EF

R_4 (ABD)

Boyce Codd Normal form

BCNF (advanced 3NF) P/NP → P P shouldn't be of RHS

handles Problems of overlapping CKs AB, BC, CD

Eg: R(ABC)

AB → C , C → B

A^+ X $(AB)^+$ = ABC ✓ AC^+ = ABC ✓

2NF ✓ 3NF ✓

X BCNF

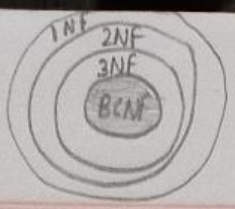
PA = A, B, C NPA = X

Condition : α should be super key.

→ It says prime attributes shouldn't be determined by any non prime or part of CK.

Rules : • It must be in 3NF
• Prime attributes shouldn't be determined by any non prime attribute or part of CK.
or α → SK
bcz if α = SK 3NF ✓ & α ≠ NP or part of CK

If all CK are simple (single attr.) it will be in 2NF
If all attr. of a relation are prime attr., it will be in 3NF
X If relation is in 3NF & all CKs are simple, it will be in BCNF
R(ABCD) Let AB & C are CK & one of FD is C → B 3NF ✓ CK are simple but still not in BCNF



→ R(ABC)

$AB \rightarrow C$ ✓ , $C \rightarrow B$ ✗

CK = AB, AC

3NF ✓ BCNF ✗

A	B	C
a	1	x
b	2	y
c	2	z
c	3	w
d	3	w
e	3	w

R_1 (CB)

R_2 (AC)

One table should contain any one of the CKs

A	C
a	x
b	y
c	z
c	w
d	w
e	w

C	B
x	1
y	2
z	2
w	3

R_2 (AC) — C is common, C is PK

R_3 (AB) — B is common, but B isn't PK

Problems : A relational table is always in 1NF

Check for BCNF → 3NF → 2NF → 1NF

- BFNF → α is SK
- 3NF → $\alpha \rightarrow$ SK or β is P
- 2NF → check for partial dep
- 1NF → always ✓

→ R(ABCDEFGH)

$AB \rightarrow C$ ✓ , $A \rightarrow DE$ ✗ , $B \rightarrow F$ ✗ , $F \rightarrow GH$ ✗

CK : $(AB)^+$ = ABCDEFGH AB is CK

BCNF ✗ 3NF ✗ 2NF ✗ 1NF ✓

→ R(ABCDE)

$CE \rightarrow D$ ✓ , $D \rightarrow B$ ✗ , $C \rightarrow A$ ✗

CK : $(CE)^+$ = ABCDE CE is CK

BCNF ✗ 3NF ✗ 2NF ✗ 1NF ✓

* → R(ABCDEF)
AB → C , DC → AE , E → F
CK: (BD)+ = BD
(ABD)+ = ABCDEF ✓ (BCD)+ = BCDAEF ✓
(BDE)+ = BDEF ✗ (BDF)+ = BDF ✗
(BDEF)+ = BDEF ✗
So ABD & BCD are CKs
BCNF ✗
3NF ✗
2NF ✗
1NF ✓

→ R(ABCDEFGHI)
AB → C , BD → EF , AD → GH , A → I
CK: (ABD)+ = ABCDEFGHI ✓ ABD is CK
BCNF ✗ 3NF ✗ 2NF ✗ 1NF ✓

→ R(ABCDE)
AB → CD D → A , BC → DE
CK: B ✗
(AB)+ = ABCDE ✓ (BC)+ = ABCDE ✓ (BD)+ = BDACE ✓ (BE)+ = BE ✗
AB, BC, BD are CK
BCNF ✗ 3NF ✓ 2NF ✓ 1NF ✓

→ R(ABCDE)
BC → ADE , D → B
CK: C ✗
(AC)+ = AC ✗ (BC)+ ABCDE ✓ (CD)+ = CDBAE ✓ CE ✗
(ACE)+ = ACE ✗
BC & CD are CK
BCNF ✗
3NF ✓
2NF ✓
1NF ✓

→ R(VWXYZ)
X → YV Y → Z , Z → Y , VW → X
CK: W ✗ ,
(XW)+ = XYVZW ✓ (VW)+ = VWXYZ ✓
(YW)+ = YWZ ✗ (ZW)+ = ZWY ✗
(YWZ)+ = YWZ ✗
XW & VW are CK
BCNF ✗
3NF ✗
2NF ✗
1NF ✓

attaining BCNF is too expensive

Theoritical concept

3NF is far too good

2NF too is OK

→ R(ABCDEF)

ABC → D , ABD → E , CD → F , CDF → B , BF → D

CK : $(AC)^+ = AC$ ✗

PA = A, B, C, D

NPA = E, F

$(ABC)^+ = ABCDEF$ ✓ $(ACD)^+ = ACDFBE$ ✓

$(ACE)^+ = ACE$ ✗ $(ACF)^+ = ACF$ ✗

$(ACEF)^+ = ACEF$ ✗

ABC & ACD are CKs

BCNF ✗ 3NF ✗ 2NF ✗ 1NF ✓

→ R(ABC)

A → B , B → C , C → A

BCNF ✓ 3 ✓ 2 ✓ 1 ✓

CK : $A^+ = ABC$ ✓ $B^+ = BCA$ ✓ $C^+ = CAB$ ✓

→ R(ABCDEF)

A → BCDEF , BC → ADEF , DEF → ABC

BCNF ✓ 3NF ✓ 2NF ✓ 1NF ✓

CK = $A^+ = ABCDEF$ ✓ $B^+ = C^+ = D^+ = F^+$ ✗

$(BC)^+ = ABCDEF$ ✓ BD ✗ BE ✗ BF ✗ CD ✗ CE ✗ CF ✗ DE ✗ DF ✗ EF ✗

BDE ✗ BDF ✗ BCD ✗ BEF ✗ BCE ✗ BFC ✗ CDE ✗ CDF ✗ CFE ✗ DEF ✓

BDEF BDEC BDFC BEFC CDEF

∴ A, BC & DEF are CK

→ R(ABC)

AB → C , C → A

BCNF ✗

3NF ✓

2NF ✓ 1NF ✓

CK : $B^+ = X$ AB ✓ BC ✓

→ R(ABCDE)

A → B , BC → E , DE → A

BCNF ✗

3NF ✓

2NF ✓

1NF ✓

CK : CD ✗

CDA → ABCDE ✓ BCD = BCDEA ✓ CDE = ACDEB ✓

ACD, BCD & CDE are CK.

→ R(ABCDE) BCNF ✗ 3NF ✓

AB → CD , D → A , BC → DE

CK : A ✗ B ✗ C ✗ D ✗ E ✗

AB ✓ AC ✗ AD ✗ AE ✗ BC ✓ BD ✓ BE ✗ CD ✗ CE ✗ DE ✗

ACD ✗ ACE ✗ ABE ✗ ADE ✗ ADB ✗ ~~ACF~~ BEC BDE CDE ✗ ACDF

→ R(WXYZ) BCNF ✗ 3NF ✓

Z → W , Y → XZ , XW → Y ,

CK : W ✗ X ✗ Y ✓ Z ✗

WX ✓ WZ ✗ XZ ✓

→ R(ABCDE) BCNF ✗ 3NF ✗ 2NF ✗ 1NF ✓

A → B , B → E , C → D

CK : AC = ACDBE ✓

→ R(ABCDEF) BCNF ✗ 3NF ✗ 2NF ✗ 1NF ✓

AB → C , DC → AE , E → F

CK : BD ✗ , (ABD)⁺ = ABDCE ✓ (BCD)⁺ = BCDAE ✓ BDE ✗ BDF ✗

~~ABDE~~ ~~ABDF~~ ~~BDEF~~ BDEF ✗

ABD & BCD

→ R(VWXYZ) BCNF ✗ 3NF ✗ 2NF ✗ 1NF ✓

Z → Y , Y → Z , X → YV , VW → X

CK : W ✗ (VW)⁺ = VWXYZ ✓ (WX)⁺ = WXYVZ ✓ (WY)⁺ = WYZ ✗

(WZ)⁺ = WZY ✗

WYZ = WYZ ✗

→ R(ABCDEF) BCNF ✗ 3NF ✗ 2NF ✗ 1NF ✓

ABC → D , ABD → E , CD → F , CDF → B , BF → D

CK : (AC)⁺ = ✗ ABC ✓ ACD ✓ ACE ✗ ACF ✗

ACEF ✗

→ R (A B C D E F)

C → F , E → A , EC → D , A → B

CK : $(CE)^+ = CEFADB$

BCNF ✗
3rd ✗
2NF ✗
1NF ✓

→ R (A B C D E H)

A → B , BC → D , E → C , D → A

CK : $(EH)^+ = EHC$

AEH BEH CEH DEH

~~BEHC~~

∴ EH , AEH , DEH , BEH

BCNF ✗
3rd ✗
2NF ✗
1NF ✓

→ R (A B C D E P G)

AB → CD , DE → P , C → E , P → C , B → G

CK : $(AB)^+ = ABCDEPG$

BCNF ✗
3NF ✗
2NF ✗
1NF ✓

→ R (A B C D E F G H)

CH → G , A → BC , B → CFH , E → A , F → EG

CK : D

AD ABCDFHEG ✓ ~~AB~~ BD BDCFHGEA ✓ CD ✗ ED ABCDEF ✓ FD ABCD FHEG ✓ GD ✗ HD ✗

CDG ✗ CDH ✗ GDH ✗

CDGH ✗ ~~CD~~

∴ CK = AD , BD , ED , FD

BCNF ✗ 3NF ✗ 2NF ✗

→ R (A B C D)

Here CK is not Composite so 2NF ✓

A → B , B → C , C → BD

$(A)^+ = ABCD$ ✓

BCNF ✗
3NF ✗
2NF ✓
1NF ✓

→ R (A B C D E F)

AB → CD , CD → EF , BC → DEF , D → B , CE → F

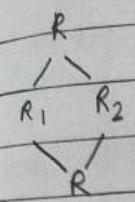
(A) AB ✓ AC ✗ AD ✓ AE ✗ AF ✗

ACE AcF AEF

AC EF ✗

BCNF ✗ 3NF ✗ 2NF ✓ 1NF ✓

DECOMPOSITION



Lossless join decomposition / non additive
This property guarantees that the extra or less tuple generation problem doesn't occur after decomposition.

- Decomposition must be lossless

If a relation R is decomposed into two relations R_1 & R_2, then it will be loss-less iff

- $attr(R_1) \cup attr(R_2) = attr(R)$

A	B	C	D
=	=	=	=

R

decompo-sition →

A	B
=	=

R_1

D
=

R_2

Lossy
C is missing

- $attr(R_1) \cap attr(R_2) \neq \phi$

A	B	C	D
1	a	p	x
2	b	q	y

R

Decompo-sition →

A	B
1	a
2	b

R_1

C	D
p	x
q	y

R_2

Combine (cartesian product) →

A	B	C	D
1	a	p	x
1	a	q	y
2	b	p	x
2	b	q	y

extra tuples added (loss) — Lossy

- $attr(R_1) \cap attr(R_2) \rightarrow attr(R_1)$
  or
  $attr(R_1) \cap attr(R_2) \rightarrow attr(R_2)$

common attr must be super key of atleast one of the subtables

common attr must have distinct values i.e it must be CK in one of the decomposed tables

A	B	C
1	a	p
2	b	q
3	a	r

R

Decompo-sition →

A	B
1	a
2	b
3	a

R_1

B	C
a	p
b	q
a	r

R_2

$R_1 \bowtie R_2$ →

A	B	C
1	a	p
1	a	r
2	b	q
3	a	p
3	a	r

extra tuples added — lossy

Examples

→

lossy ✗ – $R_1(AB)$ $R_2(CD)$ 1st ✗

lossy ✗ – $R_1(ABE)$ $R_2(DE)$ 2nd ✗

lossy – $R_1(ABC)$ $R_2(CDE)$ 1st ✓ 2nd ✓ 3rd ✗

lossless ✓ – $R_1(ABCD)$ $R_2(ACDE)$

lossless ✓ – $R_1(ABCD)$ $R_2(DE)$

lossless ✓ – $R_1(ABC)$ $R_2(BCD)$ $R_3(DE)$

BC common & CK

$R(ABCD)$ $R_3(DE)$

D common & CK

union = ABCDE ✓✓

A	B	C	D	E
a	122	1	p	w
b	234	2	q	x
a	568	1	r	y
c	343	3	s	z

whether common key is distinct

→ $R(VWXYZ)$

$Z \to Y$, $Y \to Z$, $X \to YV$, $VW \to X$

✓ – $R_1(VWX)$ $R_2(XYZ)$

✗ – $R_1(VW)$ $R_2(YZ)$ 1st ✗

✗ – $R_1(VWX)$ $R_2(YZ)$ 2nd ✗

(may/may not be lossy but not lossless) ✗ – $R_1(VW)$ $R_2(WXYZ)$ 3rd ✗

whether common key is CK

→ ✗ $R_1(ABC)$ $R_2(DE)$ ✗ common ✗

✓ $R_1(ABC)$, $R_2(ABDE)$

✗ $D(ABC, CD)$ ✗ E ✗

✓ $R_1(ABC)$ $R_2(CDE)$ ✓

✗ $R_1(AB)$ $R_2(BCDE)$ ✗

✗ $R_1(AB)$ $R_2(CD)$ $R_3(DE)$ ✓

combine any 2 first

$R_1(AB)$ $R_{23}(CDE)$ ✗

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

→ $R(ABCD)$ Lossless

$F: \{A \to B, B \to C, C \to A\}$

$R_1(ABC)$ $R_2(CD)$ ✓

$C \to ABC$ ∴ C is CK of R_1

→ $R(ABCDEF)$ ✗ Lossy

$F - \{AB \to C, C \to D, D \to EF, F \to A, D \to B\}$ DE → lossless

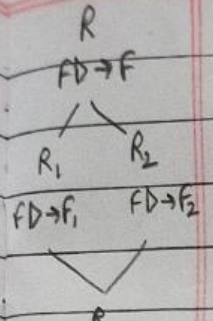
$D(ABC, CDE, EF)$

$C^+ = CDEFAB$

$R_{12}(ABCDE)$ $R_3(EF)$

$E^+ = E$

Dependency Preserving Decomposition



If a table R having FD set F is decomposed into two tables R_1 & R_2 having FD sets F_1 & F_2 then,

$F_1 \subseteq F^+$

$F_2 \subseteq F^+$

$(F_1 \cup F_2)^+ = F^+$

$R_1, R_2, R_3 \ldots R_n = R$

$F_1 \cup F_2 \cup F_3 \cup \ldots \cup F_n \equiv F$

$\subset F$ ✗

// F must not be superset union can be

→ R(ABC)

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

Also lossless

$R_1(AB)$ $R_2(BC)$ Dependency preserved

F_1: $A \to B$, $B \to A$ F_2: $B \to C$, $C \to B$

$F_1 \cup F_2$: $A \to B$, $B \to C$, $B \to A$, $C \to B$

$C^+ = CBA$

→ R(ABCD)

$AB \to CD$ ✗ Dependency not preserved !!

$D \to A$

$R_1(AD)$ $R_2(BCD)$

→ Find closures of all combos to find F_1 & F_2 avoid trivials

F_1: $A \to x$, $D \to A$ F_2: $B \to x$, $D \to x$, $C \to x$, $BC \to x$, ~~$BD \to x$~~, $CD \to x$, $BD \to C$

$F_1 \cup F_2 = D \to A$, $BD \to C$

$(AB)^+ = AB$

Note : This decomposition is lossless hence it is valid

Lossless → mandatory

Dep pres → not mandatory but must be (sometimes not fulfilled in BCNF)

→ R(ABCDEFGHIJ)

A → DE, B → F, F → GH, AB → C, D → IJ

- Candidate keys : $(AB)^+$ = ABCDEFGHIJ
  Therefore AB is CK
- Prime attributes : A, B
- Non-prime — : C, D, E, F, G, H, I, J

Not in 2NF, 3NF & BCNF

Decomposition → 2NF, 3NF & BCNF

R_1(AB C) — AB → C

R_2 (A D E I J) — A → DE, D → IJ

R_3 (B F G H) — B → F, F → GH

[2nd →

[3rd → also in BCNF →

R_{21} (A D E) — A → DE

R_{22} (D I J) — D → IJ

R_{31} (B F) — B → F

R_{32} (F G H) — F → GH

DEPENDENCY PRESERVING

→ R(A B C D E)

F - {A → B, B → C, C → D, D → A}

check only F ⊆ G

R_1(A, B, C) R_2(C, D E)

F_1: A^+ → ABCD

B^+ → BCDA

C^+ → CDAB

redundant: AB^+ → ABCD, BC^+ → ABCD, AC^+ → ABCD

as only A, B, C can derive all no need to take their combos

F_2: C^+ → CDAB

D^+ → ABCD

E^+ → E

CD^+ → ABCD

CE^+ → ABCDE

DE^+ → ABCDE

So F_1 : A → BC, B → AC, C → AB

F_2 : C → D, D → C

$G = F_1 \cup F_2$ = A → BC, B → AC, C → AB, C → D, D → C

Here F ⊆ G & G ⊆ F (obvious)

∴ F ≡ G

presumed

→ R(A B C D E)

F = { A → BCD, B → AE, BC → AED, D → E, C → DE }

R_1(A B) R_2(BC) R_3(CDE)

F_1: $A^+ \to ABCDE$ F_2: $B^+ \to ABCDE$ F_3: $C^+ \to CDE$ $DE^+ = DE$

$B^+ \to BAECD$ $C^+ \to CDE$ $D^+ \to DE$

$E^+ \to E$

F_1: A → B, B → A F_2: B → C F_3: C → DE, D → E

$F_1 \cup F_2 F_3$ = A → B, B → A, B → C, C → DE, D → E

$A^+ = ABCDE$ $B^+ = BACDE$ $BC^+ = BCDEA$ $D^+ = DE$

$C^+ = CDE$

∴ F ⊆ G_1

F will cover G_1 always ∴ G_1 ⊆ F

∴ F ≡ G_1 preserved

→ R(A B C D)

F : { A → B, C → D }

R_1(AC), R_2(BD)

F_1: A → AB F_2: B → B

C → CD D → D

F_1 is empty F_2 is empty $F_1 \cup F_2$ is empty

not preserved

→ R(ABCDE)

F : { A → BCDE, BC → AED, D → E }

R(AB) R_2(BC) R_3(CDE)

A → ABCDE B → B C → C CE → CE DE → DE

B → B C → C D → DE CD → CDE

F_1: A → B F_2: Φ F_3: D → E ∅

$G = F_1 \cup F_2 \cup F_3$ = A → B, D → E

→ R(ABCD)

F: {A→B, B→C, C→D, D→A}

$R_1(AB)$	$R_2(BC)$	$R_3(CD)$
$A^+ \to AB$	$B^+ \to ABC$	$C^+ \to ABCD$
$B^+ \to AB$	$C^+ \to ABC$	$D^+ \to ABCD$
$F_1: \{A \to B, B \to A\}$	$F_2: \{B \to C, C \to B\}$	$F_3: \{C \to D, D \to C\}$

$G = F_1 \cup F_2 \cup F_3 = \{A \to B, B \to A, B \to C, C \to B, C \to D, D \to C\}$

$F \subseteq G$ & $G \subseteq F$

$\therefore F \equiv G$ preserved

→ R(ABCDEF)

F: {AB→C, C→D, D→EF, E→A, D→B, E→F}

$R_1(ABC)$	$R_2(CDE)$	$R_3(EF)$
$A^+ \to A$ $AB^+ \to ABCDEF$	$C^+ = CDEFAB$	$E^+ = EFA$
$B^+ \to B$	$D^+ = DBEFAC$	$F^+ = FA$
$C^+ \to CDEFAB$	$E^+ = EFA$	
$F_1: \{C \to AB, AB \to C\}$	$F_2: \{C \to DE, D \to F\}$	$F_3: \{E \to F\}$

$G = F_1 \cup F_2 \cup F_3 = \{C \to AB, AB \to C, C \to DE, D \to E, E \to F\}$

DECOMPOSITION

1NF → 2NF

- Make separate table for PD
- 1 table of CK mostly

→ R(ABCD) F: { A→B, B→C }

Find CK → $AD^+ = ABCD$ ∴ AD is CK

Decompose → Prime attr. = A, D

NPA = B, C

Find depen. for new tables → $A^+ = ABC$

CKs for new table → R_1 { A B C }

check if decomp into 2NF if not decompose further → $A^+ = ABC$

$B^+ = BC$

$C^+ = C$

F_1: { A→BC, B→C }

CK = A

Now it is in 2NF

R_2 { AD } BCNF✓

$A^+ = ABCD$

$D^+ = D$

F_2: ~~{ D }~~ ϕ

Dependency preserved & Lossless

→ R(ABCD)

F: { A→B, C→D }

$AC^+ = ABCD$ ∴ AC is CK

PA = A, C NPA = B, D

$A^+ = AB$ $C^+ = CD$

R_1(AB) R_2(CD) R_3(AC)

$A^+ \to B$ $C^+ \to CD$ $A^+ = AB$

$B^+ \to B$ $D^+ = D$ $C^+ = CD$

$F_1 = \{A \to B\}$ $F_2 = \{C \to D\}$ $F_3 = \phi$

CK = A CK = C

Dep. pres ✓, Lossless

2NF → 3NF

→ R(A, B, C, D)

F : { A → B, B → C, C → D }

$A^+ = ABCD$ ∴ A is CK

PA = A , NPA = B, C, D

$B^+ = BCD$ $C^+ = CD$

R_1 (BCD) R_2 (CD) R_3 (AB) ACX

$B^+ = BCD$ $C^+ = CD$ $A^+ = AB$

$C^+ = CD$ $D^+ = D$ $B^+ = BC$

$D^+ = D$

$CD^+ = CD$

2NF → F_1 : { B → CD, C → D } BCNF F_2 : { C → D } BCNF F_3 : { A → B }

$B^+ = BCD$ C is CK A is CK

∴ B is CK

$C^+ = CD$

BCNF R_{11} (CD) R_{12} (BC) BCNF

$C^+ = CD$ $B^+ = BC$

$D^+ = D$ $C^+ = CD$

F_{11} : { C → D } F_{12} : { B → C }

R_{12} (BC) R_2 (CD) R_3 (AB)

F_{12} : { B → C } F_2 : { C → D } F_3 : { A → B }

Dep pres & lossless

→ R(ABCDEFGH)

F : {A→BD , B→C , E→FG , AE→H

AE^+ = AEBDCFGH

∴ AE is CK PA = A, E | NPA = B, C, D, F, G, H

1NF ✓

2NF ✗ : A→BD , E→FG (PD)

3NF ✗ : B→C (TD)

BCNF ✗

R(ABCDEFGH) 2NF ✗

R1(ABCD)	R2(EFG)	R3(HAE)
A^+ = ABCD	E^+ = EFG	H^+ = H
B^+ = BC	F^+ = F	A^+ = ABDC
C^+ = C	G^+ = G	E^+ = EFG
D^+ = D	FG^+ = FG	AH^+ = ABDCH
BC^+ = BC		HE^+ = HEFG
CD^+ = CD		AE^+ = ABCDEFGH
BCD^+ = BCD		
A→BCD B→C	E→FG	AE→H
CK = A	CK = E	CK = AE
2NF ✓	2NF ✓	2NF ✓
PA = A , NPA = B,C,D	PA = E NPA = F,G	PA = A, E NPA = H
B→C TD	3NF ✓	3NF ✓
	BCNF ✓	BCNF ✓

R_{11}(BC)	R_{12}(ADB)
B^+ = BC	A^+ = ABCD
C^+ = C	D^+ = D B^+ = BC DB^+ = DBC
B→C	A→BD
CK = B	CK = A
BCNF ✓	BCNF ✓

Lossless

F1 ∪ F2 ∪ F3 = { B→C , A→BD , E→FG , AE→H } Dep. Preserved

A^+ = ABDC B^+ = BC E^+ = EFG AE^+ = AEHFGBDC

Problem → ER Diagram → Relational Model → Normalisation → Hashing / Indexing
→ storing, retrieving, modifying data

Page No
Date

Query Languages DML

Languages used to retrieve, store or modify the data stored in DBMS

Types :

1) Procedural : While using procedural languages, user instruct the system to perform a sequence of operations in order to produce desired result.
- User tells what data is to be retrieved & how to be retrieved.

2) Non-procedural : Here user describe the desired info without giving the specific procedure for obtaining that info

Imp points :

Theoretical [Relational Model → RDBMS
RA, RC → SQL
(Algo) → (code)] Practical

Query language (DML)
- Procedural
  - Frameworks: Relational Algebra (90%)
- Non-Procedural
  - Relational Calculus (10%)

SQL (Structured Query Lang)

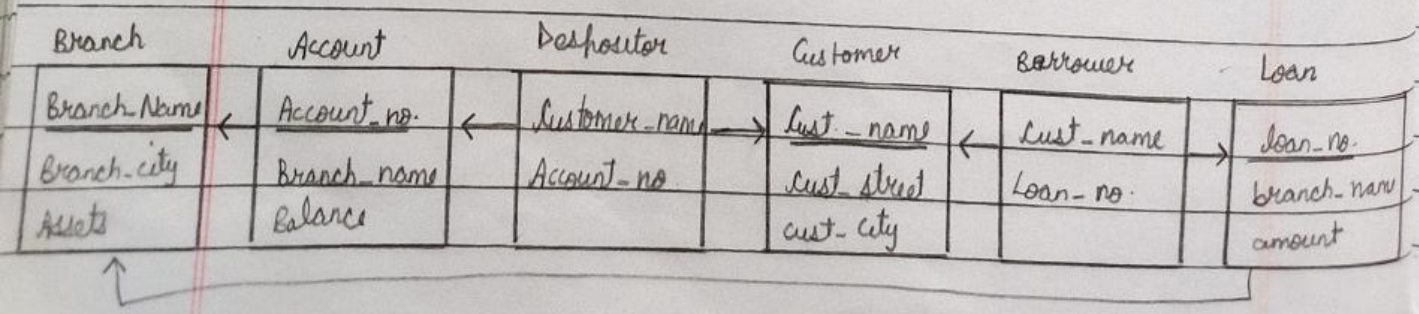
RELATIONAL ALGEBRA

Theoretical framework / algo

- It is a formal query language associated with relational model.
- It defines some operators & use relations (tables) as operands.
- Every operator in relational algebra take one or two relations as input argument & generate a single relation as a result (with no name).
- RA doesn't consider duplicacy as it is based on set theory.
- In each query, we describe a step by step procedure for computing the desired result. So it is procedural.
- English keywords are not used.

OPERATORS

Basic / fundamental	Derived
• Select (σ)	• Natural join ($\bowtie$)
• Project (π)	⋈ ⟕ ⟖
• Union ($\cup$)	• Intersection ($\cap$)
• Set difference ($-$)	• Division ($\div$)
• Cartesian product (X)	
• Rename (ρ)	



relational : $>, <, =, \geq, \leq, \neq$

logical : $\wedge$ (and), $\vee$ (or), (not)

(calculus)
(algebra)

Select (σ) horizontal partitioning

It is a fundamental unary operator which is used to find tuples (rows) in a relation based on some conditions.

Syntax : $\sigma_{\text{predicate (condition)}}$ (table name)

- works same as 'where' clause in SQL.
- Min no of tuples selected = 0
- Max no of tuples selected = all

Student

S_id	Name	Branch
1	A	CS
2	B	ME
3	C	CS
4	D	CS
5	E	EE
6	F	EC
7	G	EE

Examples : $\sigma_{sid \leq 7}$ (Student) all

$\sigma_{Branch = 'Civil'}$ (Student) 0

Examples:

→ Find the details of accounts having balance ≥ 10,000?

$\sigma_{Balance \geq 10,000}$ (Account)

→ Find details of the customers who live in Delhi

$\sigma_{cust\_city = 'Delhi'}$ (Customer)

→ Find the details of those loans having amount ≤ 5000 & from north_delhi branch

$\sigma_{amount \leq 5000}$ ($\sigma_{Branch\_name = 'North\_Delhi'}$ (loan))

(nested query)

U can change order

or

$\sigma_{amount \leq 5000 \text{ and } Branch\_name = 'North\_Delhi'}$ (loan)

→ Find those branch details which are in Delhi or have assets more than 10,00,000

$\sigma_{Branch\_city = 'Delhi' \text{ or } Assets > 10,00,000}$ (Branch)

Project (Π)

It is a fundamental unary operator used to select desired columns of a relation

Syntax : Π_{Column_name} (table name)

- works same as 'select' clause in SQL

Examples : Π_{name} (Student)

→ all Branch name of the bank

Π_{Branch_name} (Branch)

→ Find all the account_no along with their balance

$\Pi_{accountno., Balance}$ (Account)

→ Find the name of all the customer who have loan

Π_{Cust_name} (Borrower)

→ Find all the details about branch.

(Branch)

Select & project combined Select → Project

→ Find the names of all students from CS branch

$$\Pi_{name}\left(\sigma_{Branch='CS'}(Student)\right)$$

→ Find those account no. where balance is less than 1000

$$\Pi_{account\_no}\left(\sigma_{balance<1000}(Account)\right)$$

→ Find those loan no. which are from CP branch with amount > 1000

$$\Pi_{loan\_no.}\left(\sigma_{branch\_name='CP' \text{ and } amount>1000}(loan)\right)$$

→ Find branch_name and branch_city with assets more than 100000

$$\Pi_{branch\_name,\ branch\_city}\left(\sigma_{Assets>100000}(Branch)\right)$$

$A \cap B = A - (A - B)$

Page No
Date

- r & s must have same attributes
- duplicates eliminated

$r \cup s$ Union / $r \cap s$ Intersection and $r - s$ set difference (only on compatible tables) — Binary operators

→ Find the name of customers who have loan or amount or both.

→ If both ∩
→ only account −

$$\Pi_{cust\_name}(depositor) \cup \Pi_{cust\_name}(borrower)$$

→ Find the name of a branch who have amount but not loan?

$$\Pi_{branch\_name}(Account) - \Pi_{branch\_name}(loan)$$

→ Find the name of a customer who neither have a loan nor an account

$$\Pi_{cust\_name}(Customer) - (\Pi_{cust\_name}(Depositor) \cup \Pi_{cust\_name}(Borrower))$$

Demerit
Unnecessary/ redundant/ surplus tuples

coz order of columns doesn't matter
Commutative in DBMS | Also associative

Cartesian Product (X)

Tables may not contain common attr.

It is a fundamental binary operator used to combine two tables

Rows → $|R_1| = m$ $|R_2| n$ $|R_1 \times R_2| = mn$

Columns → $R_1 (1, 2 \ldots n)$, $R_2 (1, 2 \ldots m)$

$R_1 \times R_2$ $(1, 2 \ldots m+n)$

R_1

A	B
1	a
2	b
3	c

R_2

C	D	F
p	101	y
q	102	z

$R_1 \times R_2$

A	B	C	D	F
1	a	p	101	y
1	a	q	102	z
2	b	p	101	y
2	b	q	102	z
3	c	p	101	y
3	c	q	102	z

Example :

→ Find customer name having account balance < 100

$$\Pi_{cust\_name} \left(\sigma_{Account.Ano = Depositor.Ano \ \wedge \ balance < 100} (Account \times Depositor) \right)$$

→ Find those account no. which are in Delhi

Π → Account
σ → Branch

$$\Pi_{Account\_no} \left(\sigma_{Account.Branch\_name = Branch.Branch\_name \ \text{and} \ \wedge \ Branch\_city = 'Delhi'} (Branch \times Account) \right)$$

Π → Borrower
σ → Branch

→ Find those customer names who have a loan from a branch having assets more than 10,00,000

$$\Pi_{cust\_name} \left(\sigma_{assets > 1000000 \ \text{and} \ \wedge \ Branch.Branch\_name = loan.branch\_name \ \text{and} \ \wedge \ loan.loan\_no. = borrower.loan\_no.} (Branch \times Borrower \times loan) \right)$$

2 tables combine – 2 conditions to remove redundancy

3 tables — – 3 —

Rename (ρ)

used to rename a relation

example : ρ (A , B)
(It will rename relation A to B)

Joins = cartesian product + selection condition

Joins are used to combine relations based on some conditions.
Two tuples from the relations, on which join is applied, pairs only if given condition satisfies.

Types :

Theta Join ($\bowtie_\theta$)

Combines tuples from diff relations only if given theta condition is satisfied.
syntax : $R_1 \bowtie_\theta R_2$
eg : Student $\bowtie_{Student.Std = Subject.Class}$ Subject

SID	Name	Std
101	A	10
102	B	11

Student

Class	Subject
10	Math
10	Music
11	Music
11	Sports

Subject

→

SID	Name	Std	Class	Subject
101	A	10	10	Math
101	A	10	10	Music
102	B	11	11	Music
102	B	11	11	Sports

2) Equijoin ($\bowtie_{condition}$)
Combines tuples from diff relations based on some equality condition.

3) Natural Join ($\bowtie$)
Joins tuples from diff relations only if common attributes have equal values.
- Works only if two relations have atleast one common attribute with same name & domain.
Syntax : $R_1 \bowtie R_2$

4) Outer Joins
Includes all the tuples from the participating relations.

i) Left Join ($\bowtie$)
All the tuples from left relation are included. If right relation doesn't have matching tuple, NULL is displayed.
Syntax : $R \bowtie S$

ii) Right Join ($\bowtie$)
All tuples from right relation are included. If ⊄ left relation doesn't have matching tuple, NULL is displayed
Syntax : $R \bowtie S$

iii) Full outer join ($\bowtie$)
All the tuples from both the relations are displayed. If matching tuples are not found in any of the relation, NULL is displayed
Syntax : $R \bowtie S$

blocks – many bits
records – 1000 – 5000

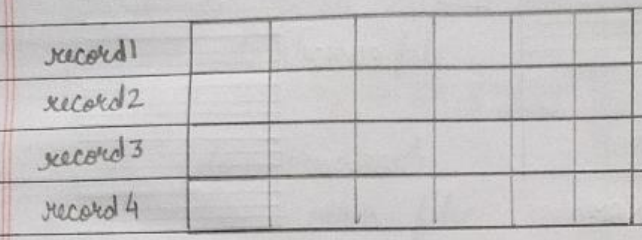
(secondary storage)

- Contiguous
- Indexed
- Linked

File Structures

(data structure – very flexible)
(set of records)

SRS
(Software Req. Specification)
↓
E-R diagram
↓
Relational model
↓
Normalisation
↓
File Structure
(Indexing (optional) | Physical str.)



orientation:

sorted file

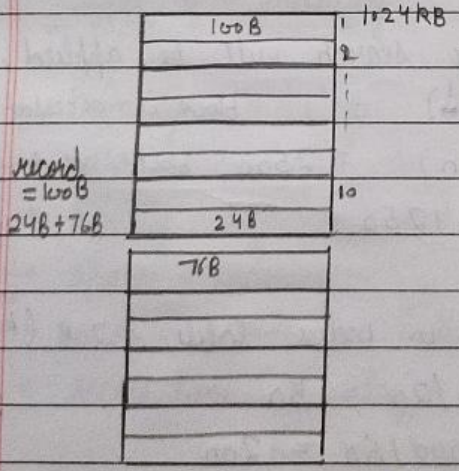
- can be sorted only according to one attribute (search key)
- searching will be fast
- Insertion & deletion will be difficult

unsorted file

- random order, any record can be placed anywhere
- ~~ser~~ searching will be slow
- Insertion and deletion will be easy

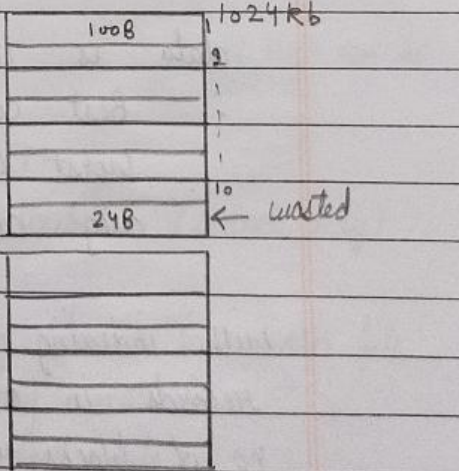
time is given priority over memory

spanned mapping



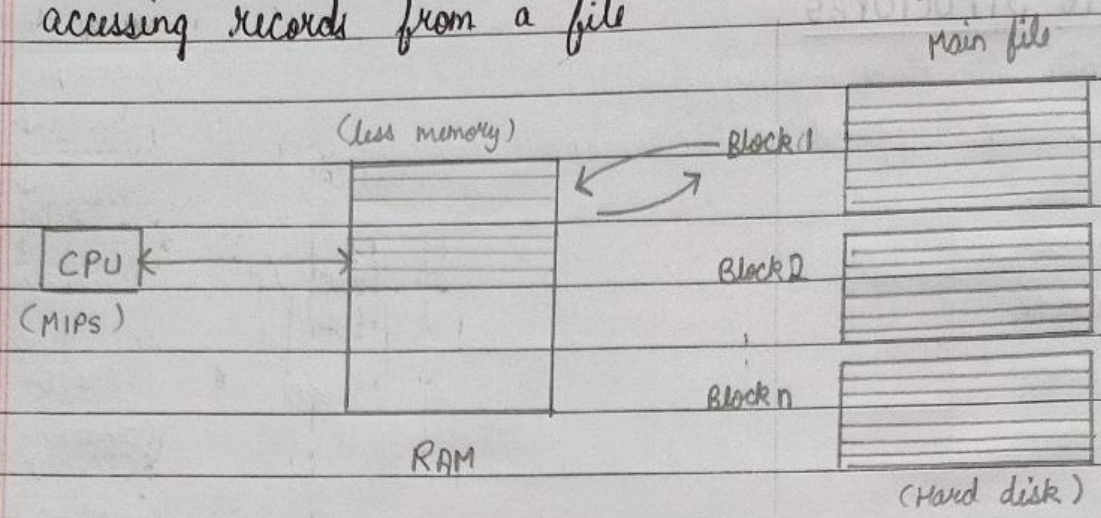
- File is undergoes internal fragmentation & stored in next ~~multiple~~ blocks in case if previous block is full.
- efficient storage but search time ↑ (2 blocks need to be searched)

unspanned mapping



- File is stored in next block
  No internal fragmentation
- not efficient in case of storage but search is fast

→ accessing records from a file



Secondary memory

When a record needs to be searched, complete blocks are transferred to RAM one by one & searched. Cost req. for the process - I/O cost

eg: A hard disk has block size = 1000 B, each record size = 250 B
If total records = 10,000 & file is unsorted. What is avg TC?
no. of records we can put in a block = 1000/250 = 4
no. of blocks required to store 10,000 records = $\frac{10,000}{4}$ = 2500

data is unsorted, linear search will be applied,
∴ Best case TC = O(1) = 1 block is accessed
worst case TC = O(n) = 2500 blocks are accessed
avg case = $\frac{2500}{2}$ = 1250

with indexing (record size in index table = 20B (10B key + 10B ptr)
records in a block = 1000/20 = 50
no. of blocks required = 10,000/50 = 200
no. of blocks accessed = $1 + \log_2 200 \simeq 1 + 8 \simeq 9$

Block search time >>> record search time (negligible)

Index file must be sorted.

Even binary search takes time in case of large files

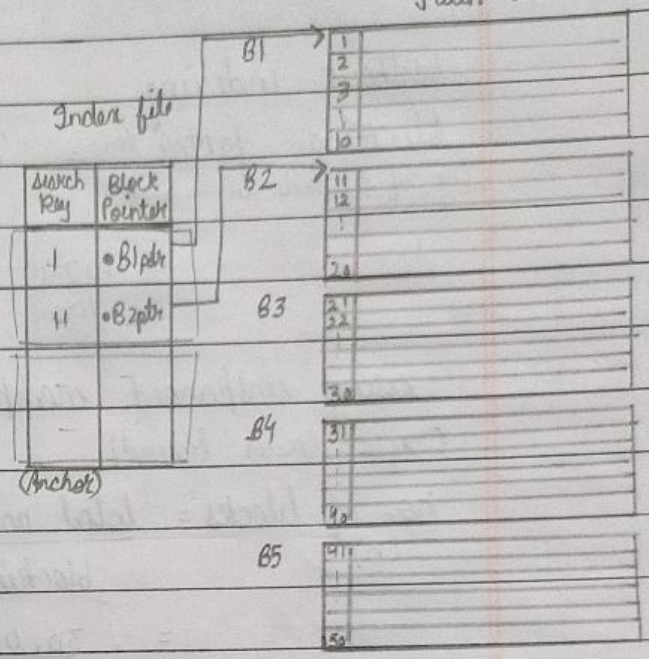
INDEXING (optional)

It is a technique which allows to access records from a database file quickly by maintaining idx files of memory blocks

Index files are much smaller than main files

∴ access time ↓

- no. of blocks of index file are less bcz record size ↓ as well as no. of records dec (except sec.)
- search is always applied on blocks



size of index file is small

- dense index - unsorted file, each ~~rec~~ value tht in ind file
- sparse index - sorted file, few rec tht in ind file (eg 1 rec from a block)

Types of Indexing

exact block can be searched.

1) Primary Indexing (above eg)

Block is searched

- when main file is sorted (acc. to PK)
- Primary key is used as anchor attribute (search key)
- sparse indexing
- no of records in index file = no. of blocks in main file
- access time = $\log_2 n + 1$ (binary search)

n = no. of blocks in index file

In oracle automatically made when u make PK

block size is hardware property
record size is decided by the user

one record → 1 block

Total block size = 1024B

Numerical

without indexing

blocking factor (no. of records in a block) = $\frac{\text{Total block size}}{\text{record size}}$

$= \frac{1024}{100} = 10.24$

using unspanned mapping = 10 records (use lower bound)

no. of blocks (n) = $\frac{\text{total no. of records}}{\text{blocking factor}}$

$= \frac{30,000}{10} = 3000$ blocks [always take upper bound]

primary (sorted) ∴ binary search

access time (blocks accessed) $= o(\log_2 n)$

$= \log_2 3000 \approx 11.- \approx 12$ units

with indexing

blocking factor = $\frac{\text{total block size}}{\text{record size}} = \frac{1024}{15} = 68$ ← more records (lower bound)

no. of blocks = $\frac{\text{total no. of records in index file}}{\text{blocking factor}}$

primary → ordered → sparse ∴ 1 record for each block

$= \frac{\text{total no. of blocks in main file (n)}}{\text{blocking factor}}$

records ↓ Block ↑

$= \frac{3000}{68} = 45$ ← block reduced to much extent

index file (always sorted) ∴ binary search

access time (no. of blocks accessed) $= \log_2 45 + 1 = 5.- + 1 \approx 6 + 1 \approx 7$

Hence, access time decreased using primary indexing

Primary / Cluster index - atmost 1

many record → 1 block
exact block can't be searched

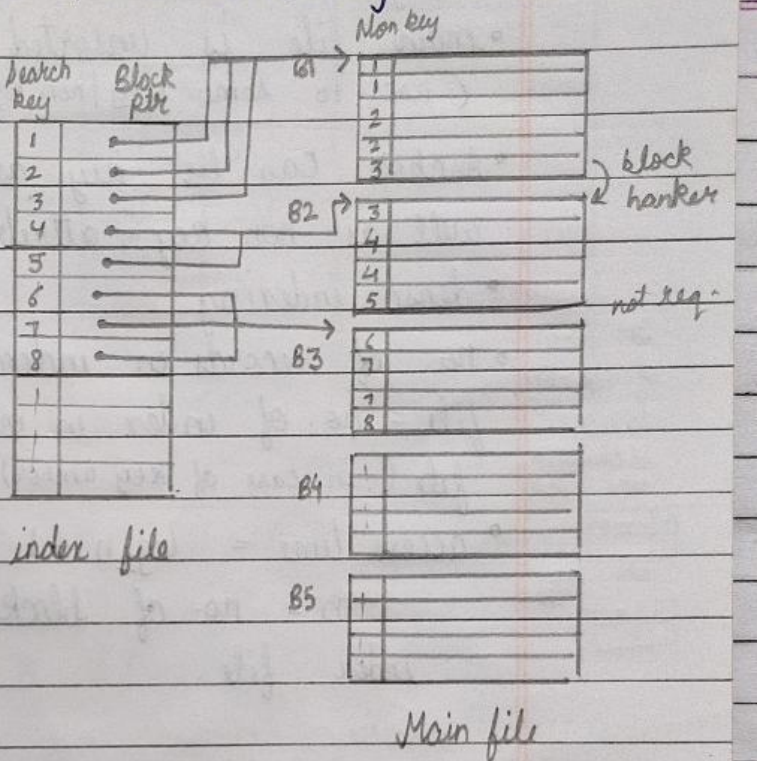
2) Clustering index

- when main file is sorted (acc. to non key attribute)
- there will be one record for each unique value of non-key attribute
- sparse (as well as dense)
- Non key attribute is used as anchor attribute
- access time $\geq \log_2 n + 1$

  n = no. of blocks in index file

  block hanker is used

= if 1 block
> if multiple blocks.



there may be multiple index files for a main file
a file can be sorted acc to one attr. but unsorted acc. to another

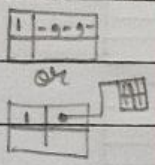
3) Secondary Index

mostly primary indexing is done

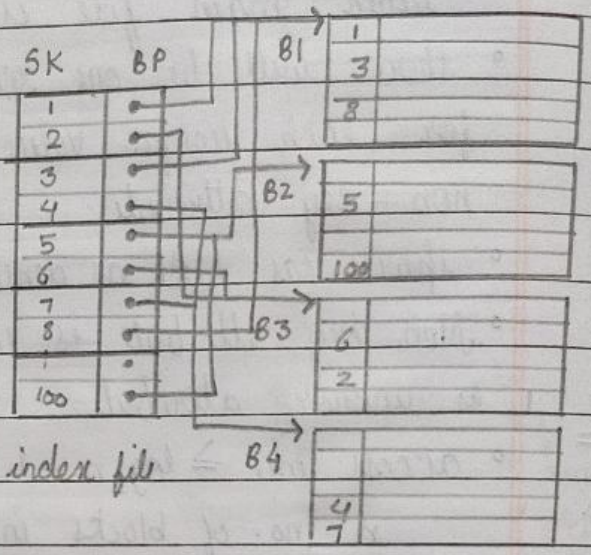
- main file is unsorted (acc. to some key / non key attr.)
- Anchor can be key as well as non key attribute.
- dense indexing
- no. of records in index file = no. of index in main file (in case of key anchor)
- access time = $\log_2 n + 1$

  n = no. of blocks in index file.

In case of non-key various techniques are used



many records → 1 block
exact block is searched



no of pointers to a block in main file = no of records in it

→ without secondary indexing:

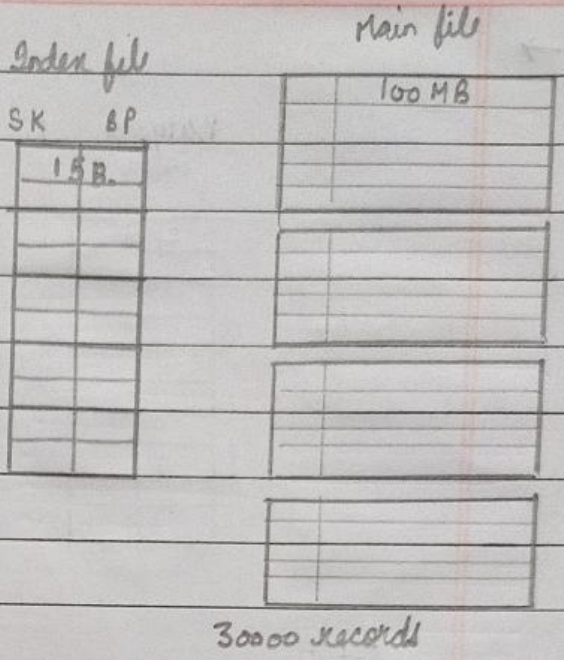
blocking factor = $\frac{1024}{100} = 10$

no. of blocks for main file(n) = $\frac{\text{total records}}{\text{blocking factor}} = \frac{30,000}{10}$

= 3000

unsorted linear search

access time = $O(n) = 3000$



total block size = 1024B

with secondary indexing:

blocking factor = $\frac{1024}{15} = 68$

no of blocks for index file = $\frac{\text{total records in index file}}{\text{blocking factor}}$

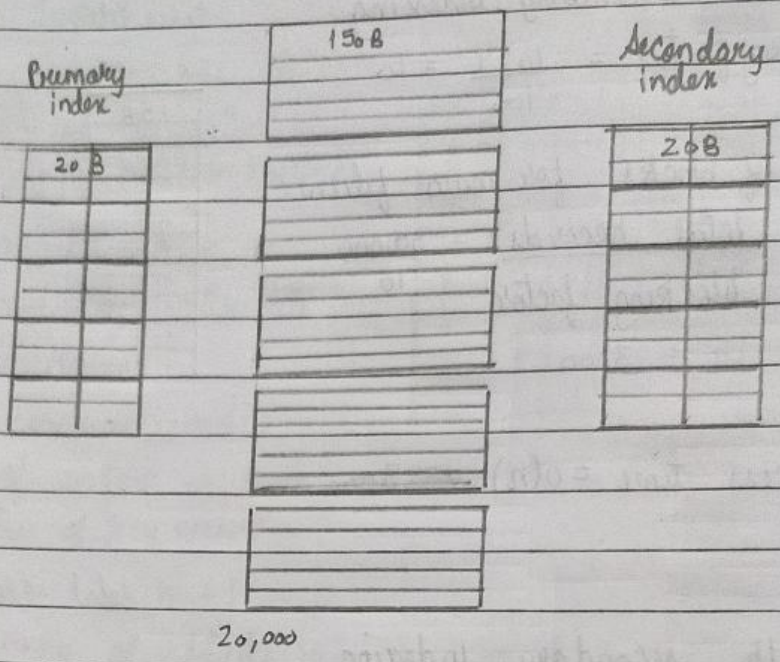
= $\frac{\text{total records in main file}}{\text{blocking factor}}$

= 30000/68 = 442

index file sorted binary search

access time = ~~of~~ $\log_2 442 + 1 \approx 9 + 1 \approx 10$

→ Block size = 2048 B



Blocking factor (main file) = $\frac{2048}{150} \approx 13$

no of blocks = $\frac{20,000}{13} = 1538.4 \approx 1539$

In case of sorted file, access time = $O(\log_2 n)$

$\log_2 1539 \approx 11$

In case of unsorted file, access time = $O(n)$

≈ 1539

Primary indexing:

Blocking factor (primary index file) = $\frac{2048}{20} = 102.8 \approx 102$

no. of blocks = $\frac{1539}{102} = 15.- \approx 16$

access time = $\log_2 16 + 1 = 4 + 1 = 5$

Secondary indexing:

Blocking factor (sec. index file) ≈ 102

no. of blocks = $\frac{20,000}{102} \approx 196.- \approx 197$

access time = $\log_2 197 + 1 = 7.-+1 \approx 8 + 1 \approx 9$

sparse ← dense sparse ← sparse

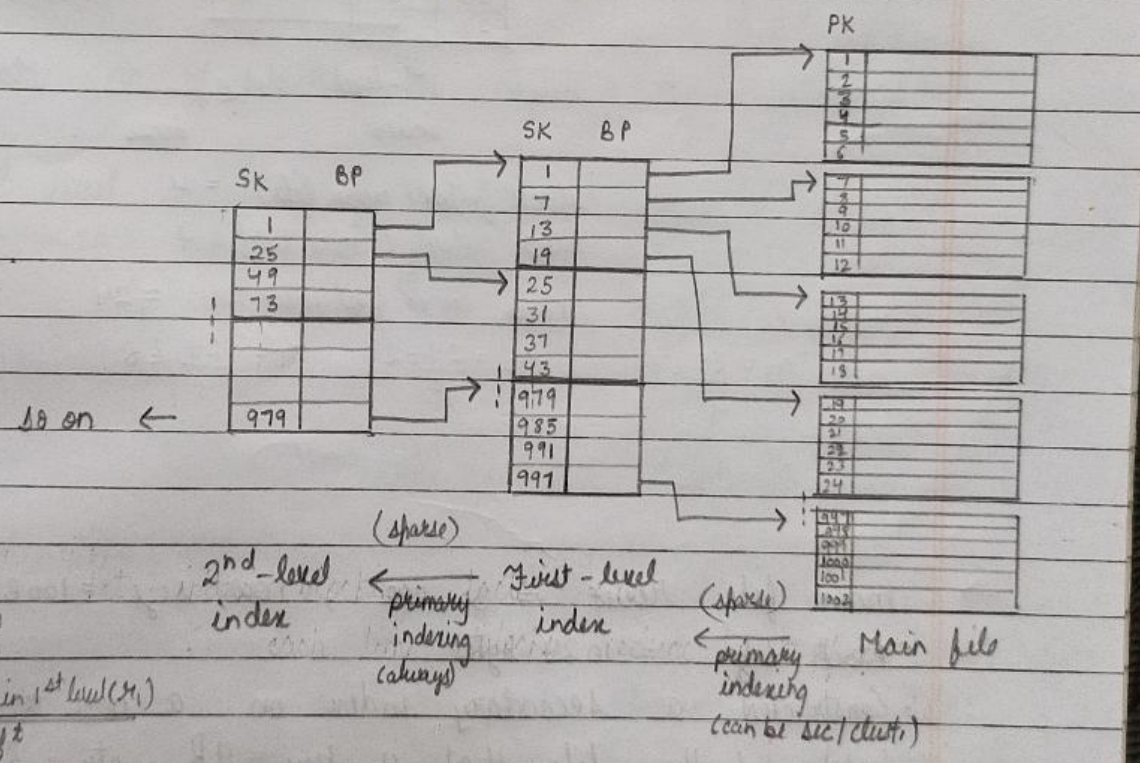
Multilevel Indexing

(index of index)

- It is a technique in which index (first-level) is broken down into further smaller indices in order to make outer most level small enough to fit in a single block.
- ~~access~~ time complexity ↓
- It can be created for any type of first-level index (primary, sec or clustering) as long as the first-level index consists of more than one disk block.
- A third level index is required only if 2nd needs more than 1 block.
- last level index → top index level (root)

2nd level index

3rd level



$n_1 = \frac{\text{rows in 1st level}}{f}$

$n_2 = \frac{n_1}{f} = \frac{\text{rows}}{f^2}$

$n_3 = \frac{n_2}{f} = \frac{\text{rows}}{f^3}$

$\therefore n_t = \frac{\text{rows in 1st level } (r_1)}{f^t}$

& n_t should be 1

- Multi-level indices of first-level index file is always primary bcz index file is always sorted and search key is unique.
- Let index blocking factor = f_o, then each time no. of blocks reduce by a factor f_o

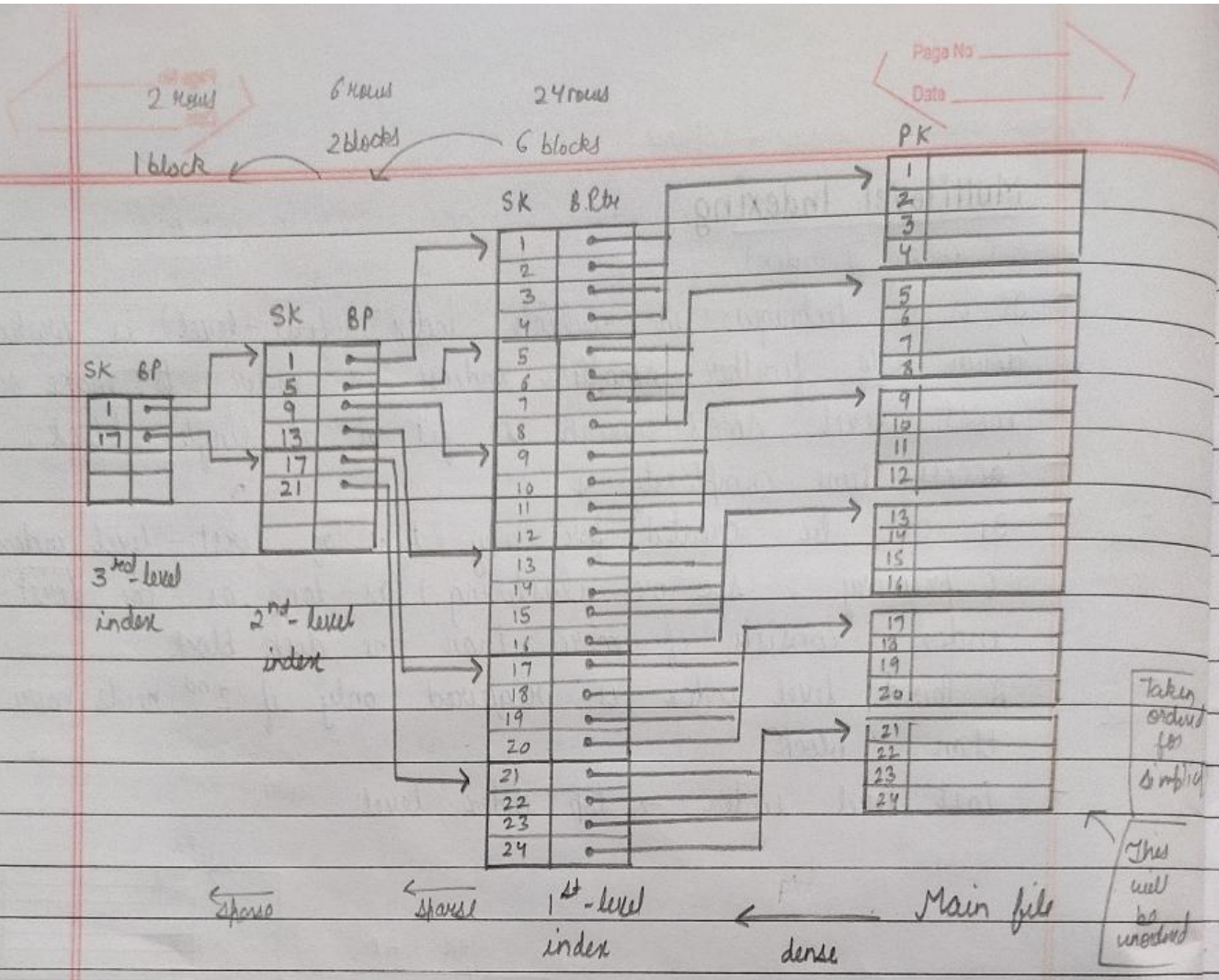
$f^t = r_1$

$t = \log_{f_o} r_1$

no. of blocks at t^{th} level $= n_t = \frac{\text{rows in 1st level } (r_1)}{f_o^t}$

n_t should be 1 $\therefore 1 = \frac{r_1}{(f_o)^t} \Rightarrow \boxed{t = \log_{f_o}(r_1)}$

$\boxed{\text{block access} = t + 1}$



block factor (main file) = 4

block factor (index file) f_o = 4 (Assume) (not possible practically)

rows in 1st level index (r_1) = 24

take upper limits

$n_3 = \frac{24}{4^3} = 1$ $\quad n_2 = \frac{24}{4^2} = 2$ $\quad n_1 = \frac{24}{4} = 6$

$t = \log_4 24 = 3$ (upper limit)

→ main file rows = 30,000, row size = 100B

Block size = 1024 bytes

We Constructed a secondary index on a non ordering key field of the file that is key = 9B, ptr = 6B

Levels = ? index block size $f_o = 1024/(6+6) = 1024/15 = 68$

~~record~~. $t = r_1 = 30000$ (sec. on key attr. → dense)

$\therefore t = \log_{68} 30000 = 3$ levels.

block access = $t + 1 = 4$

or

1st level rows = 30000, block factor = 68 $\quad n_1 = \frac{30000}{68} = 442$

2nd level rows = 442, block factor = 68 $\quad n_2 = 442/68 = 7$

3rd level rows = 7, block factor = 68 $\quad n_3 = 7/68 = 1$

→ Consider a file of 16384 records. Each record is 32 B & key field is 6 B. The file is ordered on a non-key field and the file organization is unspanned. The file is stored in a file system with block size 1024 B & size of block pointer is 10 B
If sec. index is built on the key field of the file & a multi level index is used to store the sec. index, the no. of first level & 2nd level blocks are — , — ?

1st level (dense): no. of records $r_1 = 16384$
index block factor $f_o = \frac{1024}{10+6} = \frac{1024}{16} = \frac{2^{10}}{2^4}$
$= 2^6 = 64$
no. of blocks $n_1 = \frac{16384}{64} = 256$ blocks

2nd level (sparse): no. of records $r_2 = 256$
index block factor $f_o = 64$
no. of blocks $n_2 = \frac{256}{64} = \frac{2^8}{2^6} = 2^2 = 4$ blocks

→ Block size = 1024 index = 16 B records = 100000
record size = 100 B
suppose we convert sparse index into multi level index then which level will be top level of multi level index
rows in 1st level (sparse) (r_1) = blocks in main file $= \frac{100,000}{1024/100} = \frac{100000}{10} = 10,000$
index block size $f_o = \frac{1024}{16} = 64$
$t = \log_{64} 10000 = 3$
3rd level is the top level.

Balanced Search Trees

TRANSACTION

Transaction is a independent unit of work (set of instructions) performed by DBMS against a database.

A transaction usually changes the state of database.

ACID Properties These properties must be followed by a transaction

- Atomicity : It ensures that either the entire transaction takes place at once or doesn't happen at all.
  - It should work as an atomic unit & must not occur partially
  - It is managed by 'transaction management component' of DBMS

any uncommitted transaction will be aborted after some time automatically

eg :

X = 500	Y = 200
Read (X)	Read (Y)
X = X - 100	Y = Y + 100
write (X)	write (Y)
X = 400	Y = 300

X + Y (before transaction) = X + Y (after transaction)

500 + 200 = 400 + 300

- Consistency : If database ~~was~~ is consistent before transaction, it must be consistent after the transaction
  - It ensures the correctness of database.
  - In the above example, the total amount before and after the transaction is constant ∴ It is consistent
  - All the constraints should be maintained

logical isolation
parallel → serial

- Isolation Transactions must occur in isolation concurrently without leading to the inconsistency of DB.
  - The intermediate states of must be invisible to other transactions
  - It is managed by 'concurrency control component' of DBMS

eg: X = Y = 500

T	T'
R(X)	R(X)
X = X * 100	R(Y)
w(X)	Z = X + Y
R(Y)	w(Z)
Y = Y - 50	
w(Y)	

If T' reads X, Y from intermediate state of T, where X is updated but Y is still not updated, it will lead to inconsistency

- Durability : It ensures that once the transaction has been completed, the updates & modifications in the database must persist even if a system failure occurs.
  - Changes must be made permanent.
  - Maintained by 'recovery management component' of DBMS

recover through backups

failures : comp. failure, transaction error, disk failure, power failure

Operations :

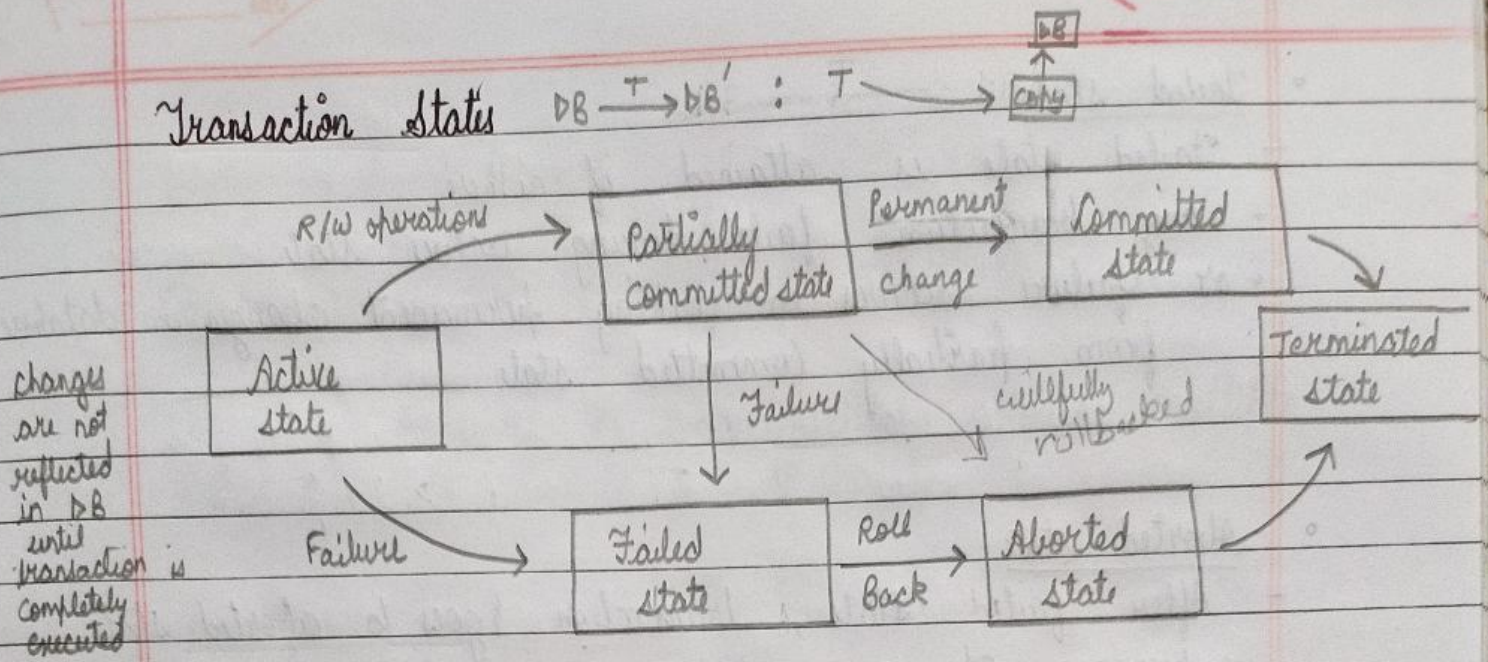
→ Read (A) Reads value of A from database and stores it in buffer in main memory.

→ Write (A) writes the value of A into the buffer

→ Commit After all instructions are executed, changes are made permanent in database.

→ Rollback If transaction doesn't execute completely, all changes made by it (in buffer) are undone. (database remains unchanged)

Transaction States



- **Active state** :
  - This state begins at the execution of first read/write instruction and ends at the last one
  - If all the instructions are performed without any error, it goes to 'partially committed state', if any instruction fails, it goes to 'failed state'.

- **Partially committed state** :
  - After completion of all the read/write operations, changes are made in local buffer (main memory).
  - If changes are made permanent, it goes to committed state else, it goes to failed state.

- **Committed state** :
  This state is attained when changes are made permanent in the database after the complete execution of transaction

- Failed state :
  - Failed state is attained if either
    - any transaction fails during active state.
    - or failure occurs in making permanent changes in database from partially committed state.

- Aborted state
  - After failed state, transaction goes to aborted state wherein changes which are made in local buffer are deleted or roll back and database remains in the actual state.

- Terminated state deallocation of resources
  - ~~After committed~~ Terminated state is attained if,
    - changes are made permanent in database (from committed)
    - changes are roll back & database remains same (from aborted)

Advantages of Concurrency multiple transaction run parallelly

- waiting time ↓
- response time ↓
- resource utilization efficient
- efficiency ↑

Disadv.
- complex management
- may lead to inconsistency

~~Dirty read problem~~

Concurrency

It is a property of DBMS which allows multiple transactions to execute simultaneously.

Advantages:

- Improved throughput: no of transactions that can be executed in a given amount of time is called as throughput. When multiple transactions execute simultaneously, throughput inc.
- Efficient resource utilisation: Various system resources & like disks and CPU are utilised concurrently. When a transaction is using CPU, other can use disk to fetch data.
- Reduced waiting time: In serial execution of transactions, waiting time for the transactions present in queue is more But for concurrent execution waiting time is reduced.

Main conflicts
- WR
- RW
- WW

Page No.
Date

Problems (due to concurrency)

When only read operations are performed, concurrency works well. But a transaction involves both read and write operations.
∴ when multiple transactions execute concurrently in an uncontrolled manner, it leads to several problems:

- **Temporary update / Dirty read problem:**

A dirty read problem occurs when temporarily updated value of a transaction (not committed yet) is used by another transaction but the ~~latter~~ former fails.
Now the former transaction reverts back and database remains unchanged but the temporarily updated value of its intermediat state is used which creates a problem.

X = 10 (initially)

	T1	T2	
X = 10	R(X)		
X = 11	X = X+1		
X = 11	W(X)		
		R(X)	X = 11
		X = X + 2	X = 13
	(failure) rollback	W(X) commit	X = 13

Now T1 is roll back ∴ no changes in database but do on executing T2 X = 10+2 = 12
But at the end of T2 X = 13 is committed

- **Unrepeatable Read Problem:**
Unrepeatable problem occurs when read operations of the same transaction read different values of the same variable due to ongoing write operations in some other transaction.

X = 10

	T1	T2	
X = 10	R(X)		
		R(X)	X = 10
X = 11	X = X+1 W(X)		
		R(X)	X = 11

conflict come from lower line → higher

T_1	T_2
	R
Del	R

- **Phantom Read problem:**
Phantom read problem occurs when a transaction reads a variable once but when it tries to read that same variable again, an error occurs saying the variable doesn't exist bcz it is already deleted by some other transaction in between.

T_1	T_2
R(X)	
	R(X)
Delete(X)	
	Read(X)

- **Incorrect summary problem:**
It occurs when a transaction is applying the aggregat function on some record while another transaction is updating these records.
The aggregat function may calculate some values before updation while others are already updated

	T_1	T_2
A=1, B=2, C=3		
		Sum = 0
Sum=1		R(A)
		Sum = Sum+A
B=3	R(B)	
	B = B+1	
	W(B)	
		R(B)
Sum=4		Sum = Sum+B
Sum=7		R(C)
		Sum = Sum+C
C=4	R(C)	
	C = C+1	
	W(C)	

Here sum should be either 6 or 8
but sum = 7

- **Lost update problem:**
In this problem, updati done to a data item by a transaction is lost as it is overwritten by the updati done by another transaction

	T_1	T_2
X=1		
	R(X)	
X=2	X = X+1	
		X = 10
		W(X)
	Commit	

X=2 needed to be stored
but X=10 is stored

Schedule

set of transactions

It is a series of operations of different transactions executing concurrently.

- If there are n transactions with $n_1, n_2, \dots, n_n$ instructions respectively then, schedule has $n_1 + n_2 + \dots + n_n$ instructions.
- order of instructions of a transaction must remain the same in the schedule.

There must be only 1 instruction in 1 row.

Total no. of schedules = $\frac{(n_1 + n_2 + n_3 + \dots + n_n)!}{n_1!\, n_2! \dots n_n!}$

Types of schedules

No concurrency ∴ if all transactions are following ACID properties, inconsistency ✗

Serial schedule: In this type of schedule, one transaction is completely executed before starting another transaction.

- no. of schedules possible for n transactions: n!
- No concurrency problems
- But inefficient

S

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

Non-serial schedule : In this type of schedule, operations of different transactions are interleaved.

- May lead to concurrency problems
- no. of schedules possible for n transactions with $n_1, n_2 \dots n_n$ instructions:

$$\underbrace{\frac{(n_1 + n_2 + \dots + n_n)!}{n_1!\, n_2! \dots n_n!}}_{\text{Total}} - \underbrace{n!}_{\text{serial}}$$

S

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

Types of Schedules

Based on Serializability
- Serializable (Result equi) - Check o/p
- Conflict Serializable (Conflict equi)
- View Serializable (View equi)

Based on Recoverability
- Ir-Recoverable
- Recoverable
- Cascade Rollback-Recoverable
- Cascade Less Rollback-Recoverable
- Strict Recoverable

If a schedule is conflict serializable, it is consistent but vice versa is not true

Conflict serializability

We do not swap instruction of same Transaction this will change the order of transaction- voilate condition of schedule

- A serial schedule is always consistent
- A non-serial schedule is said to be conflict serializable if it can be ~~converted~~ to made equivalent to serial schedule by swaping ~~to~~ non-conflicting instructions.

A conflict serializable schedule is consistent

Conflicting instructions:

Two operations are said to be conflicting if
- they belong to different transactions
- They operat on same data item
- at least one of them is writ operation.

change in T		change in T		Changes in Database
R(A)		W(A)		W(A)
	W(A)		R(A)	W(A)
	W(A)		R(A)	W(A)
R(A)		W(A)		W(A)

eg:

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

diff data items

non-conflicting

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

Non-conf.

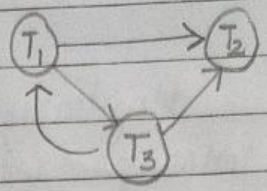
R(A)	
	W(B)
R(A)	
	R(A)

We will find conflicting instructions & try to obtain a valid serial order if cycle forms ✗ else ✓
We are not swapping here (becomes lengthy)

Page No.
Date

Examples

T_1	T_2	T_3
R(X)		
		R(z)
		w(z)
	R(y)	
R(y)		
	w(y)	
		w(x)
	w(z)	
w(x)		



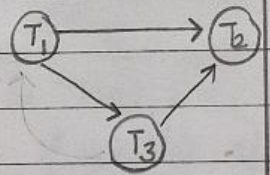
$T_1 \rightarrow T_2$ edge depicts that T_1 will execute before T_2

Stop as the cycle completes – Not conflict serializable

(If cycle doesn't form & all the operations are scanned – conflict serializable)

(Total $3! = 6$ serial schedules can be formed. Swapping the non-conflicting operations to form any 1 of those serial schedule is diff.)

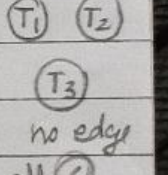
T_1	T_2	T_3
R(X)		
	R(Y)	
		R(Y)
	w(Y)	
w(X)		
		w(X)
	R(X)	
	w(X)	



not conflict serializable

order : T_1, T_3, T_2

T_1	T_2	T_3
R(a)		
	R(b)	
		R(c)
		w(c)
	w(b)	
w(a)		



no edge

all 6 combos are possible

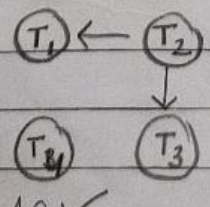
conflict serializable

If $T_1 \rightarrow T_2$, T_3 : CS ✓ (3) combos

$T_1\ T_3\ T_2$
$T_1\ T_2\ T_3$
$T_3\ T_1\ T_2$

If $T_1 \rightarrow T_2 \rightarrow T_3$ CS ✓ (1) Combo

T_1	T_2	T_3	T_4
	R(X)		
		w(X)	
		C	
w(X)			
C			
	w(Y)		
	R(Z)		
	C		
			R(X)
			R(Y)
			C



AC ✓

$O(n^2)$