DBMS

DOMS Page No.
Date / /

Ch:1 Introduction

1.1 Intro & application of DBMS
1.2 Purpose of database
1.3 Data independence
1.4 Database-System architecture - level
1.5 mappings
1.6 Database users & DBA

1.1 Intro & application of DBMS.

DBMS

Data
factual info which can be used as basis for reasoning, discussion or calculation

Database
organized & managed collection of data

Management
act of getting the things together to achieve desired goals and objectives efficiently, & effectively using available resources

System
set of components & relationships among them that produces desired results when same inputs are given.

* **Database Management System =**

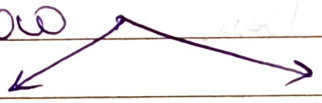
Collection of interrelated data and a set of programs and procedures to provide mechanisms for storage, access, manipulation & safety of those data.

→ Database can be seen in various fields like banking, educational institution, financial institutions, airlines, railways telecommunication, etc.

⇒ Difference between Data & information

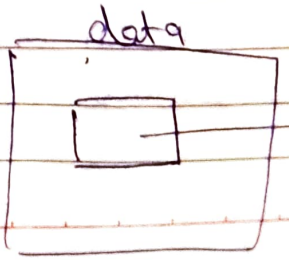
→ Data = It is the total set.
→ Information = Data that is usable according to your situation is information, rest is data.

⇒ eg:
There are 10,000 trains in Indian railway and have data of those 10,000 trains.



friend 1
- he has to travel through Chennai express, so that data related to Chennai express is information for friend 1 & rest is data which is not needed to friend 1 this time.

friend 2.
- for him, only data related to Shatabdi express is information, rest is just data.



data which is usable for that time is information

DBMS Page No. Date / /

1.2 Purpose of Database

→ To store large number of data, it was difficult to manage this task using file system.
→ Finally, we got rid of it with what is called DBMS

* File System Vs DBMS

1. Data Redundancy

→ It means repetition of data.
→ For example: A student registered himself in two different sports but has only one address.

The address gets repeated in both records.

Cricket

Name: Shub Address: 123 manne st Stret Jaipur
Name: Ramesh Address: Udaipur

Chess

Name: Aarya Address: Jaipur
Name: Shub Address: 123, manne stred Jaipur

→ This repetition requires more storage & cost

In DBMS, there is only one copy of a particular data, that is accessible from all places, wherever needed.

id	name	sports
1	Shub	Cricket
2	Ramesh	Cricket
3	Aarya	chess
1	Shub	chess

id	address
1	123, manest
2	Udaipur
3	Jaipur

2. Inconsistency:

→ It means, A thing is existed in the form in which it should not be.

→ For example: A student changes his address, so the authority changed it in Cricket records but forget to change in Chess records. So there will be two addresses of a student which is not possible.

File system

Cricket

Name: Shub
Address: Delhi
Name: Ramesh
Add: Udaipur

address changed →

chess.

Name: Aarya
Address: Jaipur
Name: Shub
address: 123, marine street, Jaipur.

↑ address forget to change.

→ This leads to inconsistency in data base.

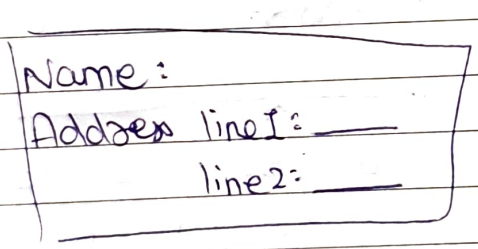
DBMS This does not occur in DBMS, because there is only one (common) table of address, so we need to change once only.

DOMS Page No
Date / /

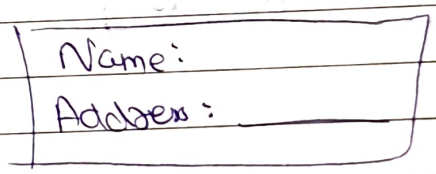
3. Data Isolation

→ In file system, different people can go with different formats for storing the data.

→ For example: An authority suggest particular format of storing data & resigns.
  → Another authority will be recruited and suggest another format of storing data
  → If same thing goes on there will be various formats by the time.
  → we will need different programs to access & manipulate the data stored in different format.
  → Hence, the data will be scattered & isolated.



format suggested by 1st person



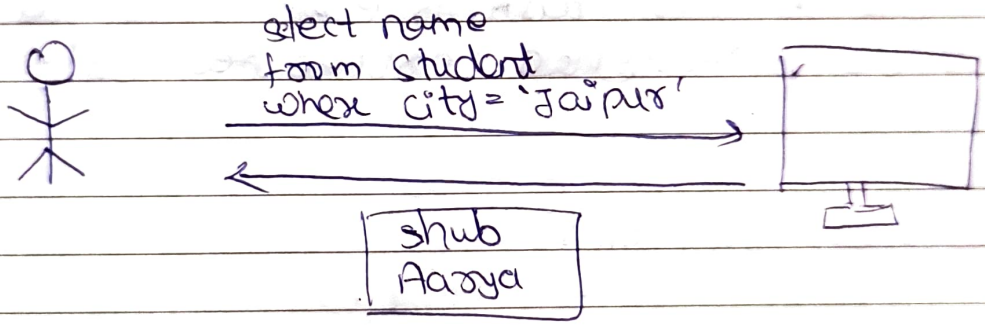
format suggested by 2nd person.

→ In DBMS, this can be avoided since only one fixed format can be decided and stored in system for long term.

4. More effortful Access & Manipulation of Data

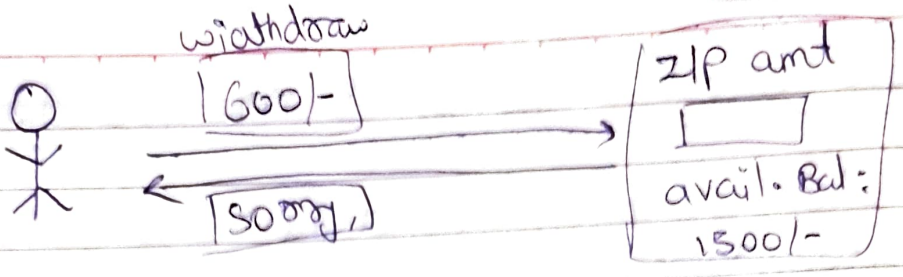
→ In file system, the data reside in different formats, so it is not easy to access the data according to our needs.
→ For example: If an authority asks an employee to get list of student who live in Jaipur, so the employee has to get names manually as there is no such program available. Time taking task.

→ In DBMS, employee has to type one or two lines and list will be generated.



5. Integrity problems

→ Integrity = putting constraint/restriction.
→ In file system, we can't impose constraints on our data.
→ In DBMS, we can put constraint.
→ For example: We put a constraint that in an account the minimum balance should be 1500
→ If you have ₹1500 in account & want to withdraw ₹600 then system will deny due to integrity constraint.



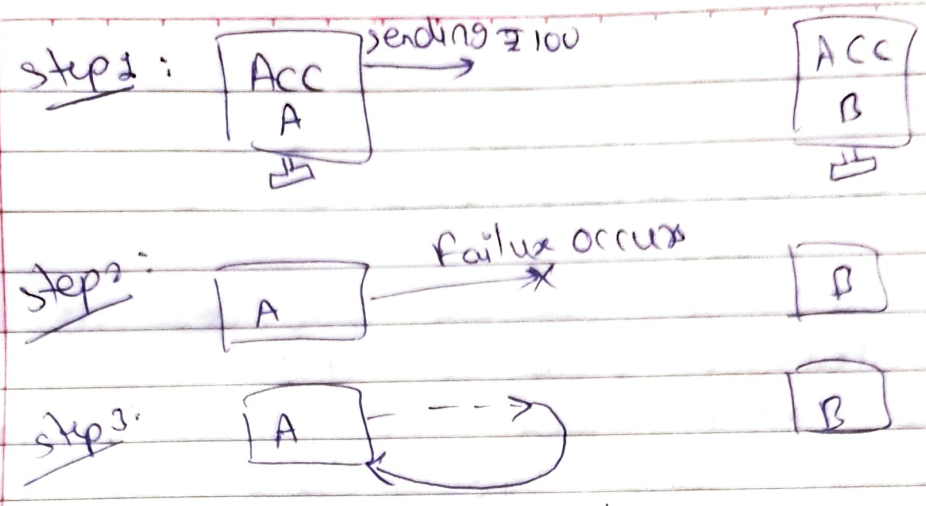
6. Backup Problems:

→ In file system, it is not easy to have a backup of all data.
As in file system different formats are existed and data exist is scattered form, so backing up data becomes a typical task.

→ In DBMS, this task is easy, we just need to follow predefined procedures only.

7. Recovery Problems:

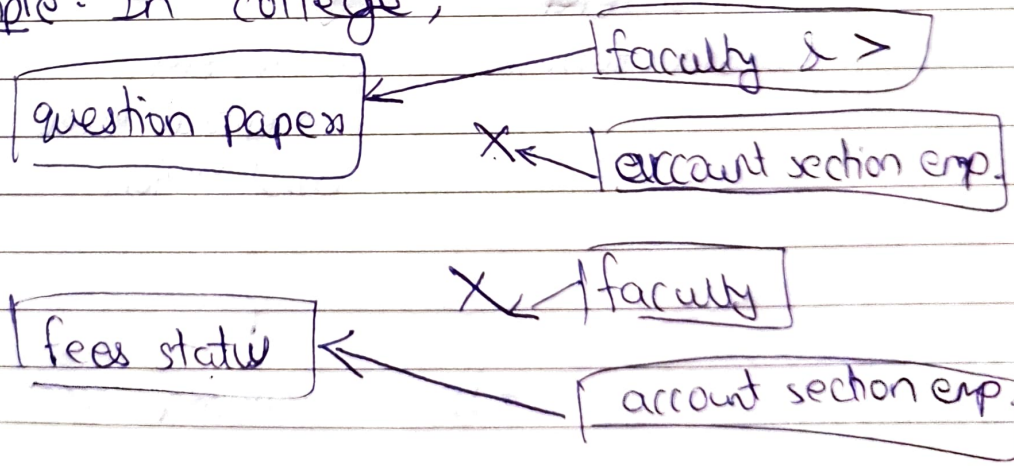
→ When there is a system failure, we need some strategy to recover the data or the system has to be restored to the same state, that existed before failure.
→ In file system, it is not possible.
→ For example: If ₹100 is transferred from account A to B, and failure occurs when 100₹ is deducted from A but does not deposited to B.
→ Then ₹100 should be deposited back to account A, otherwise the system will become inconsistent.



8. Security Issues:

→ In an organization, it is not necessary that all people have same level of access authorization for all data.

→ For example: In college,



→ Such constraint can be applied with DBMS by creating different usernames & passwords for different people in organization.

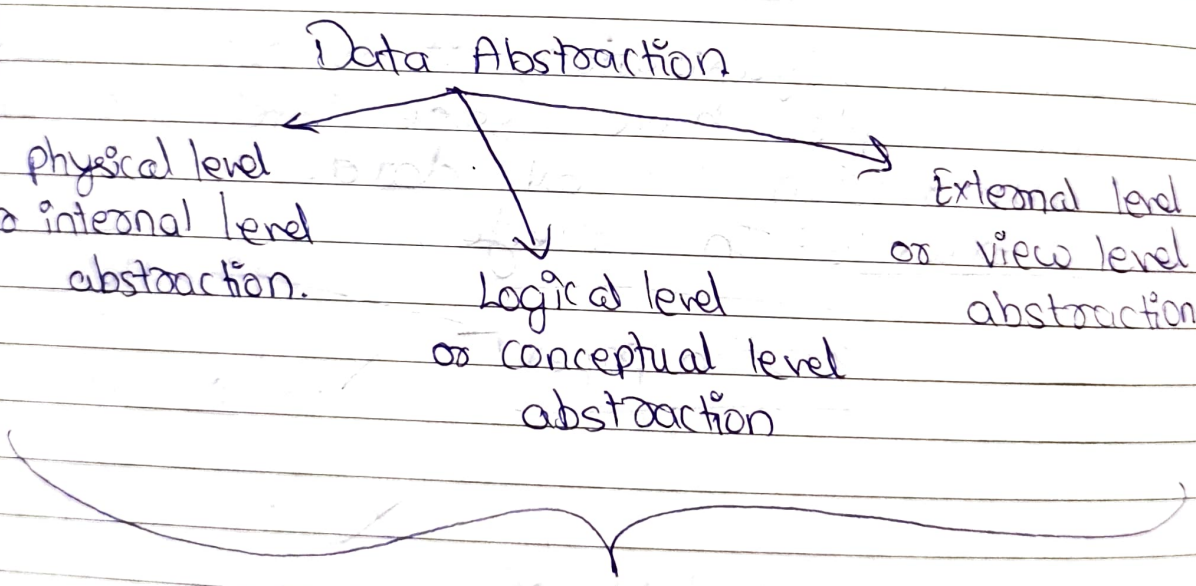
DOMS Page No.
Date / /

15 ~~Mapping~~

1. Data Abstraction / View of Data

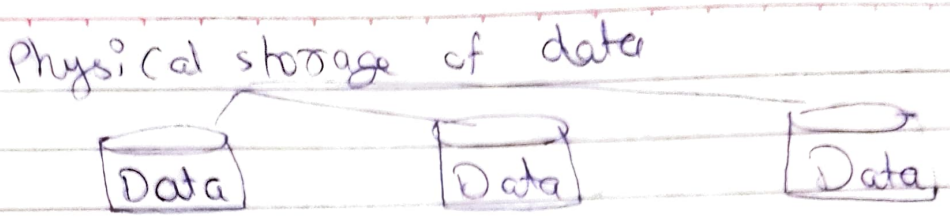
→ The database system provides its users an abstract view of data i.e. the system hides the internal implementation structure and type of data for providing the feature of simple and easy interfacing.

→ There are 3 levels of abstraction



(a) Physical level abstraction:

→ It tells how data is stored
→ Lowest level of abstraction
→ Describes database structure in detail
→ A Database Administrator (DBA) may have some idea about physical level storage

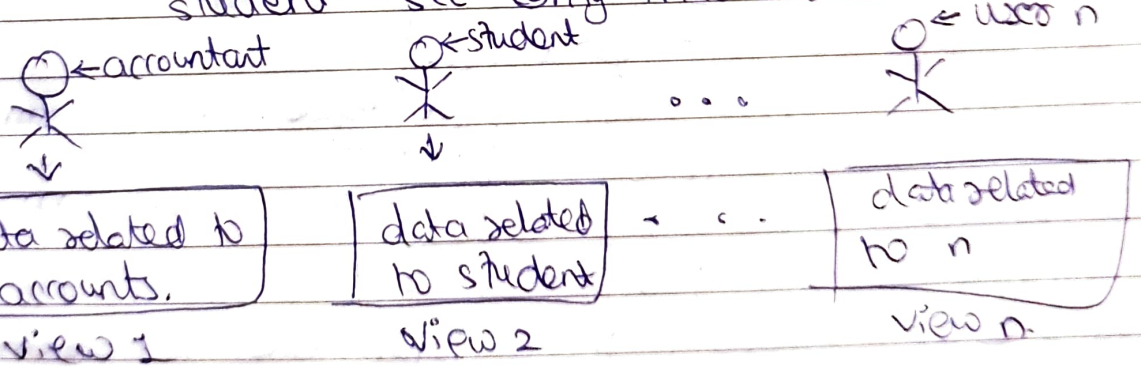


(b) Logical level abstraction:

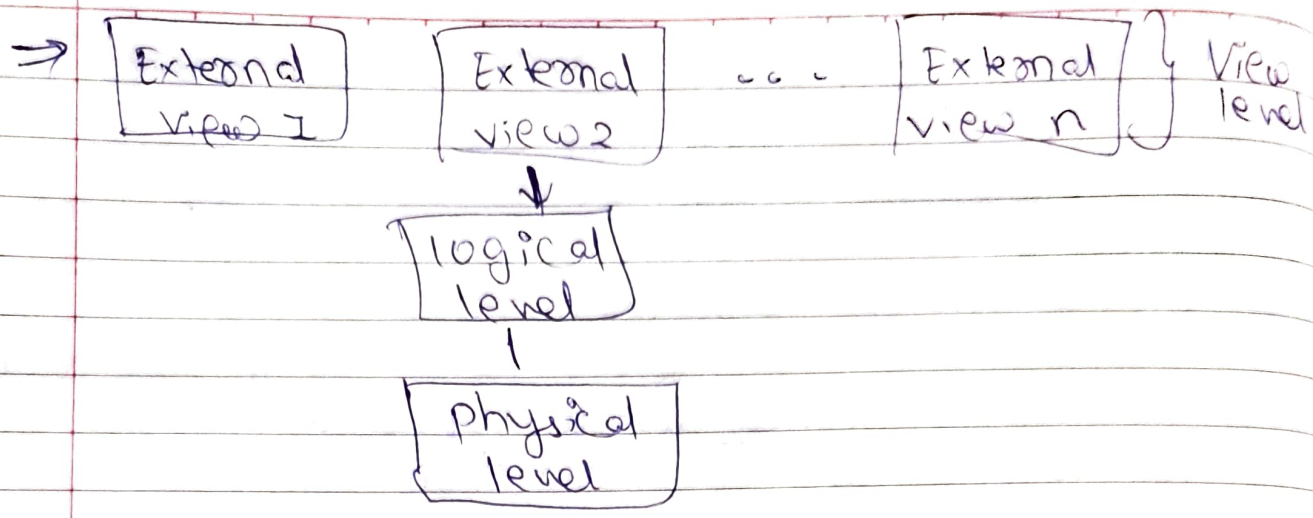
→ It tells what type of data is stored and also constraints associated with data.
→ middle level of abstraction
→ Describes Relationships among Data.
→ The decision of datatype and Constraint is taken by programmer and it is hidden from external user.

(c) View level abstraction:

→ It tells how the data is looked to external user
→ There can be number of different views.
→ For example: Accountant can see account section only student see only marks & fees status.

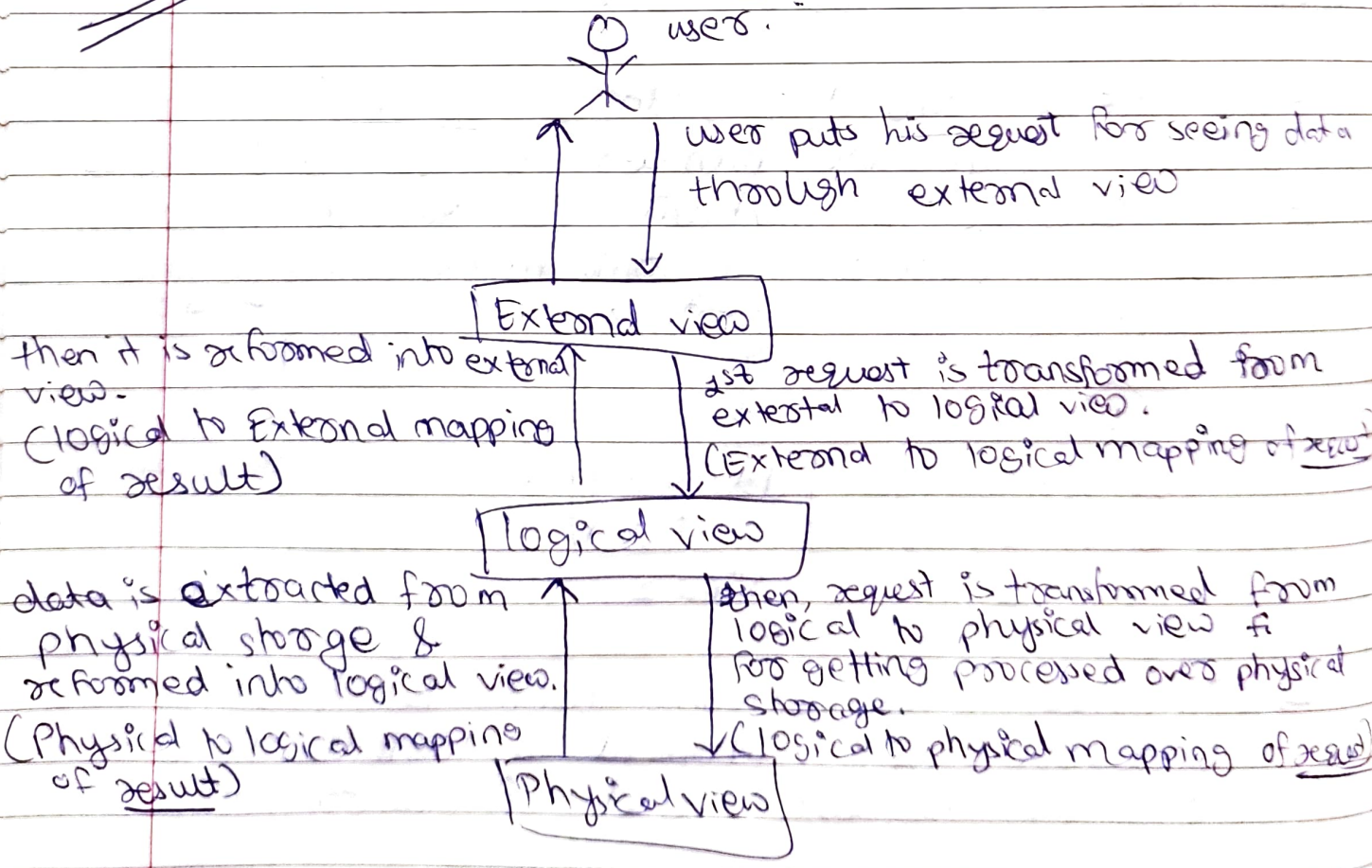


→ At this level external user is totally unaware of physical & logical level ∴ it is highest level of abstraction.



· Hierarchy of abstraction.

1. Concept of mapping



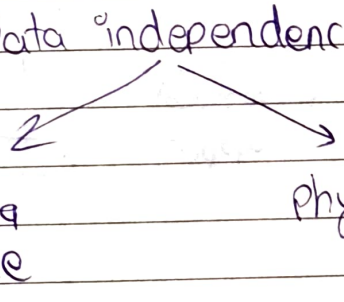
→ This process of transforming request & result from one level to another is called mapping.

1.3 Data Independence

→ Data independence is a feature or power to change the schema at one level without getting other level schema affected.

→ If we make changes at one level, then we do not need to make change at other level schema.

→ It means upper levels are unaffected by changes in lower level.



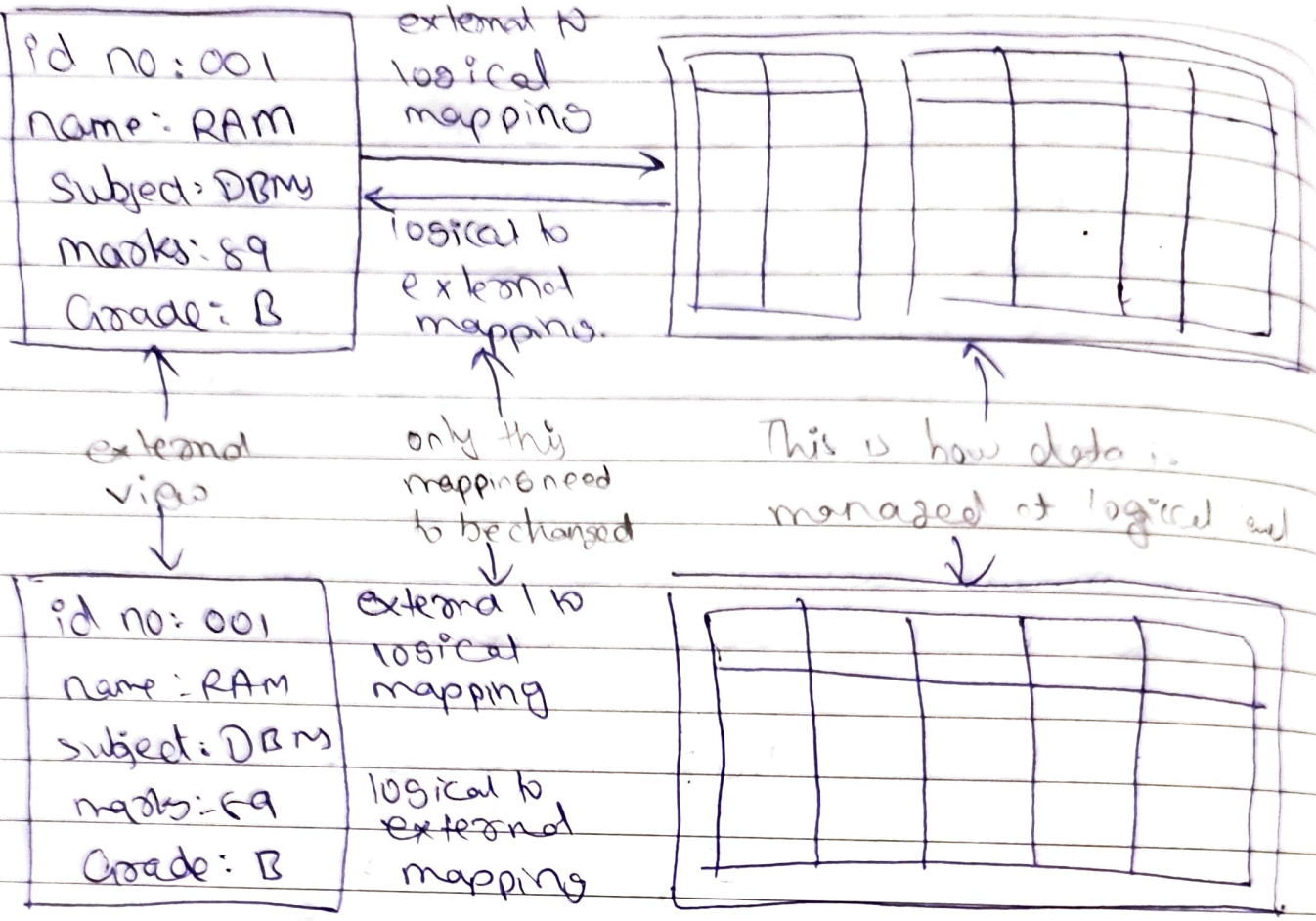
(a) Logical data independence.

→ It is a power to change logical schema without any need of changing external schema.

→ If we change structure at logical level then we do not need to change external structure

→ Difficult to achieve

(a)



(b) Physical data independance

→ Power to change physical level schema without making changes in logical level.

→ If we change structure at physical level then do not need to change structure at logical level

→ It is needed for improving performance in retrieval insertion or updation of data.

→ Example of changing physical schema: relocation of data, merging, splitting, providing new access path to data, etc.

→ Easy to achieve.

DOMS

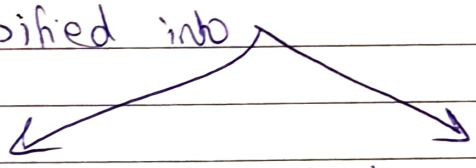
Database system is made up of few subsystems or components.

The basic building components are:

users & DBA	Query processor	storage manager	disk storage.

* Users & Database Administrator:

→ The people who interact with database are classified into



database users

database administrator

(a) Database users

→ There can be different users who use data base in different manners.

(b) Database administrator: (DBA)

→ DBA is a person who monitors and controls database.

→ Responsible for deciding level of access (read, read & write, read & write & edit) to a particular person.

→ The responsibilities that come under DBA :

- **Schema definition**

→ DBA has to define structure of DB by writing queries.
→ applying constraints through keys.

- **Modification of logical & physical level schema**

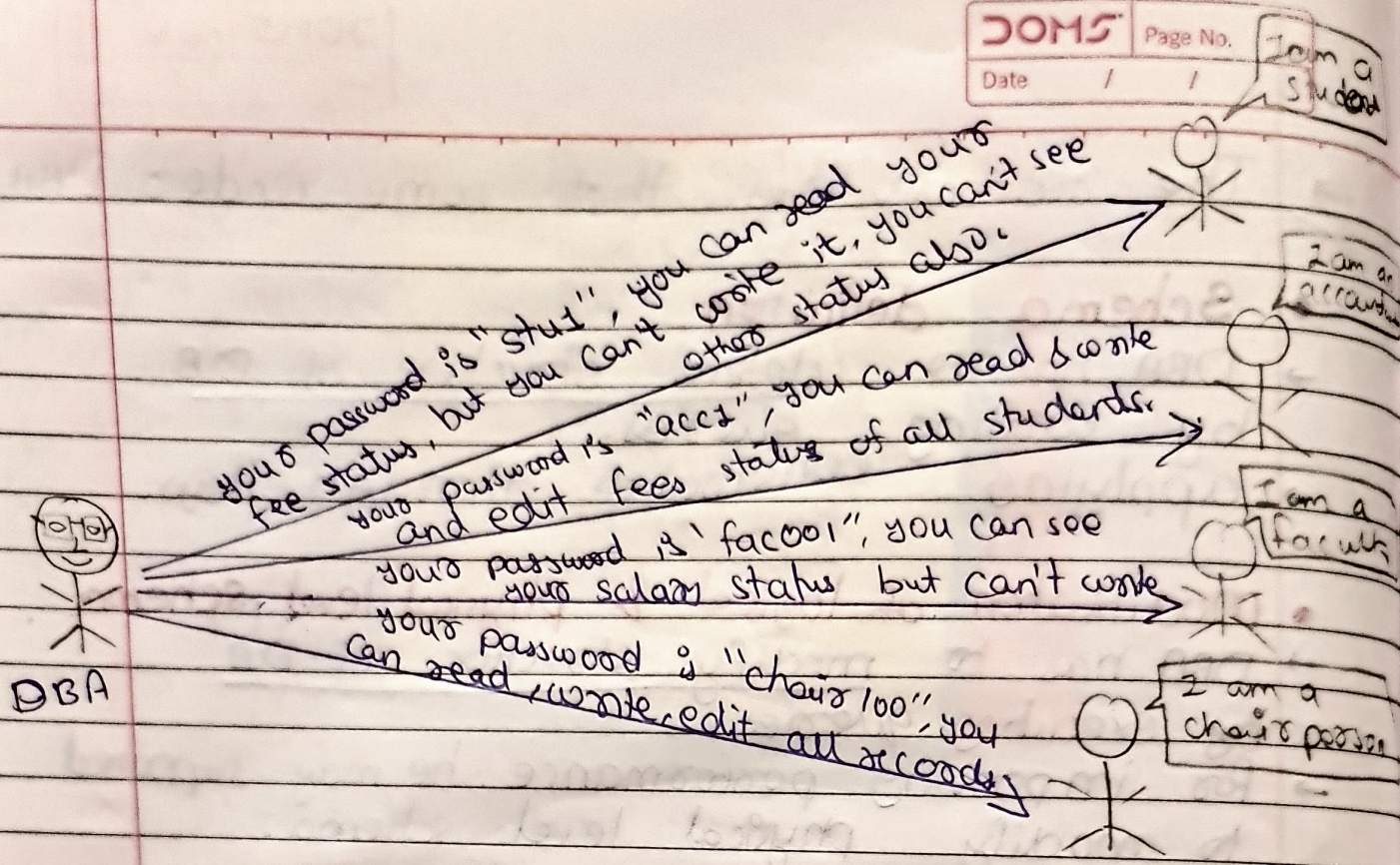
→ DBA has to modify structure of DB by executing queries.
→ for improving performance, he may supposed to modify physical level schema.

- **Routine maintenance**

→ The activities which are done on a regular or on fixed time interval are routine activities.
→ It includes - taking backup of data
- monitoring resource consumption by specific process.
- Availability of enough disk storage
- prevention of data by unauthorized parties

- **Granting different level of authorization for person**

→ within an organization every person should not be able to access all data.
→ DBA has to assign different level of access to different people.



* Query processor

→ It is a component of database architecture which process a query coming from one of the database users.

→ Query processor is made up of 3 main sub component:

- DDL interpreter
- DML compiler
- query evaluation engine

• DDL Interpreter

→ It interprets DDL statements (Data Definition Language statements)
→ Records data definition in data dictionary.
→ Data dictionary contains meta data (data abt data)
→ Eg. name → char(24)
data | metadata (this is kept in data dictionary by DDL interpreter)

• DML compiler

→ It translates DML statements (Data Manipulation Language statement)
DML statements are queries written for manipulating data.
→ It provides feature of query optimization (execution of query with lowest cost)
it picks the (form) which consume lowest cost

DML statements can be translated in many forms and all of them give same results.

→ Here, lower the time consumed, lower the cost will be } time ∝ cost
∴ cost can be seen as factor of time.

DML statements → [DML compiler] → Low level instructions

• Query evaluation engine

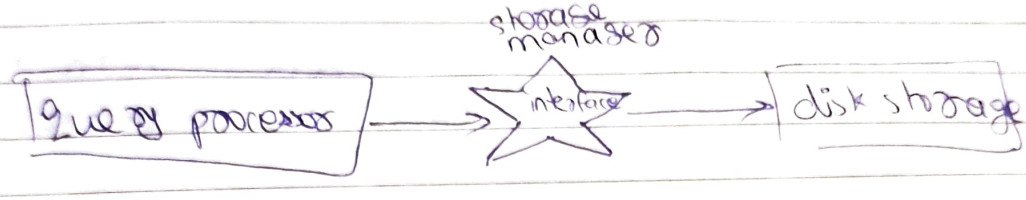
→ These (low level instruction) are executed by query evaluation engine.

Low level instruction → [Query evaluation engine] → evaluated query ready for fetching data from storage with

DOMS Page No. Date / /

* Storage Manager

→ It is a program module that provides interface between query coming from query processor and data that is stored in disk storage.



→ Storage manager program is made up of
- file manager
- buffer manager
- Transaction manager
- Authorization & integrity manager.

→ The work carried out by sub-components are:

- **File manager**

→ responsible for - managing space on disk
- deciding storage structure & data structure to store data.

- **Buffer manager**

→ Decides the data those have to be in cache memory
→ whenever a query is put then
① data is fetched from storage
② & same data put in cache memory for specified time.

So if other user or same user ask for same data then it can be fetched from cache memory

→ ∴ time for fetching reduces.

- **Transaction manager**

→ monitors - transaction that goes on particular time.
ensure - that transaction occurs fully or not at all.
- & consistency in database.

- **Authorization & integrity manager.**

→ ensures - integrity to be maintained in Database system.
→ Authorization manager checks - the user, weather it is authorized for specific access or not.

* Disk storage

→ It contains - the data itself &
- metadata
→ Different parts that can be seen in disk storage are:

- **Data files**

→ These are main storage, where actual data resides.

- **Data dictionary**

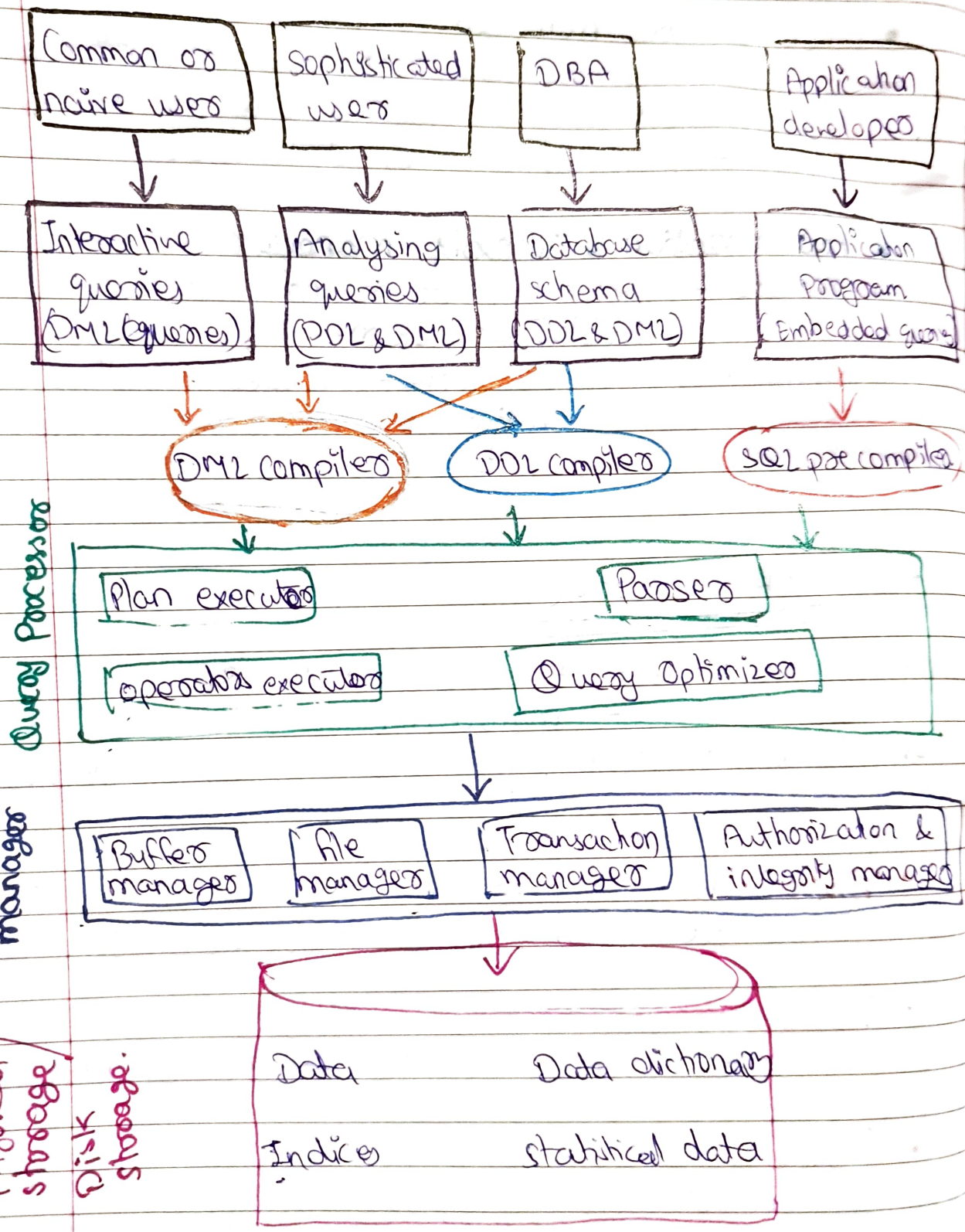
→ It stores metadata ; like structure of database

- **Indices:**

→ In textbook, index helps to find topics
→ similarly, database has indices to find the data fast & efficiently.

DOMS Page No. Date / /

* System architecture of DBMS



Ch:2 Relational Model

2.1 Structure of Relational DB
2.2 DB schema
2.3 Schema diagram
2.4 Domains, Relations
2.5 Relational Query Language
2.6 Relational operations.

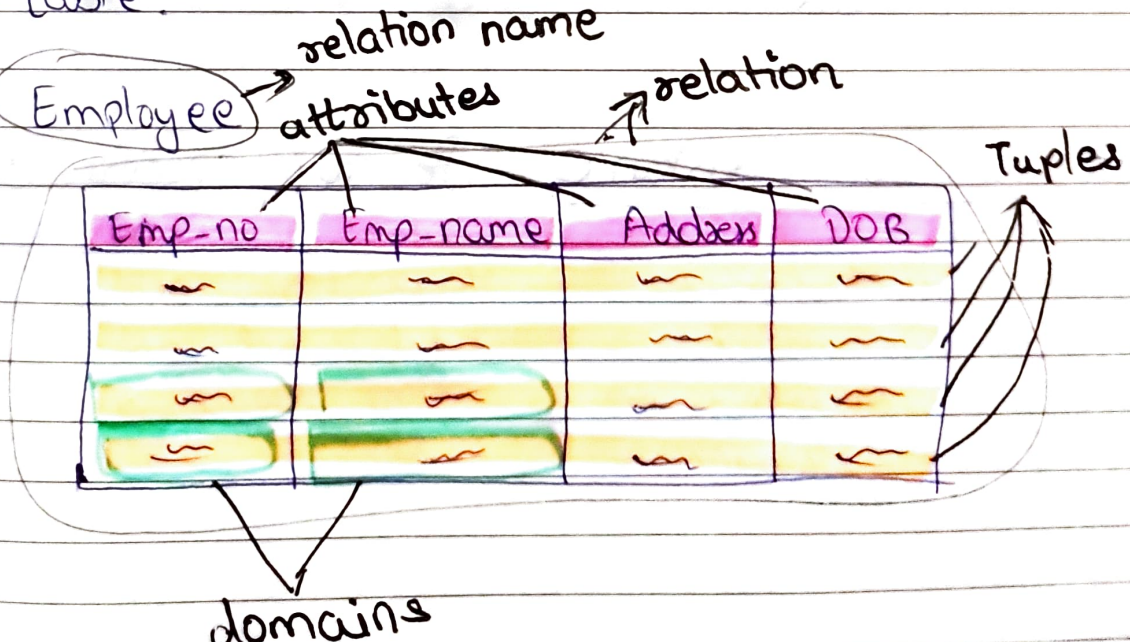
→ Relational model is primary data model.
→ It is used for data storage & processing.

2.1 Structure of Relational Database

→ Relational database consist of collection of tables.

→ Relation is a table with columns & rows.
- It is based on mathematical concept of relation which is represented physically as table.

→ Employee



→ Relation refer to table
→ tuple refer to row
→ attributes refer to column
→ Domain refer to all possible values of column

- Null value of domain – value which is unknown or does not exist
- Atomic domain – values of domain are considered to be indivisible units.

*. Properties of Relation:

1. Relation has unique name
2. each cell contains exactly one value.
3. each attribute has distinct name.
4. each tuple is distinct
5. The order of attributes & tuples has no significant.

⇒ Relation instance refer to specific set of rows & not whole table

Database instance: snapshot of data in DB at given instant in time

eg 4/7/2000 {40 entries} 9/10/2001 {400 entries}

2.2 Database Schema

→ Schema → logical representation.
→ It defines how the data is organized & how relations among them are associated.

→ It contains descriptive detail of database, which is shown using schema diagrams.

→ DB designer, designs the schema to help programmers understand the database.

→ Database schema are divided into.

Physical DB schema	Logical DB schema
→ It is actual storage of data in form of files, indices, etc.	→ It defines logical constraints that need to be applied on data stored.
→ It defines how data will be stored in secondary storage.	→ It defines tables, views & integrity constraints

→ schema describes the attributes & not the data.

→ eg: Student relation

SID	Name	dept_name	semester
001	aashka	CE	2
002	shubham	CSE	4
003	kartik	IT	3
004	om	mech	2

The schema is.

Student (SID, Name, dept_name, semester)

2.3 Schema diagrams

→ pictorial representation of database that shows
- relations in database
- their attributes
- primary keys
- foreign keys.

Primary key
→ values of attribute should be unique

Foreign key
→ reference of primary key

Table 1: Department

DID	DName	Address
D1	-	-
D2	-	-
D3	-	-
D4	-	-

primary key

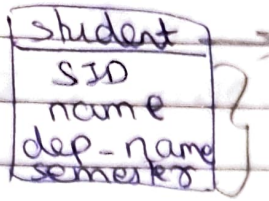
Table 2: Employee

EID	Ename	Address	DID (foreign key)
001	-	-	D1
002	-	-	D1
003	-	-	D4
004	-	-	D3

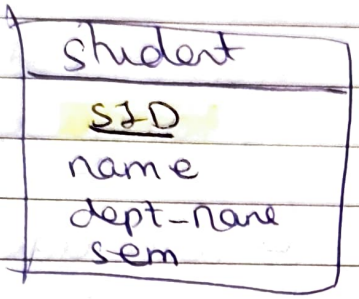
referenced

⇒ Points to be noted:

- Each relation appears as box

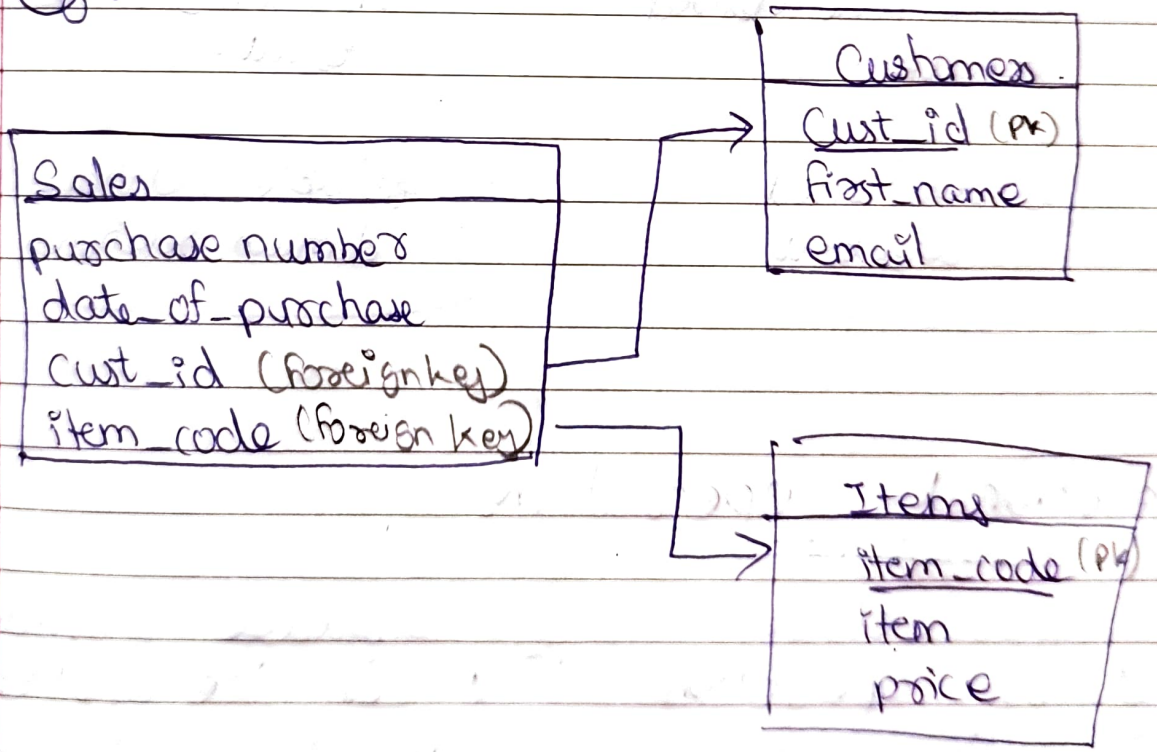


- Primary key attributes are shown underlined.



- Foreign key dependencies appear as arrows from foreign key to primary.

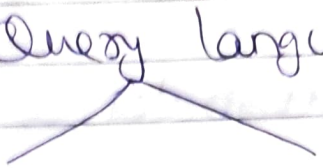
→ eg:



2.5 Relational Query Languages

→ It is a language in which a user request information from database.

→ Query languages



Procedural language

- → user instructs the system to perform sequence of operⁿ on database to compute desired results.
- → eg: relational algebra

non-procedural language.

- → users describes desired results without giving specific procedure for obtaining that result.
- → eg: tuple relational calculus, domain relational calculus

2.6 Relational operations

→ All procedural query language provides set of operations that can be applied on either a single relation or pair of relations.

→ relational algebra: provides set of operations that take one or more relations as i/p & return a relation as o/p. SQL is based on relational algebra.

Outline of few operations.

Operation	Description.
Selection (σ)	Return rows of i/p relation that satisfy predicate. $\sigma_{salary>=8500}(instructor)$
Projection (Π)	Return specified attributes from all rows of i/p relation. Remove duplicate tuples. $\Pi_{ID,salary}(instructor)$
Natural join ($\bowtie$)	It matches tuples whose values are same on all attribute names, that are common in both i/p relations. instructor $\bowtie$ department.
Join	Combine two relation by merging pair of tuples.
Cartesian product ($\times$)	Returns all pair of rows from two i/p relation. $\Pi_{name}(instructor) \times \Pi_{name}(student)$
Union ($\cup$)	Returns union of tuples from two i/p relation. $\Pi_{name}(instructor) \cup \Pi_{name}(student)$

Ch:3 Entity-Relationship model

3.1 Basic concepts
3.2 Design process
3.3 Constraints
3.4 Keys
3.5 Design issues
3.6 ER diagrams
3.7 Weak entity sets
3.8 Extended ER features, generalization, specialization, Aggregation.
3.9 Reduction to ER database schema

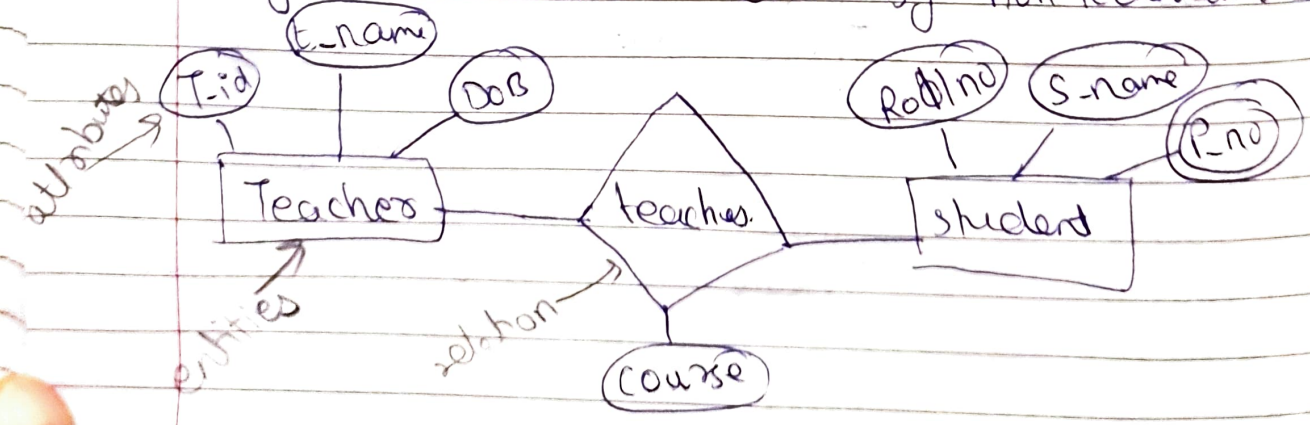
3.1 Basic Concepts

→ ER model = conceptual view of database

→ Introduced by : Dr Peter Chen
in : 1976

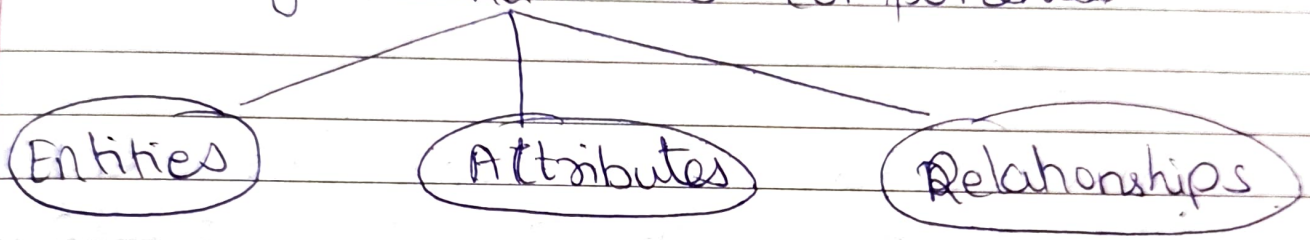
→ It is a non-technical designing method.
→ works on conceptual level
→ works around real-world entities

→ It is a diagrametic representation
→ Easy to understand even by non-technical users.



* ER diagrams are used to
→ Identify data that must be captured, stored & retrieved in order to support business.
→ Activities performed by an organisation.
→ Identify data required to derive and report on the performance measures, that an organisation should be monitoring.

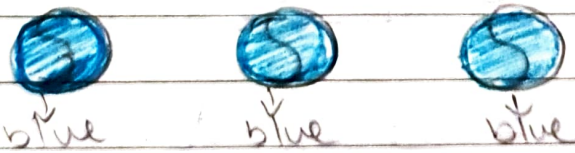
* ER diagram have 3 components.



* Entity

⇒ An entity is a thing / an object in real world, that is distinguishable from other object based on the values of attributes.

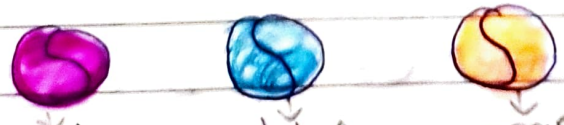
→ eg:



these are objects ✓
but entity ✗ because all are blue and we can't distinguish

Here, object = balls.

eg:



Attributes : [pink

these are objects ✓
& entity ✓ because we can

DOMS Page No.
Date / /

Types of Entities.

1) Tangible : Entities having physical existance in real word.
eg: Car, Pen, Bank-locker.

2) Intangible : Entities which exist logically
eg: Bank-account.

Entity set

⇒ Collection/set of same type of entities that share same properties or attributes.

→ It is represented by table in relational model

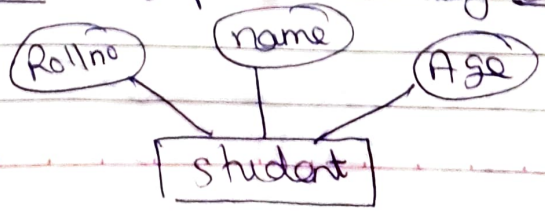
entity set

Student

RollNo	name	age
1	A	18
2	B	19
3	C	20

So, student = entity set
& student A
" B
" C
} = entities.

→ Entity set is represented by rectangle in ERD.



* Entity cannot be represented in ERD.
* Entity can be represented in relational model by row / tuple / record.

* Attributes.

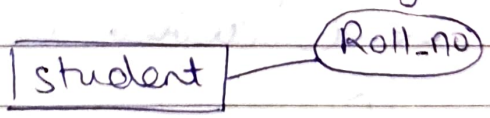
⇒ Attributes = units that describes the characteristics of entities

→ For each attribute there is set of permitted values called domains.

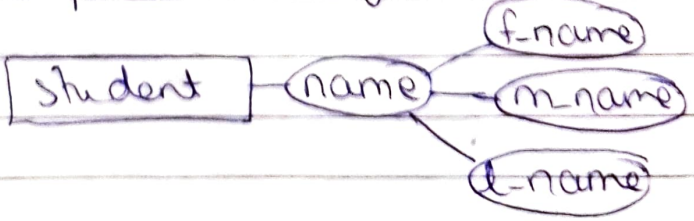
→ It is represented as: Oval in ERD
& seperate column in Relational model

* Types of Attributes.

1) Simple : can't divide
: represented by simple oval.



Composite : can be divided into simple attribute.
: represented by oval connected to oval.



2) Single : have one value at a time
: eg: Roll no.

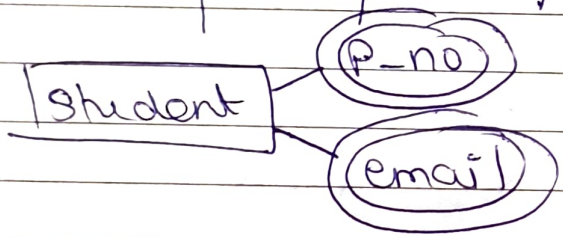
multivalued : have >1 value.
: eg. phone no., email addzes.
: represent by seperate column in Relational model
: " " double oval in ERD)

eg: Student

Roll_no	Name	DOB

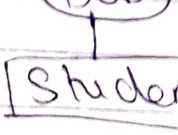
Roll_no	p_no

Roll_no	email



3) stored : stored permanently
: eg: DOB

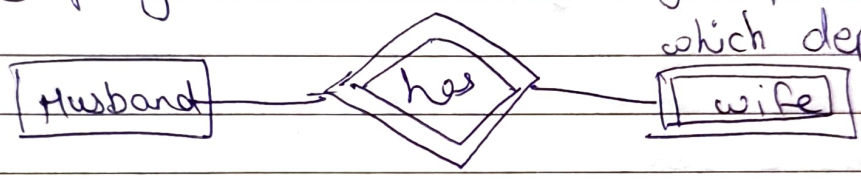
Derived : can be calculated based on other attribute
: eg : age
: age can be calculated at runtime by subtracting ~~current~~ DOB from current date.
: represented by dotted oval in ERD.



DOMS Page No.
Date / /

* Regular Entity Type Vs Weak Entity Type.

Regular Entity Type	Weak Entity Type
→ has its own key attribute	→ doesn't have key attribute of its own
→ does not depend on any other entity for its existance	→ Its existance depends on existance of other entity.
→ eg: employee, student	→ eg: Spouse of employee which depend on employee



* Null values

→ If attribute does not have a value for a particular entity then it is null value.

→ [0] ✗ , [ ] ✗ , [Null] ✓

3.2 Design process

step1 Determine DB purpose
step2 Discover & collect info
step3 Divide the entities into table
step4 Turn data points into column
step5 Identify primary & foreign keys
step6 Normalize & Refine.

DB design principles
- Duplicate information is bad ☹
- Accuracy & completeness is good ☺

3. ~~Constraints~~ continue.

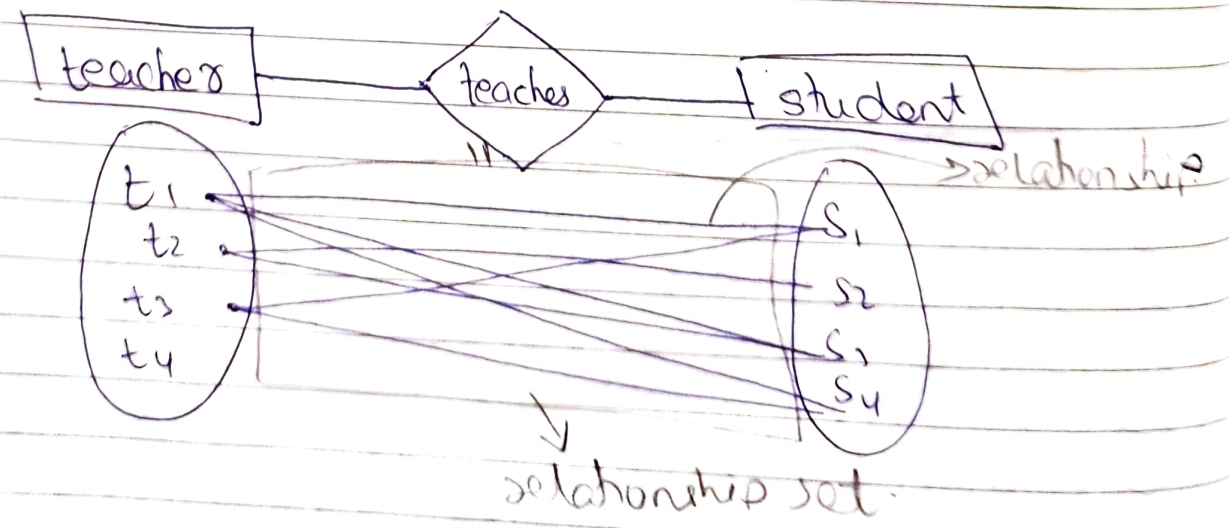
* Relationship

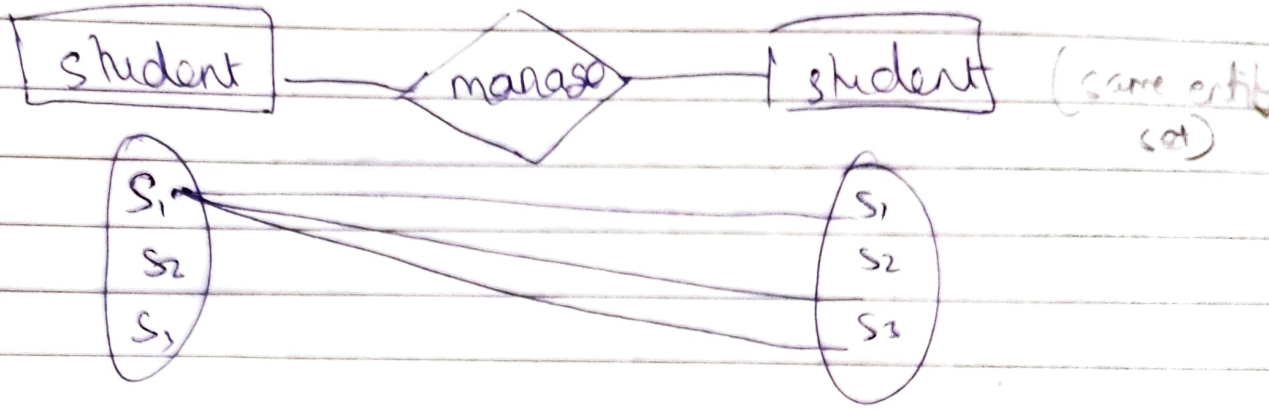
⇒ Relationship = association between two or more entities of same or different entity set.

→ no representation in ERD.
→ represented using row in a table in relational model.

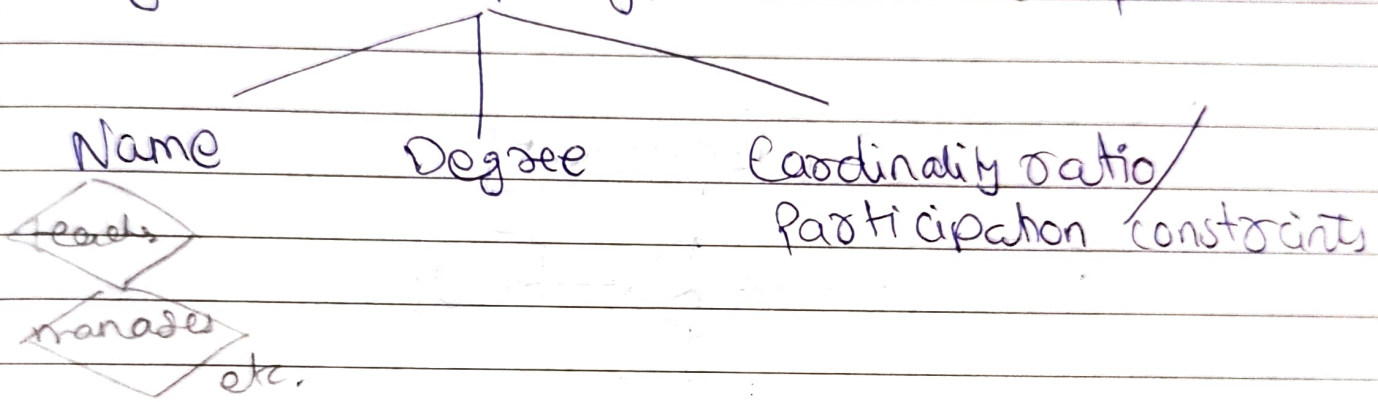
* Relationship set

→ Set of similar type of relationship.
→ represented by diamond in ERD
→ In relational model, represented either by a seperate table or seperate column (foreign key)





- Every relationship type has 3 components.

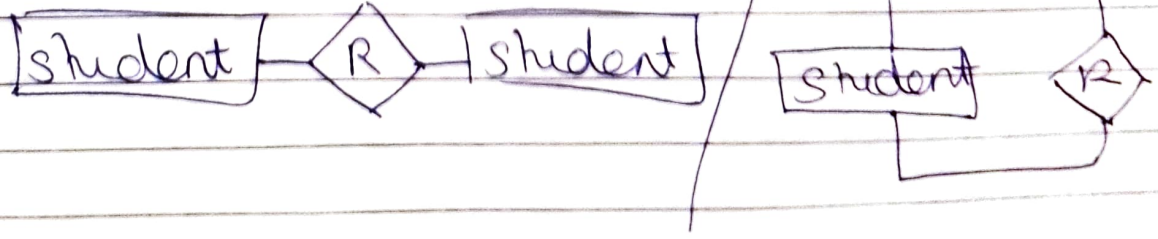


* Degree of relationship set

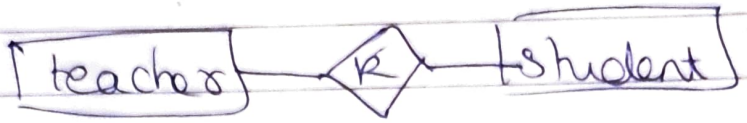
⇒ no. of entity set associated (Participated) in relationship set.

→ most of the relationship sets in ERD are binary (in industries).

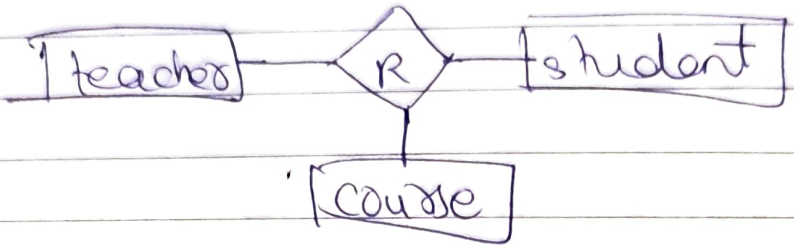
- unary. (minimum)



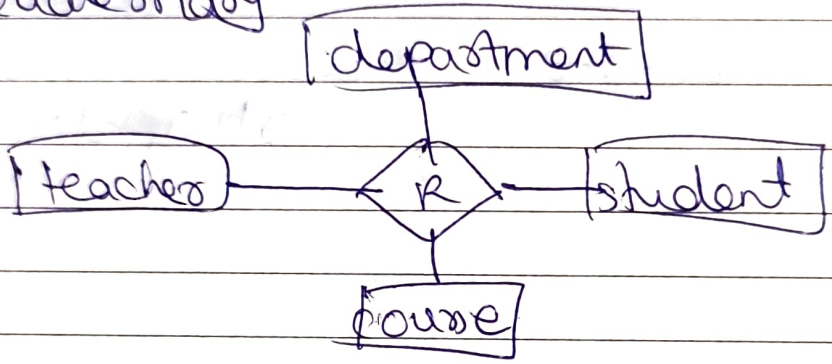
- binary (✗)



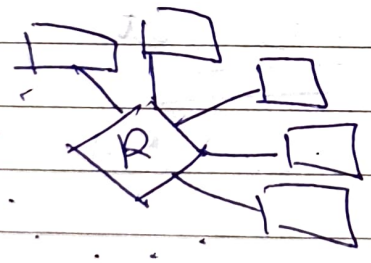
- Ternary



- Quaternary

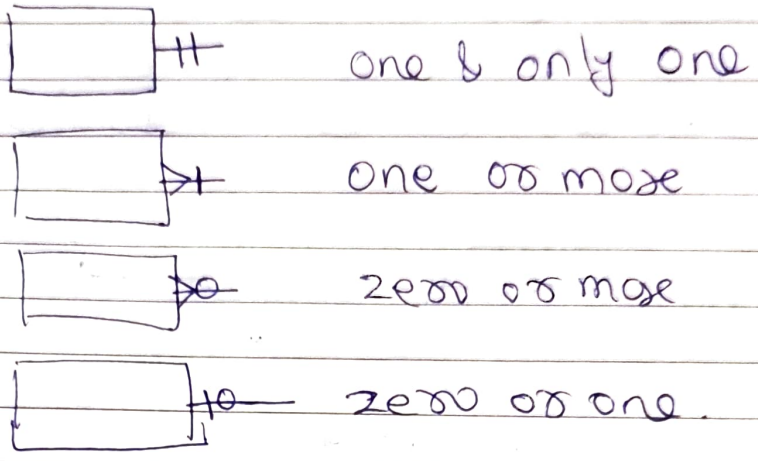


- N-ary.



33 Constraints

* Cardinality



0 = zero
1 = one
> = many
|| = (min = 1 & max = 1)

- eg:

① A associated with minimum 0 B & max 1 B.

② A asso with min 1B & max 1B

③ A asso with min 0B & max >1

④ A asso with min 1B & max >1

⑤ A asso >1 B

- One-to-one (1:1) Relationship (←→)

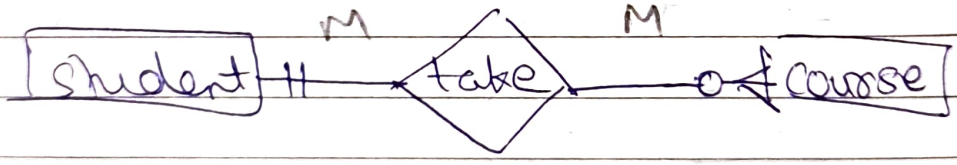
eg:

- One-to-many (1:M) (←—)

eg.

- many-to-many (M:N) (——)

eg.



* Relationship participation

1) Partial:

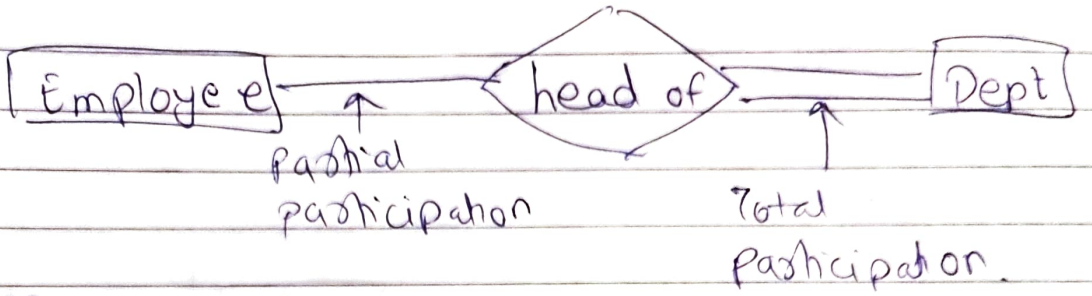
→ all Employee don't participate in relationship (head of), because every employee is not head of department. ~~Hence~~

→ Hence partially participated.

2) Total:

→ All department have heads.

→ Hence total participation.



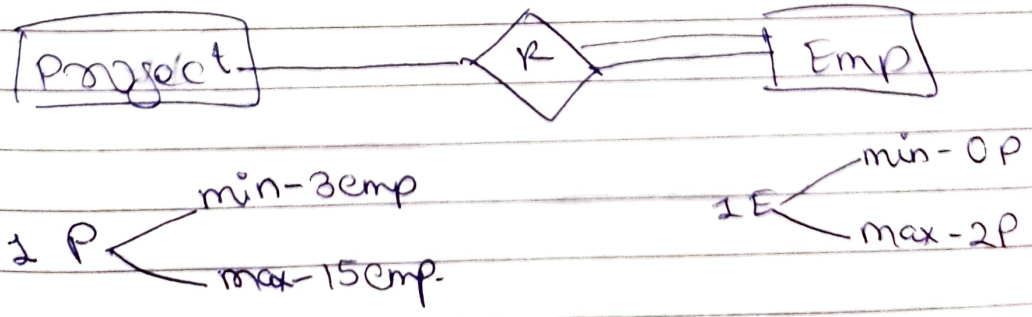
If there is atleast one entity that doesn't participate is partial.

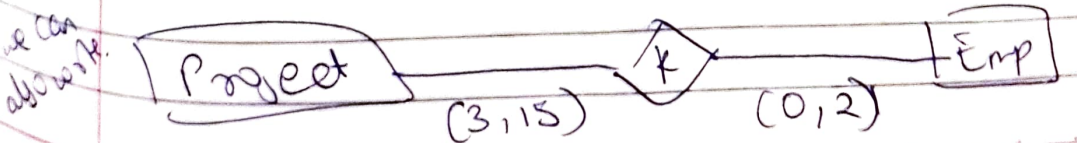
* Participation constraints.

→ Specifies minimum & maximum no. of relationship instances that each entity can/must participate.

→ max Cardinality: max no. of times an entity occurance participating in relationship.

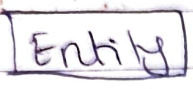
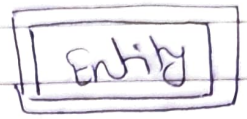
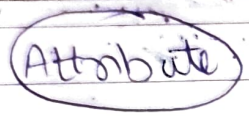
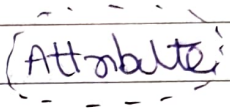
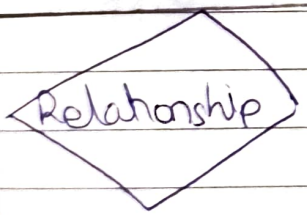
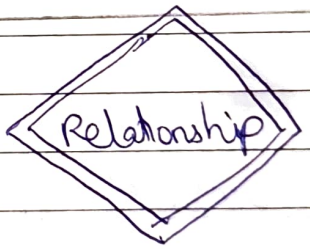
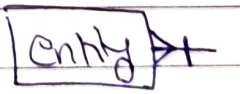
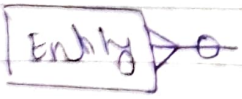
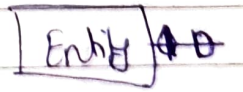
→ min Cardinality: min no. of times an entity occurance participating in relationship.

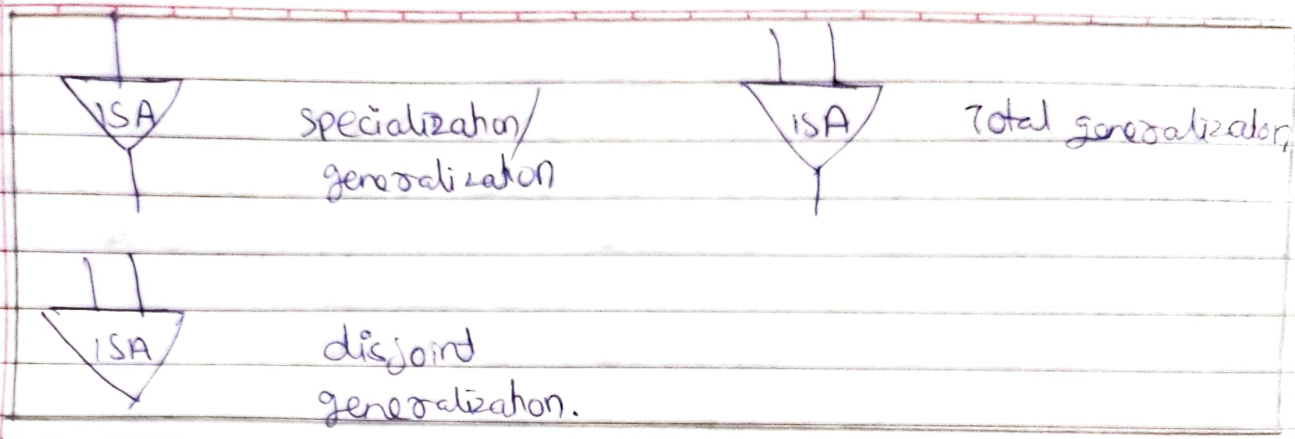


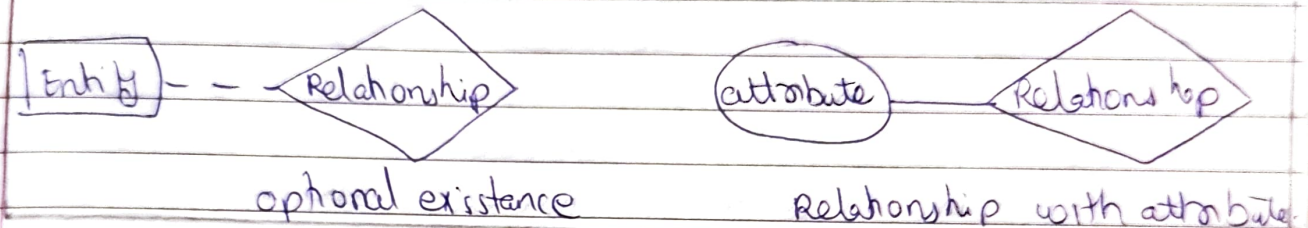


3.6 ER diagrams

* Notations.

	entity set		weak entity set
	attribute		PK attribute
	multivalue att.		Derived att.
	Discriminating att. of weak entity set		
	Relationship set		Identifying relationship set for weak entity set.
	1 & only 1		1 or more
	0 or more		0 or 1
	>1 always.		





* Case studies

(1) Notown Records:

(i) Each musician has M-id, address & phone no.
Each musician plays several instrument &
given instrument is played by several musician

(ii) Each instrument has name & musical key

(iii) Each album has title, copyright date, format
& album identifier
Each album has no. of songs but
no song may appear more than once in album.

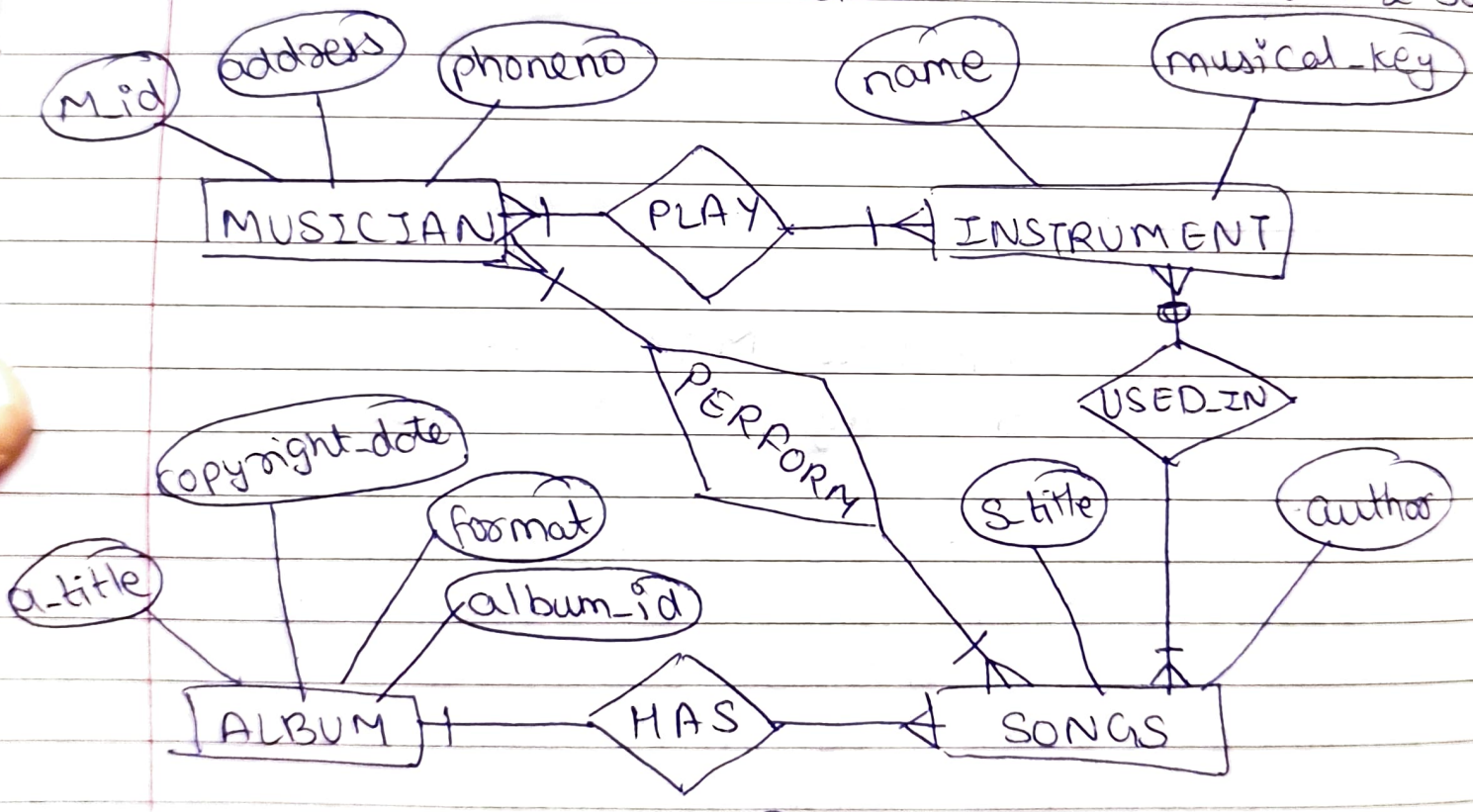
(iv) Each song is performed by one or more musician
& musician may perform no. of songs.
Each song has title & author.

Entity-sets for company.

1) MUSICIAN, with attributes M-id, address & phoneno
2) INSTRUMENT, with attributes name & musical-key
3) ALBUM, with attributes title, copyright_date, format & albumid.
4) SONGS, with attributes title & author.

Relationship-sets for company

1) 'PLAY', M to M between MUSICIAN & INSTRUMENT
2) 'HAS', 1 to M between ALBUM & SONG
3) 'PERFORM', M to M between MUSICIAN & SONGS
4) 'USED_IN', M to M between INSTRUMENT & SONGS



(2) Car insurance company.
has set of customers
each owns 1 or > cars.
Each car 0 or > accident.

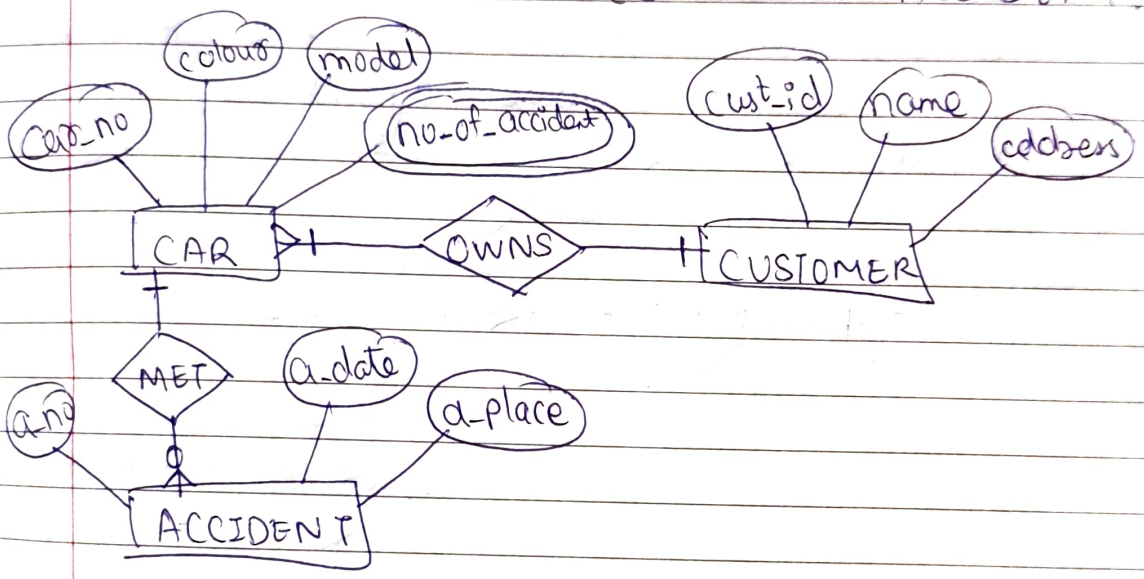
DOMS Page No.
Date / /

Entity-sets for Insurance company

1) CAR, with attributes car-no, colour, model & multivalued attribute of no of accidents
2) CUSTOMER, with attributes cust_id, name, address.
3) ACCIDENT, with attributes a-no, a_date, a_place.

Relationship-sets for Insurance company

1) 'OWNS', M to M between CAR & CUSTOMER
2) 'MET', M to M between CAR & ACCIDENT.



---

3) ART gallery.

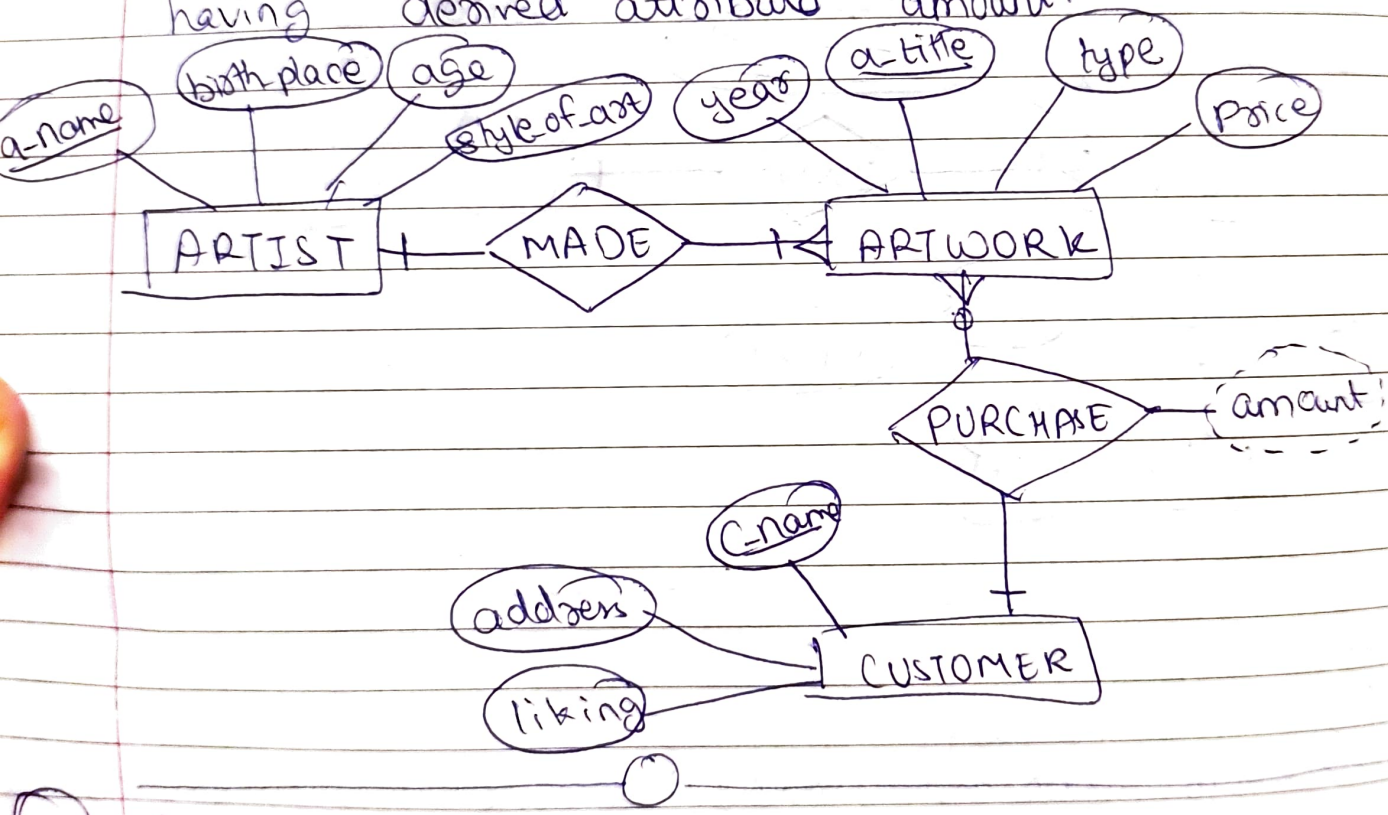
→ keeps info abt 'artist', their name(unique), birthplace, age, style of art.
→ 'artwork', the artist, year it was made, unique title, type of art & price.
→ ~~'artwork' is classified into~~
→ keeps info abt 'customers', unique names, address, amount of dollars & likings.

Entity-sets for gallery

1) 'ARTIST', with attributes a-name(unique), birth_place, age, style_of_art.
2) ARTWORK, with attributes year, a_title(unique), type & price
3) CUSTOMER, with attributes C-name(unique), address, liking, total amt.

Relationship-sets for gallery

1) 'MADE', 1 to M between ARTIST & ARTWORK.
2) 'PURCHASE', 1 to M between CUSTOMER & ARTWORK having derived attribute amount.



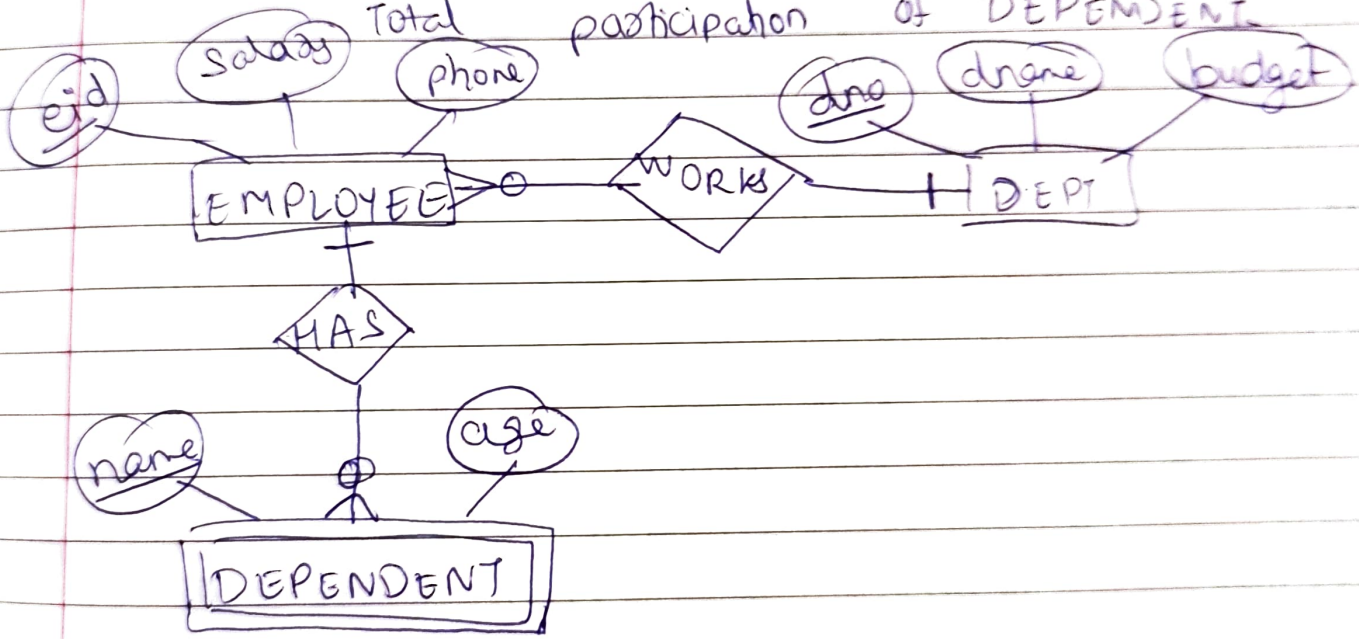
(4) Company stores info about
- employees (eid, salary, phone)
- department (dno, dname, budget)
- dependent of employee (name & age)
- Employees work in department
each department is managed by an employee
a dependent must be identified uniquely by name if employee is known.

Entity-sets for company

1) EMPLOYEE, with attributes eid, salary, phone
2) DEPT, with dno (unique), dname, budget.
3) DEPENDENT, with (name, age)

Relationship-sets for company

1) 'WORK', M to 1 between EMPLOYEE & DEPT.
2) 'HAS', 1 to M between EMPLOYEE & DEPENDENT with partial participation of EMPLOYEE & Total participation of DEPENDENT



DOMS

Ch: 6 Transaction & Recovery Management

6-1 Transaction concept
6-2 Properties of transaction
6-3 Serializability of transaction
6-4 Testing for serializability
6-5 System recovery,
6-6 Two-phase commit Protocol.
6-7 Recovery & atomicity
6-8 Log-based recovery.
6-9 Concurrent execution of transaction & related problems
6-10 Locking mechanism
6-11 Solution of concurrency related problem
6-12 Deadlock
6-13 Two-phase locking protocol
6-14 Intent Locking.

6-1 Transaction concept

⇒ Transaction = Set of instructions/operation that perform a logical unit of work.

Transaction

set of instruction

R(A)
A = A - 10
W(A)
R(B)
B = B + 10
W(B)

transfer ₹10 from acc A to B } logical unit of work from client side.

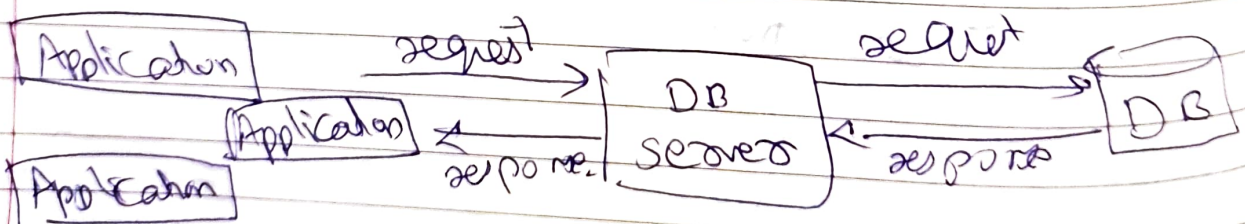
- main issues to deal with transaction.

failures
like hardware failures, system crashes

Concurrency.
execution of multiple transaction

- Example of transaction:
  - → withdraw money from ATM
  - → money transfer
  - → update address at online profile
  - → change password.

→ A transaction is often initiated by an application = program (API)
→ API contacts → DB server.
→ DB servers update → DB.



Redd Write Commit

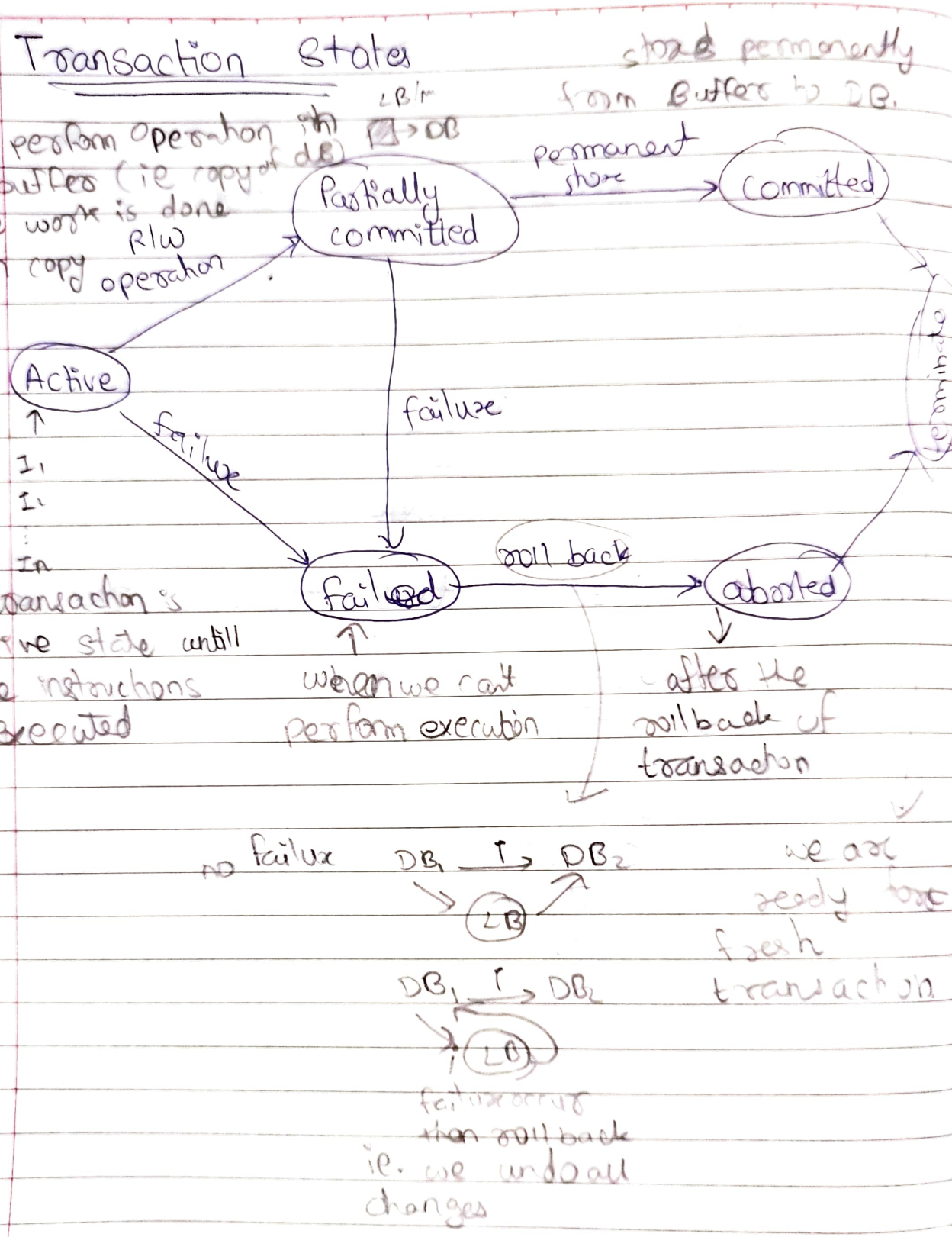
have high probability than write

used to change/ update in Data value

eg : Alice wants to transfer 700 to Bob.

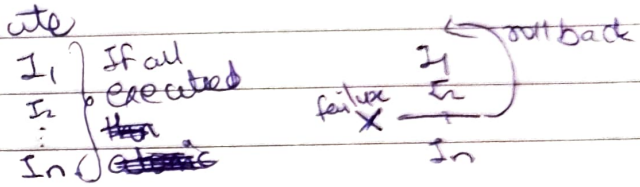
```
read(A)
A = A-700
write(A)
read(B)
B = B+700
```

* Transaction States



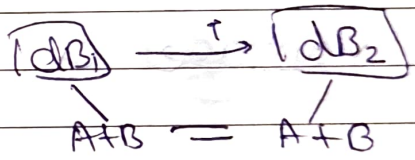
6.2 ACID properties

(A) Atomicity → All the instructions OR none in transaction should execute



→ Transaction management component ensures atomicity.

(C) Consistency. → If dB is consistent b4 transaction then it is consistent after transaction



→ If Atomicity, Isolation & Durability works properly then consistency automatically holds good.

(I) Isolation → It means logical isolation
→ i.e. no physical isolation.
→ Here, All transaction runs parallely but ~~other~~ transactions does not affect each other.

→ wrt ↓ Resource utilization ↑ efficiency ↑

→ Concurrency control component ensures isolation.

→ eg: There are lakhs of transaction occurring during Tatkal, but the transaction of 2nd person won't affect 1st person.

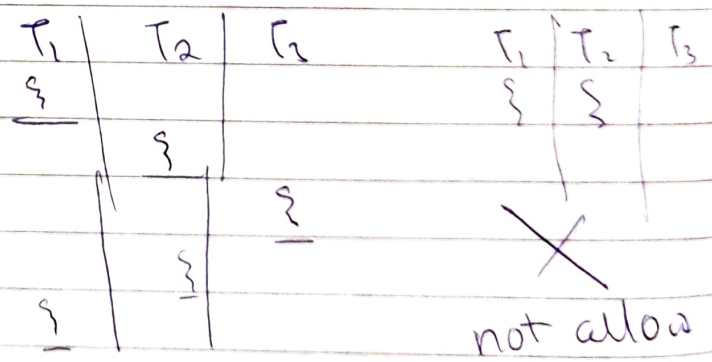
DOMS Page No.
Date / /

(d) Durability. → After the transaction is committed then the change in db should persist. irrespective of sw/Hw failure
→ eg. initially i = 10
then i = 20 committed
so i = 20 should remain as it is.
→ Recovery Management component ensures durability.

6.3 Serializability (it is a method to check, which schedules are consistent)

* Advantage of Concurrency.

→ Concurrency = multiple transaction at a time
→ it means context switching between transaction.

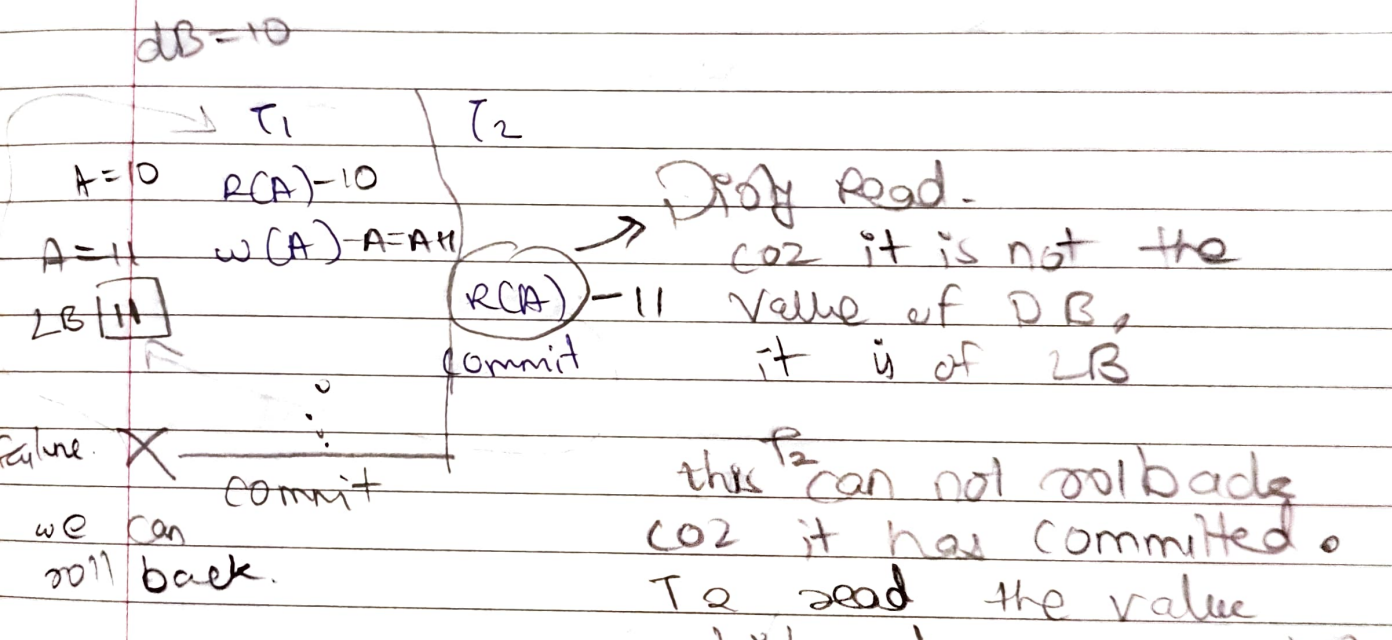


1. waiting time ↓
2. Response time ↓
3. Resource utilization ↑
4. efficiency ↑

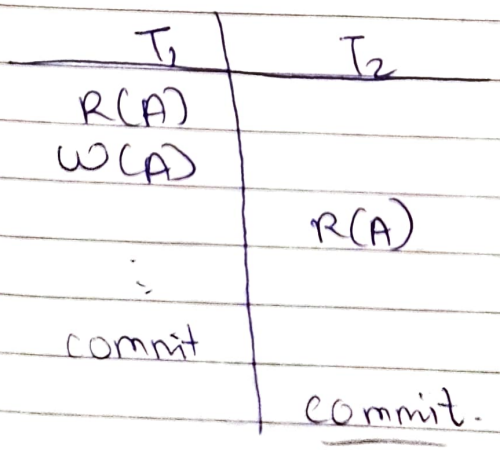
The problems that occur when we run transaction parallelly/concurrently

* Dirty Read Problem (dependency)

→ When a transaction reads a value ~~written~~ returned by an uncommitted transaction the it is said to be dirty read.



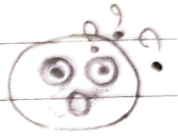
→ Solution



* Unrepeatable read problem.

DB [x = 10]

T_1	T_2
x = 10 R(x)	
	R(x) — 10
LB [x = 11] W(x) - x++	
	R(x) — 11



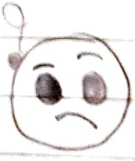
E ka ho gaya be ???!

Here, isolation property does not satisfy. bcoz change in T_1 affects T_2

* Phantom read problem

DB [x = 10]

T_1	T_2
LB [x = 10] 10 — R(x)	
	R(x) — 10
LB [ ] delete(x)	
	R(x) — variable not found



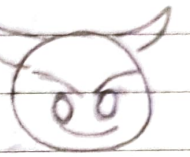
How is this possible?

abhi to 10 tha not found kese ho gaya

* Lost update problem (write-write conflict)

DB [A=10]

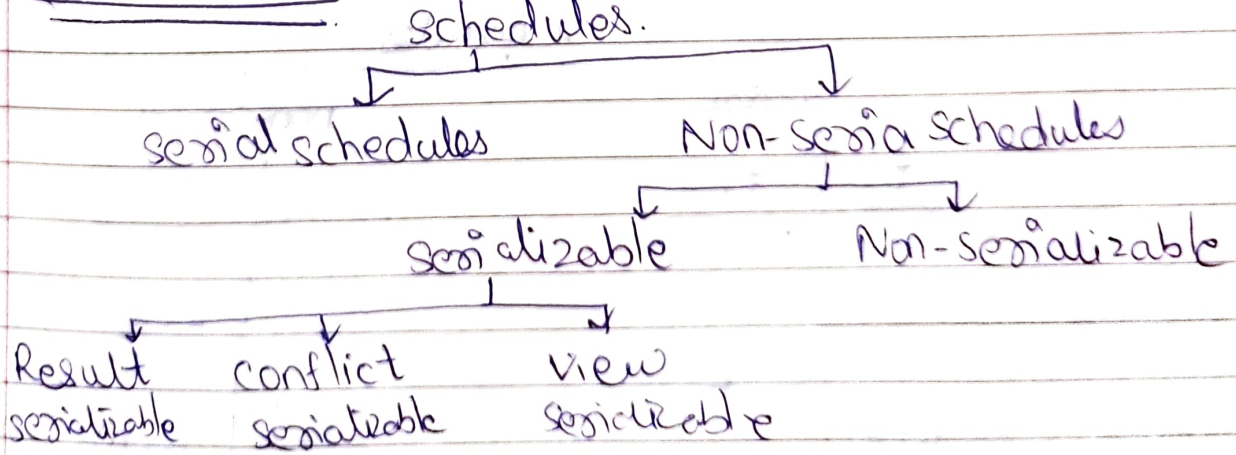
	T_1	T_2
LB [A=10]	10 — R(A)	
LB [A=11]	A++ — W(A)	
LB [A=50]		W(A) — A=50
DB [A=50]		commit.
DB [A=50]	commit.	



I performed blind write without reading I am writing the value.

my updated value is lost i.e 11, Idk how the value of A = 50, according to me it should be 11.

* Schedules



Schedules

→ Chronological order in which operations of concurrent transaction are executed.

T_1	T_2
i1	i1
i2	i2
i3	i3

These are two transaction.

T_1	T_2
	i1
	i2
i1	
i2	
i3	
	i3

This is called schedule.

Here contact switching is done bet^n T_1 & T_2 but the order of instruction should be maintained

i1	i3
i2	i1
i3	i2
✓	✗

we can't swap instructions of transaction.

Schedules

- serial
- non-serial.

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

consistent ✓
concurrency problems ✗
efficient ✗
WT ↓
RT ↓

Here, contact switching is done only after completion of 1 transaction.

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

consistent ✗
concurrency problem ✓
efficient ✓
WT ↑
RT ↑

Here, contact switching is done before completion of transaction, so that we get concurrency.

non-serial

Serializable

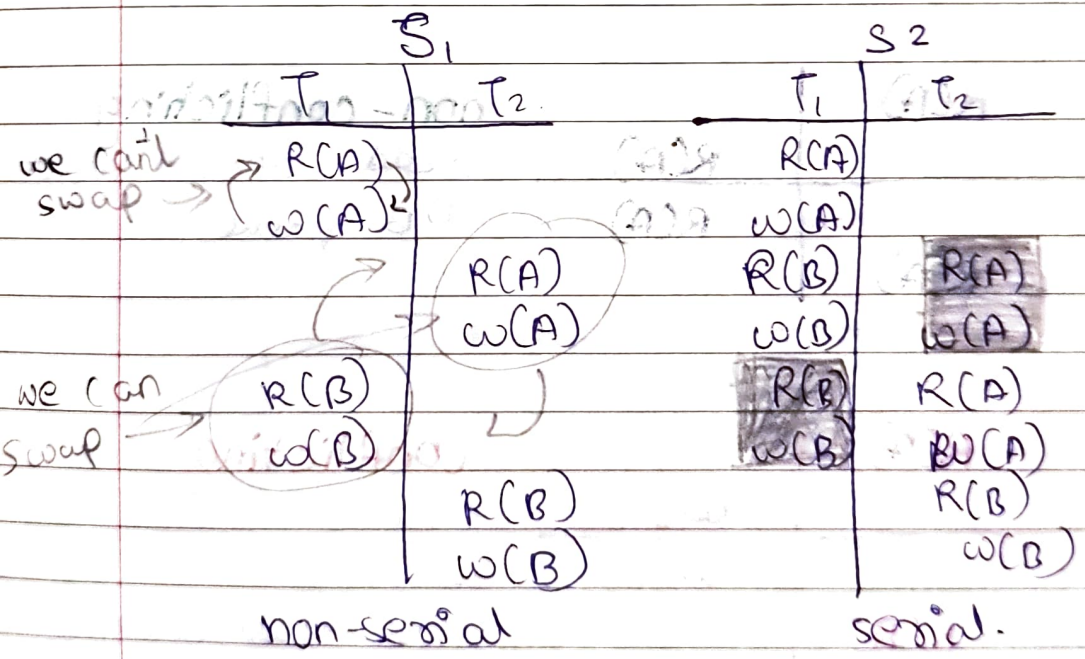
conflict — view

DOMS Page No. Date / /

* **Conflict Serializability** (A schedule is C.S if we can convert it into a serial schedule after swapping its non-conflicting operations)

→ To know that any schedule is consistent or not we have to prove that non-serial = serial (inconsistent) (consistent) always

→ To prove this we use conflict serializability (by swapping the instructions between transactions.



⇒ If swap ✓ Then non-conflicting So it can be in serial

If swap ✗ Then conflicting it cannot be in serial.

- How to find conflicting ?

	T_1	T_2
	i_1	i_2

Case 1

	T_1	T_2
Before swap	R(A)	
		W(B)
after swap		W(B)
	R(A)	

non-conflicting
coz data items are different.
Read on A & write on B

Case 2

	T_1	T_2
Before swap	R(A)	
		R(A)
after swap		R(A)
	R(A)	

non-conflicting
Read doesn't affect DB value.

DB A = 10

Case 3

	T_1	T_2
Before	10 - R(A)	
		W(A) - 20
after		W(A) - 20
	20 - R(A)	

conflicting
DB A = 20
DB A = 20
Hex result changes

Case 4

	T_1	T_2
Before	W(A)	
		W(A)
after		W(A)
	W(A)	

conflicting
Before swap
- DB A = 12
After swap
- DB A = i_1
∴ It affects the final value in DB.

no-cycle → Conflict serializable → serial schedule convert

⇒ from those cases we conclude that a schedule is conflicting

1. They must belong to diff^n transaction
2. must operating on same data value
3. Atleast one of them should be write

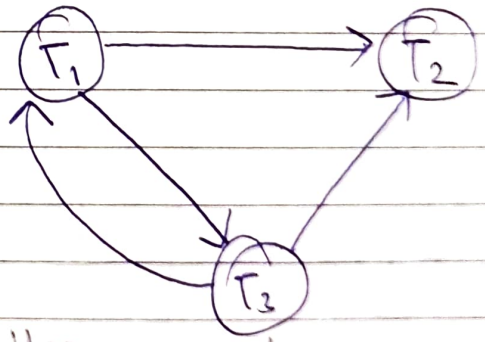
* Questions on Conflict Serializability

→ Finding whether a schedule is conflict serializable or not.

Q1

S		
T_1	T_2	T_3
R(x)		
		R(z)
		w(z) ×
	R(y) ×	
R(y)		
	w(y)	
		w(x)
	w(z)	
w(x)		

Here we have to prove non-serial = serial. by swapping

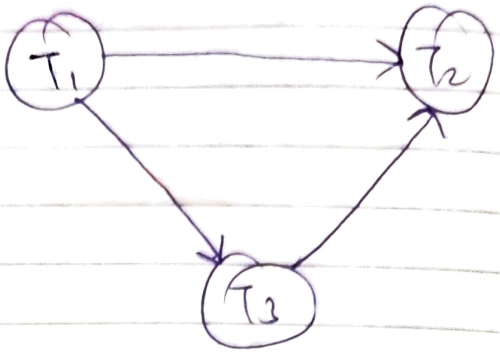


Here cycle occurs.
∴ serial order conversion is not possible.

Hence it is **not conflict serializable**.

Q2

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



→ It is conflict serializable.
∵ non serial → serial is possible.

→ order of serializability.

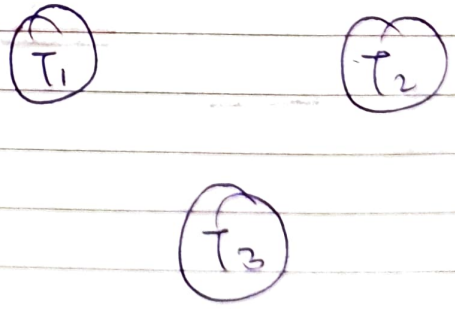
$T_1 \rightarrow T_3 \rightarrow T_2$

all outgoing edge

all incoming edge.

Q3

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



→ It is conflict serializable
→ diff" conflict serializable possible then

3! schedule are possible.

Note :- $T_1 \rightarrow T_2$ (like T_1 depends on T_2)

possibilities : $T_1\ T_2\ T_3$

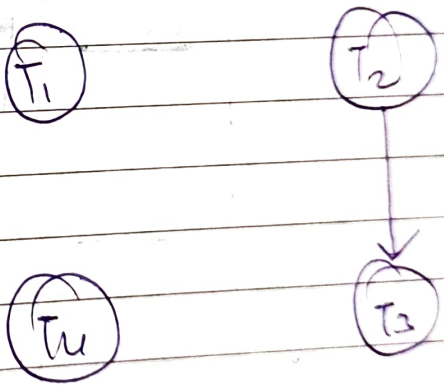
$T_3\ T_1\ T_2$

$T_1\ T_3\ T_2$

3 diff^n schedule are possibles.

Transaction with commit after commit we don't have to check the clash.

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

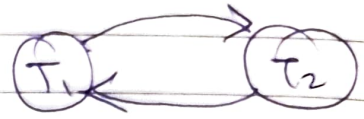


Conflict serializable

Q5 → Transactions:
- T_1 : $R_1(x)$ $W_1(x)$
- T_2 : $W_2(x)$

→ Schedule S
$R_1(x)$ $W_2(x)$ $W_1(x)$

T_1	T_2
$R_1(x)$	
	$W_2(x)$
$W_1(x)$	



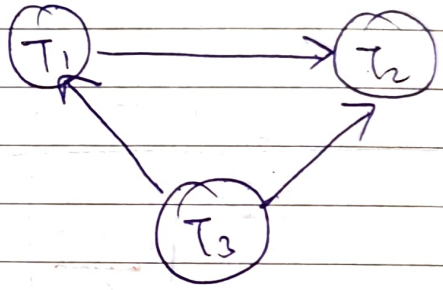
not conflict serializable

Q6

S1

T_1	T_2	T_3
R(x)		
W(x)		
	R(y)	
	R(x)	~~R(x)~~
		R(y)
		R(z)
	W(y)	
		W(z)
R(z)		

we have to compare S1 & S2



$T_3 \rightarrow T_1 \rightarrow T_2$

Equivalent

Conflict

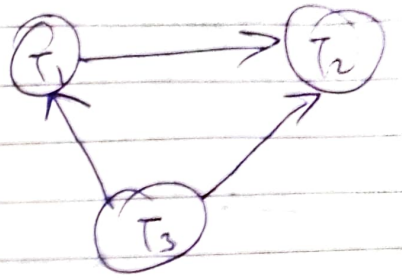
S2.

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

not conflict equivalent

Q7

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

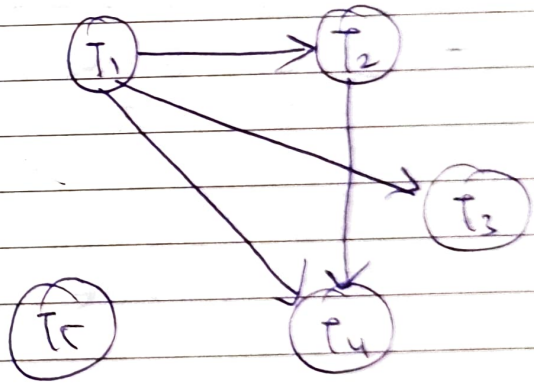


$T_3 \rightarrow T_1 \rightarrow T_2$

Conflict serializable

Q8

T_1	T_2	T_3	T_4	T_5
	R(x)			
R(y)				
R(z)				
				R(v)
				R(w)
	R(y)			
	W(y)			
		W(z)		
R(w)				
			R(y)	
			W(y)	
			R(z)	
			W(z)	
R(w)				
W(w)				

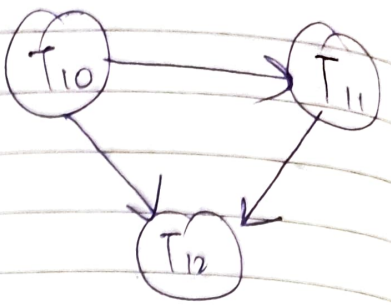


$T_5 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$

Conflict serializable

Q9

T_{10}	T_{11}	T_{12}
R(A)		
R(B)		
W(A)		
	R(A)	
	W(A)	
		R(A)
abort		



$T_{10} \rightarrow T_{11} \rightarrow T_{12}$

Conflict pairs.

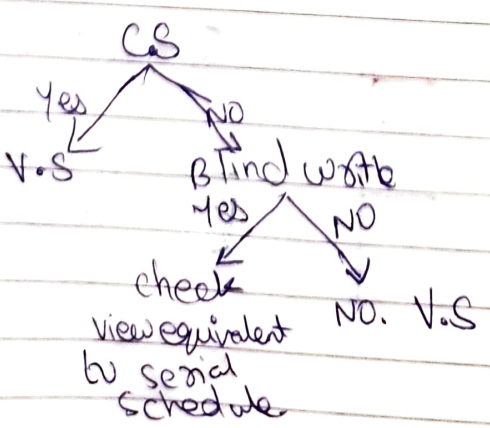
$\langle R_{10}(A), W_{11}(A) \rangle$, $\langle W_{10}(A), R_{12}(A) \rangle$, $\langle W_{11}(A), R_{12}(A) \rangle$

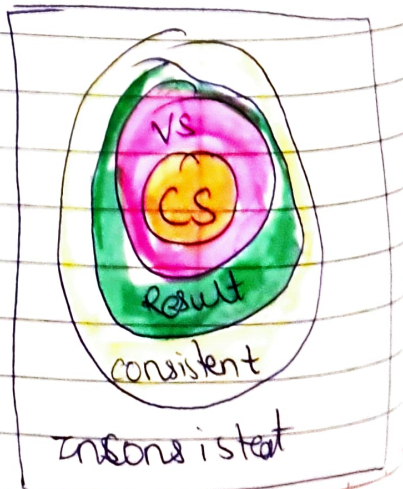
* View Serializability

→ If a schedule is conflict serializable → then it will be view serializable.

→ If the view serializable which does not conflict serializable contains blind write.

→ A schedule will view serializable if it is view equivalent to serial schedule.





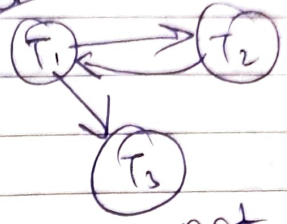
if conflict → V.S ✓

- Conditions to check view equivalent
  1. Initial Read (same)
  2. final write (same)
  3. intermediate Read order (same).

S

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

step 1



not conflict serializable.

step 2 check blind write = Yes.

step 3 all possible serial schedule

1 2 3 ✓
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

S'

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

∴ S ≡ (view equivalent) S' $T_1 \rightarrow T2 \rightarrow T_3$

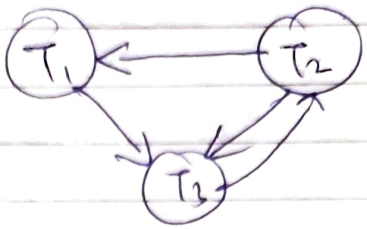
∴ view serializable.

Data object	Initial Read	final write	update Read.	Serial order
a.	T_1	T_3	-	$T_1 \rightarrow T_2 \rightarrow T_3$

Q2

S1

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



non, conflict serializable.

Data object	Initial Read	final write	Updated Read order	Serial order
A	T_1	T_2	$T_3 \rightarrow T_2$	$T_1 \rightarrow T_3 \rightarrow T_2$
B	T_2	T_2	$T_2 \rightarrow T_1$, $T_2 \rightarrow T_3$	$T_2 \rightarrow T_1 \rightarrow T_3$, $T_2 \rightarrow T_3 \rightarrow T_1$

not view serializable schedule

no common serial orders

Q3 Check view equivalence betⁿ S1 & S2.

S1

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

S2

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

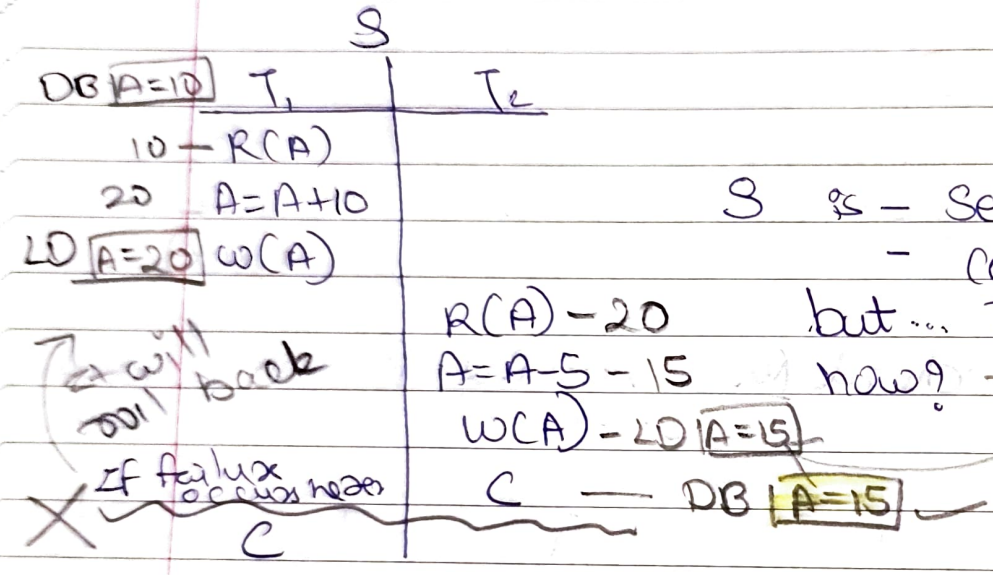
Data object	Schedule No.	Initial Read	Final write	Updated Read
X	S1	T_1	T_3	$T_1 \rightarrow T_2$
	S2	T_1	T_3	$T_1 \rightarrow T_2$
Y	S1	T_2	T_2	–
	S2	T_2	T_2	$T_2 \rightarrow T_3$
Z	S1	T_3	-	-
	S2	T_1		

Not view equivalent.

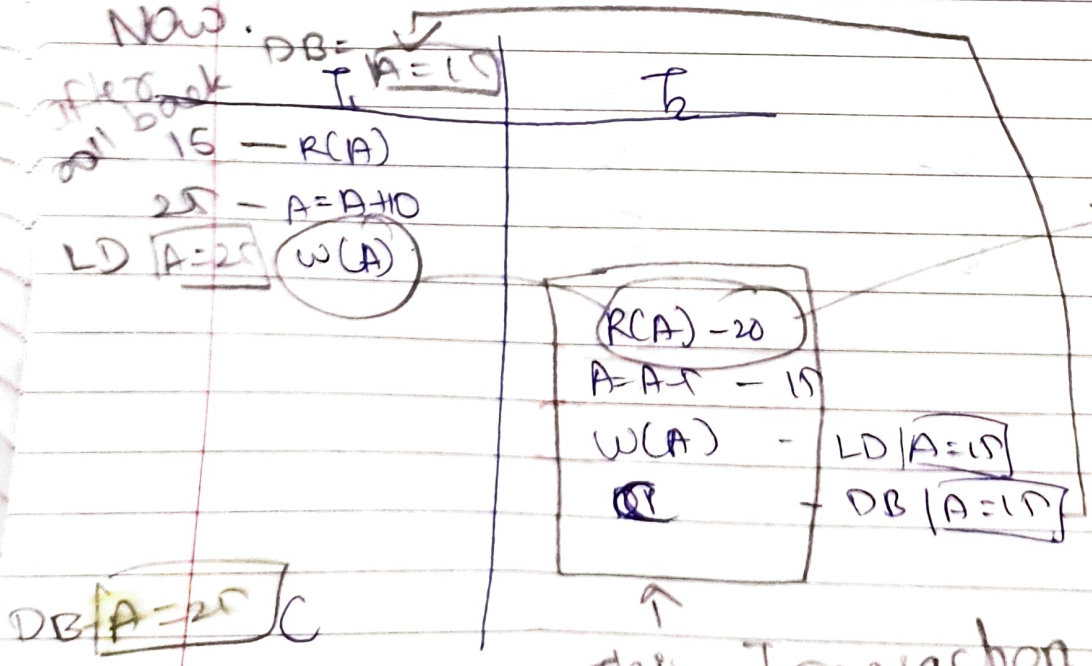
* Recoverable Schedule

Schedule

Recoverable Irrecoverable



S is - Serial schedule
- Conflict serializable
but... Irrecoverable.
how?

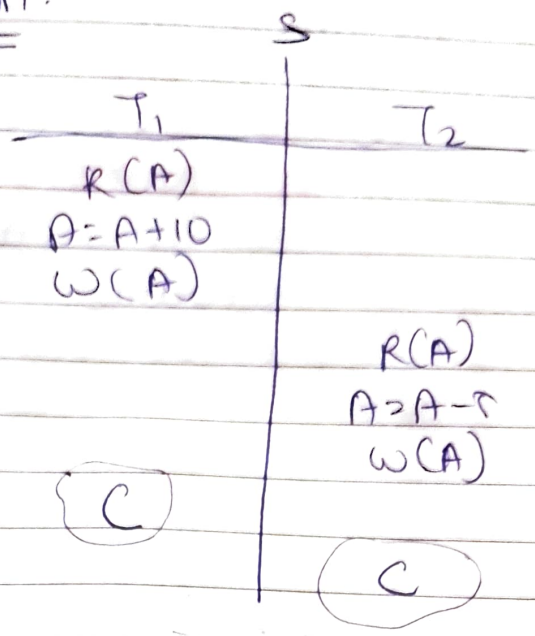


↑ This Transaction is committed & done. so it dont need to roll back

Here the value in DB changes & irrecoverable.

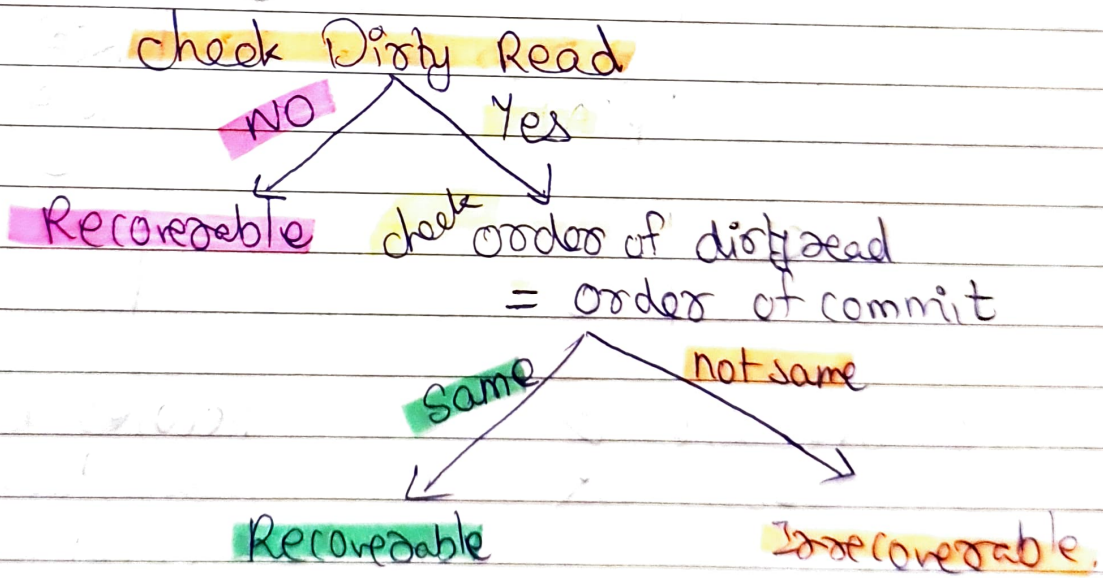
DOMS

solution:



→ It is recoverable

⇒ Now how to find whether a schedule is recoverable or not.



Recoverability is a mandatory property of a schedule.

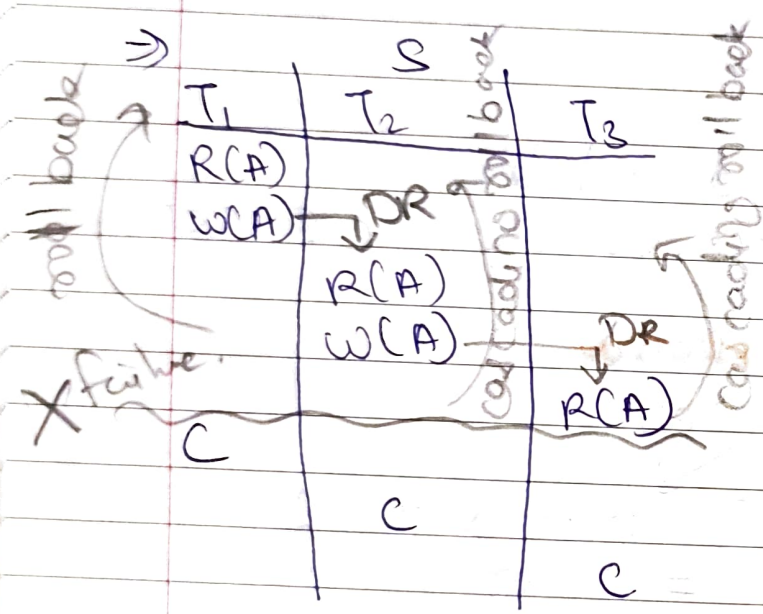
DOMS Page No. Date / /

* Cascadeless schedule

→ Cascadeless schedule ⇒ If there are no cascading roll backs then it is cascadeless schedule.

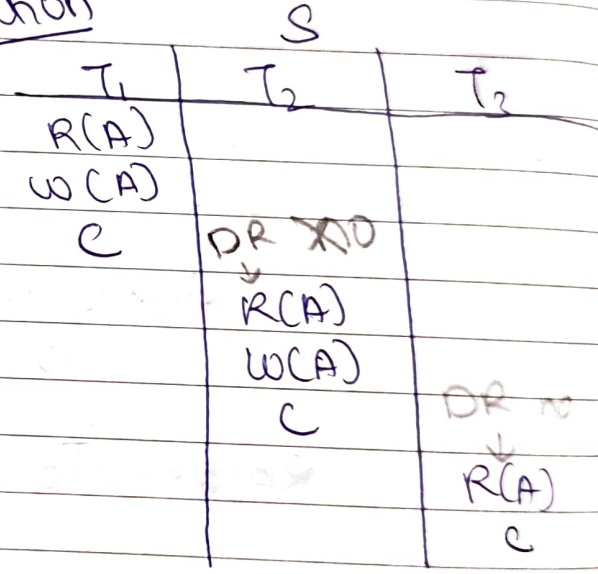
The transaction T_2 which dirty read from T_1, If T_1 roll backs then T_2 have to roll back this is called cascading roll back

⇒



recoverable ✓
Cascadeless ✗

solution

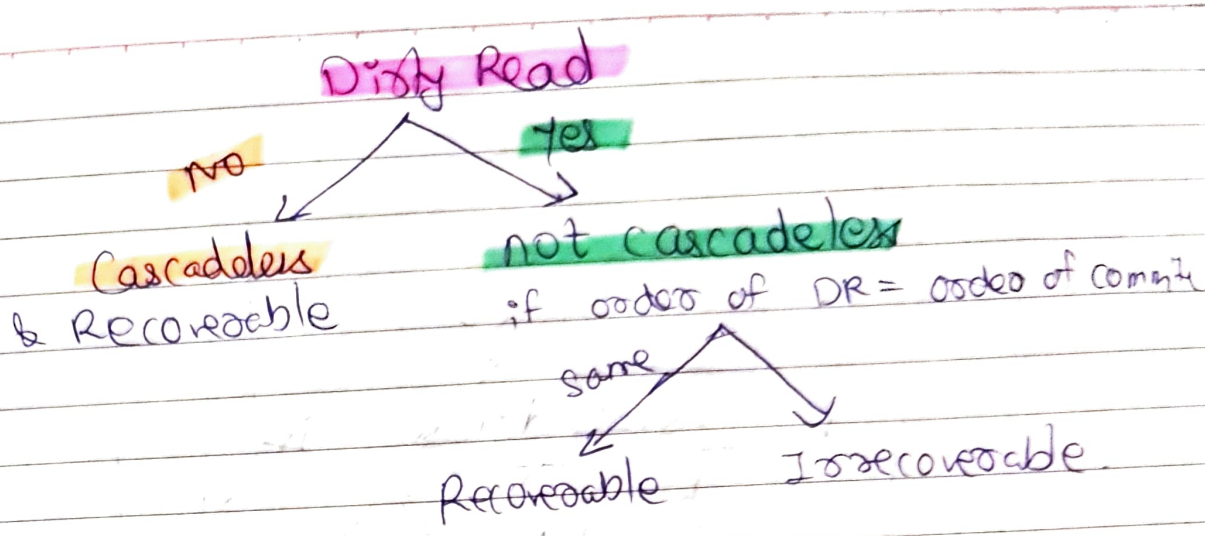


recoverable ✓
Cascadeless ✓
degree of concurrency ↓

→ Cascadeless is optional property

→ If there is dirty read then compulsorily there is cascading roll backs & hence it will not be a cascading schedule

DR = Dirty Read



* Strict Schedule

S1

T_1	T_2
R(a)	
w(a)	
	w(a)
C	
	R(a)
	C

S2

T_1	T_2
R(a)	
w(a)	
C	
	w(a)
	R(a)
	C

S3

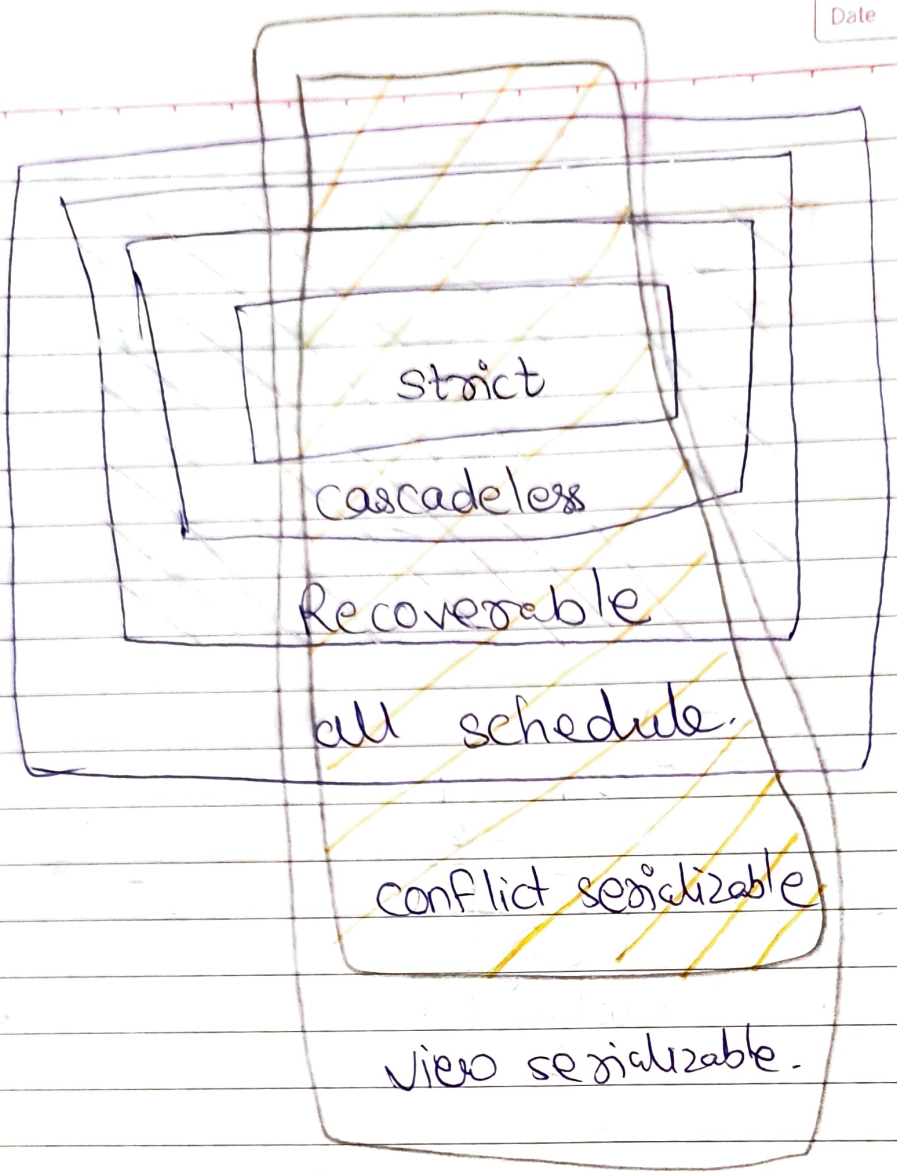
T_1	T_2
R(a)	
	R(b)
w(a)	
	w(b)
C	
	R(a)
	C

Recoverable:	✓	✓	✓
Cascadeless:	✓	✓	✓
Strict:	X	✓	✓

Rule for Strict schedule:

- If a transaction is working on a data item and doesn't commit till that no other transaction can read or write that data item.

after committing we can work on that data item

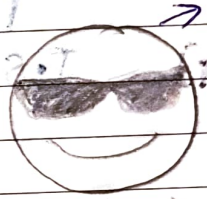


If All these properties satisfies then we can guarantee consistency of a schedule.

* Concurrency Control techniques

→ Till now we were checking whether a schedule maintain consistency or not.
→ Now we will use protocols that guarantee to generate schedule which satisfy the properties (C.S, V.S, Recoverability, Cascadelessness, strict etc), Specially C.S.

Conflict serializability



Kabhi kabhi to lagta hein
ki apun ich bhagwaan hein

* Time Stamping protocol

→ Basic idea = to decide the order between the transaction before that enters into the sys.
So in case of conflict, we can resolve that using ordering.

→ ~~Timestamp = System uses~~

→ For ordering, we use value of system clock. [hh:mm:ss dd/mm/yyyy] (it is always unique & never repeat itself).
Hence we call it Timestamp

DOMS | Page No.
Date / /

⇒ 2 ideas of time stamping

1) Time stamping with transaction:

→ With each transaction T_i, we associate a time stamp denoted by $T.S(T_i)$

It is the value of System clock when transaction enters into system

→ If new transaction T_j enters after T_i then, $T.S(T_i) < T.S(T_j)$

bcoz. if T_i enters at 2:20 pm 3/3/2020
& T_j enters at 4:10 pm 3/3/2020
then $T.S(T_i) < T.S(T_j)$

→ Also determine serializability order if $TS(T_i) < T.S(T_j)$ then, system ensure that in the resultant conflict serializable schedule, T_i will execute first (before T_j)

2) Time stamp with data item:

R(A)

→ For each data item A, Protocol maintains two time-stamps.

(i) Write-timestamp(A): The largest timestamp of any transaction that executed write(A) successfully

(ii) Read-timestamp(A): The largest timestamp of any transaction that executed read(A) successfully

allow serial
deny serial

DOMS

⇒ Timestamping Protocol

Ti request for Read(A)

- If T.S(Ti) < W.T.S(A)
  Ti It means Ti needs to read a value of A that was already overwritten. Hence, request must be rejected & Ti must rollback.

- If T.S(Ti) ≥ W.T.S(A)
  Ti: Operation can be allowed and RTS(A) will be max(RTS(A), T.S(Ti))

Ti request for Write(A)

- If T.S(Ti) < R.T.S(A)
  It means value of A that Ti is producing was needed previously and the system assumed that the value would never be produced hence reject & rollback.

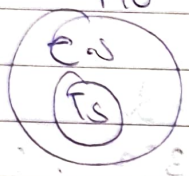
- If T.S(Ti) < W.T.S(A)
  Ti is attempting to write an obsolete value of A.
  reject & rollback

- Otherwise T.S(Ti) ≥ R.T.S(A) or T.S(Ti) ≥ W.T.S(A)
  then allowed
  & W.T.S(A) = max(W.T.S(A), T.S(Ti))

DOMS | Page No.
Date / /

* Properties of Time stamping protocol.

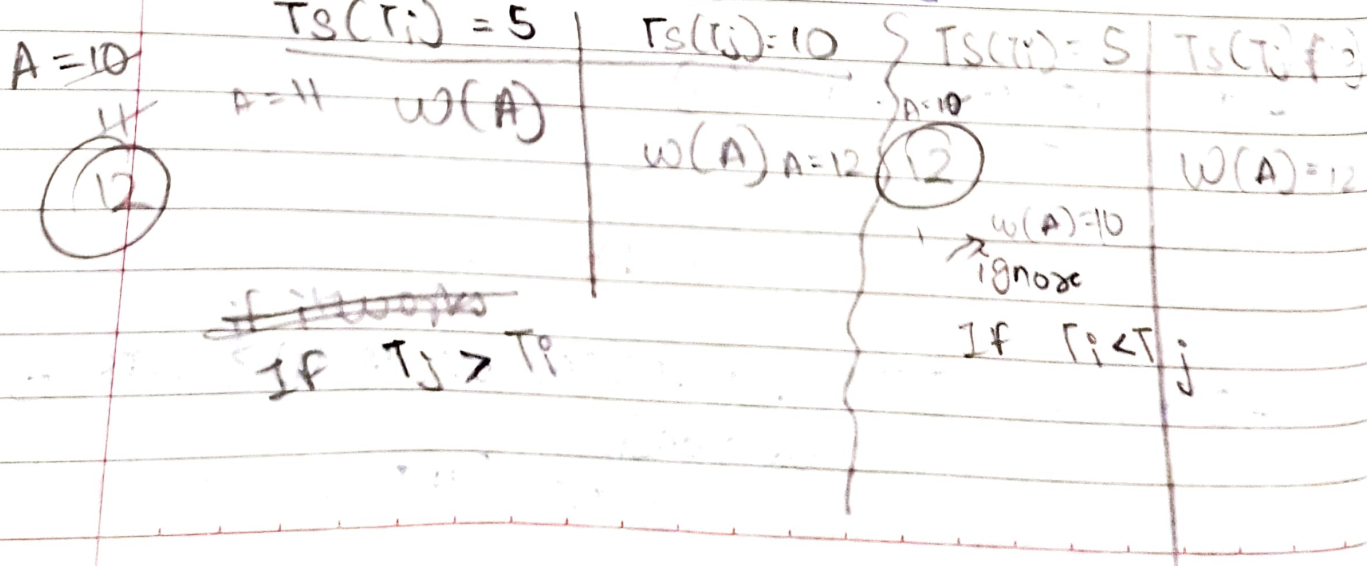
→ Ensures conflict serializability
→ Ensures view serializability
→ Possibility of dirty read no restriction on commit, irrecoverable schedules & cascading rollbacks are possible.

→ Here, we either allow or reject, so no idea of deadlock.
→ 

using Time stamping (TS) we can generate conflict serializable schedule but, not all C.S can be generated with TS

→ may suffer from starvation,
→ relatively slow.

* THOMAS WRITE RULE



→ Modify time stamping protocol to make some improvements and may generate those schedules that are V.S but not C.S and provide better concurrency.

→ modify time-stamping protocol in absolute write case when T_i request write(A) if $T.S(T_i) < WTS(A)$

→ Here T_i attempts to write absolute value of A. Rather than rolling back T_i, write operation is ignored.

eg.

T_1	T_2	T_3
R(A)		
	W(A)	
ignored → W(A)		
		W(A)

not conflict serializable but V.S.

DOMS | Page No.
Date / /

* Lock based Protocol

→ To achieve consistency, isolation is most important.

→ Locking is a simple concept to achieve isolation
i.e. first obtain a lock on data item
then perform desired operation
& then unlock.

• modes of lock.

1) Shared lock : denoted by lock-S(A)
Transaction can perform Read
Any other transaction can also obtain same lock on same data item at same time.

T_1	T_2
lock-S(A)	
	lock-S(A)

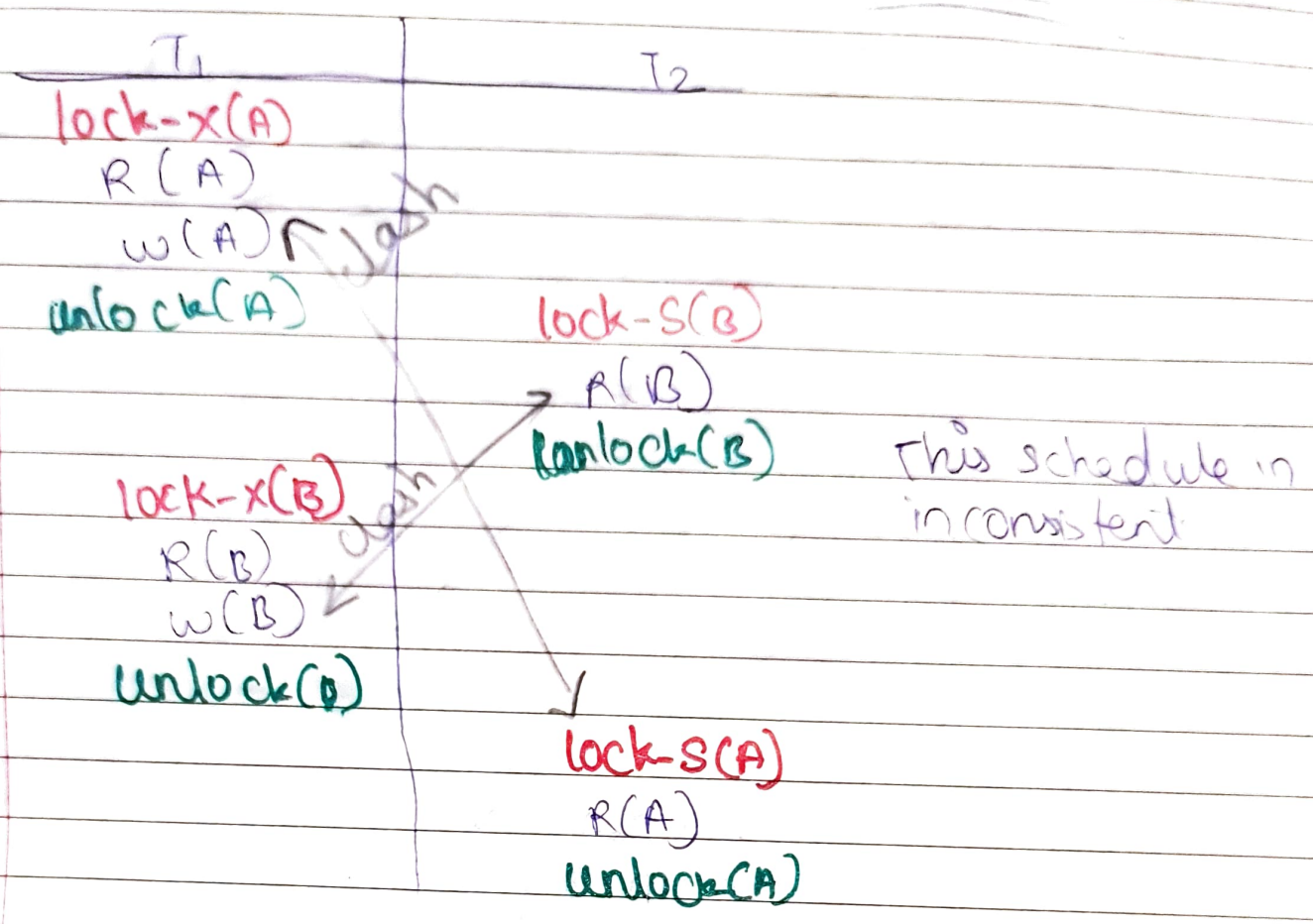
2) Exclusive lock: denoted by lock-X(A)
Transaction can perform Read / write
Any other transaction cannot obtain either shared / Exclusive locks.

T_1	T_2
lock-X(A)	
	lock-S(A) ✗ not allowed

⇒ Compatibility table

	shared	Exclusive
shared	T	F
exclusive	F	F

* Properties of Lock based protocol



→ If we do unlocking, inconsistency will occur if we don't unlock then, concurrency will poor.

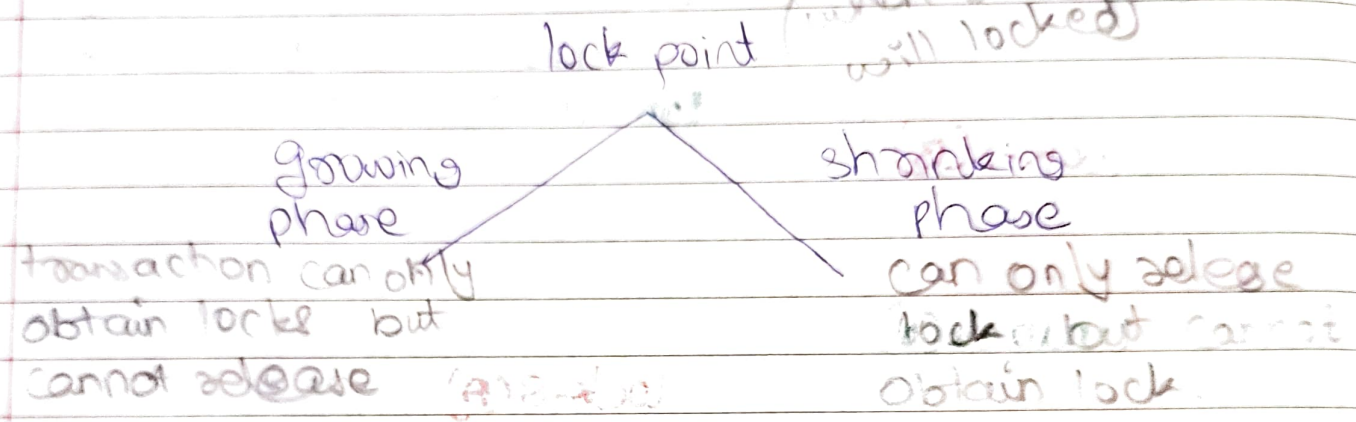
→ We require that, transaction follows some set of rules for locking & unlocking of data eg: 2PL or graph based protocol

→ We say a schedule is legal under a protocol, if it can be generated using the rules of protocol.

DOMS | Page No.
Date / /

* Two-phase locking (2PL) / Basic 2PL

→ This protocol requires that each transaction in a schedule will have 2 phase : growing phase & shrinking phase.



transaction can Perform read/write in both growing/shrinking phase.

→ Properties

→ ensures C.S / V.S, the order of serializability → order in which transaction reaches lock point.

→ may generate irrecoverable schedule & Cascading roll back.

Q

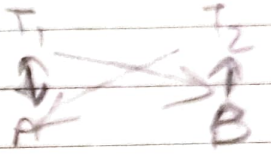
T_1	T_2
lock X(A)	
R(A)	
W(A)	
unlock(A)	lock S(A)
	R(A)

clash because dirty read of T_1 not yet committed

3. Do not ensure freedom from deadlock.

T_1	T_2
lock S(A)	
Read(A)	
	lock X(B)
	R(B)
	W(B)
lock S(B)	
not get	
	lock X(A)
	not yet unlocked

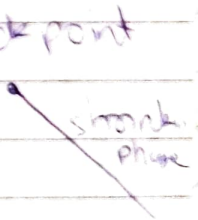
deadlock situation



→ 2PL can be C.S but not all C.S can generate by 2PL

* Conservative / static 2PL:

→ There is no growing phase, 1st acquire all locks required & directly start from lock point.



→ If all locks are not available, then transaction must release the lock acquired so far & wait.

→ shrinking phase works as usual.

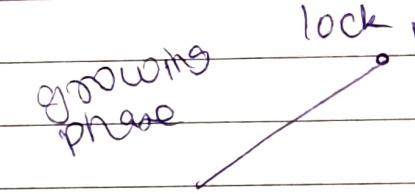
* Properties:

→ Ensures CS/VS
→ Independent from deadlock
→ Irrecoverable schedule & Cascading rollback

* Rigorous 2PL

→ It is improvement over 2PL protocol. where we try to ensure recoverability & cascadelessness.

→ Here, all locks must be held until commit i.e there is no shrinking phase.



T_1	T_2
lock X(A)	
R(A)	
W(A)	
commit ~~unlock(A)~~	
	lock S(A)
	R(A)

then there will be no dirty read.

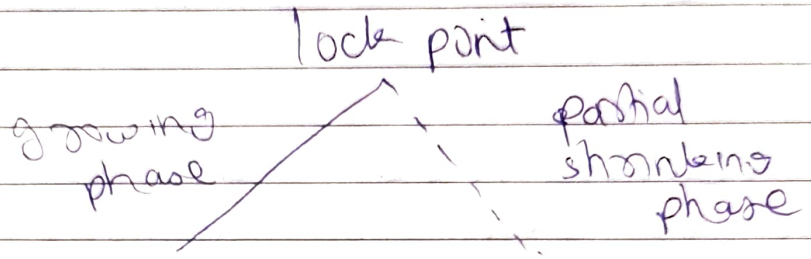
* properties.

→ Ensure CS/VS
→ " recoverability & cascadelessness
→ Suffer ~~from~~ from deadlock & inefficiency

* Strict 2PL

→ It is an improvement over Rigorous 2PL

→ In shrinking phase
exclusive lock cannot unlock but
shared lock can be unlock.



* Properties.

→ Ensures CS/VS, recoverability, cascadeless, efficiency

→ suffer from deadlock. but its ok

* Log based Recovery

1) Deferred Database modification.

DB

A = ~~100~~ 200
B = 200 ~~400~~

T_1
R(A)
A = A + 100
w(A)
R(B)
B = B + 200
w(B)
commit

log (it is like record file)

<T_1, start> → new value
<T_1, A, 200>
<T_1, B, 400>
<T_1, commit>
redo
(save new values)

i.e.

DB
A = 200
B = 400

→ It is also called NO undo/Redo

→ <T1, start>
<T1, A, 200>
<T1, B, 400>
<T1, commit>
<T2, start>
<T2, C, 500>

If failure occurs then,
T1 will redo i.e.
DB | A=200, B=400

T2, will roll back
DB | C0

If commit found then new value updated
If not then roll back.

After commit only we save changes.

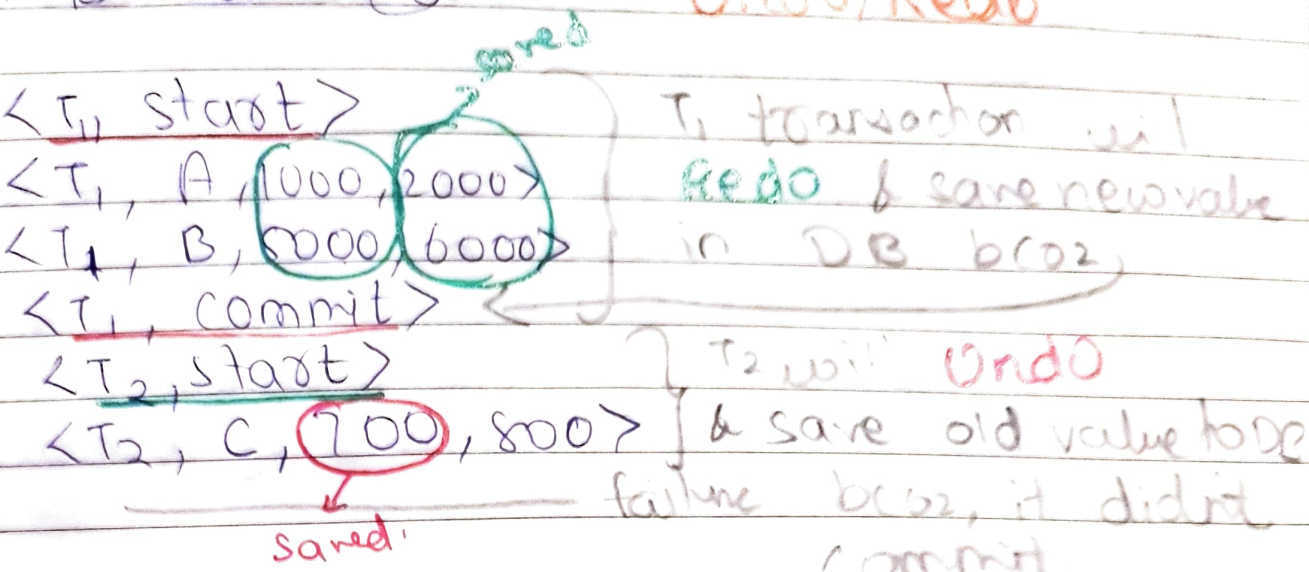
* Immediate Database modification

→ Here, at the time of write operation value is updated in DB & don't wait for commit.

<T1, start>
<T1, A, 100 (old value) 200 (new value)>
<T1, B, 200 (old value) 400 (new value)>
<T1, commit>

T1	DB
	A=100, B=200
R(A)	
A = A+100	
W(A)	A=200, B=200
R(B)	
B = B+200	
W(B)	A=200, B=400
Commit	

→ It is also called Undo/Redo



* Deadlock

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

→ If there exist a set of waiting transaction T_0, T_1 to T_{n+1} such that $T_0 \to T_1, T_1 \to T_2, \ldots T_{n-1} \to T_n$ so no transaction can processes in such situation

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

→ System must have proper methods to deal with deadlock otherwise
- → it may lead to life & money in real time
- → will reduce resource utilization & increase inefficiency

DOMS | Page No.
Date / /

⇒ There are two principle for dealing with deadlock problem.

→ Prevention: ensures system never enters into deadlock.

→ Detection & Recovery: allow system to enter deadlock then try to recover.

Ch:4 Formal Relational Query Language

4.1 The relational algebra
4.2 Tuple Relational Calculus
4.3 Domain Relational Calculus.

4.1 Relational algebra

→ It is a procedural query language.
→ Consist of set of operation that take one or two relations as input & produce new relation as o/p.

→ Fundamental operations are:
1) select
2) project
3) union
4) set difference
5) Cartesian product
6) Rename
7) set intersection
8) natural join
9) assignment.

1) Select

→ Select tuples that satisfy given predicate
→ denoted by (σ)

Q1 : select instructor whose deptname = 'physics'

$\sigma_{deptname = 'physics'}(instructor)$

Q2: select instructor where salary > 90000;

$\sigma_{salary>90000}(instructor)$

Q3: select instructor where dept_name = 'physics' and salary > 90000;

$\sigma_{dept\_name="physics" \wedge salary>90000}(instructor)$

2) Project

→ Allows us to produce relation, list the attributes
→ denoted by (Π)

Q1: $\Pi_{ID, name, salary}(instructor)$

ID	name	salary

Q2: Select sname from student where city='jaipur'

$\Pi_{sname}(\sigma_{city='jaipur'}(Student))$

3) Union

→ Include all tuples in table A or in B. also eliminates duplicate tuples.
→ denoted by (∪)

- Conditions:
  1) A & B have same no. of attributes

2) Attribute domain must be compatible
3) Duplicate tuples are automatically eliminated.

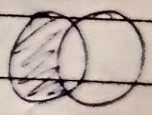
Q1: $\Pi_{author}(Books) \cup \Pi_{author}(Articles)$
displays name of author who wrote books or articles or both.

4) Set difference

→ Result is tuples which are present in one relation but not in second.
→ Denoted by (−)

Q1: list name of author who wrote books & not articles.

$\Pi_{author}(Books) - \Pi_{author}(Articles)$



5) Cartesian Product

→ Combines ~~relation~~ information of 2 different relation into 1.
→ Denoted by (X)

Q1: display all the books & articles written by XYZ.

$\Pi_{author='XYZ'}(BOOKS \times ARTICLES)$

6) Rename

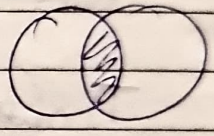
→ Allows to rename o/p of relation.
→ denoted by (ρ)

Q1: result of expression E should be saved as x.

ρx(E)

* Additional Relational Algebra operations

7) Set-intersection



→ To find tuple that ax in both table.
→ Denoted by (∩)

Q1: list authors that write both books & articles.

Πauthor (Books) ∩ Πauthor (articles)

8) Natural join

* Join
→ Combination of Cartesian product followed by selection process.
*Pairs 2 tuples from different relations, iff given join condition is satisfied.
→ Denoted by (⋈θ) → join condition

DOMS Page No. Date / /

Eg:

student

SID	Name	std
101	A	10
102	B	11

subjects

Class	Subject
10	math
10	Eng
11	music
11	sports

Q1 join student & subject where std = sub

student ⋈ (student.std = subject.class) Subject

↓

θ

* Natural join

→ does not require any comparison operator.
→ we can perform natural join only if 1 common attribute exist bet^n tables.
→ denoted by (⋈)

Eg:

Courses

CID	Course	Dept

HOD

Dept	Head

Q1 Join courses & HOD

Courses ⋈ HOD

* Outer join

→ Include all tuples from participating relations in resulting relation.

Q: Left (courses)

A	B
100	DBMS
101	OS
102	CO

Right (HOD)

A	B
100	A
102	B
104	C

1) Left outer join (R ⟕ S)

→ All tuples of left are included.

Courses ⟕ HOD

A	B	C	D
100	DBMS	100	A
101	OS	–	–
102	CO	102	B

2) Right outer join (R ⟖ S)

→ All tuples of right are included.

Courses ⟖ HOD

A	B	C	D
100	DBMS	100	A
102	CO	102	B
–	–	104	C

3) Full outer join (R ⟗ S)

→ All tuples from both are includ.

Courses ⟗ HOD

A	B	C	D
100	DBMS	100	A
101	OS	–	–
102	CO	102	B
–	–	104	C

Assignment operation

- works like assignment in programming languages denoted by (←)

temp 1 ← R x S

* Extended Relational-Algebra Operations

1) Generalized projection

- Extends the projection operation by allowing operations such as arithmetic & string functions to be used in projection list
- Can use arithmetic operations (+, -, *, ÷) also permits operation on other data types (concatenation of strings).

eg: $\Pi_{ID, name, dept\_name, salary \div 12}(instructor)$

2) Aggregation

- Permits use of aggregate functions such as min, average, sum, count, max, distinct
- denoted by $\mathcal{G}$

eg: $\mathcal{G}_{sum(salary)}(instructor)$

$dept\_name\ \mathcal{G}_{average(salary)}(instructor)$

* Relational Calculus

→ It is non procedural query language.
→ User is concerned with details of how to obtain end results.
→ tells what to do but not how to do.

Relational Calculus

Tuple Relational Calculus

Domain Relational Calculus.

4.2 Tuple Relational Calculus (TRC)

→ Specified to select tuples in relation.
⇒ The result of relation can have one or more tuples.

$\{T \mid P(T)\}$ or $\{T \mid condition(T)\}$

resulting tuples — condition to fetch T

eg: $\{T.name \mid Author(T) \in Author \text{ AND } T[subject] = 'data'\}$

Select tuples from Author relation returns tuple with name from author who wrote article on data

TRC can be quantified we can use Existential (∃) & Universal (∀) quantifiers.

eg: {R | ∃ T ∈ Author (T.article = 'dotd' And R.name = T.name)}

eg: Find ID, name, dept_name, salary for those instructor whose salary > 80000

{T | T ∈ instructor ⇒ T.salary > 80000}

Domain Relational Calculus (DRC)

- Provides only the description of the query but does not provide method to solve it.
- In DRC, query is expressed as

$$\{\langle x_1, x_2, x_3 \ldots x_n \rangle \mid P(x_1, x_2, x_3 \ldots x_n)\}$$

$\langle x_1, x_2, x_3 \ldots x_n \rangle$ ↑ resulting domain variable

$P(x_1, x_2, x_3 \ldots x_n)$ ↑ condition to predicate calculus

- Predicate calculus formulas
  1) set of all comparison operators
  2) set of connectives ∧, ∨, ¬
  3) set of quantifiers

eg: find loan no., branch, amt from LOAN of >= 100

$$\{\langle l, b, a \rangle \mid \langle l, b, a \rangle \in LOAN \wedge (a \geq 100)\}$$

DOMS Page No.
Date / /

eg: Find loan no. for each loan of amt ≥ 150.

$\{\langle l \rangle \mid \exists b, a \ (\langle l, b, a \rangle \in LOAN \wedge (a \geq 150))\}$

Ch:5 Relational Database Design

5.1 Functional dependency & types.
5.2 Closure of FD set
5.3 Closure of attribute set
5.4 Irreducible set of FD

5.5 normalization & normal forms
- 1NF, 2NF, 3NF, BCNF, 4NF, 5NF

5.1 Functional Dependency (FD)

→ It is a relationship between attributes of a table dependent on each other.

→ It helps in preventing data redundancy and gets to know about bad design.

→ FD is represented by (→)

B is functionally dependent on A

A → B

determinant → dependent

R

A	B

→ $f: \alpha \rightarrow \beta$

R

α	β
a	1
a	2
c	3
d	4

✗

K

α	β
a	1
b	1
c	3
d	4

✓

∴ If $\alpha \subseteq R$ & $\beta \subseteq R$

— If $t_1[\alpha] = t_2[\alpha]$
then $t_1[\beta] = t_2[\beta]$

R	α	β
t_1 →	a	b
t_2 →	a	b

t_1 = tuple 1
t_2 = tuple 2.

⇒ If
→ two or more determinants are same and dependents are different then not valid.

→ determinants are different & dependents are same then valid.

→ two or more determinats are same & their dependents are also same then valid.

* Types of Functional Dependency

Trivial	Non-trivial	completely non-trivial
→ occurs when B is subset of A in A → B	→ occurs when B is not subset of A in A → B	occurs when A intersection B is null in A → B
→ eg: {ID, name} → ID	eg: ID → name	
B ⊂ α	B ⊄ α	

DOMS

* FD examples.

• find valid dependencies.

1)

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

~~(a)~~ $A \to BC$

~~(b)~~ $DE \to C$

~~(c)~~ $C \to DE$

~~(d)~~ $BC \to A$

step1: In $\alpha \to \beta$
Check if α is distinct
If yes then valid.
else

step2: Check if β are same
If yes then valid
else

step3: You have to check every dependency

For step1.

R

A	B	C	D	E
a	2	3	4	5
b	a	3	4	5
c	2	3	6	5
d	2	3	6	6

↑ all are distinct

$\therefore A \to B$
$A \to D$
$A \to BCDE$
all are valid.

To prove step2

R

A	B	C	D	E
a	2	3	4	5
b	a	3	6	5
c	2	3	6	5
d	2	3	0	6

↑ all are same

$\therefore A \to C$
$BD \to C$
$ABDE \to C$
all are valid

2)

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

(a) ~~XY→Z && Z→Y~~
(b) YZ→X && Y→Z
(c) ~~YZ→X && X→Z~~
(d) ~~XZ→Y && Y→Z~~

3)

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

(a) ~~A→B && BC→A~~
(b) ~~C→B && CA→B~~
(c) B→C && AB→C
(d) ~~A→C && BC→A~~

* Armstrong's axioms (inference rules)

1) **Reflexivity**: If $Y \subseteq X$ then, $X \rightarrow Y$

2) **Augmentation**: If $X \rightarrow Y$ then ~~$X \rightarrow YZ$~~ $XZ \rightarrow YZ$

3) **Transitivity**: If $X \rightarrow Y$ && $Y \rightarrow Z$ then $X \rightarrow Z$

Primary rules (RAT)

4) **Union**: If $X \rightarrow Y$ && $X \rightarrow Z$ then $X \rightarrow YZ$

5) **Decomposition**: If $X \rightarrow YZ$ then $X \rightarrow Y$ && $X \rightarrow Z$

wrong. ~~$WX \rightarrow Y$~~ ~~$W \rightarrow Y$~~ $X \rightarrow Y$

6) **Pseudo-transitivity**: If $X \rightarrow Y$ && $WY \rightarrow Z$ then $WX \Rightarrow Z$

Secondary rules

7) Composition: If $X \to Y$ && $Z \to W$ then $XZ \to YW$

8) Self determination: $X \to X$

* Closure of set of FDs

→ Given a set F, set of FDs, there are certain other FDs that are logically implied by F.

Eg: $F = \{A \to B$ & $B \to C\}$
then we can infer that
$A \to C$

→ The set of FDs that is logically implied by F is called closure of F

→ Denoted by F^+

Q Check whether the closure set of FD is valid or not.

R (A, B, C, G, H, I)

F (A → B, A → C, CG → H, CG → I, B → H)

DOMS | Page No
Date / /

① $F^+ \{A \to H\}$ ✓

$A \to B$
$B \to H$
$\therefore A \to H$ (transitivity)

② $F^+ \{CG \to HI\}$ ✓

$CG \to H$
$CG \to I$
$CG \to HI$ (union)

③ $F^+ \{AG \to I\}$ ✓

$A \to C$
$CG \to I$
$AG \to I$ (pseudo transitivity)

OR. $A \to C$
$AG \to CG$ (Augmentation)

$AG \to CG$
$CG \to I$
$AG \to I$ (transitivity)

$\therefore F^+ = (A \to H, CG \to HI, AG \to I)$

Q Compute closure of following set F of FD

① $R = (A, B, C, D, E, F)$
$F = (A \to B, A \to C, CD \to E, CD \to F, B \to E)$

$A \to BC$	$A \to B$ && $A \to C$	Union rule
$A \to E$	$A \to B$ && $B \to E$	transitivity
$CD \to EF$	$CD \to E$ && $CD \to F$	Union
$AD \to E$	$A \to C$ && $CD \to E$	pseudo transitivity
$AD \to F$	$A \to C$ && $CD \to F$	pseudo transitivity

$\therefore F^+ = [A \to BC, CD \to EF, A \to E, AD \to E, AD \to F]$

(2) $R(A, B, C, D, E)$
$F = \{AB \to C,\ D \to AC,\ D \to E\}$

$D \to A, D \to C$ $D \to AC$ Decomposition
$D \to ACE$ $D \to AC$ & $D \to E$ Union

$\therefore F^+ = \{D \to A, D \to C, D \to ACE\}$

* Closure of Attribute set

→ It is defined as set of attributes which can be functionally determined from it
→ Denoted by F^+.

⇒ Algorithm to compute α^+, the closure of α under F.

Step 1: result = α
Step 2: while (changes to result)
do
for each $\beta \to \gamma$ in F
do
begin
if $\beta \subseteq$ result then
result = result $\cup$ γ
else
result = result
end.

DOMS Page No. Date / /

1) R(ABC)
F = {A → B, B → C}
A^+ = A
= AB
= ABC

2) R(ABCDEF)
F = {A → B, C → DE, AC → F, D → AF, E → CF}
D^+ = D
ADF
ABDF
DE^+ = DE
ADEF
ABDEF
ABCDEF

3) R(ABCDEFG)
F = {A → B, BC → DE, AEG → G}
$(AC)^+$ = AC
ABC
ABCDE

4) R(ABCDE)
F = {A → BC, CD → E, B → D, E → A}
B^+ = B
BD

5) R(ABCDEFGH)
F = {A → BC, CD → E, E → C, D → AEH, ABH → BD, DH → BC}
check, BCD → H ?
$(BCD)^+$ = BCD
BCDE
ABCDEH

6) F = {A → B, A → C, CG → H, CG → I, B → H}
$(AG)^+$ = AG
ABG
ABCG
ABCGH
ABCGHI

DOMS

1) $R = (A, B, C, D, E)$
$P = \{ A \to BC,\ CD \to E,\ B \to D,\ E \to A \}$

1) find closure of A i.e. A^+
2) $(CD)^+$
3) B^+
4) BC^+
5) E^+

$A^+ = A$
ABC
$ABCD$
$ABCDE$

$(CD)^+ = CD$
CDE
$ACDE$
$ABCDE$

$(B)^+ = B$
BD

$(BC)^+ = BC$
BCD
$BCDE$
$ABCDE$

$E^+ = E$
AE
$ABCE$
$ABCDE$

* Equivalence of functional dependency

$R(A C D E H)$

1) $F = \{ A \to C,\ AC \to D,\ E \to AD,\ E \to H \}$

$G = \{ A \to CD,\ E \to AH \}$

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

$A^+ = ACD$
$AC^+ = ACD$
$E^+ = EAHCD$

$A^+ = ACD$
$E^+ = EADHC$

$F \subseteq G$

$G \subseteq F$

$\therefore F = G$

DOMS

2) R(P Q R S)

X: P → Q
Q → R
R → S

Y: P → QR
R → S

$P^+ = PQRS$
$Q^+ = QR$ ✗
$R^+ = RS$

$\therefore X \not\subseteq Y$

$P^+ = PQRS$
$R^+ = RS$

$Y \subseteq X$

Options
(a) $X \subseteq Y$
~~(b)~~ $Y \subseteq X$
(c) $X = Y$
(d) $X \neq Y$

3) R(A B C)

F: A → B
B → C
C → A

G: A → BC
B → A
C → A

$A^+ = ABC$
$B^+ = ABC$
$C^+ = ABC$

$F \subseteq G$

$A^+ = ABC$
$B^+ = BCA$
$C^+ = CAB$

$G \subseteq F$

$\therefore F = G$

4) R(V W X Y Z)

F: W → X
WX → Y
Z → WY
Z → V

$W^+ = WXY$
$(WX)^+ = WXY$
$Z^+ = ZWXY$ ✗

$F \not\subseteq G$

G: W → XY
Z → WX

$W^+ = WXY$
$Z^+ = ZWXY$

$G \subseteq F$

DOMS

* Irreducible sets of FD / canonical cover

→ It is simplified and reduced version of the given set of FD.

- **Extraneous attributes**

→ If we can remove an attribute from a FD without changing the closure of set of functional dependencies then that attribute is extraneous Attribute.

* Steps to find canonical Cover.

step1: Write FD in such a way that each FD contains exactly 1 attribute at right side.

Step2: consider each FD one by one from set obtain in step1
& To determine whether it is essential or not
- compute closure of its left side
  1st time: consider that particular FD is present in set
  2nd time: consider that particular FD is not present in set.

IF results (of both 1st & 2nd) are same Then eliminate that FD immediatly & don't consider further.

If results are different then It is essential.

Step 3: Consider the newly obtained set of FD after performing step 2

- Check if any FD contains more than 1 attribute on its left side.

IF NO
then obtained set in step 2 is canonical cover.

IF YES
then consider all FD one by one
- Check if their left side can be reduced.
  ↳ To check
  - Consider FD
  - compute closure of all possible subset on left side
  - If any subset produce same closure result as produced by entire left side, then replace left side with that subset

eg

① R(WXYZ)

$X \to W$

$WZ \to XY$

$Y \to WXZ$

Step1:

- ✓ $X \to W$
- ~~$WZ \to X$~~
- ✓ $WZ \to Y$
- ~~$Y \to W$~~
- ✓ $Y \to X$
- ✓ $Y \to Z$

Step2:

$(X)^+ = XW$ → considering $X \to W$

$X_W^+ = X$ → Ignoring $X \to W$

$WZ^+ = WZXY$ } producing same result

$WZ_X^+ = WZYX$

$WZ^+ = WZYX$

$WZ_Y^+ = WZ$

$Y^+ = YWXZ$

$Y_W^+ = YXZW$

$Y^+ = YXZW$

$Y_X^+ = YZ$

$Y^+ = YXZW$

$Y_Z^+ = YXW$

Step3:

$X \to W$

$WZ \to Y$ → $WZ^+ = WZYX$, $W^+ = W$, $Z^+ = Z$

$Y \to X$

$Y \to Z$

~~Steps~~: Canonical cover is

$X \to W$, $WZ \to Y$, $Y \to X$, $Y \to Z$

② R(ABCD)

$A \to B$

$C \to B$

$D \to ABC$

$AC \to D$

Step1:

- ✓ $A \to B$
- ✓ $C \to B$
- ✓ $D \to A$
- ~~$D \to B$~~
- ✓ $D \to C$
- ✓ $AC \to D$

Step2:

$A^+ = AB$

$A_B^+ = A$

$C^+ = CB$

$C_B^+ = C$

$D^+ = DABC$

$D_A^+ = DBC$

$D^+ = DBAC$

$D_B^+ = DACB$

$D^+ = DCAB$

$D_C^+ = DAB$

$AC^+ = ACDB$

$AC_D^+ = ACB$

Step3:

$A \to B$

$C \to B$

$D \to A$

$D \to C$

$AC \to D$ → $AC^+ = ACDB$, $A^+ = AB$, $C^+ = CB$

∴ Canonical cover is

$A \to B$

$C \to B$

$D \to A$, $D \to C$ } $D \to AC$

$AC \to D$

(3) R(VWXYZ)

V → W	step1:	step2:	step3:
VW → X			
Y → VXZ	V → W	$V^+ = VWX$	V → W
		$V_W^+ = V$	VW → X → $VW^+ = VWX$, $V^+ = VWX$
	VW → X	$VW^+ = VWX$	Y → V
		$VW_X^+ = VW$	Y → Z
	Y → V	$Y^+ = YVWXZ$	∴ V → W
		$Y_V^+ = YXZ$	V → X
	Y → X	$Y^+ = YXVZW$	Y → V
		$Y_X^+ = YVZWX$	Y → Z
	Y → Z	$Y^+ = YZVWX$	
		$Y_Z^+ = YVWX$	∴ Canonical covers = V → WX, Y → VZ

* Keys

→ It is used to uniquely identify any tuple of data

- Super key: It is used to uniquely identify any tuple of data.

$$(\text{Super key})^+ = R$$

- Candidate key: It is a super key but efficient.

eg: $(ABC)^+ = ABCD$
$(AB)^+ = ABCD$
$(A)^+ = ABCD$

} all are SK → but this is efficient & hence it is CK

If subset of SK = R, then it is ... closure of ... Ck

DOMS

→ There can be more than 1 candidate key (among the FD) or in a table

- Primary key :- must contain unique value
  - Cannot have NULL value
  - There can be only 1 PK in table which can consist of single or multiple fields/attributes.

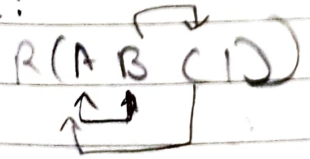
Eg:

① R(ABCD)	② R(ABCD)	③ R(ABCD)	④ R(ABCD)
A → BC	ABC → D	B ↛ ACD	AB → C
	AB → CD	ACD → B	C → BD
	A → BCD		D → A
Soln: $A^+ = ABC$	$ABC^+ = ABCD$ SK	$B^+ = ABCD$ SK, CK	$AB^+ = ABCD$ SK, CK
not a key	$AB^+ = ABCD$ SK	$ACD^+ = ABCD$ SK, CK	$C^+ = CBDA$ SK, CK
coz $[key^+] = R$	$A^+ = ABCD$ SK, CK		$D^+ \rightarrow DA$ ✗

Eg: Find candidate key. Imp

① R(ABCD)
A → B
B → C
C → A

Soln:



not a CK. coz, D is not there

Now, making combination with D

$(AD)^+ = ABCD$ CK
$(BD)^+ = ABCD$ CK
$(CD)^+ = ABCD$ CK

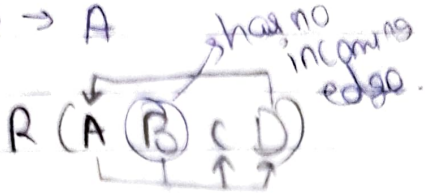
} we have made only 2 combination coz it is minimum

∴ CK = AD, BD, CD

(2) R(ABCD)

AB → CD

D → A



check $B^+ = \{B\}$ ✗

making combo of 2

$(AB)^+ = ABCD$ ✓

$(BC)^+$ ✗

$(BD)^+ = BDAC$ ✓

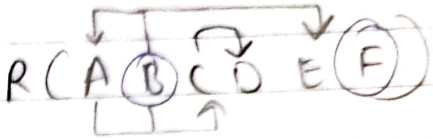
∴ **Ck = AB & BD**

(3) R(ABCDEF)

AB → C

C → D

B → AE



$BF^+ = BAFECD = R$

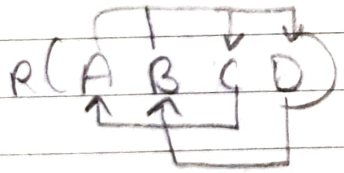
∴ **Ck = BF**

(4) R(ABCD)

AB → CD

C → A

D → B



all have incoming edge

so trying all combo

$A^+ = A$ ✗

$B^+ = B$ ✗

$C^+ = AC$ ✗

$D^+ = DB$ ✗

$AB^+ = ABCD$ → ACB ✗

$AC^+ = AC$ ✗ → ACD ✗

$AD^+ = ADBC$ → BDA ✗

$BC^+ = BCAD$ → BDC ✗

$BD^+ = BD$ ✗

$CD^+ = CDAB$

∴ **Ck = AB, AD, BC, CD**

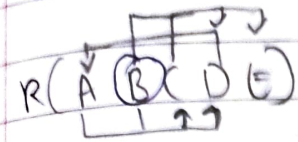
DOMS

(5) $R(ABCDE)$

$AB \to CD$

$D \to A$

$BC \to DE$



$B^+ = \{B\}$ ✗

$AB^+ = ABCDE$ ✓

$BC^+ = BCDEA$ ✓

$BD^+ = BDACE$ ✓

$BE^+ = BE$ ✗

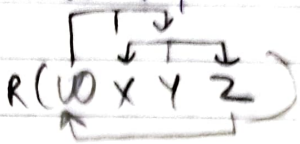
$\therefore$ CK = AB, BC, BD

(6) $R(W\ X\ Y\ Z)$

$Z \to W$

$Y \to XZ$

$XW \to Y$



$W^+ = W$ ✗

$X^+ = X$ ✗

$Y^+ = YXZW$ ✓

$Z^+ = ZW$ ✗

$WX^+ = WXYZ$ ✗

$XZ^+ = XZWY$ ✗

$WZ^+ = WZ$ ✗

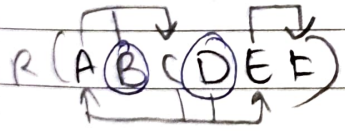
$\therefore$ CK = Y, WX, XZ

(7) $R(A\ B\ C\ D\ E\ F)$

$AB \to C$

$DC \to AE$

$E \to F$



$B^+ = B$ ✗

$D^+ = D$ ✗

$BD^+ = BD$ ✗

$ABD^+ = ABCDEF$

$BCD^+ = ABCDEF$

$BDE^+ = BDEF$ ✗

$BDF^+ = BDF$ ✗

$BDEF = BDEF$ ✗

$\therefore$ CK = ABD & BCD

Note:

If $R(A\ BC)$

So, possible combo ~~are~~

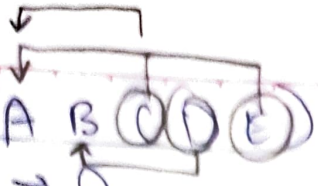
If A is key then don't check AB, AC & ABC

(A) ~~AB~~

B BC ~~ABC~~

C ~~AC~~

(8) R(A B C D E)

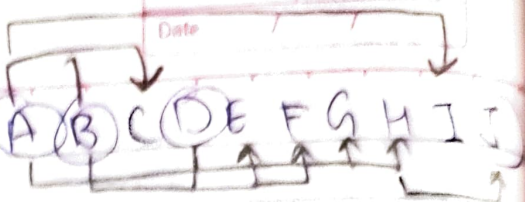


CE → D
D → B
C → A

~~C⁺ = A C~~
~~D⁺ = D B~~
~~E⁺ = E~~
CE⁺ = A B C D E = R
~~DE =~~

Ck = CE

(9) R(A B C D E F G H I J)

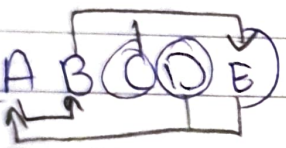


AB → C
AD → GH
BD → EF
A → I
H → J

ABD⁺ = A B C D E F G H I J
= R

Ck = ABD

(10) R(A B C D E)

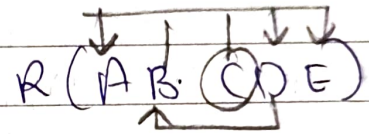


A → B
BC → E
DE → A

× CD⁺ = CD
ACD⁺ = A B C D E
BCD⁺ = A B C D E
CDE⁺ = A B C D E

∴ **Ck = ACD, BCD, CDE**

(11) R(A B C D E)



BC → ADE
D → B

× C⁺ = C
× AC⁺ = AC
~~× BC⁺ = A B C D E~~
~~CD⁺ = A B C D E~~
× CE⁺ = C E
× ACE⁺ = A C E

Ck = BC & CD

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

AB → C
C → D
D → BE
E → F
F → A

A⁺ = A
B⁺ = B
C⁺ = ABCDEF
D⁺ = A BCDEF
E⁺ = A EF
F⁺ = A F

AB⁺ = ABCDEF
AE⁺ = A EF
AF⁺ = A F
BE⁺ = ABCDEF
BF⁺ = ABCDEF
EF⁺ = A EF

CK = C, D, AB, BE, BF

(13) R(A B C D E F G H)

CH → G
A → BC
B → CFH
E → A
F → EG

AD⁺ = ADBCFHEG
BD⁺ = A BCDEFGH
CD⁺ = CD
DE⁺ = A BC DEFGH
DF⁺ = A BC DEFGH
DG⁺ = D G
DH⁺ = D H

CDG⁺ = C D G
CDH⁺ = C D GH
DGH⁺ = D G H
CDGH⁺ = C D GH

CK = AD, BD, DE, DF

* Normalization.

DOMS | Page No. | Date / /

⇒

Redundancy

student_info

S_id	name	age	Br. code	Br. name	HOD name
1	A	18	101	CS	ABC
2	B	19	101	CE	ABC
3	C	18	101	CE	ABC
4	D	20	102	EC	XYZ
5	E	21	102	EC	XYZ
6	F	18	103	ME	KLM

→ In this table, we tried to store entire data of student.

→ Here entire branch data is being repeated.

⇒ Redundancy: → It is the root cause of all problems
→ It means: when same data is stored multiple times unnecessarily in a database.

⇒ Disadvantages:

(i) Insertion, deletion & updation anomalies.
→ Insertion anomalies :- when certain data cannot be inserted in DB without presence of another data.
→ deletion anomalies :- If we delete some data (unwanted) it cause deletion of some other data (wanted)
→ updation anomalies :- when we want to update a single piece of data, but it must be done at all of its copies.

(ii) Inconsistency
(iii) ↑ in dB size & ↑ in time (slow)

monitor

⇒ Solution for student_info redundancy

student_info

S_id	name	age	Br_code
1	A	18	101
2	B	19	101
3	C	18	101
4	D	20	102
5	E	21	102
6	F	18	103

Branch_infos

Br_code	Br_name	HOD_name
101	CE	ABC
102	EC	XYZ
103	ME	KLM

normalization :- It is a process which we use to remove redundancy

↳ To do normalization we use FD.

* 1 Normal Form :-

1) Single value in a single cell.

Roll	name	course
1	Ash	DS, CO
2.	Shub	DSA, OS

2) order of column & order of rows are insignificant

↓ not allowed

Roll	name	course
1	Ash	DS
1	Ash	CO
2	Shub	DSA
2	Shub	OOT

3) In every column, all value must belong to same domain.

4) Every column should have unique name.

* 2NF

→ Let's consider a relation.

R (A B C D)

So, $(AB)^+ = ABCD$

∴ AB is candidate key.

A, B are prime attributes as it ∈ candidate key.

C, D are non-prime attributes as it ∉ CK.

AB → D D is dependent on entire CK.

B → C C is dependent on only a Part of CK.

Partial dependency.

~~Here~~ now, we know that,

A & B makes PK here.

So, PK's total value cannot be null but any one value is allowed to be null

Or any one value must not be null

So,

A	B	
1	—	✓
—	2	✓
—	—	X
3	4	✓

Partial dependency :- when a proper subset of candidate key identify non-prime attribute then it is PD.

Now in this case,

$B \rightarrow C$.

IF

A	B
1	(–) → null.

The value of C is dependent on B

But B is null.

hence | partial dependency is not allowed. |

∴ R(A B C D) is not in 2NF.

To decompose it

R(A B C D)

Create 1st table of candidate key and also their dependents which are entirely dependent on CK.

R_1 (A B D)

Create 2nd table.

R_2 (B C)

eg:- R(A B C)

$R_1(AB)$ $R_2(BC)$

R

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

→ redundancy

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

Hence using normalization redundancy is removed.

eg:- R(A B C D E)

$AB \rightarrow C$

$D \rightarrow E$

$(ABD)^+ = ABCDE = R_1$

∴ CK = ABD.

Now, $AB \rightarrow C$ (PD)

$D \rightarrow E$ (PD)

- R_1 (A B C)
- R_2 (D E)
- R_3 (A B D)

There must always a table for CK.

Q) R(A B C D E)

A → B
B → E
C → D

$(AC)^+ = R$ = CK.

A → B (PD)
C → D (PD)

R_1(A B E)
R_2(C D)
R_3(A C)

Q) R(A B C D E F G H I J)

AB → C (PD)
AD → GH (PD)
BD → EF (PD)
A → I (PD)
H → J

$(ABD)^+ = R$ = CK.

R_1(A B C)
R_2(A I)
R_3(A D G H J)
R_4(B D E F)
R_5(A B D)

* 3NF

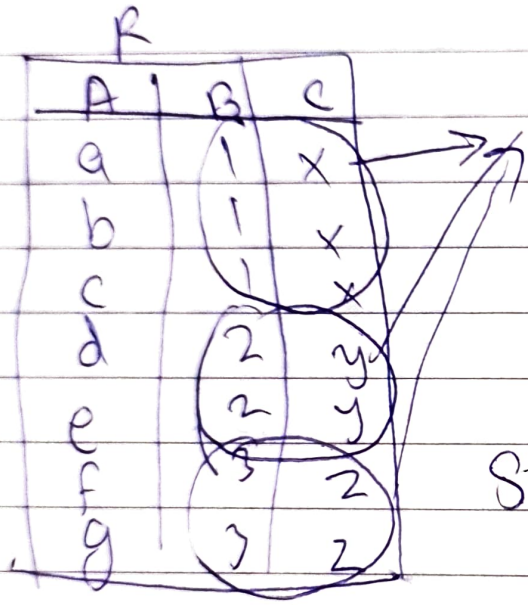
Let R(A B C)

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

Here CK = A

This relation is in 2NF.

BUT:- Lets suppose.



⇒ **Transitive dependency:-**

If is a FD from $\alpha \to \beta$ is called transitive if $\alpha, \beta \in$ non-prime attributes.

For 3NF: A relation R is in 3NF if
a) if it is 2NF
b) no transitive dependency.

Now, converting R(ABC) in 3NF.

R_1 (B C)
R_2 (A B)

R_2

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

R_1

B	C
1	x
2	y
3	z

Hence redundancy is removed

PD	$P \Rightarrow NP$
TD	$NP \rightarrow NP$

→ Now we can also prove that the relation is in 3NF by proving any A the following condition true.

- every dependency from $\alpha \rightarrow \beta$
  (i) either α is super key
  (ii) OR β is a prime attribute.

This is used to directly find 3NF or not, without finding 2NF

eg: R(A B C D E)

not SK — A → B — not P

B → E ∴ R is not

C → D in 3NF

(AC)⁺ = ABCDE = R.

= CK

- R_1 (A B E) TD
  - R_{11} (A B) ✓
  - R_{12} (B E) ✓
- R_2 (C D) ✓
- R_3 (A C) ✓

Here we'll get 4 tables

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

SK — AB → C

not SK — A → DE → not P

- B → F Hence
- F → G H not in
- D → I J 3NF.

(AB)⁺ = R = CK

- R_1 (A D E I J)
  - R_{11} (A D E) ✓
  - R_{12} (D I J) ✓
- R_2 (B F G H)
  - R_{21} (B F)
  - R_{22} (F G H)
- R_3 (A B C) ✓

eg: R(A B C D E)

SK — AB → C not P

not SK — B → D not

- D → E in 3NF

(AB)⁺ = R = CK.

- R_1 (B D E)
  - R_{11} (B D)
  - R_{12} (D E)
- R_2 (A B C)

eg:

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

- AB → C → not P
- AD → GH
- BD → EF
- A → I
- H → J

hence not in 3NF

CK: ABD

- R_1 (A B C I)
  - R_{11} (A B C) ✓
  - R_{12} (A I) ✓
- R_2 (A D G H J)
  - R_{21} (A D G H) ✓
  - R_{22} (H J) ✓
- R_3 (B D E F) ✓
- R_4 (A B D) ✓

* BCNF (Boyce Codd NF)

A relation R is in BCNF if in $\forall\ \alpha \rightarrow \beta$

α is a super key.

eg: R (A B C)

The R is in 3NF.

SK ✓

AB → C

C → B

not SK

Candidate key = AB & AC.

∴ prime attr = ABC

- R_1 (C B)
- R_2 (A C)

R_2 (A B) ×

R

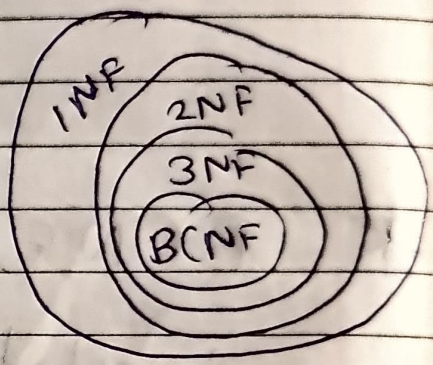
A	B	C
a	1	x
b	2	y
c	2	z
d	2	z
e	3	w
f	3	w
g	3	w

R_1

C	B
x	1
y	2
z	2
w	3

R_2

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



we solved
In 2NF : P → NP
3NF : NP → NP
BCNF : P/NP → P

Q. Consider the following relation with FD & identify the normal form.

How to check?
step1: check BCNF
If yes then ✓
else goto step2.
step2: check 3NF
If yes then ✓
else goto step3
step3: check 2NF
If yes then ✓

① R(ABCDEFGH)

AB → C BCNF

A → D BCNF fails not P/2NF

B → F

F → GH

Ck = AB = Sk

It is 1NF

② R(ABCDE)

~~BCNF~~ CE → D

~~2NF~~ D → B

~~3NF~~ C → A

Ck = Sk = CE

It is 1NF

③ R(AB C DE F)

BCNF AB → C

~~2NF~~ DC → AE

~~2NF~~ E → F

Ck = Sk = ABD BCD

It is 1NF

④ R(ABCDEFGHI)

AB → C ~~BCNF~~

BD → EF ~~3NF~~

AD → GH ~~2NF~~

A → I

Ck = Sk = ABD

It is in 1NF

⑤ R(ABCDE)

AB → CD ~~BCNF~~

D → A 3NF

BC → DE

Ck = Sk = AB BD BC

It is in 3NF

⑥ R(ABCDE)

BC → ADE ~~3NF~~

D → B ~~2NF~~

Ck = Sk = BC CD

It is in 3NF

⑦ R(VWXYZ)

X → YV ~~BCNF~~

Y → Z ~~3NF~~

Z → Y ~~2NF~~

VW → X

Ck = Sk = VW XW

It is in 1NF

⑧ R(ABCDEF)

ABC → D ~~BCNF~~

ABD → E ~~3NF~~

CD → F ~~2NF~~

CDF → B

BF → D

Ck = Sk = ABC ACD

It is in 1NF

⑨ R(ABC)

A → B BCNF

B → C

C → A

Ck = Sk = A B C

It is in BCNF

(10) R(ABCDEF)
A → BCDEF
BC → ADEF
DEF → ABC
(A), (BC), (DEF)
It is BCNF

(11) R(ABC)
AB → C
C → A
(AB), (BA)
It is 3NF

(12) R(ABCDE)
A → B
BC → E
DE → A
(ACD), (BCD), (CDE)
3NF

(13) R(ABCDE)
AB → CD
D → A
BC → DE
(AB), (BC), (BD)
3NF

(14) R(WXYZ)
Z → W
Y → XZ
XW → Y
(Y), (XW), (XZ)
3NF

(15) R(ABCDE)
A → B
B → E
C → D
(AC)
INF

(16) R(ABCDEF)
AB → C
BC → AE
E → F
(ABD), (BCD)
INF

(17) R(VWXYZ)
Z → Y
Y → Z
X → VY
VW → X
(VW), (XW)
INF

(18) R(ABCDEF)
ABC → D
ABD → E
CD → F
CDE → B
BE → D
(ABC), (ACD)
INF

(19) Normalize from INF to BCNF.

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

AB → C
A → DE
D → IJ
B → F
F → GH

CK = AB

DOMS

⇒ Decompose to 2NF.

R_1 (A B C)

R_2 (A D E I J)

R_3 (B F G H)

⇒ Decompose to 3NF

R_1 (A B C) ✓

R_2 (A D E I J)
- R_{21} (A D E) ✓
- R_{22} (D I J) ✓

R_3 (B F G H)
- R_{31} (B F) ✓
- R_{32} (F G H) ✓

This is also in BCNF.

⇒ ~~Decompose to BCNF~~

Hence we'll have 5 tables.