# DBMS

→ Data: Raw, unprocessed facts.
E.g. Suresh, Banglore.

→ Information: Processed data
E.g. The age of Suresh is 25

→ Database: Collection of related data
E.g. Online banking system, Library management system

→ Meta-data: The data-base def^n

DBMS: Collection of programs that enables users to create & maintain database.

Functionalities

→ Define: Specifying the data-type, structures & constraints for the data to be stored

→ Construct: Process of storing data on some storage medium

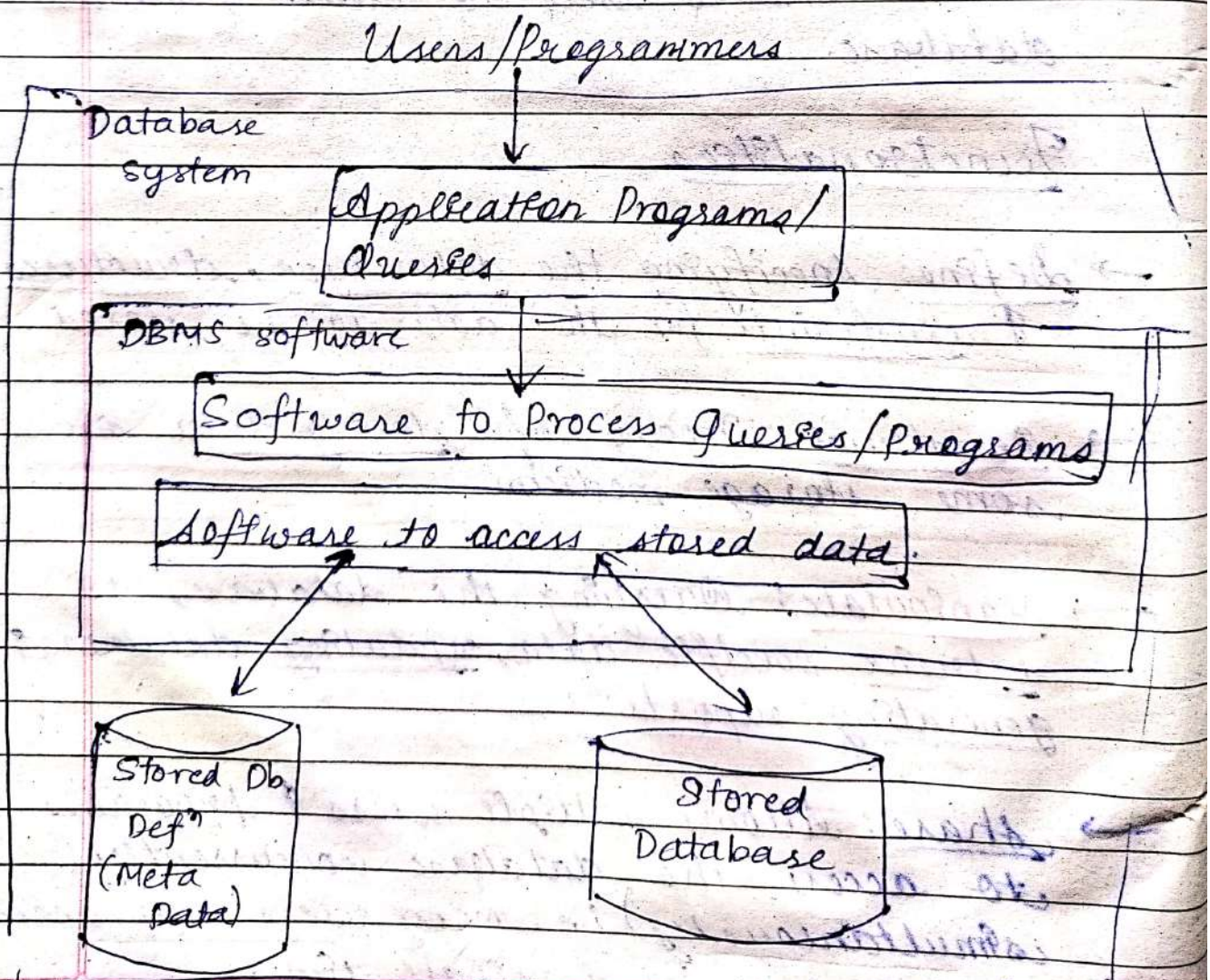
→ Manipulate: Querying the database, to retrieve specific data, updating database & generating reports.

→ Share: Allows multiple users & programs to access the database concurrently (simultaneously) i.e. many users can access the database at a single time.

Protection of the database from unauthorized access or from hardware or software failures & also maintainance of the database for a long period of time.

PROPERTIES

→ A database represents some aspects of the real world (mini world)
→ It is a logically coherent collection of data with some inherent meaning
→ Designed built & populated with data for a specific purpose



| Student | Name | Roll no. | Class | Major |
|---|---|---|---|---|
| | Smith | 17 | 1 | CS |
| | Brown | 8 | 2 | CS |

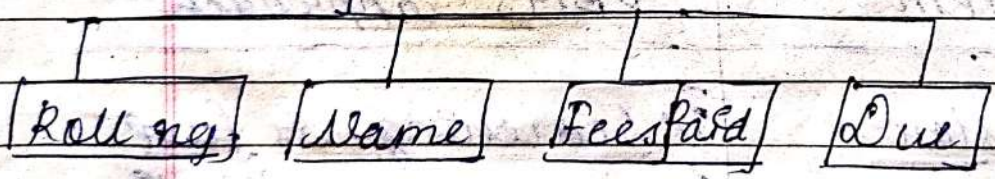
| Course | Course Name | Course No. | Dept |
|---|---|---|---|
| | DS | 123 | CS |
| | DSGT | 234 | MATH |
| | DBMS | 567 | CS |

| GRADE-REP. | Roll-no. | Courseno. | Grade |
|---|---|---|---|
| | 17 | MATH240 | B |
| | 17 | CS1310 | A |
| | 8 | CS1310 | A |

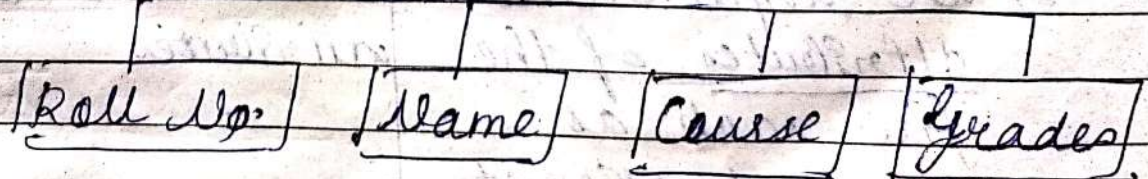
A Database storing student & course info.

* File System Approach

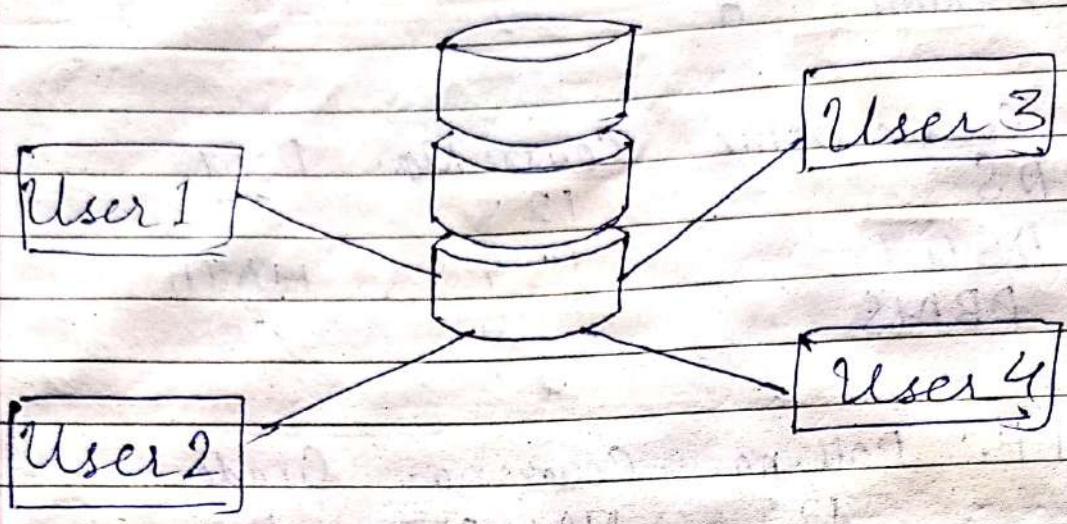
Accounts Dept.




Exam Dept



DBMS approach




You use file-system approach on your PC.
DBMS is used for client-server architecture. i.e. the data is on a centralized location & accessed by users all over the world.

| File System | DBMS approach |
|---|---|
| 1) Searching slow<br>High memory usage | 1) Searching is fast<br>less memory utilization |
| 2) We require the attributes of the file such as name, location etc. | 2) We don't require attributes |
| 3) No protocols for concurrency, hence no protocol guarantee | There exists protocols for concurrency in DBMS, whereas no |

| | |
|---|---|
| of consistency of data | which leads to consistency of data |
| No ~~not~~ role based security | Role based security is provided, so that a person of a particular role can access a data which is invisible to other |
| There is redundancy | There is no redundancy, i.e. there is no duplication of data. for eg. two files with diff. name & same data can't exist This is due to the concept of foreign key & primary key |

# Characteristics of DBMS approach

→ Self Describing nature of Data-base system

→ Insulation betn Programs & Data, & Data abstraction

→ Support of Multiple views of Data

→ Sharing of data & Multiuser Transaction Processing

- Self Describing nature of DB system

⇒ Database system:

→ Data-base + Meta-Data (DB def$^n$)

↓

Stored in: DBMS catalogue

↓

Used by: DBMS software & DB users

→ DBMS software must work equally well with any number of database applications

→ In traditional file processing, data def$^n$ → part of application programs → work only with one specific DB

- Insulation bet$^n$ programs & Data & Data abstraction

→ In traditional file processing → structure of data files is embedded in the application programs

→ In DB approach → structure of data files is stored in DBMS catalogue → seperate from access programs (Program data-independence)

→ The characteristic that allows program data-independence is called data abstraction

→ DBMS users with conceptual representation of data (i.e. it hides the behind scene details which are not interest to DB users)

→ Data model → type of data abstraction → provides conceptual representation to users such that they can understand & hides

- Support of Multiple views of data

→ A database has many users, each of whom may require a diff. view of DB. (Provide only necessary data req^d by the user)

→ A view → subset of DB → contains virtual data derived from DB (not explicitly stored).

- Sharing of Data & Multiuser Transaction Processing

→ A multiuser DBMS allows multiple users

→ Must include concurrency control (i.e. Multiple users are sharing the same DB at the same time, to prevent two users editing the same data)
for e.g. ~~Ira~~ Train reservation

→ Such type of applications are called OLTP (Online Transaction Processing) → major part of DB application.

→ DBMS must enforce several transaction properties.

Types of Transaction Prop.

→ Isolation
If one agent assigns a seat to a passenger, then that seat is blocked or isolated from other agents

→ Atomicity
DBMS must ensure that the transaction is executed completely or none at all.
For e.g. If an agent has assigned a seat to a pass. & transaction is stuck midway or incomplete then that seat has to be released for other agents)

# DATABASE USERS

Actors on the scene (People whose jobs involve day to day usage of a large data-base)

→ Database Administrators

→ Database Designers

→ End users

→ System Analysts & Application Programmers (Software Engineers)

Workers behind the scene (Whose job is to maintain DB system environment)
[They deal with Design, ~~Dev.~~ & Development & Operation of DB system environment)

→ System Designers & Implementers

→ Tool developers

→ Operators & Maintainance personnel

---

⇒ Actors on the scene

## (1) Database Administrators

(Now in any large org. we may have resources & we always need a man personnel to manage these resources)

→ In data-base environment, ~~primary resource~~ → ~~database~~,
primary resource → database
secondary resource → DBMS & related software

→ Database Administrator (DBA) responsibilities.

1) Administering primary / secondary resources

2) Authorizing access to the database.

3) Co-ordinating & Monitoring use of DB.

4) Acquiring hardware & software resources as needed.

5) Troubleshooting of any problem arises

6) Deal with security issues.

(2) Database Designers

→ Responsible for:

1) Identifying the data to be stored in the database.

2) Choosing appropriate structure to represent & store data.

3) Communication with DB users → understand their requirements → designs database.

4) They develop views for each group of users & each view is linked with the views of other group of users & that final DB design should fulfill the req. of all user groups.

(3) End Users:

People whose roles require access to DB → for querying, updating & generating reports

Several categories of end users:

- Casual end users: Accesses DB occasionally → typically mid middle or high-level managers or other occasional browsers.

- Naive or parametric end users:
constantly querying & updating DB used using canned transactions.
(Standard types of queries & updates to query the DB for some data or to update the DB for some changes)

For e.g. Reservation agents

- Sophisticated end users:
Engineers, scientists, business analysts

- Stand-alone users:
Maintain personal databases → using ready made program packages

For-e-g. A person running a business would ~~be~~ keep a track of his personal financial data for tax purposes.

System Analysts & Application Programmers (Software Engineers)

→ System analysts → determine the req. of end users → develop specifications for canned transactions

→ Application programmers → test, debug, document & maintain these canned transactions

# WORKERS BEHIND THE SCENE

→ System Designers & Implementers:
Design & implement DBMS modules & interfaces as a package.

→ Tool Developers:
Persons who design & implement tools (Software packages)
These software packages are optional & can be purchased separately to improve the performance of a DB system.

→ Operators & maintainance personnel:
Responsible for actual running & maintainance of Hardware & software of DB system environment

## Three-Schema Architecture

→ Goal: to seperate user applications & the physical database.

→ 3 Levels:

- Internal Level: (Physical)

→ Describes the physical storage structure of database.

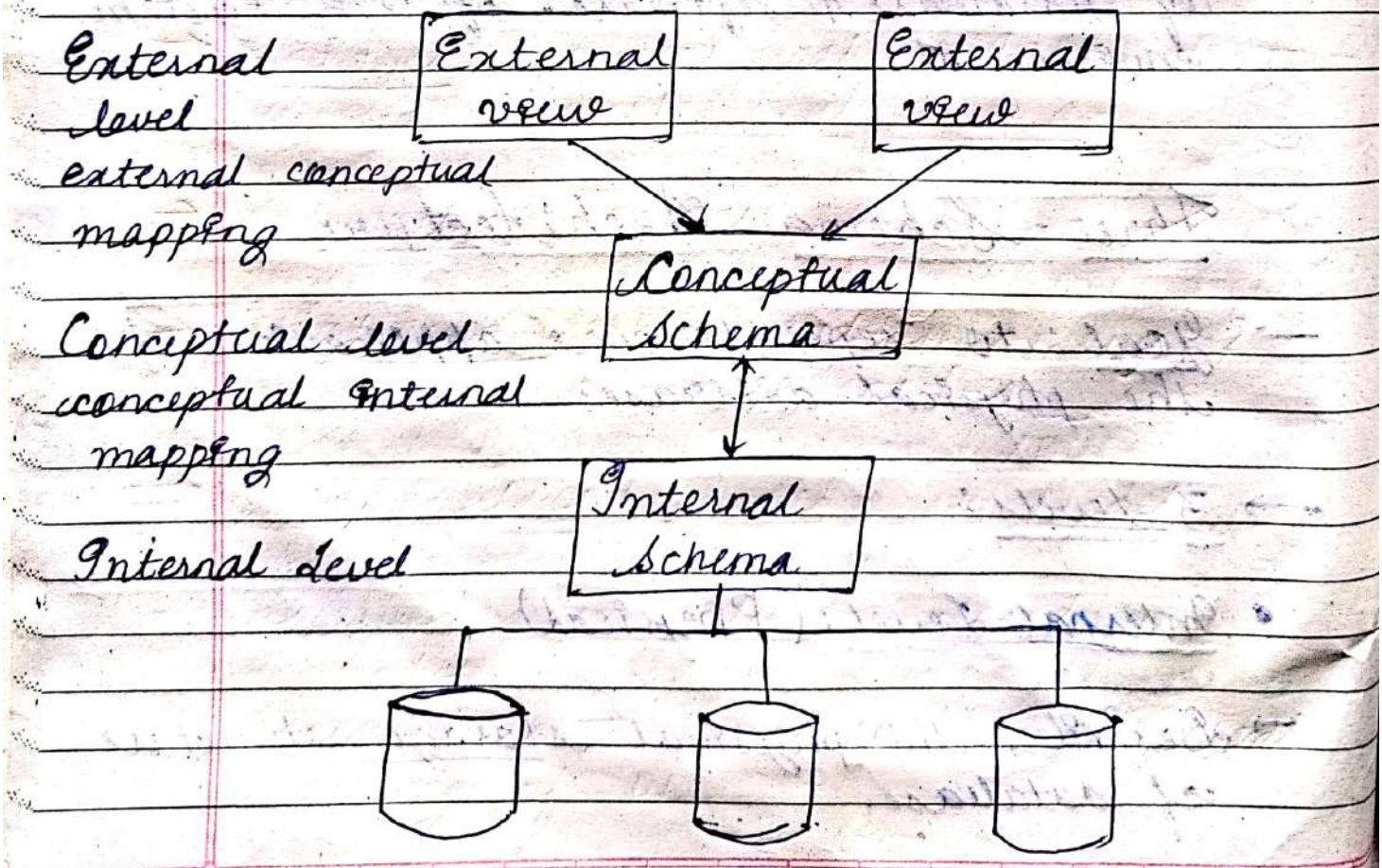
→ Describes complete details of data storage & access paths

- **Conceptual Level:** (Logical

→ Hides the details of the physical storage structure & concentrates on describing entities, data types, relationships, constraints.

- **External Level:**

→ Describes the part of the DB that a user is interested in & hides the rest of the DB from the user group.



# DATA INDEPENDENCE:

Def$^n$: Capacity to change the schema at one level of a database system without having to change the schema at the next higher level.

## Types of Data Independence:

1. ~~Logical~~
2. Logical Data Independence

Ability to modify the conceptual schema without changing external schemas or application programs.

2. Physical Data Independence

→ Ability to modify the internal schema without changing conceptual schema
→ Changes may be needed to improve performance.

# DATA MODEL

Most imp. feature of DBMS approach is it provides data abstraction → (complex details not needed by users)
For this purpose data model is required

## Data Models:

→ Used to describe the structure of the database → helps to achieve data abstraction

- Includes a set of basic operations for specifying retrievals or updates on the DB
- Includes concepts to specify the behaviour of a DB application

### ⇒ High level or conceptual Data Model:

- Provides concepts that are close to the way many users perceive data.
- Use concepts such as entities, attributes & relationships

⇒ Entities: represent real world object or concept

⇒ Attributes: further describe an ~~en~~ entity

⇒ Relationships: association among 2 or more entities

### ⇒ Low-level or Physical Data Model:

- Describes how data is stored on computer
- Access path → structure for efficient searching of DB.

⇒ Representational (or Implementation) Data Model:

- Represent data using record structures: record based data models

✓ Terminologies:

- Database schema: Description of a database

→ Meta-data is data about data whereas schema is just a plan or a layout or a blueprint of the DB & it's a component of the meta-data.

- Schema Diagram: Displayed schema

E.g.:

Student

| NAME | Roll no. | Branch | address |
|---|---|---|---|

- Schema Construct: Each object within the schema

E.g. Student, course etc.

- Database State (Instance or snapshot): The data in the DB at a ~~parted~~ particular moment.

# DBMS Languages

⇒ In DBMS where no strict seperation of levels is maintained, Data Definition Language (DDL) is used to define the external & conceptual schemas.

→ ~~DDL~~ DDL is used by DBA's & DB designers & DDL compiler compiles the DDL statements

⇒ In DBMS, where a clear seperation is maintained betⁿ conceptual & external levels:

→ DDL → used to specify the conceptual schema only

→ Storage Definition Language (SDL) → used to specify the internal schema.

Mappings betⁿ external level & conceptual level can be specified with the help of either DDL or SDL.

→ View Definition Language (VDL) → to specify user views & their mappings to the conceptual schema,

→ shows mapping betⁿ external level & conceptual level.

→ Data Manipulation Language (DML) → for manipulation of data in DB.

| High-level (Non-procedural DML) | Low-level (Procedural DML) |
|---|---|
| → Used to specify complex ~~dat~~ DB operations in a concise manner. | → Embedded in a general purpose programming language. (C/C++/Java etc.) |
| → Called as set-at-a-time DMLs. (process multiple records at a time) | → Called as record-at-a-time DML. (Uses looping to retrieve each record from a set of records) |

→ DBMS Interfaces

- Menu based interfaces:

→ These interfaces present the users with a list of options (menus)
→ Most popular → Pull-down menus

- Forms-Based Interfaces:

→ Displays a form to each user
→ Designed & programmed for the naive users.

- Graphical User Interface

→ Displays a schema to the user in diagrammatic form.

→ Utilizes both menus & forms.

→ Uses a pointing device like a mouse to pick certain parts of the displayed diagram.

- Natural Language Interface

→ Has its own schema & dictionary
→ Refers to them while interpreting a request.
→ If interpretation is successful → high level query is generated → Query processing
→ Otherwise a dialogue is started with the user for further clarification

- Interfaces for the Parametric users

→ Parametric users: have small set of operations that they must perform repeatedly.
→ System Analysts & Programmers: Design & implement a special interface for these users

Page No.:
Date: / /

- Interface for the DBA:

→ → ~~Privi~~ Privileged commands (like for creating accounts, granting access) that can be used only by DBA's staff.

Criteria used to classify DBMS:

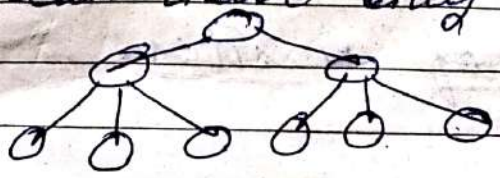
→ Data model
→ No. of users
→ Cost of DBMS
→ No. of sites
→ Types of access paths
→ Generality

- ~~Datat~~.
- Data models:

→ Relational data model:
represents database as a collection of tables. (Most popular)

→ Object-Oriented DB model:
defines DB in terms of objects, their prop. & their operations. (Not used widely)

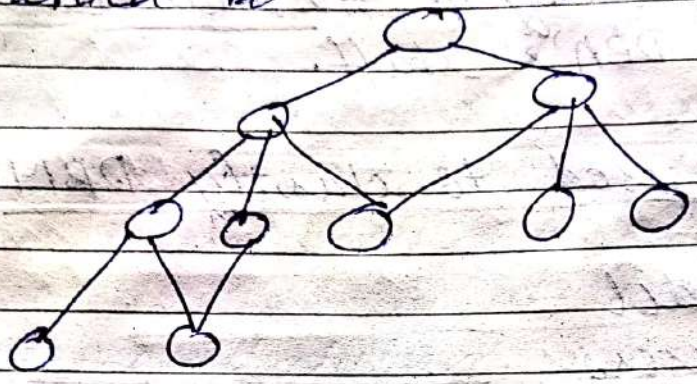
→ Hierarchical model:
→ represents data as tree structures.
→ a child can have only one parent (Not common)



⇒ Network Model:

→ represents objects & their relationships in the form of graph.

→ A child can have more than one parent (Not common)



o Number of Users

→ Single User systems: supports only one user at a time.

→ Multi User systems: supports multiple users concurrently.

o Cost of DBMS:

→ Low cost: Cost of 100$ – 3000$

→ Medium cost: 10,000$ – 100,000$.

→ High cost: 100,000$ & more

Classification of DBMS

* Number of sites:

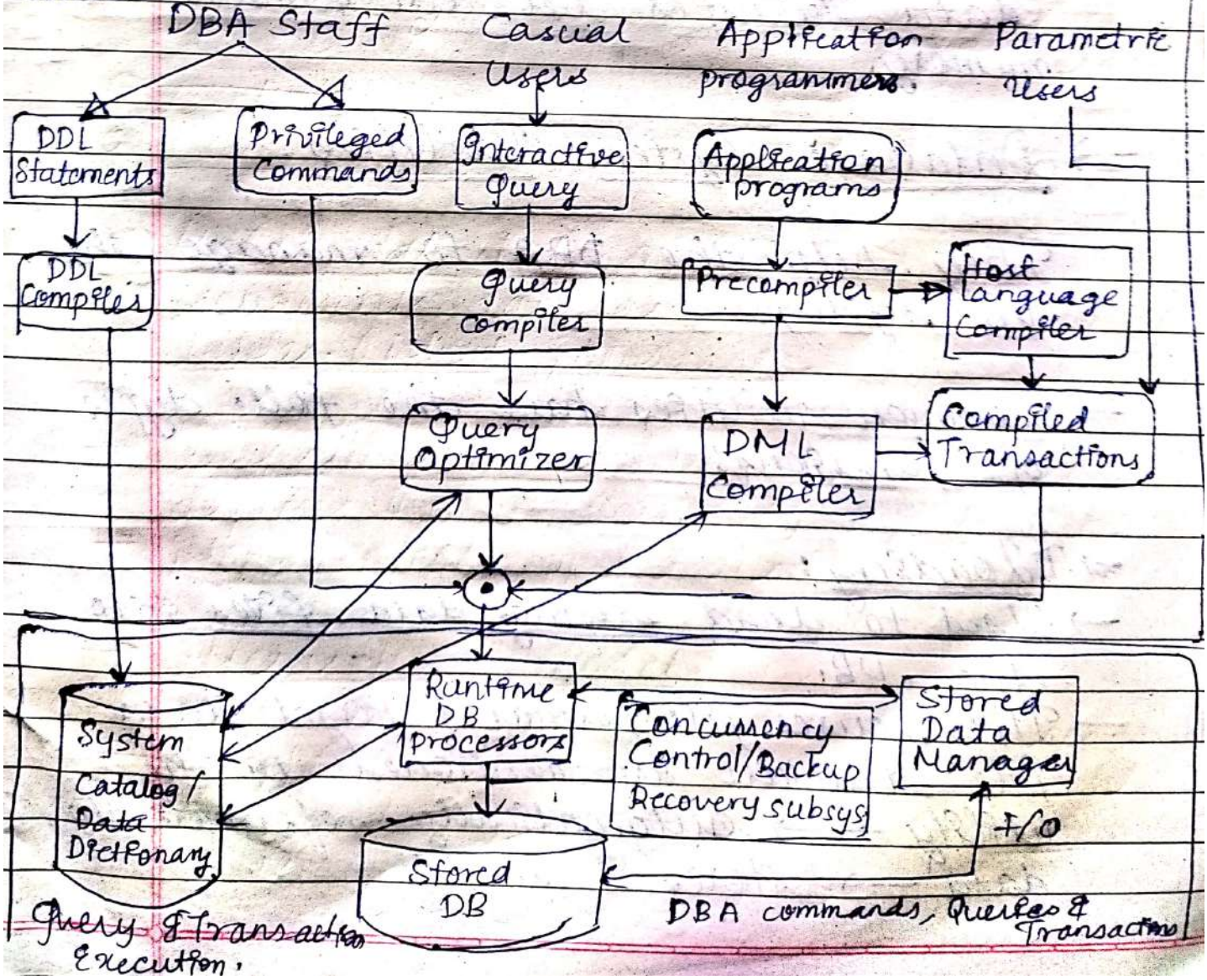
- **Centralized data base system:**

DBMS & database → reside at a single computer site.

- **Distributed DB system:**

DBMS & database → distributed over many sites, connected by a computer network.

# DBMS Component Modules



→ DDL compiler → processes schema def" → stores it in the catalogue

→ Query compiler → handles high level queries

→ Pre-compiler → extracts DML comm commands from an application program.

→ Runtime database processor:
handles DB accesses at runtime.

→ Stored data manager:
data transfer betⁿ desc & main memory.

⇒ Database system Utilities:

They help the DBA to manage the DB system.

→ Common utilities have the foll. types of functions:

⇒ Loading:
→ Used to load existing data files into the DB.
→ The source file format & target data file structure are mentioned to the utility → Automatically reformats the data → stores it in DB

2) Backup
→ Creates a back up copy of the DB.
→ Incremental backups are often used. Though it is complex it saves space.

3) File re-organization:
→ Used to re-organize a DB file into a different file organization.

4) Performance monitoring
→ Monitors DB usage & provides stats to the DBA.

Other tools:

⇒ CASE (Computer aided Software Engg.) tools:
→ Used during design of DB.

⇒ Data Dictionary (data repository) system→
→ Storing catalogue information
→ It also stores other info. like descriptions of application program, user information etc.
→ It is called as info. repository because it has info. which can be directly accessed users or DBA's as & when req^d. It's similar to a catalogue just has more information.

⇒ Application development Environments:
→ provides an environment for developing DB applications
E.g. JBuilder, Power Builder systems

⇒ Communication Software: →
→ remote access to the DB through computer terminals or workstations or their local personal computers.
→ They are connected to the DB sites through phone lines or lan etc.

# Centralized DBMS architecture

→ Earlier mainframe computers were used to process all system functions
→ Users accessed systems via computer terminals → provided only display capabilities but no processing capabilities.
→ Processing → performed remotely on computer systems.
→ Only display info. → sent to the terminals
→ Prices of hardware declined → terminals replaced by PCs & workstations.

Slowly DBMS systems started making use of the client ~~server architecture~~ processing ability of the systems at the user side also & not just at the server's side. & this led to client-

server architecture.

# CLIENT SERVER ARCHITECTURE

It was developed to deal with large no. of personal computers, workstations, printers, file severs etc.

⇒ Goal → define specialized severs with specific functionalities.
For e.g. we can connect a no. of PC's etc & file severs & they are responsible for maintaining the files of the client machines

⇒ Client → User machine that provides user interface capabilities & local processing
→ When a client req. access to any additional functionality that does not exist on its machine, it connects to a server that provides the necessary functionality as required by the user

⇒ Server → provides services to client machines

## Two-Tier Client/Server Architecture

→ In RDBMS, user interfaces & application programs → moved to client side

→ Query & transaction functionality → on server side (query server/ transaction server)

→ When DBMS architecture is req$^d$. → program establishes a ~~set~~ connection with the DBMS (server side).

⇒ Open Data base Connectivity (ODBC) → provides API → allows programs (client-side) to call DBMS on server side, In this way a client program can connect to several RDBMS & send query & transaction requests using ODBC API, which are processed at the servers side.

→ ODBC API is used to establish a connection betⁿ the client & the server.

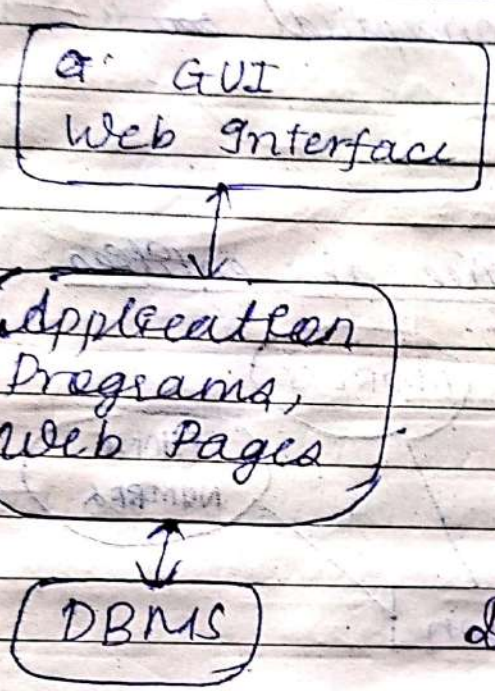
→ So once program is est. the program can communicate with the DBMS on the server side.

⇒ Advantages

① Simplicity ② Compatibility

Emergance of world wide web → led to three tier architecture

## Three-Tier client/server Architecture




GUI Web Interface → client

Application Programs, Web Pages → Application server or Web server

DBMS → Database server

⇒ Additional Intermediate layer betⁿ client & DB server →

Application server or Web server

↓

Stores rules used to access data

↓

Accepts requests from client

↓

Processes the requests

↓

Send commands to DB server

# ER MODEL

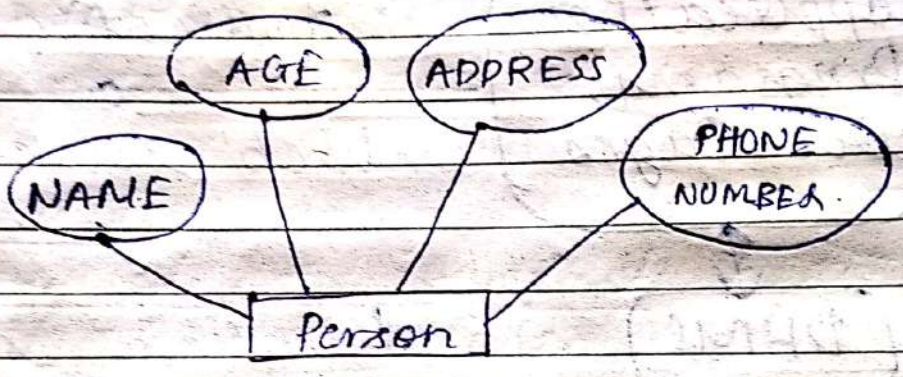
→ Entity: A 'thing' in the real world with an independent existence

→ E.g. house, person (Physical existence)
course, goal (Conceptual ~~on~~ " )

Attributes:

→ Properties that describe the entities.

E.g.




| Composite Attributes | Simple Attributes |
|---|---|
| → Can be divided into further parts<br>→ E.g. Name → First Name, Lastname | → Cannot be divided.<br>E.g. Weight |
| **Single Valued** | **Multi-valued** |
| → Have a single value for a particular entity<br>E.g. Age | Can have set of values for a particular entity.<br>E.g. Languages known |

| Derived | Stored |
|---|---|
| → Can be derived from other attr.<br>→ E.g. Age - derived from DOB | → From which value of other attr. are derived<br>→ E.g. DOB |

⇒ Complex attributes :

→ Has multi-valued & composite components in it.

→ Multi-valued attributes → represented with '{ }'.

→ Composite attributes → represented with '( )'.

→ E.g. {CollegeDegrees (College, Year, Degree, Field)}

⇒ Null Values

→ Null is something which is not applicable or unknown.

| Student_ID | NAME | AGE | PHONE NO. |
|---|---|---|---|
| 1 | Sam | 20 | 12345678 90 |
| 2 | Pam | 22 | NULL |
| 3 | Ram | NULL | 1098765432 |

⇒ Entity Type:
→ A collection of entities having same attributes.
E.g.: Student

STUDENT

| ID | NAME | AGE |
|---|---|---|
| 1 | RAM | 20 |
| 2 | SHYAM | 22 |
| 3 | KESHAV | 18 |

⇒ Entity Set:
→ Collection of entities of a particular type at a point in time.

⇒ Key Attribute:
→ That attribute that is capable of identifying each entity uniquely.
→ E.g. Roll no. of student

STUDENT

| ID | NAME | AGE |
|---|---|---|
| 1 | A | 20 |
| 2 | B | 21 |
| 3 | C | 20 |

⇒ Value set of Attributes
→ Set of values that can be assigned to an attribute.

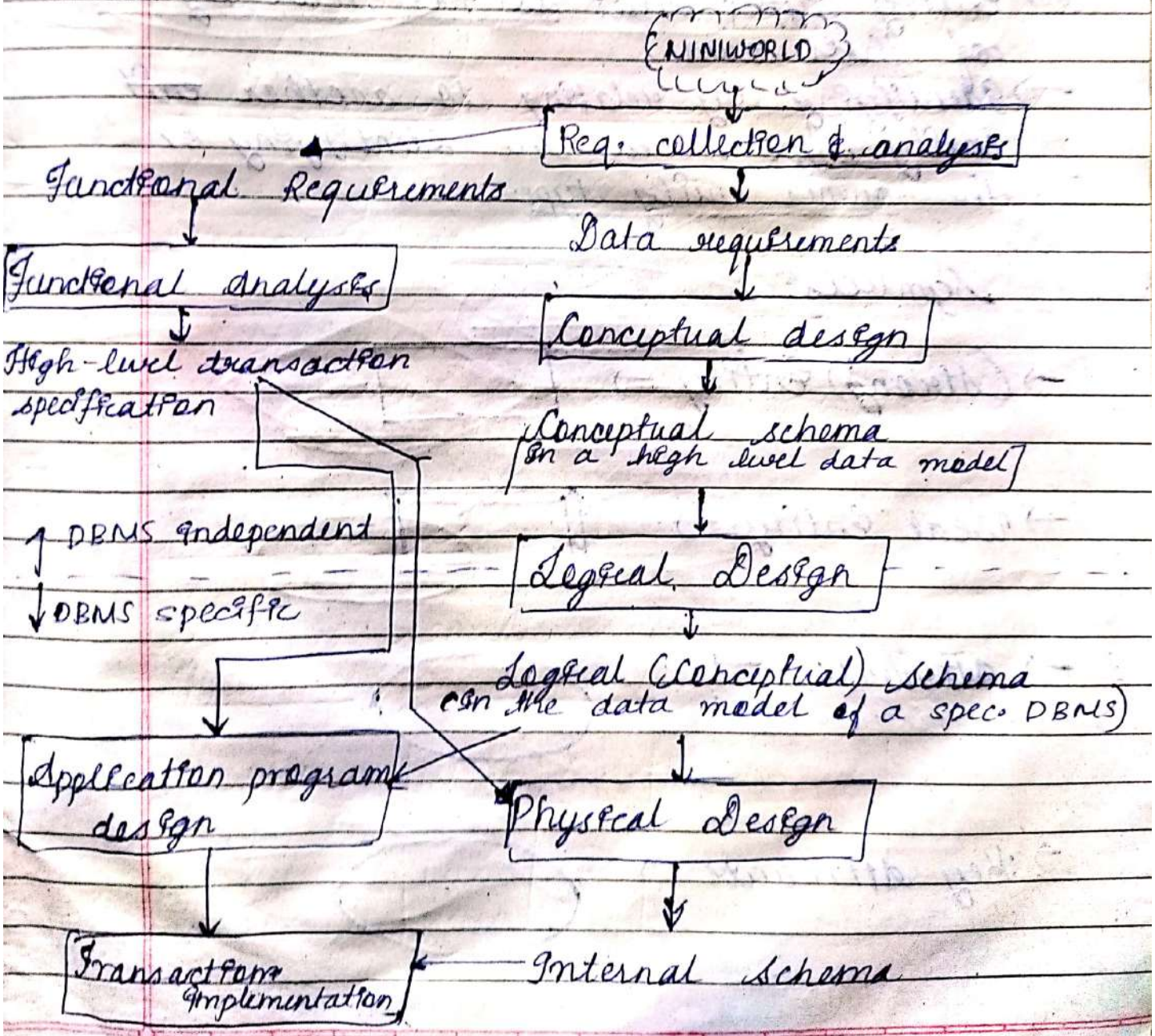
# Database Design

=> Requirements Collection & Analysis

→ Database designers understand & document the data requirements of the DB users.

=> Functional Requirements

→ Consists of user-defined operations



⇒ Logical Design:
→ Actual Implementation of the DB, using commercial DBMS

⇒ Physical Design:
→ The internal storage structures, indexes, access paths → specified.

⇒ Weak Entity

→ Entity types that do not have key att. of their own.
→ Identifying by relating to another entity type called the identifying or the owner entity type

Symbols:

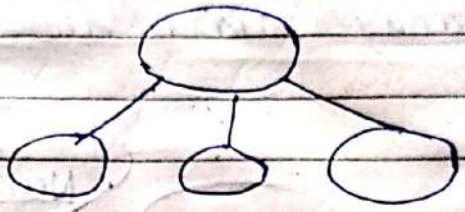
→ (Strong) Entity →

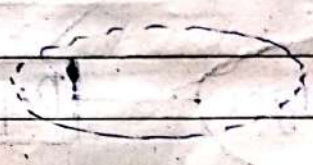
→ Weak Entity →

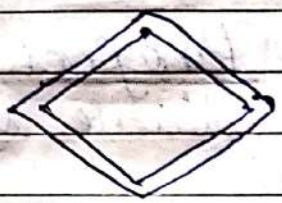
→ Attribute →

→ Key Attribute →

→ Multivalued dtt. ⟶ 

→ Composite dtt. ⟶ 

→ Derived dtt. ⟶ 

→ Identifying Relationship → 
Instances of weak entity type
eg. an IR type that relates
them to identifying entity types

E.g. COMPANY DB

Requirements gathered

→ Company is organized into departments. Each dept. has a unique name, unique no. & a particular employee who manages the dept. We also keep track of the start date of manager & dept. may have several locations.

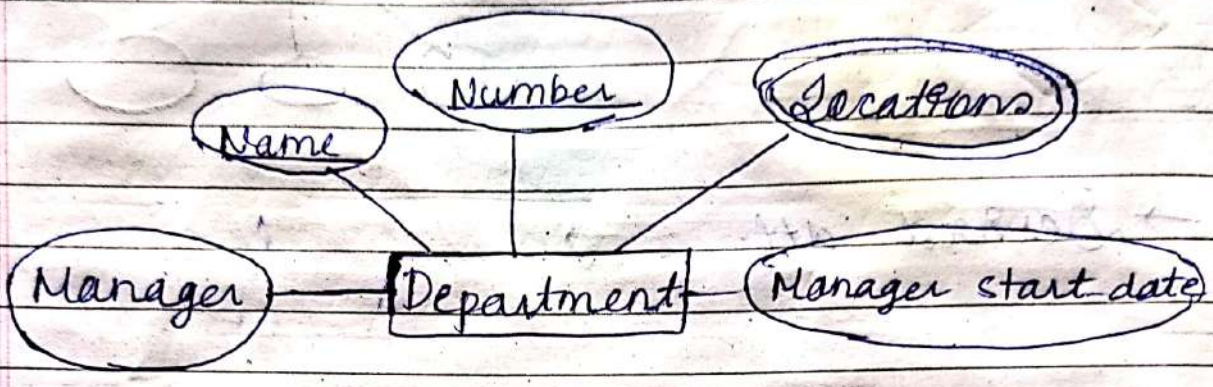
→ A dept. controls no. of projects, each has unique name, no. & location.

→ Employee details → name, SSN, sex, salary. Track of his per week on each project.

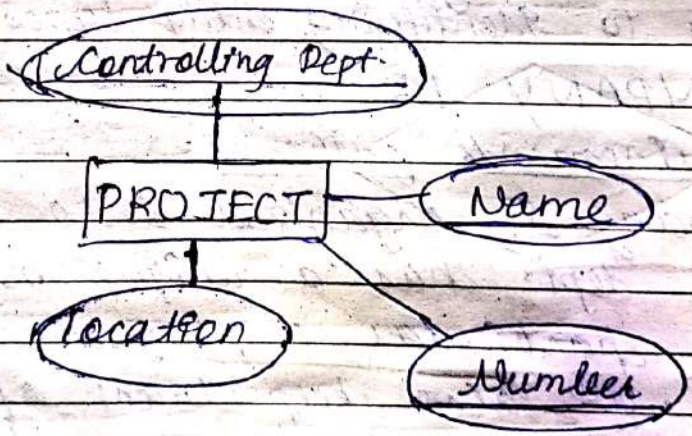
→ Employee's dependents (name, sex, rel?)

# Gnttal Conceptual Design of DB

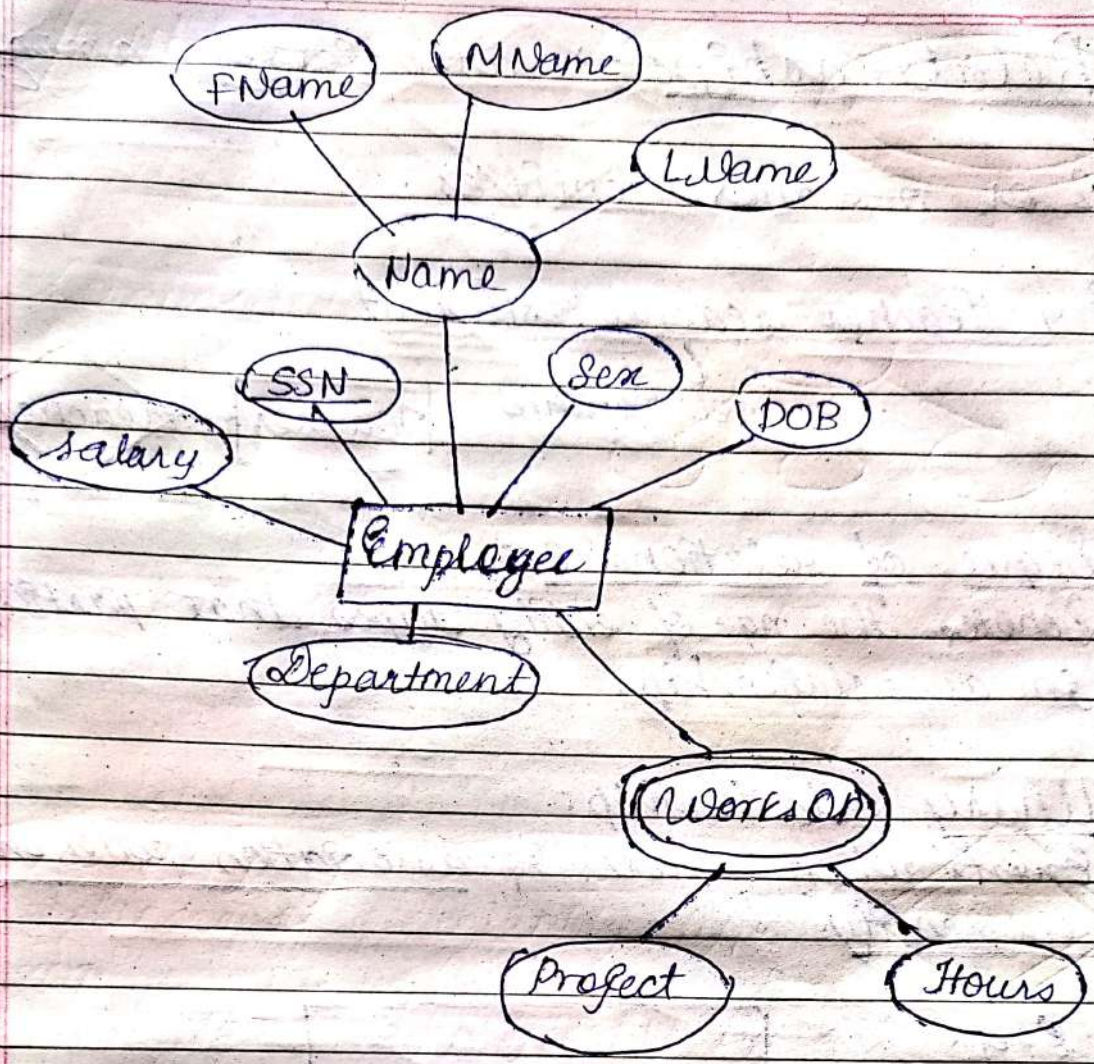
→ Dept.
Name, Number, {Locations}, Manager, Manager Start Date




→ Project.
Name, No. location, Controlling Dept.

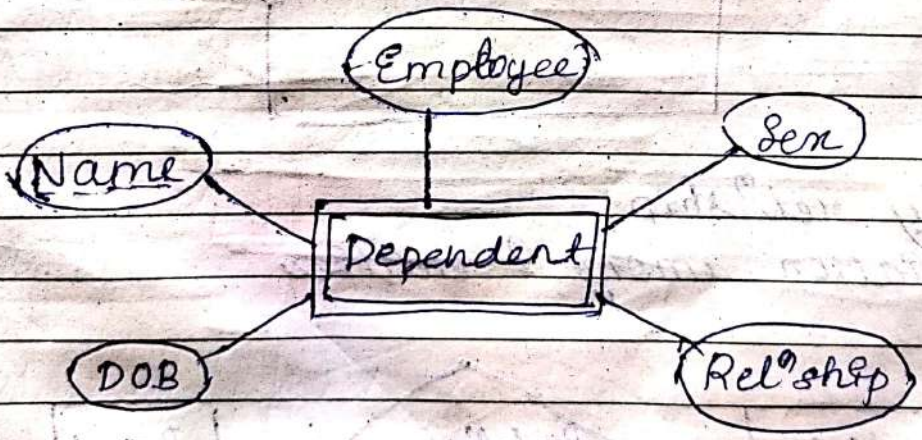



→ Employee
Name(F Name, M Name, L Name)
SSN, Sex, Salary, DOB, {Works on(Project, Hours)}




→ Dependent

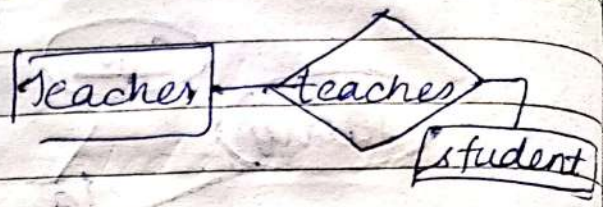
Employee, Dep. Name, Sex, DOB, Rel$^{n}$



Relationships → association among 2 or more entities.

E-g. Teacher teaches student

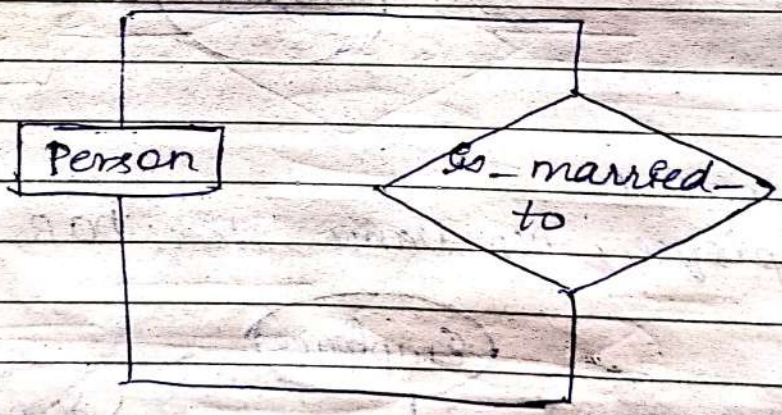
relationship



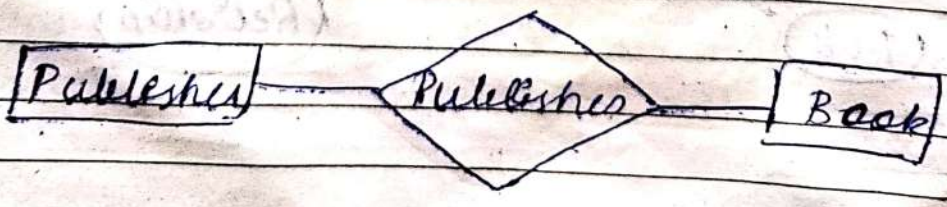

→ Degree of relationship:
Denotes the no. of entity types that participate in a relationship.

1) Unary relationship:
→ Exists when there is association with only one entity.

E-g:

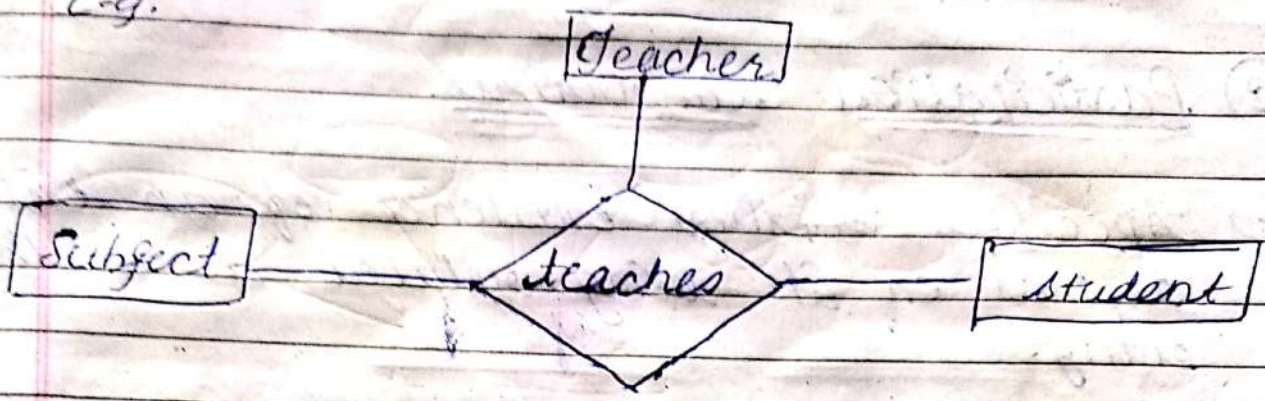



2) Binary rel$^{n}$ship:
Association among 2 entities



3) Ternary relationship:
Rel. among 3 entities.

E.g.

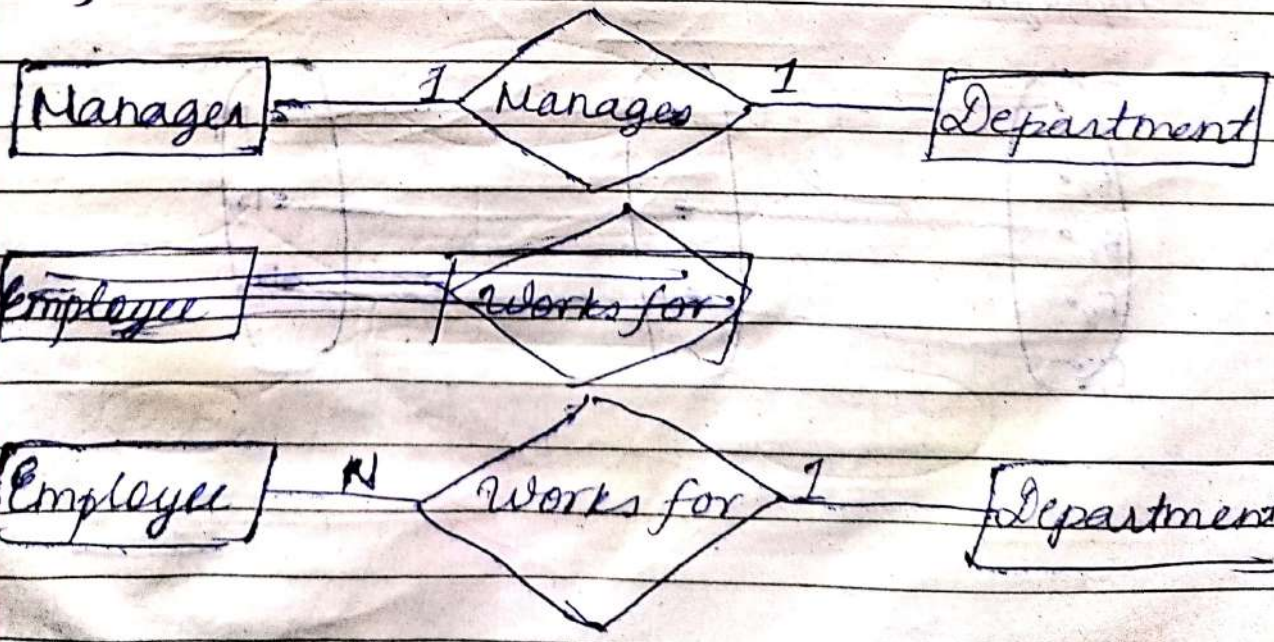


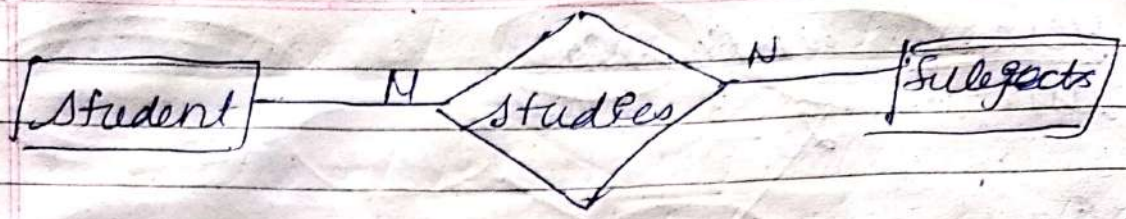

Constraints:

1) Cardinality Ratio:

→ Max. no. of relationship instances that an entity can participate in
→ Possible cardinality ratios for binary relationships: 1:1, 1:N, N:1, M:N

E.g.





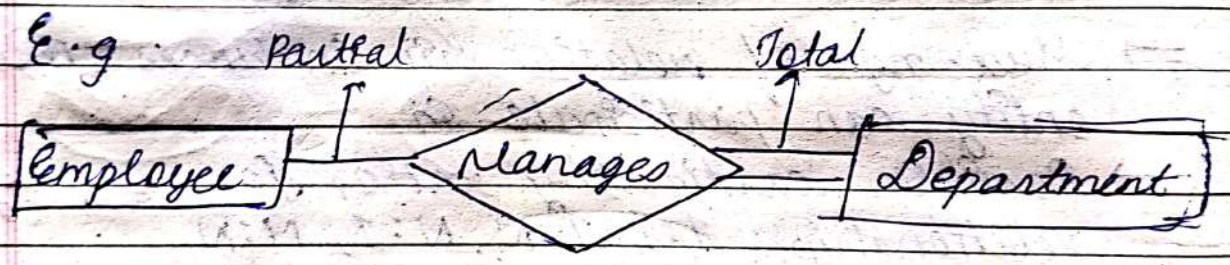

## 2) Participation Constraints

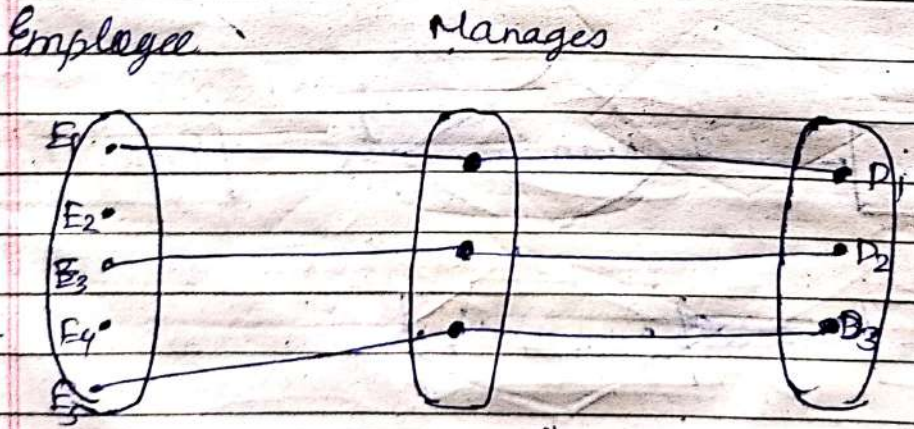
→ Specifies whether existence of an entity depends on its being related to another entity.

→ Types:

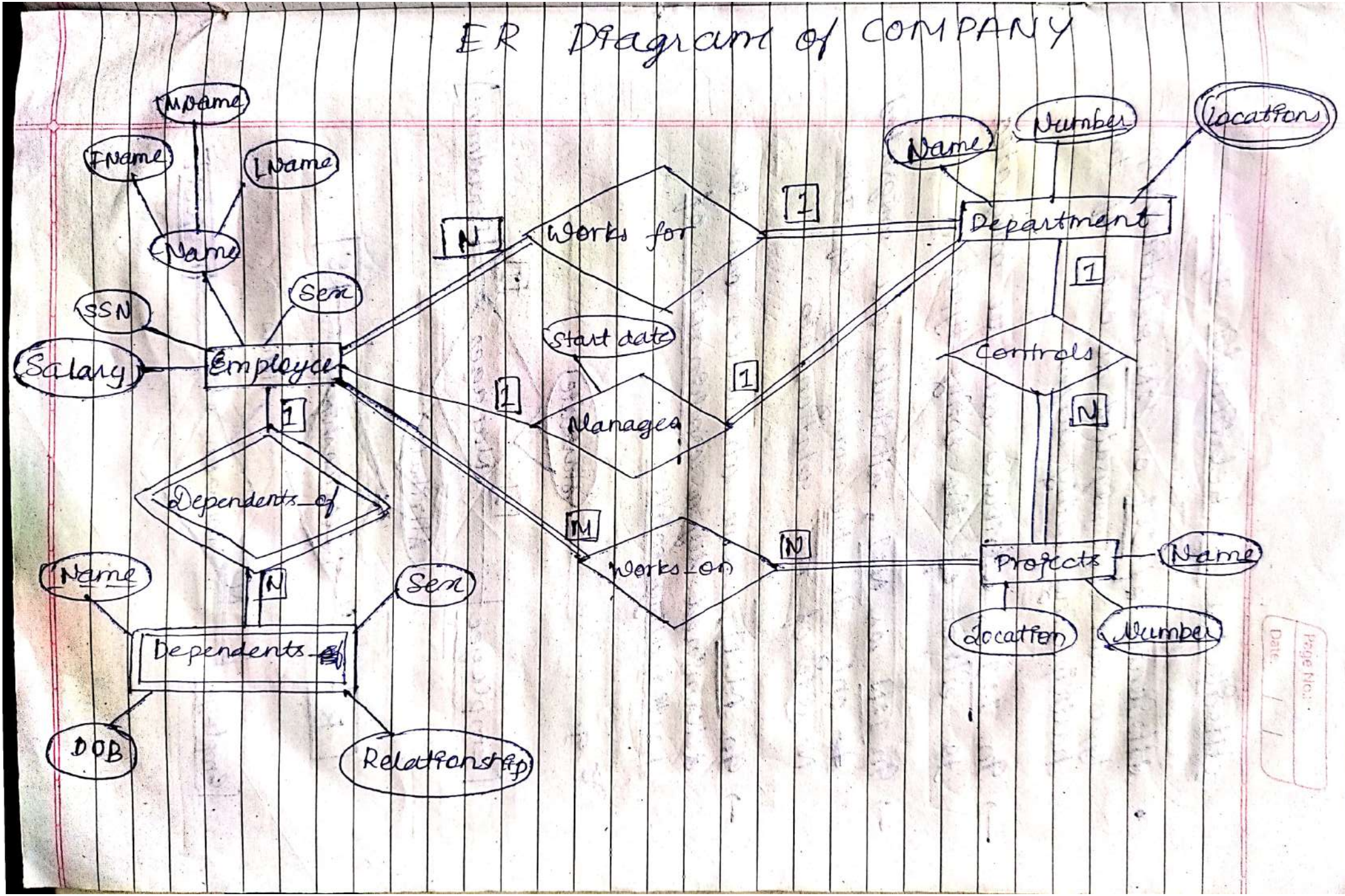
→ Total → Partial

E.g.



# ER Diagram of COMPANY



## Attributes of Relationship types

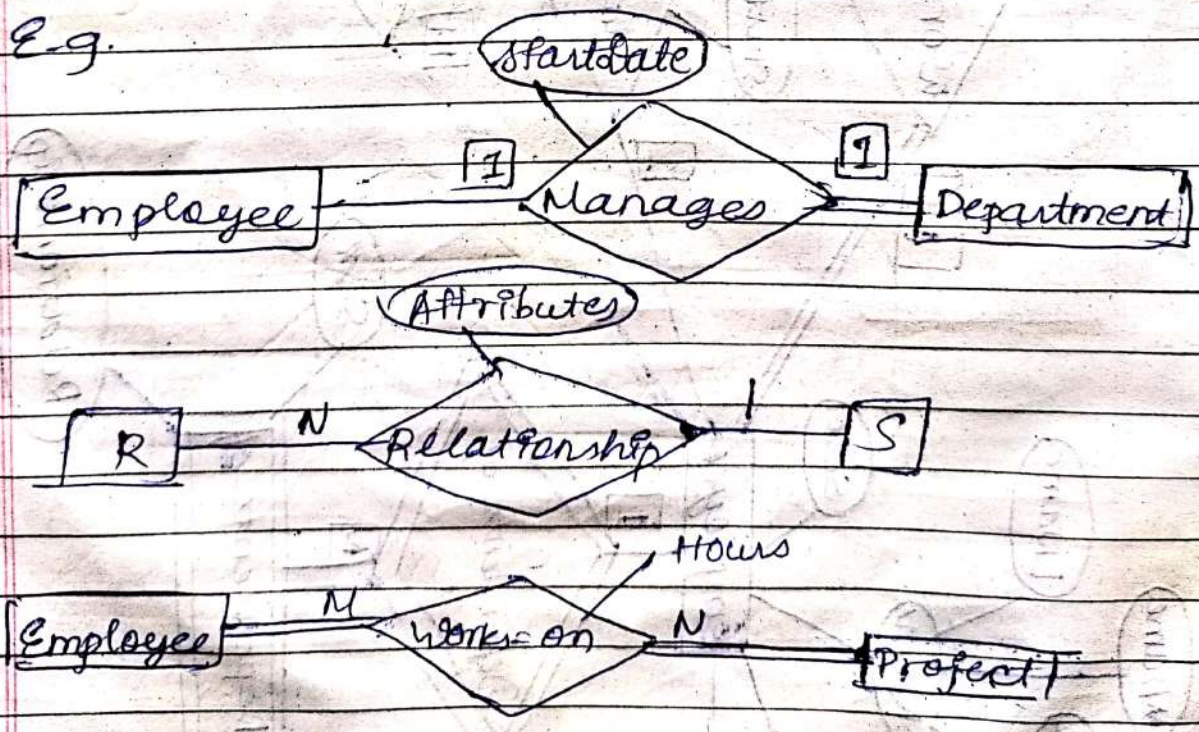
- Attributes of 1:1 or 1:N relationship types can be migrated to one of the participating entity types.

→ In 1:1, attributes can be migrated to either of the entity types.

→ In 1:N or N:1, attributes are migrated only to the entity type of the N-side of the relationship.

→ In M:N, some attributes can be determined by a combination of participating entities.

E.g.



Page No.:
Date. / /

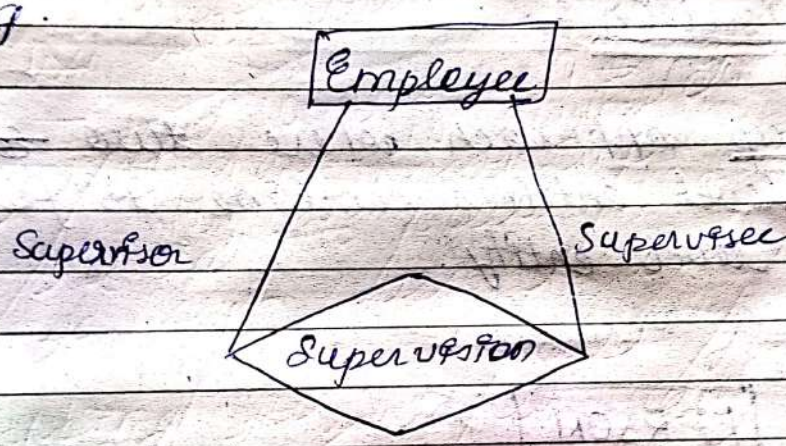
⇒ Role Names

- Signifies the role that a participating entity plays in each relationship instance.

⇒ Recursive relationships

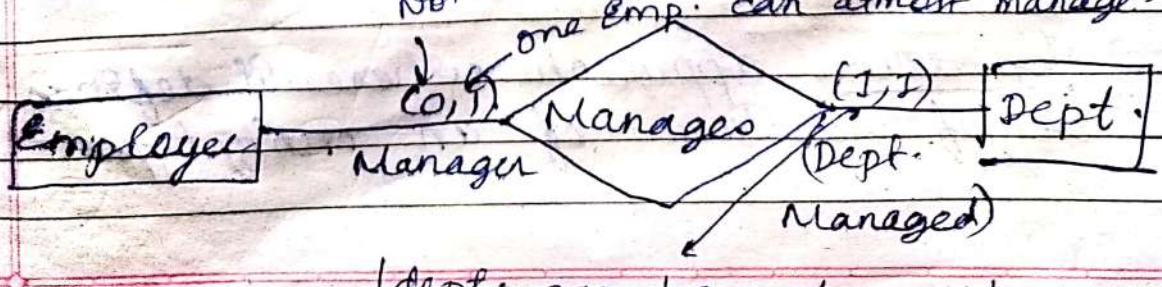
- Same entity type participates more than once in relationship type in different roles.

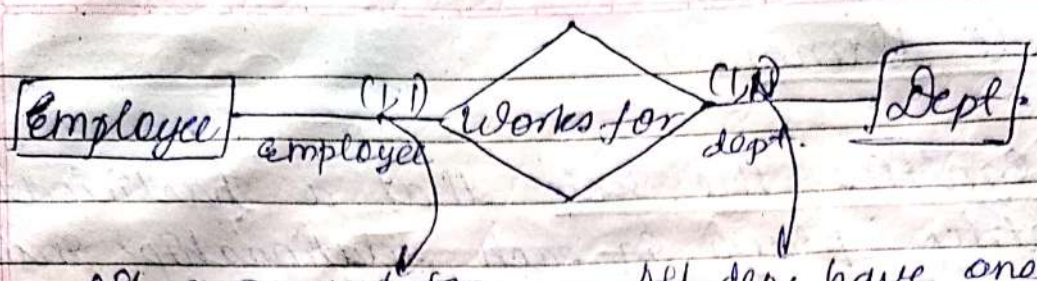
E.g.




# Alternative Notations for ER Diagrams

Associates a pair of integers (min, max) with each participation of an entity type in a relationship type, where $0 \leq min \leq max$ & $max \geq 1$.






All emp. work for one or the other dep.

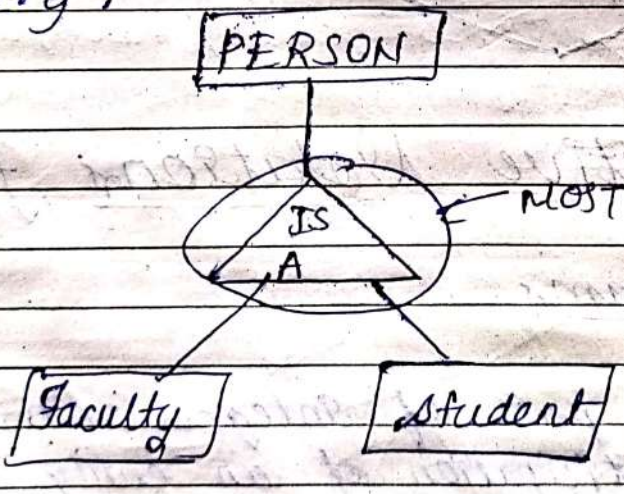
All dep. have one or more emp.

## Enhanced ER Model

### → Generalization:

→ Bottom-up approach where two ~~lower~~ lower level entities combine to form a higher level entity.

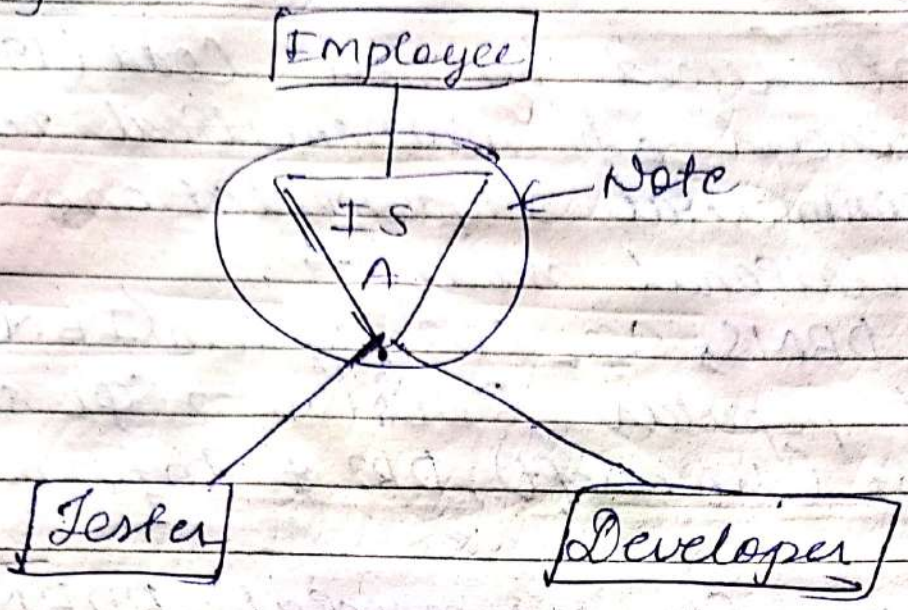
→ E.g.,




### → Specialization

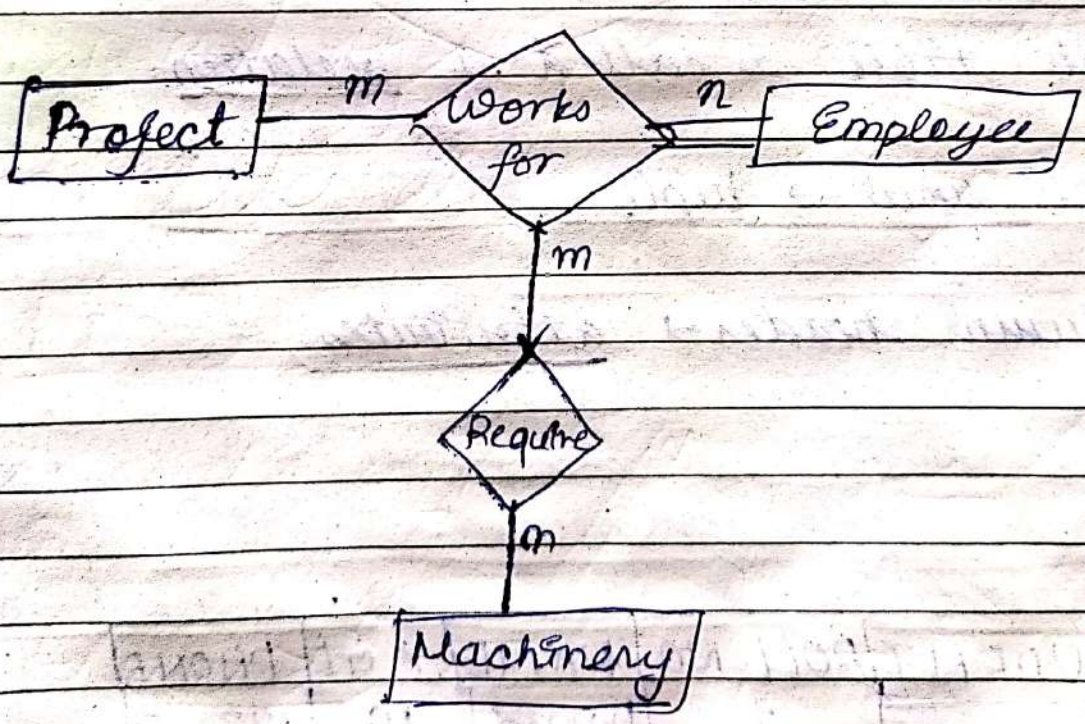
→ Top-down approach where it defines set of subclasses of an entity type.

E-g.




## Aggregation

It's a relationship set of an entity & relationship. Here relationship with corresponding entities is aggregated to a higher level.



# Relational DBMS

History:

→ First introduced by Ted Codd (1970).
→ Uses concept of mathematical relation.
→ First commercial implementations of the relational model → Oracle DBMS, SQL/DS system (IBM).
→ Current popular RDBMSs → SQL server & Access (Microsoft), DB2 & Informix (IBM) etc.
→ Standard for commercial RDBMS → SQL query language.

Terminologies:

→ Relational model represents data as a collection of tables.
→ Each table is called a relation.
→ Each row → tuple
→ Column header → attributes

| STUDENT | ROLL_NO | NAME | AGE | PHONE |
|---|---|---|---|---|
| | 1 | Dawid | 18 | 123... |
| | 2 | Om | 19 | 73046... |

⇒ Domain:

→ A set of atomic values allowed for an attribute

→ E.g. (a) Name: string of characters that represent ~~name~~ name of persons.

(b) Employee age: Possible ages of employees of a company (values betⁿ 20 & 70)

⇒ Relation schema:

→ Describes a relation

→ Made up of relation name R & a list of attributes $A_1, A_2, A_3 \dots A_n$

⇒ Degree (or arity) of a relation:

→ Number of ~~attribut~~ attributes in a relation schema

For. e.g.

STUDENT(Name, Rollno, Age, Add., Phone#, Grade)

No. of attr. is 6 - Degree is 6

⇒ Cardinality

⇒ Total no. of tuples present in a relation.

STUDENT

| Roll-no | Name | Age |
|---|---|---|
| 1 | A | 19 |
| 2 | B | 20 |
| 3 | C | 22 |

Card. = 3

⇒ Relational DB schema

⇒ Set of relation schemas & a set of integrity constants.

⇒ Relation state (relation instance)

⇒ Set of tuples at a given time.

⇒ Ordering of Tuples within a relation:

→ A relation is a set of tuples
→ Tuples in a relation need not have any particular order.

⇒ Ordering of values within a tuple

⇒ An n-tuple → ordered list of n values so ordering of values in a tuple is important.

→ With an alternative def$^n$ of rel$^n$, ordering of values in a tuple is unnecessary.

→ A tuple → set of (<attr.>, <value>) pairs, then ordering of attributes isn't important.

For. eg. t = <(Rollno, 2), (Name, Ben), (Age, 22)>

→ We we generally go with the 1st def$^n$, that ordering is imp.

⇒ Values & Nulls in a Tuple

→ Each value in a tuple is an atomic value

→ Composite attr. are not allowed in RDBMS, Instead they can be represented by their component attributes

✗ Student Rollno Name address
2 Ben Bengaluru, Karnataka 560051

✓ Studen Rollno Name City State Pincode
2 Ben Beng. Kar. 560051

→ Nulls → unknown or not applicable.

Page No.
Date

## Interpretation of a Rel" schema

→ The rel" schema can be represented as a declaration or assertion
→ Each tuple can be interpreted as a fact.

## ⇒ Relational Model Constraints

→ Constraints on databases:

⇒ Inherent model-based: Inherent in the data model
For. e.g. Duplicate ~~values~~ rows not allowed

⇒ Schema based: Defined directly in the schemas of data model
For e.g. Age should be bet" 20 & 60

→ Application based: Must be expressed & enforced by the application programs.

⇒ Domain constraints:

→ Must be an atomic value
→ Performs Data type check:
For. e.g. int, varchar etc.
For. e.g.

| Roll no | Name | Phone | Age | |
|---|---|---|---|---|
| 1 | A | 1---9 | A | violates |

⇒ Key constraint:

An attribute that can uniquely identify each tuple in a relation is called a key. For. e.g. Rollno. of a student

Super key:

→ Superkey specifies that no two tuples can have the same value.

→ Every rel" has ~~at most~~ atleast one superkey → set of all attributes

SK = {Rollno., Email}

{Rollno, Name}, {Rollno, Name}

In short superkey is any set of attr. that helps us identify a tuple uniquely

A key ~~satif~~ satisfies 2 constraints:

(i) Two tuples cant have identical values for ~~any attributes~~ all attributes in the key.

(ii) It is a minimal superkey

## Candidate key

→ A relation schema can have more than one key, then it's called as candidate key.
→ Set of att. that uniquely identify the tuples in a rel".

Candidate keys

| Student | RollNo. | Name | Age | Email |
|---|---|---|---|---|
| | 1 | A | 14 | - - - |
| | 2 | B | 14 | - - - |
| | 3 | C | 15 | - - - |

## Constraints on null values

Specifies whether null values are permitted or not (NOT NULL).

## Entity Integrity constraints

States that no primary key value can be null.

## Referential Integrity Constraint

→ Specified bet" 2 rel".
→ States that a tuple in one rel" that refers to another rel" must refer to an existing tuple in that rel".

for. e.g. Referencing relation

Foreign key

| Student | SID | SName | DNO |
|---|---|---|---|
| | 1001 | A | 3 |
| | 1002 | B | 1 |
| | ~~1003~~ | | |

| Department | DNO | DName |
|---|---|---|
| Referenced rel$^n$ | 1 | OS |
| | 2 | ECE |
| | 3 | CIVIL |

Foreign key:

References to the primary key of other rel$^n$.

Conditions:

(i) Same domain

(ii) Value of FK in a tuple either occurs as a value of PK i.e.

$t_1[FK] = t_2[PK]$ or is NULL.

# Unary Relational Operators:

⇒ The Selection operation:

Syntax:

$$\sigma_{<selection\_condition>}(R)$$

$\sigma$ = selection operation operator

E.g. 1 Select Employee tuples whose dep. no. = 2

| Employee | Name | Dno |
|---|---|---|
| | Tom | 3 |
| | Amy | 2 |
| | Ford | 2 |

Soln: $\sigma_{Dno=2}(Employee)$

o/p:

| Name | Dno |
|---|---|
| Amy | 2 |
| Ford | 2 |

E.g. 2 Select Employee tuples who work in Dno = 3 & with Salary >= 35000 or who work in Dno = 2 & salary > 25000

| Employee | Name | Salary | Dno |
|---|---|---|---|
| | Tom | 30k | 3 |
| | Amy | 20k | 2 |
| | Alice | 40k | 2 |
| | Frank | 38k | 3 |

Soln:

$$\sigma_{(Dno=3 \text{ AND } Salary>35K) \text{ OR } (Dno=2 \text{ AND } Salary>25K)}(Employee)$$

O/P:

| Name | Salary | Dno |
|---|---|---|
| Alice | 40k | 2 |
| Frank | 38k | 3 |

## ⇒ The Profection operation

Syntax: $\Pi_{<attribute\_list>}(R)$

$\Pi$ = Profection operator

E.g. 1) To list the Employee's First Name, Last Name & Salary

| Employee | FName | LName | Salary | Dno |
|---|---|---|---|---|
| | Tom | Ford | 30k | 3 |
| | Amy | Jones | 30k | 2 |

O/p Soln: $\Pi_{FName, LName, Salary}(Employee)$

O/p:

| FName | LName | Salary |
|---|---|---|
| Tom | Ford | 30k |
| Amy | Jones | 30k |

Profection operation removes duplicate tuples

⇒ The Rename operation (ρ)

Doing rename without ρ

| Employee | FName | LName | Salary | Dno |
|---|---|---|---|---|
| | Tom | Ford | 30K | 3 |
| | Amy | Jones | 20K | 2 |
| | Alice | Smith | 40K | 2 |
| | Frank | Smith | 38K | 3 |

$TEMP \leftarrow \sigma_{Dno=3}(Employee)$

| TEMP | FName | LName | Salary | Dno |
|---|---|---|---|---|
| | Tom | Ford | 30K | 3 |
| | Frank | Smith | 38K | 3 |

$R \leftarrow \pi_{FName, LName}(TEMP)$

| R | FName | LName |
|---|---|---|
| | Tom | Ford |
| | Frank | Smith |

$R(FirstName, LastName) \leftarrow \pi_{FName, LName}(TEMP)$

| R | FirstName | LastName |
|---|---|---|
| | Tom | Ford |
| | Frank | Smith |

Syntax:

$\rho_{S(B1, B2 \ldots, BN)}(R)$

or

$\rho_S(R)$

or

$\rho_{(B1, B2, B3 \ldots BN)}(R)$

R = Old Relation
S = New Relation Name
B1, B2 ... BN => New Attr.

## Set Theory:

### Union Operation: (R ∪ S)

⇒ Includes all tuples that are in either R or S or both in R & S.

Student → R

| FN | LN |
|---|---|
| Tom | Ford |
| Amy | Jones |
| Alice | Smith |
| Ernest | Gilbert |

Instructor → S

| FName | LName |
|---|---|
| Amy | Jones |
| John | Brown |
| Alice | Smith |

| FN | LN |
|---|---|
| Tom | Ford |
| Amy | Jones |
| Alice | Smith |
| Ernest | Gilbert |
| John | Brown |

→ R ∪ S

⇒ ~~Int~~

⇒ Intersection operation (R∩S)

- Includes tuples that are in both R & S.

⇒ ~~Minus~~

⇒ ~~Mi~~

⇒ Minus operation (R−S)

Includes all tuples that are in R but not S.

⇒ Cartesian product (R×S)

Combined att. of 2 rel".

| R | A | B |
|---|---|---|
| | | |
| | | |
| | | |

| S | C | D | E |
|---|---|---|---|
| | | | |
| | | | |

R×S

| C | A | B | C | D | E |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |

⇒ The join operation (⋈):

It combines related rows from 2 relations to a single rel" with a given join cond".

Syntax: $R \bowtie_{\langle join\_condition \rangle} S$

| Department | DName | DNo | Mgr_SSN |
|---|---|---|---|
| | Research | 2 | 553 |
| | Finance | 5 | 996 |

| Employee | SSN | FName | LName | DNo |
|---|---|---|---|---|
| | 123 | Alex | Smith | 2 |
| | 553 | Fred | Scott | 5 |
| | 996 | Elsa | David | 5 |
| | 859 | Peter | Williams | 2 |

⇒ • Join operation

~~Dep~~

Dept_Mgr ← DEPARTMENT $\bowtie_{Mgr\_SSN=SSN}$ (Employee)

OR

Temp ← DEPARTMENT X EMPLOYEE

Dept_Mgr ← $\sigma_{(MgrSSN=SSN)}$ (Temp)

Result:

Dept Mgr

| DName | DNo | Mgr_SSN |
|---|---|---|
| Research | 2 | 553 |
| Finance | 5 | 996 |

| SSN | FName | LName | DNo |
|---|---|---|---|
| 553 | Fred | Scott | 2 |
| 996 | Elsa | David | 5 |

⇒ The Theta join (θ):

<join condition> : $A_i \theta B_j$

$A_i$ = attribute of R
$B_j$ = " " of S
$\theta = =, <, <=, >, >=, \neq$

Syntax: $R \bowtie_{A_i \theta B_j} S$

| R | A1 | A2 |
|---|---|---|
| | 20 | 25 |
| | 80 | 40 |

| S | B1 |
|---|---|
| | 50 |
| | 35 |

Operat Operation: $R \bowtie_{A_2 > B_1} S$

Result :

| $A_1$ | $A_2$ | $B_1$ |
|---|---|---|
| 80 | 40 | 35 |

⇒ The Equijoin Operation:

- The only comparison operator is '='.

~~Dept Mgrs~~

For. e.g.

Dept_Mgrs ← DEPARTMENT ⋈ (Employee)
Mgr_SSN=SSN

⇒ The Natural join (~~*~~) (*):

→ It was created to get rid of duplication when it comes to Equijoin
→ Can be performed iff there is a common attribute on bet^n the relations.

| PROJECT | PID | PName | DNum |
|---|---|---|---|
| | 101 | Projectx | 1 |
| | 102 | ProjectY | 2 |
| | 103 | Project Z | ~~3~~ 2 |

| DEPART | DNo | Mgr_SSN |
|---|---|---|
| | 1 | 553 |
| | 2 | 996 |

Proj_Dept ← PROJECT * $\rho_{(DNum, Mgr\_SSN)}$ (Department)

OR

DEPT ← $\rho_{(DNum, Mgr\_SSN)}$ (DEPARTMENT)

Proj_Dept ← PROJECT * ~~Dep~~ DEPT

Proj_Dept

| PID | PName | DNum | Mgr_SSN |
|---|---|---|---|
| 101 | ProX | 1 | 553 |
| 102 | ProY | 2 | 996 |
| 103 | ProZ | 2 | 996 |

Basically, it's a equijoin followed by removal of unnecessary attributes.

⇒ ~~The~~

⇒ The Division Operation:

~~E-g. To retrieve~~

E-g: To retrieve the ~~EID~~ EID of the employees working on all projects.

Employee (R)

| EID (x) | PID (y) |
|---|---|
| 1001 | 1 |
| 1002 | 1 |
| 1003 | 2 |
| 1004 | 2 |

PROJECT (S)

| PID |
|---|
| 1 |
| 2 |

Query Res ← EMPLOYEE ÷ PROJECT

~~Result Res~~ OR

$T1 \leftarrow \Pi_x(R)$

$T2 \leftarrow \Pi_x((T1 \times S) - R)$

$Res \leftarrow T1 - T2$

Result:

| Res |
| --- |
| EID |
| 1002 |

⇒ Aggregate functions (ℱ)

Syntax:

$$\langle \text{grouping\_attributes} \rangle \mathcal{F} \langle \text{function\_list} \rangle (R)$$

⟨grouping attributes⟩ ⇒ list of attributes in R

⟨function list⟩ ⇒ list of (⟨function⟩⟨attribute⟩ pairs

| Employee | FName | SSN | Salary | DNo |
| --- | --- | --- | --- | --- |
| | Ann | 123 | 40K | 2 |
| | Jeremy | 969 | 30K | 2 |
| | Peter | 333 | 30K | 1 |
| | Elsa | 888 | 20K | 2 |

E.g. To retrieve the no. of employees & their avg. salary

$$\mathcal{F}_{COUNT\ SSN,\ AVERAGE\ Salary}(Employee)$$

O/p:

| COUNT_SSN | AVERAGE_Salary |
| --- | --- |
| 4 | 30K |

E.g. To retrieve the no. of employees & their average salary in each department

$$_{DNO}\mathcal{F}_{COUNT_{SSN}, AVERAGE_{Salary}}(Employee)$$

o/p:

| DNO | Count_SSN | Average Salary |
|---|---|---|
| 2 | 3 | 30K |
| 1 | 1 | 30K |

If we want to rename table:

$$\rho_{R(DNo, No\_of\_Employees, Avg\_Sal)}\,{}_{DNo}\mathcal{F}_{COUNT_{SSN}, AVERAGE_{Salary}}(Employee)$$

o/p:

Result

| R | DNO | No_of_Employees | Avg_Sal |
|---|---|---|---|
| | 2 | 3 | 80K |
| | 1 | 1 | 30K |

## Recursive Closure Operations

| Employee | FName | LName | SSN | Super_SSN |
|---|---|---|---|---|
| | Ann | Smith | 1236 | 8885 |
| | Jeremy | Matthew | 9696 | 1236 |
| | Peter | Williams | 3332 | 8885 |
| | Elsa | David | 8885 | null |

To retrieve SSN of all employees directly supervised by Elsa David at level 1)

S-1: Elsa_SSN ← $\Pi_{SSN}(\sigma_{FName='Elsa' \text{ AND } LName='David'}(Employee))$

o/p:

| Elsa_SSN | SSN |
|---|---|
| | 8885 |

S-2: Supervision (SSN1, SSN2) ← $\Pi_{SSN, Super\_SSN}(Employee)$

o/p:

| Supervision | SSN1 | SSN2 |
|---|---|---|
| | 1236 | 8885 |
| | 9696 | 1236 |
| | 3332 | 8885 |
| | 8885 | null |

S-3: Result1(SSN) ← $\Pi_{SSN1}(Supervision \bowtie_{SSN2=SSN} Elsa\_SSN)$

o/p:

| Result 1 | SSN |
|---|---|
| | 1236 |
| | 3332 |

Q2: To retrieve SSN of all Employees supervised by Elsa David at level 2.

Result 2 ← $\Pi_{SSN1}(Supervision \bowtie_{SSN2=SSN} Result1)$

| Result2 | SSN |
|---|---|
| | 9696 |

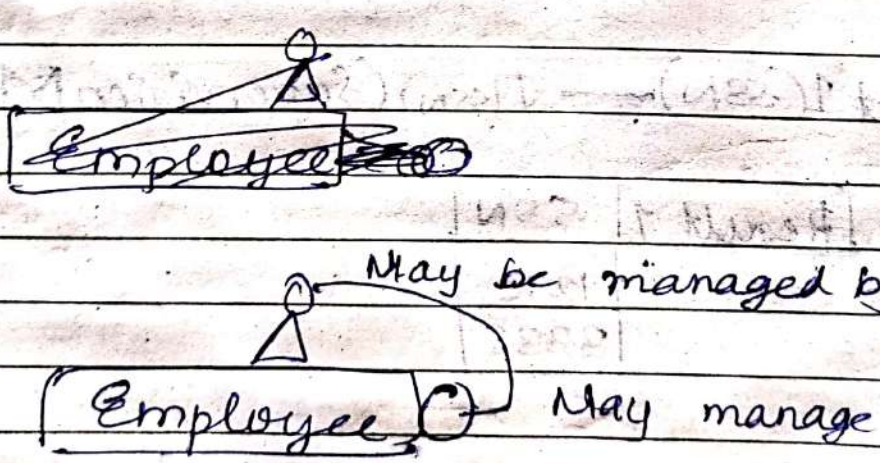
Q3: To retrieve SSN of all employees supervised by 'Elsa David'. Cat level 1 & level 2)

Result ← Result 2 ∪ Result 1

Recursive rel$^n$:
A rel$^n$ship set$^n$ 2 entities of similar entity type is called a recursive rel$^n$. Here the same entity type participates more than once in a rel$^n$ship type with a diff. role for each instance.

Cross: A rel$^n$ship has always more set$^n$ occurence in 2 diff. entities

For e.g.



## Outer Join

A join operation, it displays on the o/p only those combination of tuples that satisfies the given join cond$^n$ & tuples that don't match are also eliminated this leads to loss of info. to overcome this...

Outer join is used when we want all the tuples from rel$^n$ R or all tuples from rel$^n$ S or tuples from both rel$^n$ R & S to be displayed in the result even if tuples don't match this avoids loss of info., in inner join only the matching tuples are displayed

### ⇒ Left Outer Join :-

Keeps all tuples from left rel$^n$ & only matched tuples from rel$^n$ on the right.

Symbol: $R \bowtie_{condition} S$

E.g.

| R | A | B |
|---|---|---|
| | 1 | a |
| | 2 | c |

| S | C | D |
|---|---|---|
| | 1 | b |
| | 3 | d |

Op. = $R_{cs} \leftarrow R \bowtie_{A=C} S$

| Res | A | B | C | D |
|---|---|---|---|---|
| | 1 | a | 1 | b |
| | 2 | c | null | null |

(ii) Right outer join:

Keeps all tuples from right rel$^n$ & only the matched tuples from left rel$^n$.

Syntax: $R \bowtie_{A=C} S$

or

operation: $Res \leftarrow R \bowtie_{A=C} S$

| Res | A | B | C | D |
|---|---|---|---|---|
| | 1 | a | 1 | b |
| | null | null | 3 | d |

(iii) Full Outer join

Keeps all the tuples from both the rel$^n$s & those tuples which don't satisfy the cond$^n$ are filled with null values

Syntax: $Res \leftarrow R \bowtie_{A=C} S$

O/P:

| Res | A | B | C | D |
|---|---|---|---|---|
| | 1 | a | 1 | b |
| | 2 | c | null | null |
| | null | null | 3 | d |

operation

$Res \leftarrow R \bowtie_{A=C} S$

## ⇒ Outer Union Operation:

The 2 rel$^n$s should be union compatible i.e. the 2 rel$^n$s must have same degree or same no. of attr.
Iff. the 2 rel$^n$s are not union compatible or partially compatible

E-g.

| R | A | B |
|---|---|---|
| | 1 | a |
| | 2 | c |

| S | A | D |
|---|---|---|
| | 1 | 10 |
| | 3 | 30 |

R & S have same no. of attr.
R & S both are of deg. 2
~~Doma~~ Data types of B & D don't match, hence they are partially compatible

## ⇒ Outer Union Operation result

| Res | A | B | D |
|---|---|---|---|
| | 1 | a | 10 |
| | 2 | c | null |
| | 3 | null | 30 |

# Distance Vector Routing

Info. about network: Each router is designed to share vector data to each of its nodes throughout the network.

Routing pattern: The data shared by the routers is transmitted only to those nodes with those it has direct link.

Sharing data periodically: The nodes share vector data at regular intervals in the network.

Bellman-ford Algorithm:

$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

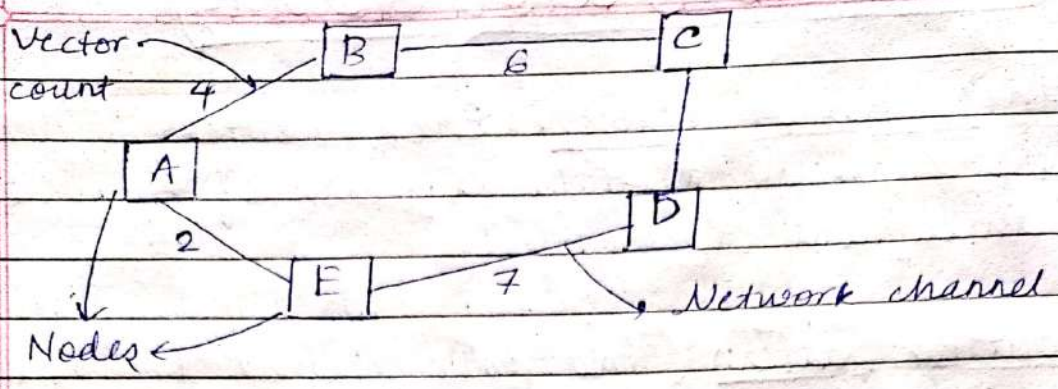
where,

$d_x(y)$ = The least distance from $x$ to $y$

$c(x,v)$ = Node $x$'s cost from each of its neighbours $v$.

$d_v(y)$ = Distance of each neighbour from initial node

$\min_v$ = Selecting the minimum distance for the data packet.

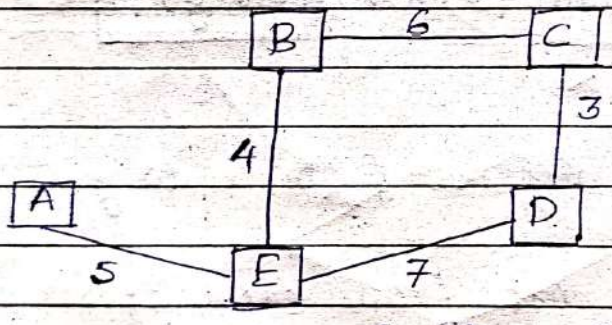



Note: The data shared is in format of a routing table.

Each node shares data with neighbouring nodes & then to the whole n/w.

# Worked Example

(1)




Initial Step

For Node A

| Destination | Vector | Hop |
|---|---|---|
| A | 0 | A |
| B | ∞ | - |
| C | ∞ | - |
| D | ∞ | - |
| E | 5 | E |

For Node E

| Dest. | Vector | Hop |
|---|---|---|
| A | 5 | A |
| B | 4 | B |
| C | ∞ | - |
| D | 7 | D |
| E | 0 | E |

C

| Dest. | Vector | Hop |
|---|---|---|
| A | ∞ | - |
| B | 6 | B |
| C | 0 | C |
| D | 3 | D |
| E | ∞ | - |

B

| D | V | Hop |
|---|---|---|
| A | ∞ | - |
| B | 0 | B |
| C | 6 | C |
| D | ∞ | - |
| E | 4 | E |

D

| D | V | Hop |
|---|---|---|
| A | ∞ | - |
| B | ∞ | - |
| C | 3 | C |
| D | 0 | D |
| E | 7 | E |

Update step

(A)

| D | V | Hop |
|---|---|---|
| A | 0 | A |
| B | 9 | E |
| C | ∞ | - |
| D | 12 | E |
| E | 5 | E |

E

| V |
|---|
| 5 |
| 4 |
| ∞ |
| 7 |
| 0 |

- A to B
- (A,E) + (E,B)
- = 5+4
- = 9

- A to C
- (A,E) + (E-C)
- = 5+∞
- = ∞

- A to D
- (A,E) + (E-D)
- = 5+7
- = 12

- A to E
- = (A,E)
- = 5

(C)

| D | V | Hop |
|---|---|---|
| A | ∞ | - |
| B | 6 | B |
| C | 0 | C |
| D | 3 | D |
| E | 10 | B,D |

(E)

| V |
|---|
| 5 |
| 4 |
| ∞ |
| 7 |
| 4 |

(D)

| V |
|---|
| ∞ |
| ∞ |
| 3 |
| 0 |
| 7 |

(B)

| V |
|---|
| ∞ |
| 0 |
| 6 |
| ∞ |
| 4 |

Likewise continue further

(B) C to E
(C,B) + (B,E)
= 6+4 = 10

(D) C to E
(C,D) + (D-E)
= 3+7 = 10

Steps:

1. For the initial step, use only node distance of immediate neighbours.
2. Nodes without direct contact have a value of infinity
3. For update step, use vector table of immediate neighbours to get new routing table.
4. Continue the update step, for roughly (n-1) iterations for most effecient routing table.

# LINK STATE ROUTING:

Link State Routing Protocol: In this, if each node in the domain has entire topology of the domain i.e. list of nodes & links, How they are connected, type, cost & cond$^n$ of link, then the node can use Dijkstra's algorithm to build routing table.

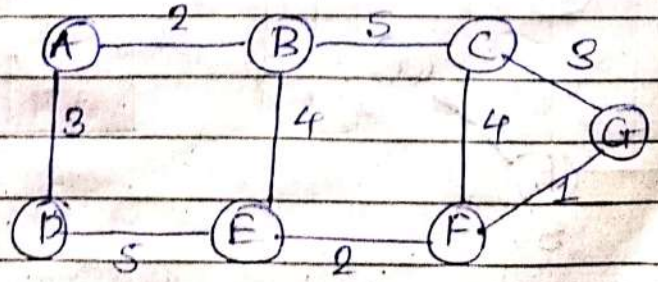
[In this, whole topology can be compiled from partial knowledge of each node]

## Building Routing Table:

Following actions are required to ensure that each node has the routing table:

→ Creation of the states of the links by each node, called Link State Packet (LSP).
→ Dissemination of LSP's to every router, called Flooding.
→ Formation of shortest path tree for each node.
→ Calculation of routing table based on the shortest path tree

E.g.



## Link state packets:

(A)

| Dest | Cost |
|---|---|
| B | 2 |
| D | 3 |

(B)

| Dest. | Cost |
|---|---|
| C | 5 |
| E | 4 |

(C)

| D | C |
|---|---|
| F | 4 |
| G | 3 |

(D)

| D | C |
|---|---|
| A | 3 |
| E | 5 |

(E)

| D | C |
|---|---|
| D | 5 |
| B | 4 |
| F | 2 |

(F)

| D | C |
|---|---|
| E | 2 |
| C | 4 |
| G | 1 |

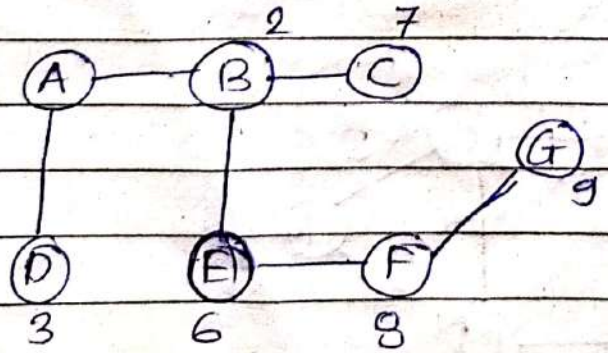
(G)

| D | C |
|---|---|
| C | 3 |
| F | 1 |

## Dijkstra's algorithm:

Source = A

| Sel: V | B | C | D | E | F | G |
|---|---|---|---|---|---|---|
| A | (2) | ∞ | 3 | ∞ | ∞ | ∞ |
| A, B | (2) | 7 | (3) | 6 | ∞ | ∞ |
| A, B, D | (2) | 7 | (3) | (6) | ∞ | ∞ |
| A, B, D, E | (2) | (7) | (3) | (6) | 8 | ∞ |
| A, B, D, E, C | (2) | (7) | (3) | (6) | (8) | 10 |
| A, B, D, E, C, F | (2) | (7) | (3) | (6) | (8) | 9 |



# Cal. of routing table

## For Node A :

| Dest | Cost | Next |
|---|---|---|
| A | 0 | - |
| B | 2 | B |
| C | 7 | B |
| D | 3 | D |
| E | 6 | B |
| F | 8 | B |
| G | 9 | B |

# Transactions

Unit of program execution that acc. & possibly updates various data items.

E.g. Transfer ₹50 from A/c A to B

1) R(A)
2) A : A - 50
3) W(A)
4) R(B)
5) B : B + 50
6) W(B)

→ Two main issues to deal with:

→ Failures like Hardware failures & system crashes
→ Concurrent execution of multiple trans.

## Atomicity

req:
→ If tran. fails after S-3 & before S-6, money will be lost leading to inconsistent DB.
→ Sys. should ensure that updates of a partially ~~exec~~ exec. trans. are not ref. in DB

## Durability:

→ Once trans. is complete (SO is succ. trans.)
→ Updates to the DB must persist even if there are software or hardware failures.

## Consistency:

→ The sum of A & B is unchanged after exec. of transaction
→ Consistency req. include;
  → Explicity spec. integ. const. (primary key, foreign key etc.)
  → Implicit integrity constraints
    e.g. sum of bal. of all acc., minus sum of loan amt. = cash in hand
→ A trans. when starting to exec. must see a cons. DB
→ DB may be temp. inconsistent during exec.
→ DB must be cons. after trans. completes successfully.

## Isolation req.

If let" steps 3 & 6 (of fund transfer), another transaction T2 is allowed to access the partially updated database, it would see an inconsistent DB.

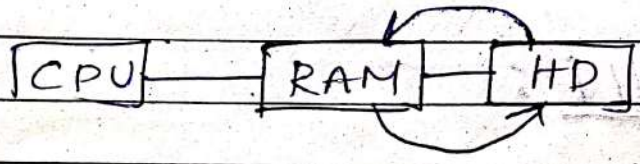
| | T1 | T2 |
|---|---|---|
| 1 | R(A) | |
| 2 | A = A − 50 | |
| 3 | W(A) | |
| | | R(A), R(B), P(A+B) |
| 4 | R(B) | |
| 5 | B = B + 50 | |
| 6 | W(B) | |

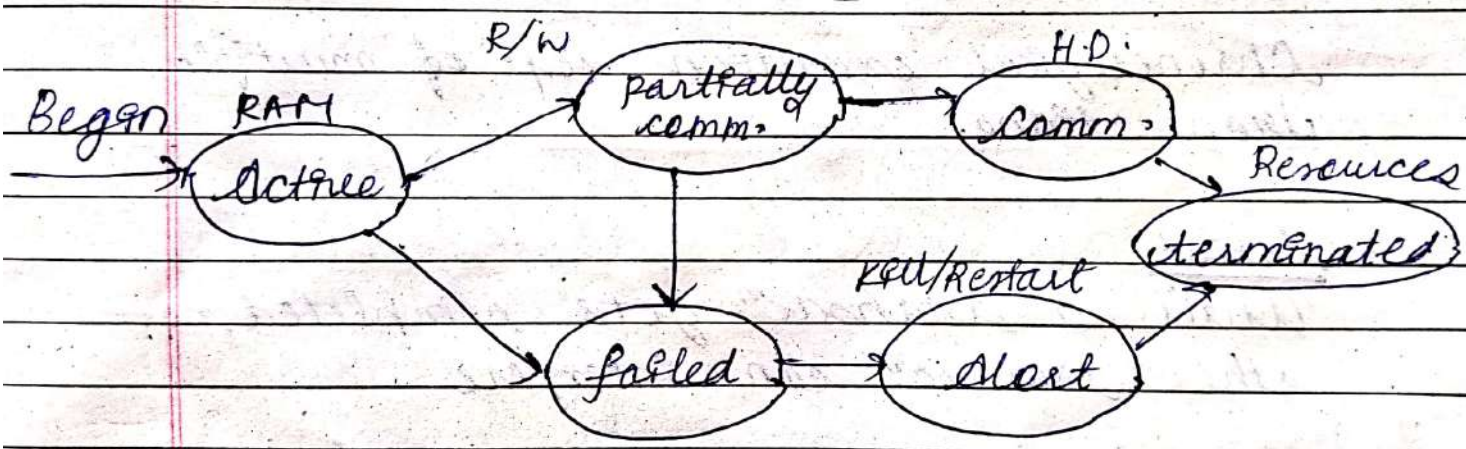
→ Isolation can be ensured trivially by running all tran. concurrently.

# Transaction State

Before start, the transac" is in passive state & as we start executing it comes into active state.

In terms of O.S.:
When we write a prog. in C & save it. Now the prog. is in the H.D. in the ideal state. But when we start executing or compiling it comes into the RAM in active state.




## Partially Comm.

All operations performed except commit.
E.g. If N oper. then (N-1) are performed

All oper. are stored in local ~~m~~ mem./ shared mem. until ~~no~~ now

→ Committed
Changes are now saved to DB.

→ Terminated
Here we deallocate all our resources i.e. free all of our resources.

→ Failed
Power failure, switch damage etc.

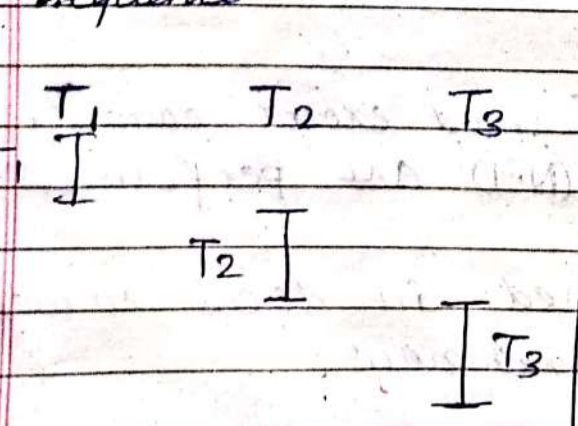
→ ~~Abo~~ Abort
Rollback & Restart the operations.

## SCHEDULE:

Chronological execution seq. of multiple transactions.

Serial:
Until a transac$^n$ gets completed, no other transac$^n$ can interfere.

All transac$^n$ are executed by a serial sequence




Advantage:
→ Consistency

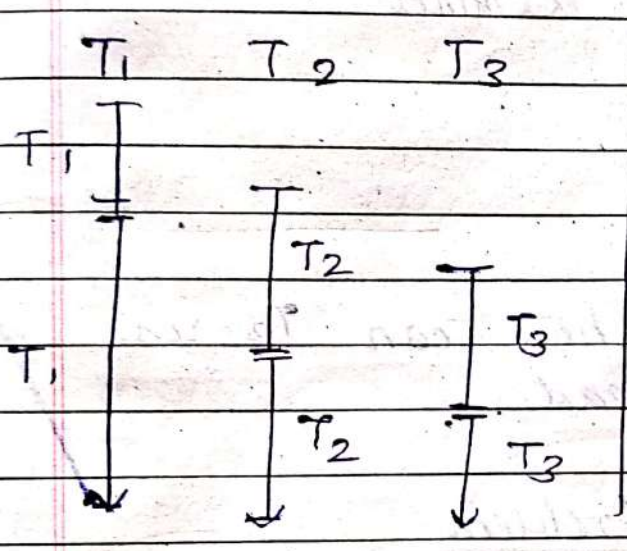
Disadvantage
→ Degraded Performance

## Parallel

We can switch to multiple transactions at a time i.e. multiple transac. can execute at the same time.

E.g Online Banking

Throughput = $\frac{\text{No. of transac. executed}}{\text{Time}}$




Advantage
Inc. Performance

Disadvantage
Inconsistent

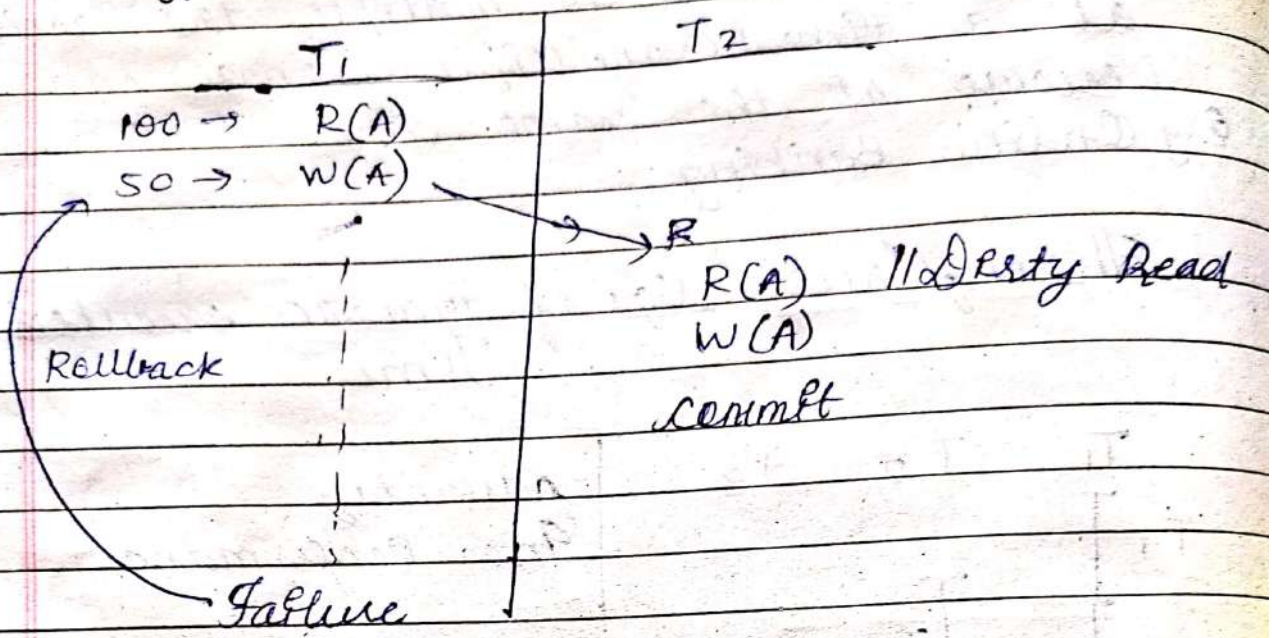
Throughput & performance

## Types of problems on Concurrency:

Concurrency means multiple transac. executed at the same time. i.e. Parallel schedule.

1) Dirty Read
2) Incorrect summary
3) Lost Update
4) Unrepeatable read
5) Phantom read

1) ~~Dirt~~ (Temp. & Update)
1) Dirty Read: Uncommitted read or RAW.




When $T_1$ gets failed how can $T_2$ use A-50, so dirty read.

2) Incorrect Summary Problem

It occurs mostly when a transac. ($T_1$) starts & other transac. ($T_2$) comes & performs the agg. functions.
then,
we get incorrect value of sum, avg. etc.

| $T_1$ | $T_2$ |
|---|---|
| | S=0 |
| | R(A) |
| | Sum+=A |
| R(X) | |
| X-=N | |
| W(X) | |
| | R(X) |
| | Sum+=X |
| | R(Y) |
| | Sum+=Y |

3) Lost Update

An update done to a data item by a transaction is lost as it is overwritten by update done by other transac.

| $T_1$ | $T_2$ |
|---|---|
| R(X) | |
| X+=N | |
| | X+=10 |
| | W(X) |

Here $T_2$ changes the value of X but it will be overwritten by the write commit by $T_1$ on X. ∴ Update done by $T_2$ would be lost. Or short write commit done by last Transac would overwrite all prev. write commits

4) Unrepeatable Read

When two or more read operations of the same transac read diff. values of same variable.

| $T_1$ | $T_2$ |
|---|---|
| R(X) | |
| | R(X) |
| W(X) | |
| | R(X) |

When T2 reads X, a write operation in T1 changes the value of X. ∴ When another read op. is performed by T2, it reads new value of X updated by T1.

## Phantom Read Problem:

When a transaction reads a variable once but when it tries to read it again, an error occurs saying that the variable does not exist.

| T1 | T2 | |
|---|---|---|
| R(X) | ~~R(X)~~ | Once T2 reads X, T1 deletes X without T2's knowledge. |
| | R(X) | ∴ when T2 tries to read it |
| Delete(X) | | again, it is not able to |
| | R(X) | do it. |

## Read-write conflict or Unrepeatable read problem

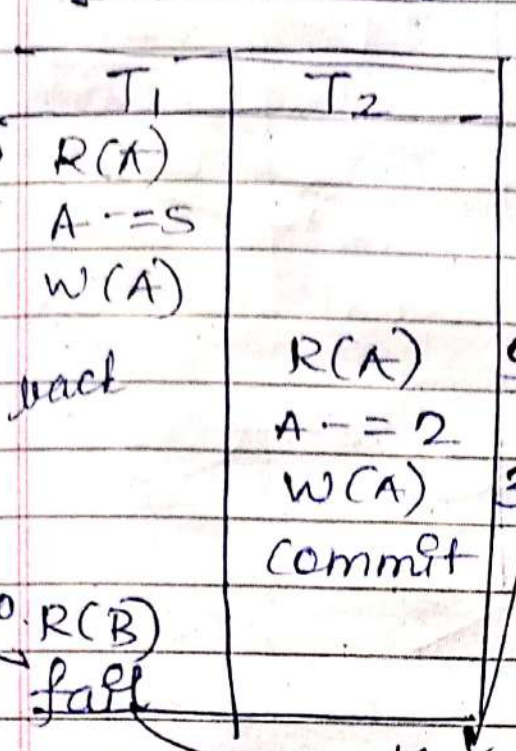
~~R(A)~~

| $T_1$ | $T_2$ | |
|---|---|---|
| R(A) | R(A) | → No conflict |
| R(A) | W(A) | } Conflict |
| W(A) | R(A) | |
| W(A) | W(A) | |

E.g.

| $T_1$ | $T_2$ | |
|---|---|---|
| 10 R(A) | | We issued 2 books but value of A is till 9 |
| ~~W(A)~~ | | |
| 9 A=A-1 | | |
| | R(A) → 10 | |
| | A--=1 | |
| | W(A) 9 | |
| | Commit | |
| W(A) 9 | | |
| Commit | | |

# Irrecoverable vs. Recoverable




After Rollback everything done by T1 is lost & A is 10 again. But T2 also does something which is lost, we can't recover it now hence it's Irrecoverable.

Now T1 rolls back due to atomicity

# Cascading Vs. Cascadeless schedule

1) Recoverable
2) Irrecoverable
3) Cascadeless
4) Cascading
5) Strict Recoverable

## Cascading:

Due to occ. of one event multiple events are occussing automatically.

A = 100

A = 100

| | $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|---|
| 100 | R(A) | | | |
| 50 | A -= 50 | | | |
| 50 | W(A) | 50 | | |
| | | R(A) | 50 | |
| | | | R(A) | 50 |
| Roll back | | | | R(A) |
| | fail | | | |

If $T_1$ fails, it rollp back due to Atomicity & again A = 100. But now in $T_2, T_3, T_4$ (A = 50). So they are working on wrong ~~da~~ data, so $T_2, T_3, T_4$ are forcefully roll-back; this is cascading

Disadvantage

→ CPU utilization by $T_2, T_3$ & $T_4$ gone waste
∴ Performance is lead

## Cascadeless

Here $T_2$ & $T_3$ can't read value of A until it's committed or roll back in T1.

A = 100

| T1 | T2 | T3 |
|---|---|---|
| R(A) | | |
| W(A) | | |
| Commit | R(A) | R(A) |

Don't allow read on T2 & T3 it becomes automatically cascadeless.
There is still W-W problem in cascadeless

A = 100

| | T1 | T2 | |
|---|---|---|---|
| | 100. R(A) | | |
| | 90 A - = 10 | R(A) | 100 |
| | 90 W(A) | A - = 20 | 80 |
| Rollback | | W(A) | 80 |
| | | Commit | |
| | fails | | |

When T1 fails, work of T2 automatically gets lost i.e. W-W problem or lost updation rule.

The value of A i.e. 90 written by T2 won't be reflected anywhere & finally A = 100

Strict recoverable tells us that we also can't write with read.

## Serializability:

It means that a schedule has the ability to become serializable.

S

| $T_1$ | $T_2$ |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| | W(A) |

S'

| T1 | T2 |
|---|---|
| | R(A) |
| | W(A) |
| R(A) | |
| W(A) | |

$T_1 \rightarrow T_2$ $\quad$ $T_1 \leftarrow T_2$

Serializability

Conflict $\quad$ View

# Conflict Equivalent Sch.

| | | |
|---|---|---|
| R(A) | R(A) | Non-conflict |
| R(A) | W(A) | Conflict pairs |
| W(A) | R(A) | |
| W(A) | W(A) | |

How to convert:
If we have adjacent non-conflict pair, then swap their pos$^n$.

| $T_1$ | $T_2$ |
|---|---|
| | R(A) |
| R(B) | |

| $T_1$ | $T_2$ |
|---|---|
| R(B) | |
| | R(A) |

Q: To check Conflict Equivalent

S

| $T_1$ | $T_2$ |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| | W(A) |
| R(B) | |
| | |

S'

| $T_1$ | $T_2$ |
|---|---|
| R(A) | |
| W(A) | |
| R(B) | |
| | R(A) |
| | W(A) |
| | |

Soln. On S we have adjacent non-conflict pairs so swap them

S

| $T_1$ | $T_2$ |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| | W(A) |
| R(B) | |

S

| $T_1$ | $T_2$ |
|---|---|
| R(A) | |
| W(A) | |
| R(B) | R(A) |
| | W(A) |

S

| $T_1$ | $T_2$ |
|---|---|
| R(A) | |
| W(A) | |
| R(B) | |
| | R(A) |
| | W(A) |

Hence $S \equiv S'$

∴ S & S' are conflict Equivalent

Note:

$S \xrightarrow{CE} S' \longrightarrow$ Serializable (Serial schedule)

CE: Conflict Equivalent

# Precedence Graph:

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| R(x) | | |
| | | R(y) |
| | | R(x) |
| | R(y) | |
| | R(z) | |
| | | W(y) |
| | W(z) | |
| R(z) | | |
| W(x) | | |
| W(z) | | |

~~S-1~~ Make Precedence Graph

S-1 Makes vertices as the transacⁿ name.
Eg. $T_1, T_2, T_3$

S-2: Check the conflict pairs on other tran. & draw edges
Like ~~for a~~ $\forall T_1$ check $T_2$ & $T_3$

Start:

Start with first operation:

In $T_1 \rightarrow R(x)$

& check conflict pairs of $R(x)$ in $T_2$ & $T_3$

Now do check for every operations in all transactions & after checking out the operations & if you find any conflict pair, then draw edge acc. to order (Transac$^n$)

| | $T_1$ | $T_2$ | $T_3$ |
|---|---|---|---|
| 1 | R(x) | | |
| 2 | | | R(y) |
| 3 | | | R(x) |
| 4 | | R(y) | |
| 5 | | R(2) | |
| 6 | | | W(y) |
| 7 | | W(2) | |
| 8 | R(2) | | |
| 9 | W(x) | | |
| 10 | W(2) | | |

~~R(x)~~ 3

3) $R(x) - W(x)$ $(T_3 \rightarrow T_1)$

4) $R(y) - W(y)$ $(T_2 \rightarrow T_3)$

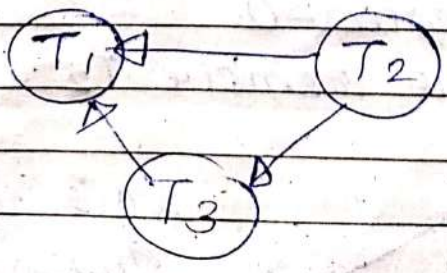
5) $R(2) - W(2)$ $(T_2 \rightarrow T_1)$

~~7~~) $W(2) - R(2)$ $(T_2 \rightarrow T_1)$

~~7~~) $W(2) - W(2)$ $(T_2 \rightarrow T_1)$

Don't draw edge for duplicates

Precedence graph:



Now come on Graph

Check that if there is any loop

Check for loop / cycle

If,

Loop / Cycle → Not Conflict Serializable

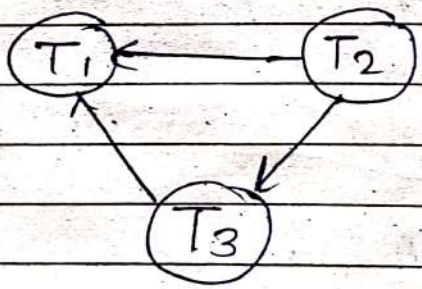
No " " → If " " "

Conflict Serializable → Serializable

Consistent

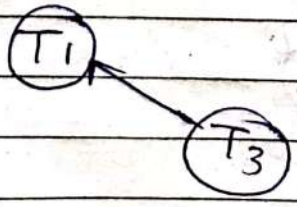
Now we need to know the sequence of execution

→ In the graph check the vertex having Indegree = 0




$T_2$ has Indegree = 0

Now remove $T_2$ ∴ $T_2$ is the first




Now we can see Indegree ($T_3$) = 0 ∴ $T_3$ is next

∴ Order: $T_2 \rightarrow T_3 \rightarrow T_1$

# View Serializability:

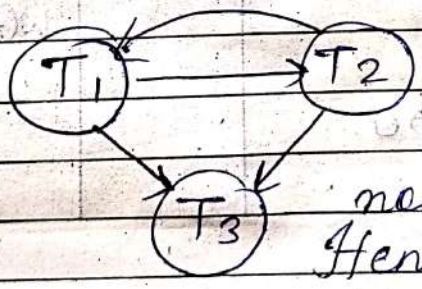
→ Check whether schedule is conflict serializable or not?
→ Make precedence graph

If there is loop/cycle the conflict serializable is not answerable, then view serializability answers

S

| | $T_1$ | $T_2$ | $T_3$ |
|---|---|---|---|
| 1) | R(A) | | |
| 2) | | W(A) | |
| 3) | W(A) | | |
| 4) | | | W(A) |

1) R(A) – W(A) $[T_1 \rightarrow T_2]$
2) W(A) – W(A) $[T_2 \rightarrow T_1]$
3) W(A) – W(A) $[T_1 \rightarrow T_3]$
2) W(A) – W(A) $[T_2 \rightarrow T_3]$




Here, we get a loop, ∴ it's not conflict serializable
Hence, we can't tell whether it's serializable or not.

Now, we have to check

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| R(A) | | |
| | W(A) | |
| W(A) | | |
| | | W(A) |

We change pos$^n$ →

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| R(A) | | |
| W(A) | | |
| | W(A) | |
| | | W(A) |

Now this is a serial schedule:

$T_1 \rightarrow T_2 \rightarrow T_3$

Now, we have to check whether there final result match each other or not.

A = 100

| | $T_1$ | $T_2$ | $T_3$ |
|---|---|---|---|
| 100 — | R(A) | | |
| | | A -=40 ~~-60~~ | |
| | | W(A) → 60 | |
| | A -=40 | | |
| 20 ← | W(A) | | |
| | | | A -=20 |
| | | | W(A) → 0 |

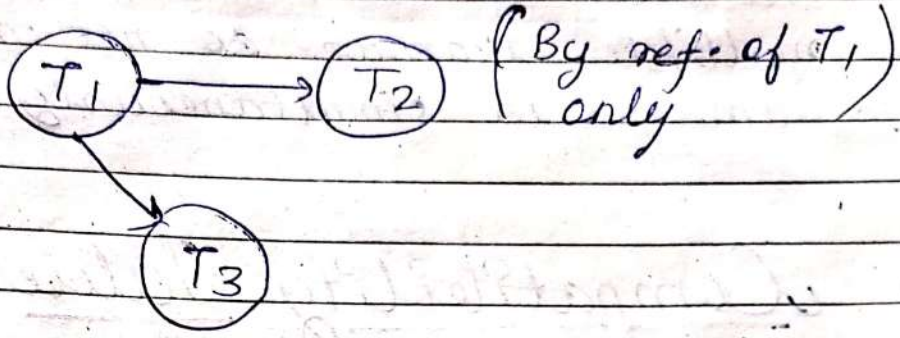
| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| R(A) | | |
| A -=40 | | |
| 60 ← W(A) | | |
| | A -=40 | |
| | W(A) → 20 | |
| | | A -=20 |
| | | W(A) → 0 |

We can see that at both places final value of A is 0.

∴ They both are view equivalent (≡)

Page No.:
Date. / /

Note: Here, finally o/p is given by A of $T_3$.
So if we adjust the pos$^n$ of A of $T_1$ & $T_2$ so there is no problem.




Hence $T_1 \rightarrow T_2 \rightarrow T_3$

# Concurrency Control Protocols: (C.C.P.)

C.C.P. is that how to make them serializable, or how to make them recoverable schedules are concurrent, but how we can make them serializable & recoverable, this comes under C.C.P.

## 1) Shared-Lock:

→ Read-only
→ It can be shared bet$^n$ the transac. because when the transac. holds a lock, then it can't update the data on the data item.

2) Exclusive lock:

→ Both Read & Write
→ The lock is exclusive, & on this lock, multiple transac. do not modify the same data simultaneously

## Compatibility Table:

| | S | X |
|---|---|---|
| S | Yes | No |
| X | No | No |

Eg: S has only R(A)
X has both R(A), W(A)

⇒ S-S (No conflict) ∵ R(A)-R(A)

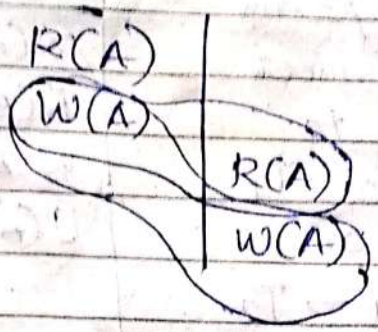
⇒ S-X (Conflict)

| R(A) | R(A) |
|---|---|
| | W(A) |

⇒ X-S (Conflict)

| R(A) | |
|---|---|
| W(A) | |
| | R(A) |

⇒ X-X




⇒ Problems on S/X locking:

→ May not be sufficient to produce only Serializable schedule.

→ May not be free from irrecoverability

→ May not be free from dead-lock

→ May not be free from saturation

## Drawbacks on S/X

⇒ Locking gives us consistent schedule. (Consistent)

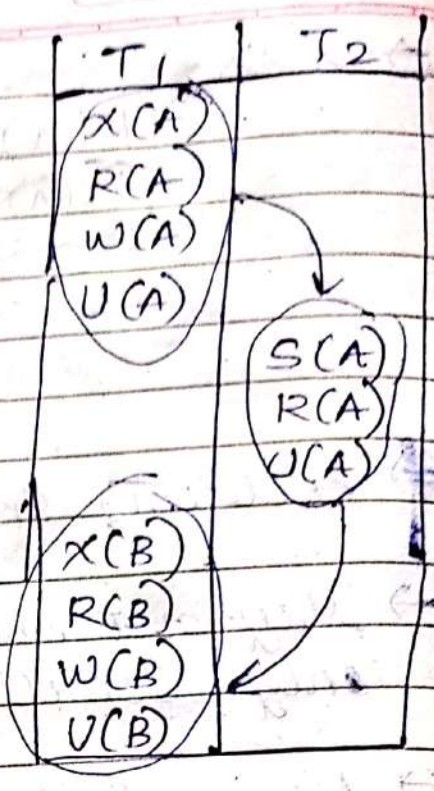
⇒ May not be sufficient to produce only serializable schedule

Eg:

E.g.

| $T_1$ | $T_2$ |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| R(B) | |
| W(B) | |

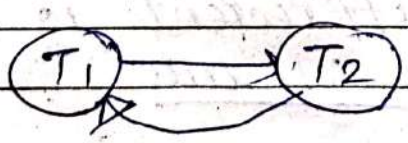
May do conflict to $T_2 \rightarrow T_1$ or $T_1 \rightarrow T_2$



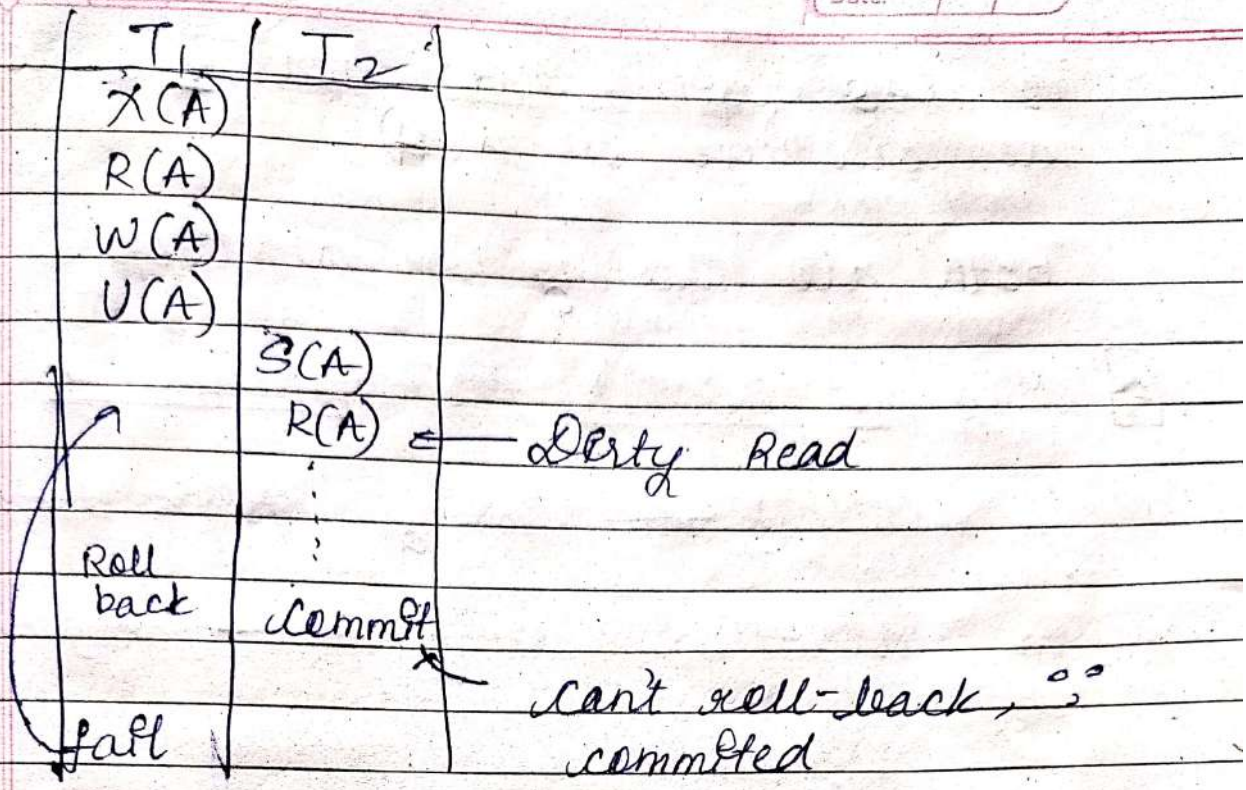

Note: Until, we do unlock, U(A) on $T_1$, we don't be able to put shared lock S(A) on $T_2$. Here by Compatibility Table.

(X-S) → No, So first unlock X(A) & then S(A) on $T_2$ data

1) ⇒ Here we don't get Serializable schedule even after applying S/X locking. As you can see on table.




2) May not be free from Irrecoverability or recoverability




3) May not be free from deadlock

Deadlock: When 2 persons wait for resources & they both are waiting on an ∞ loop. So, when we wait infinitely, it is called Deadlock State.

E.g.

| | $T_1$ | $T_2$ |
|---|---|---|
| G | X(A) | |
| | ↓ | X(B) G |
| W | X(B) | ↓ |
| | | X(A) W |

Here both are in waiting state, ∵ they aren't unlocked after use, so $T_1$ also waits that when $T_2$ unlocks he can use on X(B)

$T_2$ also waits that when $T_1$ unlocks, he can use on $x(A)$.

Both are waiting in ∞ loop.

4. May not be free from starvation

In deadlock, wait waiting upto ∞ time limit,
In starvation, waiting ~~not~~ for finite time.

~~Shared~~

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| | S(A) | | |
| X(A) | | S(A) | |
| | U(A) | | S(A) |
| Waiting time | | U(A) | |
| | | | U(A) |

S → Shared
X → Exclusive
U → Unlock

As here, $T_1$ waits for X(A) until $T_4$ unlocks, ~~so here starvation also~~. So $T_1$ starves

# 2 phase locking Protocol (2PL)

Transac: Concurrency Control:

2PL is extension of simple S/X locking.

(#) 2PL →

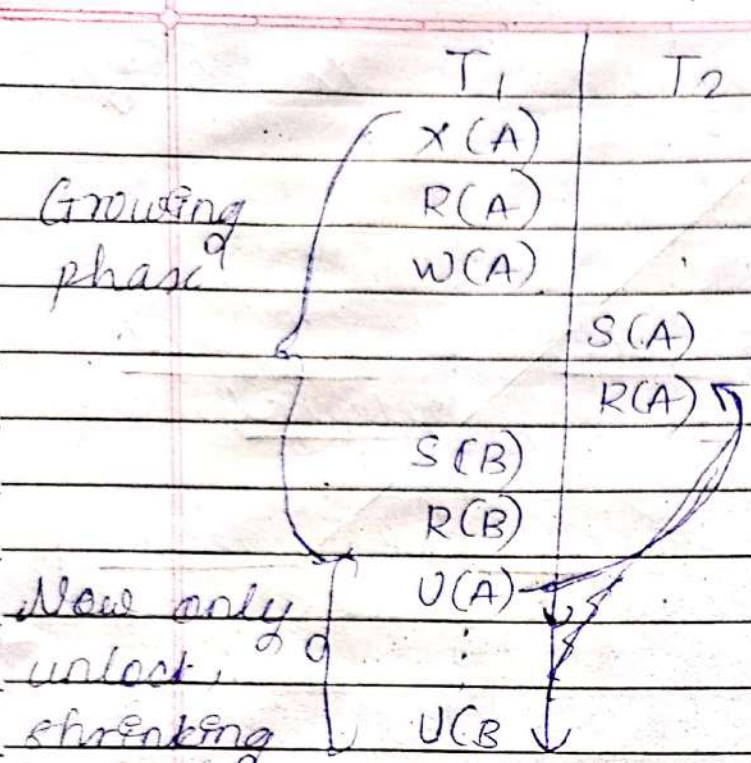
→ Growing phase: Locks are acquired & no locks are released.

→ Shrinking phase: Locks are released, & no locks acquired.

| $T_1$ | | |
|---|---|---|
| X(A) | Growing phase | Note: When growing phase starts, then only apply locks & When we first unlock, i.e. shrinking phase starts, then we only unlock (and not able to apply any lock) |
| S(B) | | |
| R(A) | | |
| W(A) | | |
| R(B) | | |
| S(A) | | |
| R(C) | | |
| S(D) | | |
| R(D) | | |
| U(A) | Shrinking phase | |
| ⋮ | | |

⇒ We achieve serializability by this, that we don't achieve in simple S/X locking

E.g.




Hence, we first start $T_1$, & if $T_2$ comes in b/w then we don't entertain them. First, we complete $T_1$ & then goes to $T_2$.

$$T_1 \longrightarrow T_2$$

Serializability achieved → So consistent

Note: Transacⁿ following 2PL is always serializable

E.g. While $T_1$ is in growing phase by using S(A) (shared lock), then at the same time $T_2$ also starts it's growing phase using S(A), by compatibility table.

| $T_1$ | $T_2$ |
|---|---|
| S(A) | |
| | S(A) |
| X(B) | |
| (*) | |
| U(A) | |
| | X(B) |
| U(B) | (*) → Lock pt. |
| | U(A) |
| | U(B) |

Growing
Lock pt.
Shrinking
Growing & Shrinking

Lock point:
Where transaction takes the last lock or where transaction is unlocked for the first time

The transaction whose lock pt. comes first first, comes first.

$T_1 \rightarrow T_2$

## Drawbacks of 2PL:

Advantage: always ensures serializability.

Drawbacks:

1) Not free from irrecoverability

| $T_1$ | $T_2$ |
|---|---|
| X(A) | |
| R(A) | |
| W(A) | |
| U(A) | |
| . | S(A) |
| . | R(A) |
| . | Commit |
| fail | |

Roll-back

We can't rollback ∵ it's already committed, Hence irrecoverable ×

2) Not free from cascading Rollback:

| E.g. $T_1$ | $T_2$ | $T_3$ | $T_4$ | |
|---|---|---|---|---|
| X(A) | | | | |
| R(A) | | | | |
| W(A) | | | | |
| U(A) | | | | |
| | S(A) | | | Dirty read |
| | R(A) | | | |
| | | S(A) | | Bad performance |
| | | R(A) | | |
| fail | | | S(A) | |
| | | | R(A) | |

∵ we don't commit here roll back is possible

(Roll back them all)

3) ~~Bad~~ Not free from ~~Deadlo~~ Deadlock:

| | $T_1$ | $T_2$ | G - Grant W → Wait |
|---|---|---|---|
| G ↓ | X(A) | | |
| | | X(B) | G ↓ ∴ ∞ waiting time here |
| W | X(B) | | ($T_1$ & $T_2$ both are waiting here to complete their trans |
| | | X(A) | W |

4) Not free from starvation:

(Wait for limited time)

# Strict 2PL, Rigorous 2PL & Conservative 2PL:

# Strict 2PL: It should satisfy basic 2PL & all exclusive locks should hold until commit/abort.

# Rigorous 2PL: Should satisfy basic 2PL & all S/X locks should hold until ~~not~~ commit/abort.

E.g.:

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| X(A) | | |
| R(A) | | |
| W(A) | | |
| | S(A) | |
| U(A) | R(A) | |
| : | | S(A) |
| : | | R(A) |
| : | | |
| fails | | |

Roll-back

We have to unlock Exclusive lock after commit.
then, the problem of cascading removes.

| $T_1$ | $T_2$ |
|---|---|
| X(A) | |
| R(A) | |
| W(A) | |
| : | |
| | S(A) |
| | R(A) |
| : | |
| Commit | |
| U(A) | |

∴, after commit on $T_1$, the $T_2$ takes data from DB & not from W(A) of $T_1$
So, no dirty read

Hence, here, Cascading Rollback is removed & it will always produce Cascadeless

Now, Irrecoverability :-

| $T_1$ | $T_2$ |
|---|---|
| X(A) | |
| R(A) | |
| W(A) | |
| | S(A) |
| U(A) | R(A) |
| | Commit |
| Commit | |

Roll back →

(But we can't roll-back it, it is already committed)

We unlock it after commit is done.
Hence, $T_2$ won't S/X get S/X lock until $T_1$ isn't committed.

Hence Ir-recoverability is removed here.
Hence DB would read from HD.
It produces strict recoverable schedule

Page No.:
Date. / /

Note: 2 problems removed here:
1) Strict Recoverable
2) Cascadeless

Regarding 2PL:

We can't release shared lock also in this.

But problem of dead-lock & starvation are also there.

## Conservative 2PL:

→ Not possible practically.
→ It says that any transaction should take all locks before starting.

→ If $T_1$ starts first, $T_2$ & $T_3$ won't get any locks, so problem of deadlock is removed here.

→ Basically, Grant all locks to $T_1$ so that $T_2$ gets nothing.

| | $T_1$ | $T_2$ | |
|---|---|---|---|
| G → | X(A) | | |
| | | X(B) | ← W |
| G. → | X(B) | | |
| | | X(A) | ← W |

$T_1$ completed here first, so no deadlock & for waiting.
No deadlock here.

# Basic Time Stamp Ordering Protocol:

→ Assign unique value to every transac$^n$
→ Tells the order (when they enter in sys.)

| 10:00 | 10:10 | 10:15 | → Time |
|---|---|---|---|
| $T_1$ | $T_2$ | $T_3$ | |
| 100 | 200 | 300 | → Time Stamp |

Older ↙ Younger ↙ Youngest

→ Read T.S (RTS): last transac$^n$ ng. which performed Read successfully

→ Write_TS(WTS): last transac$^n$ ng. which performed write ~~or~~ successfully

e.g.

| 10 | 20 | 30 | TS($T_i$) |
|---|---|---|---|
| $T_1$ | $T_2$ | $T_3$ | |
| R(A) | | | RTS(A) = ~~20~~ 30 |
| | R(A) | | i.e. : last read & of ~~30~~ $T_3$ & |
| | | R(A | T.S. of $T_3$ is 30 |

| 10 | 20 | 30 |
|---|---|---|
| $T_1$ | $T_2$ | $T_3$ |
| W(A) | | |
| | | W(A) |
| | W(A) | |

WTS $\boxed{WTS(A) = 20}$

# Rules:

In T.S.O.P, the transac$^n$ which comes first, finishes first.

$\boxed{\text{Serializability} : \text{Older} \rightarrow \text{Younger}}$

follows conflict serializability

Transac$^n$ $T_i$ issues a Read (A) Operation

→ If $WTS(A) > TS(T_i)$, Rollback $T_i$

→ Otherwise Execute R(A) operation,

Set $RTS(A) = \text{Max}\{RTS(A), TS(T_i)\}$

Transac$^n$ $T_i$ issues a Write Operation

→ If $RTS(A) > TS(T_i)$ then Roll back $T_i$

→ If $WTS(A) > TS(T_i)$ then Roll back $T_i$

→ Otherwise execute write(A) operation:

Set $WTS(A) = TS(T_i)$

Understanding these

| e.g. (old) (100) | (200) (young) |
|---|---|
| $T_1$ | $T_2$ |
| R(A) | |
| | W(A) |

| $T_1$ | $T_2$ |
|---|---|
| W(A) | |
| | R(A) |

| $T_1$ | $T_2$ |
|---|---|
| W(A) | |
| | W(A) |

(NP)

(No conflict)

∵ (old → young) or ($T_1 \rightarrow T_2$). So, ∵ $T_1$ executes first there's no problem in $T_2$

e.g. (old) (100) $T_1$ | (200) $T_2$ (young)

$T_2$: R(A) ← 10

$T_1$: 20 ← W(A); ... Commit

Roll back

$T_2$: ✗

(Case-I)

∵, now in $T_2$, A-10 has read ~~which~~ which value, ∵ $T_1$ has updated 20 in DB

($T_2$ won't let $T_1$ happen) ($T_1$ shouldn't happen)

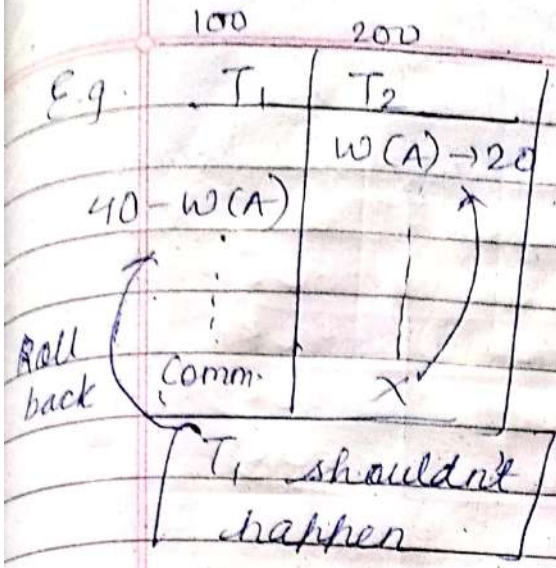
| $T_1$ | $T_2$ | |
|---|---|---|
| | W(A) | -20 |
| 20 → R(A) | | Wrong read |
| ... | ... | |
| Commit | ✗ | |

Roll back

(Case-II)

20 is nowhere. i.e. $T_1$ has read wrong data

($T_1$ shouldn't read)




(Case - III)

If $T_2$ fails, then our updation will be lost.

∴ Lost Updation

# Ques. on TS-Ordering:

Rules:

1) 1st Rule Eg. of read → has only 1 cond$^n$
∴ only R-W conflict

2) 2nd Rule of write → has cond$^n$s
$\begin{pmatrix} W-R \\ W-W \end{pmatrix}$ → 2 conflicts

| $T_1$ (100) | $T_2$ (200) | $T_3$ (300) | |
|---|---|---|---|
| R(A) | | | 0 > 100 |
| | R(B) | | 0 > 200 |
| W(C) | | | 0 > 100 |
| | | R(B) | 0 > 100 |
| R(C) | | | 0 > 300 |
| | W(B) | | 100 > 100 |
| | | W(A) | 300 > 200 |
| | Roll back | | ↳ Roll back |

| | A | B | C |
|---|---|---|---|
| | | 200 300 | 100 |
| RTS | 100 Ø | Ø | Ø |
| | 300 | | 100. |
| WTS | Ø | 0 | Ø |

⇒ Initially all values are zero

⇒ Check, for every value on the table & put that values in Rule's & then make Table.

→ For R(A) check 1st rule of Read
→ For W(B) use 2nd rule of Write

⇒ first check, R(A) of $T_1$, then
R(B) of $T_2$
W(C) of $T_1$
R(B) of $T_3$
R(C) of $T_1$
Roll-back here ← W(B) of $T_2$
W(A) of $T_3$
then make final table of RTS & WTS

OR

~~We also~~
We can also do the ques$^n$ directly, ~~with~~ without using these two rules with just the older → youngest concept & cases.

# Normalization

→ It's a technique to remove or reduce redundancy from a table.

If there is row-level duplicacy or col.-level duplicacy, it leads to 3 types of anomalies.

→ Insertion

→ Deletion

→ Updation

There are 2 types of duplicacies in DB:

1) Row-level 2) Column-level

(1) Row-level:

| S-id | S-name | Age |
|---|---|---|
| 1 | Ram | 20 |
| 2 | Varun | 25 |
| 1 | Ram | 20 |

Same

We use the concept of primary key (P.K.)
We set a P.K. to any approp. attribute.

Primary key (Unique + Not Null)

(2) Column-level:

| S-Id | S-name | C-id | C-name | F-Id | F-name | Salary |
|---|---|---|---|---|---|---|
| 1 | Ram | $C_1$ | DBMS | $F_1$ | John | 30K |
| 2 | Ravi | $C_2$ | JAVA | $F_2$ | Bale | 40K |
| 3 | Uffen | $C_1$ | DBMS | $F_1$ | John | 30K |
| 4 | Anurag | $C_1$ | DBMS | $F_1$ | John | 30K |
| 5- | Varun | $C_{10}$ | MBBS | | | |

4-col' are same in 3 rows

Anomalees:
It means that problem occurs on special occasions.

1) Insertion 2) Deletion 3) Updation

⇒ Insertion:
For e.g. we want to add data of a new student for e.g. Varun
for e.g. University starts a new course,
$C_{10}$ - MBBS

→ We can't insert this info in table, Even, we — " — " — the new faculty data
→ ∴ We only introduce the new course (∵ We dont talk about student & we don't have S-Id.

→ We can't keep S-Id equal to NULL, ∵ it's as a P.K.
∴ We can't insert directly

2) Deletion Anomaly

We have simple query - Remove the DDR of Roll.no : 1

Delete from student where S-Id = 1 → It will delete the whole row

We don't face any problem here

Now,
We have to delete the data of Roll.no : 2

Delete from student where S-Id = 2 → Row 2 is deleted fully from DB

Now, Row 2 is blank
Now, you can't tell who is teaching roll.no : 2 & what was the course-name of roll no. : 2

Likely le, There was only one student who was studying that particular course & that particular faculty is teaching that course.

⇒ We here wanted to delete the details of student only, but ; of them all the info gets deleted.

course info → lost
faculty info → lost

## 3 Updation Anomaly

Simple query → S-id = 4 change name from Amrit to amritpal

Update student
Set Sname = 'Amritpal'
where S-id = 4

No problem here

Now, if we want to change the salary of faculty f1 from 30k to 40k

Now, The no. of times F1 repeats in table, the same number of times updation query runs & changes them all from 30k to 40k.

Note: There is only 1 faculty f1, then his salary must also change 1 time. But due to the col. level ~~dy~~ duplicacy, it runs no. of times. Hence, it takes more time

∴ It's updation anamaly

& now, normalization remove redundancy

How ?

a table selection may be, it we divide that table into multiple tables.

| S-id | Sname |
|---|---|

| ~~C-id~~ | ~~Cname~~ |
|---|---|

| C-id | C-name |
|---|---|

| F-id | F-name | Salary |
|---|---|---|

# First Normal Form:

Table should not contain any multi-valued attribute.

| Roll-no. | Name | Course |
|---|---|---|
| 1 | Sai | C/C++ |
| 2 | Anurag | Java |
| 3 | Omkar | C/DBMS, |
| | | |

1st way to convert to 1NF:

| Roll-no. | Name | Course |
|---|---|---|
| 1 | Sai | C |
| 1 | Sai | C++ |
| 2 | Anurag | Java |
| 3 | Omkar | C |
| 3 | Omkar | DBMS |

P.K. = Roll no. & Course

Combined it's composite P.K.

## 2nd Way:

| Roll no. | Name | Course-1 | Course-2 |
|---|---|---|---|
| 1 | Sai | C | C++ |
| 2 | Anurag | Java | Null |
| 3 | Omkar | C | DBMS |
| | | | |
| | | | |

P.K. = Roll no.

## 3rd Way:

| Roll. no. | Name |
|---|---|
| 1 | Sai |
| 2 | Anurag |
| 3 | Omkar |
| | |

Base Table

P.K. = Roll. no.

| Roll. no. | Course |
|---|---|
| 1 | C |
| 1 | C++ |
| 2 | Java |
| 3 | C |
| 3 | DBMS |
| : | : |

Referencing table

P.K. = Roll no. + Course

F.K. = Roll no.

# Closure Method:

Helps to find all candidate keys in the table.

E.g. Candidate key (C.K.)

R(ABCD)

F.D. {A → B, B → C, C → D}

Functional dependency | Meaning of closure is that what 'A' can determine

Here, A is determining B

Closure sign $A^+ = B$

$A \rightarrow B$

$B \rightarrow C$

Transitive

$A^+ = BCDA$

(A can determine itself also)

E.g. Rolling can determine itself

Now, R(ABCD) has all 4 attributes that are in $A^+ = BCDA$

A can determine all the attributes of Table, Thus the prop. of the candidate key.

Now, $B^+ = BCD$

Here B does not determine A
i.e. B cannot be a C.K.

& $C^+ = CD$
$D^+ = D$
∴ only A is C.K.

Prime attr. = {A}
Non-prime attr. = {B, C, D}

C.K. = {A}

Note: $(AB)^+ = ABCD$

AB → itself
B → C
C → D

Here, AB can be a C.K.
But, it's not a C.K.
∵ C.K. is always minimal
&
in AB only A is C.K. & extra is super key. (S.K.)

AB → Super key

(With A add anything it becomes a S.K.)

AB is a S.K.

E.g. R(ABCD)

F.D. = {A → B, B → C, C → D, D → A}

$A^+ = ABCD$

$B^+ = BCDA$

$C^+ = CDAB$

$D^+ = DABC$

C.K. = {A, B, C, D}

all 4

Prime attribute: Attr. which is used in making of C.K.

Prime att. = {A, B, C, D} [∵ all 4 are C.K.]

Non-prime att. = {ϕ} → NULL

Q:- R(ABCDE)

FD = {A → B, BC → D, E → C, D → A}

Now, we have to check that which att. are coming on the ~~R.S.~~ right side ∵ Attr. on the right side are being determined.

E = BDCA

E = BDCAE

Note: Each & every candidate key must contain E.

∵, E is present on left side, then only it is written on the right side also.

The att. which is not on right side must be on the left side then it would be used in making C.K.

Now,

$E^+ = EC \rightarrow$ E alone is not C.K. but it is used in making C.K.

Now start:,

with A

$AE^+ = ABECD$ ∵ A is C.K.

$BE^+ = BECDA$

$CE^+ = CE$

$DE^+ = DEABC$

C.K. = {AE, BE, DE}

Trick: First we got AE as C.K., so check either (A or B) on the right side of FD.

F.D. : {$A \rightarrow B$, $BC \rightarrow D$, $E \rightarrow C$, $D \rightarrow A$}

So, directly, DE also becomes the C.K. now check if D, it is dependent on 2,

prime Aff. = {A, B, D, E}

Non-prime Aff. = {C}

# Functional Dependency

It is the method which describes the rel. ship b/w the attributes.

X → Y → Dependent Aff.

Determinant

X determines Y

Y is determined by X

e.g. S-id → S-name → Valid
1 → Ranjit
2 → Ranjit
∴ These 2 are diff.

e.g. 1 → Ranjit
1 → Ranjit
→ Same-student → Valid

e.g. 1 → Ranjit
2 → Varun
→ Valid

e.g. 1 → Ranjit
1 → Varun
→ Not valid

F.D. are of 2 types:

→ Trivial F.D. → Non-Trivial F.D.

1) Trivial F.D.:

If then,

$X \rightarrow (Y)$

Y is subset of X

These trivial F.D. are valide.
(Always true)

E.g. Sid → (S-id)

Note:

$X \rightarrow Y$

L.H.S ∩ R.H.S ≠ ϕ (Never NULL)

E.g. S-id Sname → S-id

2) Non-Trivial F.D.

If then,

$X \rightarrow Y$

Y is not a subset of X.

i.e. $\boxed{X \cap Y = \phi}$ NULL

E.g.

S-Id → S-name
S-Id → Phone no.
S-Id → Location

(Now, in this case we have to use check cases & find whether is valid or not)

## Prop. of F.D.:

1) Reflexivity: If Y is subset of X. (Trivial)

then

$X \rightarrow Y$ (S-Id → S-Id)

2) Augmentation:

If $X \rightarrow Y$, then

$XZ \rightarrow YZ$ (S-Id → Sname
S-Id phone no. → Sname,

(Sid → S-name
Sid phone → S-name phone)

3) Tra Transitive:

If $x \to y$ & $y \to z$

$x \to z$

Sid → S-name & S-name → City

S-id → City

4) Union: If $x \to y$ & $x \to z$

then $x \to yz$

5) Decomposition:

If $x \to yz$ then

$x \to y$ & $x \to z$

But, $xy \to z$ $x \to z$ & $y \to z$.

6) Pseudo Transitive:

If $x \to y$ & $wy \to z$

$wx \to z$

7) Composition:

If $X \to Y$ & $Z \to W$

$XZ \to YW$

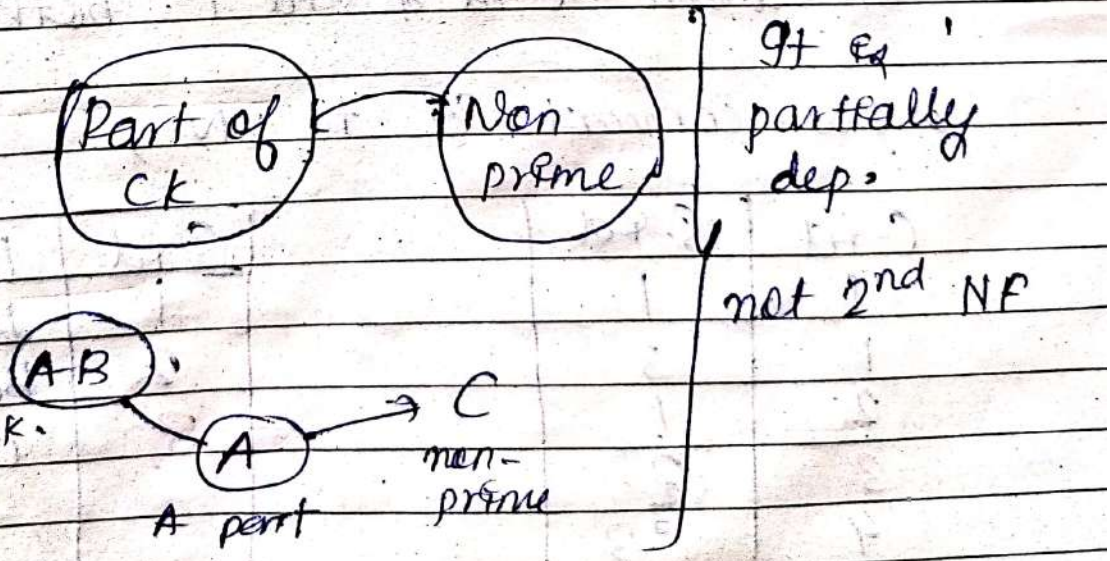
# 2 NF:

## 2 Rules:

→ Tables on Rel$^n$ must be in 1NF

→ All the non-prime attr. should be fully F.D. on C.K.
(or)

(There should be no partial dependency in ~~table~~ Rel$^n$)



Page No.:
Date: / /

E.g.

| Customer_ID | Store_ID | Locn |
|---|---|---|
| 1 | 1 | Delhi |
| 1 | 3 | Mumbai |
| 2 | 1 | Delhi |
| 3 | 2 | Bangalore |
| 4 | 3 | Mumbai |

C.K.:: Customer_ID & Store_ID

Prime att.: C_ID, Store_ID

Non-prime: Locn

Here Locn is only dependent on ~~Locat~~ Store_Id ∴ P.D.

(∵ to be in 2nd NF it should depend on both C_Id & S_Id (∵ both are C.K.))

Now, convert it to 2NF,

| C_Id | S_Id |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |

| S_Id | Locn |
|---|---|
| 1 | Delhi |
| 2 | Bangalore |
| 3 | Mumbai |

(Here it's fully dependent ∵ only 1 C.K.)

Page No.:
Date. / /

Q: R(A B C D E F)

F.D : {C → F, E → A, EC → D, A → B}

Soln: C.K :?

First check right side

S-1: F A D B | means now these attr. are det. by some of the values

So, on L.H.S, there must be ~~EE~~ CE | Whatever would become C.K. would definitely contain CE

CE = FADB

Now,

EC⁺ = ECFADB (All are present)

∴ EC is C.K.

{Now, Use Trick
Either E or C must be present at R.H.S of any F.D.
But, not neither E nor C, no-one is present

So, there is only one C.K. in the table.

$$C.K. = \{EC\}$$

Step I: is complete, here after finding C.K. = {EC}

proof: check

$A^+ = AB$

$B^+ = B$

$C^+ = CF$

proper subset is always less than a set

$\left(\begin{array}{l} X \subset XY \rightarrow \text{proper subset} \\ XY \subseteq XY \rightarrow \text{Subset} \end{array}\right)$

S-2: Prime att. {E, C}

non-prime att.: {A, B, D, F}

S-3: C.K. = ~~{E, E}~~ {EC}

proper subset of $EC \rightarrow E$ or C

Now, check

F.D. = {C→F, E→A, EC→D, A→B}

Page No.:
Date. / /

Check on LHS → Either proper subset of C.K. (Either E or C)

& AND

Check on RHS → Whether it is a non-prime attr.

Check for partial dependency i.e. not in 2NF

We get C → F ∴ partial dependency

So table not in 2NF

C → F ✓ P.D. ✓
E → A ✓ P.D. ✓
} partial

EC → D ✓ Fully dep. ✓
A → B ✗ ——— ✓

Even if there is 1 P.D. then table not in 2NF

# 3NF:

→ Table in Rel^n must be in 2NF.
&
→ There should be no transitive dependency in table

Non-prime or Non-unique
prime or Unique

Mean, Non-prime Att. can't determine other non-prime att.

(C.K. & P.A. can determine NPA)

Eg.

| Roll ng. | State | City |
|---|---|---|
| 1 | Punjab | Mohali |
| 2 | Haryana | Ambala |
| 3 | Punjab | Mohali |
| 4 | Haryana | Ambala |
| 5 | Bihar | Patna |

C.K. = {Roll ng.}

F.D. = {Roll ng. → State

State → City}

PA = {Roll ng.}

NPA = {State, City}

Here, Roll. ng. → State

State → City

NPA NPA

It's a Transitive dependency, so table not in 3NF.

Eg. R (ABCD)

F.D. : {AB → C, C → D}

C.K. = {AB}

PA = {A, B}

NPA = {C, D}

A+ = ABCD

C → D

NPA NPA

∴ Not in 3NF

Page No.:
Date. / /

Eg: R(ABCD)

F.D. : (AB → CD, D → A)

C.K. : {AB, DB}

PA : {A, B, D}

NPA = {C}

$B^+ = B$

$AB^+ = ABCD$

Now A on R.H.S so,

$DB^+ = DBAC$

Q also C.K.

Imp: Now, for each F.D.

L.H.S → C.K. or S.K

OR → Then table Q in 3NF

R.H.S → Q a P.A.

FD: (AB → CD, D → A)

∴ Table Q in 3NF

(NPA → NPA) Q not present here

# BCNF (Boyce Codel Normal form)

→ Also called as special case of BCNF.

| Roll-no. | Name | Voter-ID | Age |
|---|---|---|---|
| 1 | Ram | K0123 | 20 |
| 2 | Varun | N034 | 21 |
| 3 | Ravi | K786 | 23 |
| 4 | Rahul | D286 | 21 |

Table in 3NF

C.K. : {Roll no., Voter-ID}

F.D. :
- Roll no. → name
- Roll no. → voter id
- Voter id → Age
- Voter id → Roll-no.

Note: L.H.S of each F.D. must be C.K or S.K.

? Here, 3NF's OR cond" is always removed, in which it's OK even if R.H.S is PA.

Here we only want C.K. or S.K in L.H.S. & we don't care what's in R.H.S

Soln: So, check all F.D. one by one & in all the F.D., the L.H.S is C.K., it's in BCNF form.

# Lossless & Lossy Decomposition

We normalize table → or we decompose table into 1NF..... forms.

| A | B | C |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 2 |

→ $R_1$ (AB)

→ $R_2$ (BC)

We divides this table into $R_1$ & $R_2$ & B is common in both tables.

$R_1$

| A | B |
|---|---|
| 1 | 2 |
| 2 | 2 |
| 3 | 3 |

$R_2$

| A | B |
|---|---|
| 2 | 1 |
| 2 | 2 |
| 3 | 2 |

→ find the value of C if the value of A = '1'

→ Now, for this we have to join $R_1$ & $R_2$

Select R2.C from $R_2$ Natural join $R_1$ where $R_1.A = '1'$

1st row
cross

First take cross product: If $R_1$ has x rows & $R_2$ has y rows :

then that join has $x \cdot y$ rows

Condition :- Common column of both tables ($R_1$ & $R_2$) here B has the value in join Table

Table after join

$R_1$

| | A | B | C |
|---|---|---|---|
| Spurious tuples | 1 | 2 | 1 |
| | 1 | 2 | 2 |
| | 2 | 2 | 1 |
| | 2 | 2 | 2 |
| | 3 | 3 | 2 |

In original table $R_1$ we have only 3 tuples (rows)

but,
after joining, on R, we have 5 tuples
If is a flaw, it is called lossy decomposition.

→ Why lossy:-

Here, we get 2 xtra rows, then why lossy
Here,
we don't talk about rows. We call it lossy ∵ lossy because of inconsistency
There is a problem in DB.

⇒ In original, for A=1, C=1

but

In join table, for A=1 ← C=1
C=2

Why we get error: (2 tuples more).
∵, we take B as common in both Table

Criteria for Common:

Common att. should be C.K. or S.K. of either $R_1$ or $R_2$ or both.

& B has duplicacy in table, We have to choose Att-A for right Ans.

∴ A is unique (1, 2, 3)

$R_1(AB)$ → We get 3 tuples
$R_2(AC)$ in joining table

## Condition for lossless joining decomposition

1) $R_1 \cup R_2 = R$

$AB \cup AC = ABC$

2) $R_1 \cap R_2 \neq \phi$

$AB \cap AC = A \neq \phi$

3) $R_1$ C.K. or $R_2$ C.K. or both

To take common att.

## Minimal Cover - (Irreducable)

Q1: Find Minimal Cover

{A → B, C → B, D → ABC, AC → D}

Page No.:
Date: / /

Soln: Our R.H.S of F.D. must be single

S-1: By Decomposition prop:, seperate them

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

S-2: Remove the redundant F.D.:

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

If we remove $A \to B$, now check closure of A

$\boxed{A^+ = A}$ (not all)

∴ $A \to B$ is not redundant. We can't remove it

→ & same check this for every F.D:

$D \to B$ is redundant

If we remove $D \to B$, then

$D^+ = DABC$ (all there)

remove $D \to B$

Now while checking other tuples, don't consider $D \to B$

Now, for $AC \rightarrow D$

$$AC^+ = ACB$$

So also include $AC \rightarrow D$

Now, we get

$$\{A \rightarrow B, C \rightarrow B, D \rightarrow A, D \rightarrow C, AC \rightarrow D\}$$

(S-3): Now we want only one Att. on L-H S

Here $AC \rightarrow D$

Now, check by removing A & find closure of C

$$C^+ = CB$$

If we would have found A in $C^+$, then we would have removed A also.

$AC \rightarrow D$ won't be removed

Same check for A, by removing C

$$A^+ = AB$$

So can't remove C

So, $AC \rightarrow D$ can't be reduced

So: $\{A \rightarrow B, C \rightarrow B, D \rightarrow A, D \rightarrow C, AC \rightarrow D\}$

$\{A \rightarrow B, C \rightarrow B, D \rightarrow AC, AC \rightarrow D\}$

## Questions on Normal form:

Q: R(ABCDEF), check the highest normal form

F.D.: $\{AB \rightarrow C, C \rightarrow DE, E \rightarrow F, F \rightarrow A\}$

Soln:

Find all C.K.s in rel$^n$

By Closure:

B is not on R.H.S, so B compulsory

$$B^+ = B \quad , A^+ = A$$

$$AB^+ = ABCDEF$$

$\therefore$ AB is C.K.

check if A on R.H.S, we get

$$F \rightarrow A$$

$\therefore$ FB is also C.K.

check F on R-H-S

$E \rightarrow F$

$\therefore$ EB is also C.K.

check E on R.H.S,

$C \rightarrow DE$

CB is also C.K.

check for C on R-H-S

$AB \rightarrow C$

As AB is already our C.K.
we get got all C.K.S

C.K. $= \{AB, FB, EB, CB\}$

Step 2: Write All Prime Att. & NPA's

P.A. $= \{A, B, C, E, F\}$

N.P.A $= \{D\}$

S-3: Check F.D.'s

$\{AB \rightarrow C, C \rightarrow DE, E \rightarrow F, F \rightarrow A\}$

check 1 by 1

1NF, 2NF, 3NF, BCNF
→ Redund. dec.

Note: When table is in BCNF, then redundancy is lowest & in 1NF redundancy is highest.

Start from BCNF.

BCNF → L.H.S of all F.D.'s should be C.K or S.K.

{AB → C, C → DE, E → F, F → A}
AB ✓ C ✗ DE ✗ E ✗ F ✗ A

Not in BCNF

3NF :

Check Transitive dependency.

(NPA → NPA) ✗

L.H.S → C.K. or S.K. } → 9th's in 3NF
OR
R.H.S → is a P.A.

1st cond. is already checked,

Check whether R.H.S is P.A.

{AB → C ; C → DE ; E → F ; F → A}

| | | | | |
|---|---|---|---|---|
| BCNF | ✓ | × | × | × |
| 3NF | ✓ | × | ✓ | ✓ |

Not in 3NF

2 NF :

Check only at places where there is ×, ∵ if fds in 3NF fds already in 2NF.

Check :

LH.S is proper subset of C.K.
R.H.S is N.P.A.

for partial dep. i.e. if true then not in 2NF.

| | AB → C | C → DE | E → F | F → A |
|---|---|---|---|---|
| BCNF | ✓ | × | × | × |
| 3NF | ✓ | × | ✓ | ✓ |
| 2NF | ✓ | × | ✓ | ✓ |

for 1NF, check that we don't have multi-valued att. in table.

Page No.:
Date. / /

Table → R(ABCDEF)
All are gen- att.
we can't tell table is in INF or not.

## (34) Cover & Equivalence:

If X covers Y  $Y \subseteq X$
If Y covers X  $X \subseteq Y$

& If both are True

$X \equiv Y$

Q: $X = \{A \to B, B \to C\}$ | $Y = \{A \to B, B \to C, A \to C\}$

(1) X covers Y:  $Y \subseteq X$

(Here we'll consider L-H-S of F.D: of Y & find its closure from X)

$A^+ = ABC$  $B^+ = BC$

$Y = \{A \to B, B \to C, A \to C\}$

∴ All 3 covers in the closure forms.

∴ X covers Y

(11) Y covers X

$X = \{A \to B, B \to C\}$

(Will take closure of X from Y)

$A^+ = ABC$

$B^+ = BC$

∴ Y also covers X

∴ both symbols cancel each other & become Equivalent.

$X \not\supset Y \qquad Y \not\supset X$

$X \equiv Y$

Q $X = \{AB \to CD, B \to C, C \to D\}$

$Y = \{AB \to C, AB \to D, C \to D\}$

Soln: X covers Y

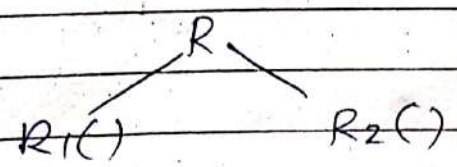
# Dependency Preserving Decomp.

Defⁿ: Let any table R(ABCD)
&
also some F.D., FD = {A → B, B → C}.

Now,
We find closure & then find C.K. &
then we also find hidden dependencies.

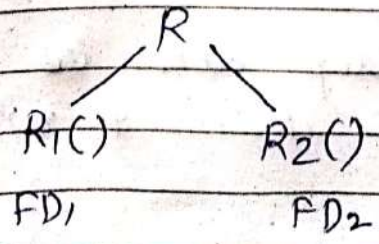
i.e., FD⁺ {A → C} by Trans. prop.

& In normalization, we do decompsⁿ
& then we divide




& we dist. att. of R in R1 & R2
&
union of all att. of R1 & R2 must be
equal to att. of R

Now, FD. also divides, like



Now check, dependency should be preserved

$$\therefore FD_1^+ \cup FD_2 = FD^+$$

Q: R(ABCD) with F.D.

{A → B, B → C, C → D, D → B}.

R is decomposed into $R_1(AB)$, $R_2(BC)$, $R_3(BD)$

| $R_1(AB)$ | $R_2(BC)$ | $R_3(BD)$ |
|---|---|---|
| ✗ A → A | B → C ✓ | B → D ✓ |
| ✗ B → B | C → B ✓ | D → B ✓ |
| A → B ✓ | | |
| B → A ✗ | | |
| ∵ $B^+ = BCD$ (don't have A) | | |

Union of att. $R_1$, $R_2$ & $R_3$ are equal to $R_1$. So, True (Right decomposition)

Now check Dependency preservation:—

$R_1(AB)$, so, it has only A, B att.

So, whatever dep. can be made from these 2, make them & write in table.

We don't want Trivial F.D. →

Trivial F.D.: in which ∩ (intersection) is not null

E.g.1 $A \rightarrow A$ has $\cap \neq \phi$ (common is A)

E.g.2 $AB \rightarrow A$ has $\cap \neq \phi$ (common is A)

We don't want these ∵, trivial are by default true.

$A \rightarrow A$

So,
only take non-trivial F.D. : →

$(\cap = \phi)$

F.D. : { $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B$ }

~~A+=~~

F.D. : { $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B$ }

$A^+ = ABCD$
$B^+ = BCD$
$C^+ = CDB$
$D^+ = DB$

} So acc. to these select from non-trivial F.D. (which we will finally select from table)

Now,
F.D. which we get from table

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

} Now check whether original F.D. are possible through these or not

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

Now, $C \to D$ is not direct
So, now take 'C' closure

$C^+ = CBD$ $\therefore C \to D$ is also preserved

all F.D. are preserved.