Data: Data can be any character, text, word, numbers, images.

→ Data is a raw and unorganized fact that required to be processed to make it meaningfull.

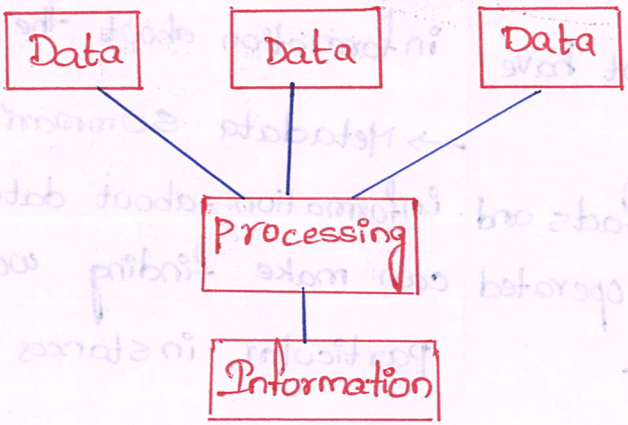
→ Data is always interpreted by a human or machine to derive meaning. so data is meaningless.

Information:

Data formatted in a manner that allows it to be utilized. By humanbeings in a significant way.

→ Information is a set of data which is processed in a meaning-full way according to the given requirement.

→ Information is processed, structured or presented in a given context to make it meaningfull and useful.



Data:

Ex: Enclave, 9505727837, laxmi, road no:4, krishna, Nagaram, near, flat no:154, Theater, Vaj, 500083, shasank.

⇓

Information:

raj laxmi
flatno 154

Road no: 4
Shashank Enclave
near krishna Theater,
Nagaram, 500083.

Data.

→ Data can be a collection of facts, words, observations, measurements, or description of something.

→ The data can be operated to obtain same meaningfull information out of it.

→ Data can or can not be informative

→ Data may or may not have been processed.

→ Data is a set of facts and statistics that can be operated referred or analyzed.

Meta Data.

→ It described as data about data.

→ It contains the informative and relevant description about the original data.

→ It is always informative.

→ Metadata is always a processed data.

→ Metadata describes relevant information about the data.

→ Metadata summarizes basic information about data which can make finding working with particular instances of data easier

Ex:- author,
data created,
data modified,
file size

} By using this it much easier to locate same specific doc.

Metadata.

Ex: to locate an object.

title,
author,
Subjects,
keywords,
Publisher

} Meta data is used for discovery & identification.

Data

Eid	first name	last name	Gender	age
111	A	Raj	F	22
112	B	Laxmi	F	25
113	C	pinky	F	26
114	D	Mahi	M	28
115	E	Mohan	M	27

⇒

Metadata.

Column	Datatype	description.
Eid	int	-
firstname	varchar(10)	-
lastname	Varchar(15)	-
gender	char(1)	-
age	int	-

DBMS UNIT-I

Introduction:

DBMS

↓ ↓

Data Base → Collection of data

Management system → Set of programs to store & retrive those data (Define, construct, manipulate).

so,

DBMS is a software which is used to store the data into the database & access the data from the database in a efficient & convenient manner.

Ex: MYSQL, ORACLE etc.... are the most popular commercial DBMS s/ws used in different applications.

Purpose of database system:

→ In previously data is stored in file system (file s). These file system have same drawbacks.

* Data Redundancy & inconsistency.
* Difficulty in accessing data.
* Data isolation (it use multiple files & formats)
* Data security.
* Translation problems. so,

→ Data base system offers solutions to all the above Problems.

Data:

Collection of facts numbers, Text, character, images, words, observations, measurements, or description of something.

Ex: 76.

Information:

Information is a set of data which is processed in a meaningfull way.

Ex: Marks = 76.

Database:

It is a set of files where application data & metadata is stored.

DBS:

Data base system is a combination of database & DBMS.

DBS = Database + DBMS.

DB structure:

→ In data base structure we have 3 levels.

1) physical level / Internal level / storage level :-

→ This is the lowest level that describes how data are actually stored.

→ It's a physical representation of database.

→ It covers the data structures and file organization.

2) Logical level / conceptual level :-

→ It's a next higher level that describes what data are stored

in DB & relationship among the data.

→ DBA who must decide what data to keep in database. will use logical level of abstraction.

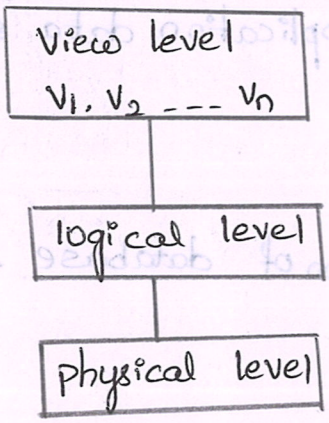
→ It represents all entities, attributes & their relationships & the constraints on the data.

→ It describes the security & Intigrity of information.

3) View/External level :-

This is the highest level of abstraction. It provides many views for same DB.

→ It is the part of DB that is relevant to each endoser.



3 schemas.

physical schema : database design at physical level

logical schema : database design at logical level

View schema : database design at view level

Sub schema :- It's a part of the schema which inherits the properties of schema.

Schema: The overall design of a database.

Instance: Database design at particular movement.

Abstraction:

→ Data abstraction refers to providing only essential information to the outside world & hiding their background details.

→ The process of hiding irrelavent details (not applicable) from user is called data abstraction.

Database applications:-

1) univercities.
- college information
- Student information
- Course information
- Staff information
- fee details.

2) Banking.
- Customer profile
- loan details
- transaction details
- Branch details
- Deposits information

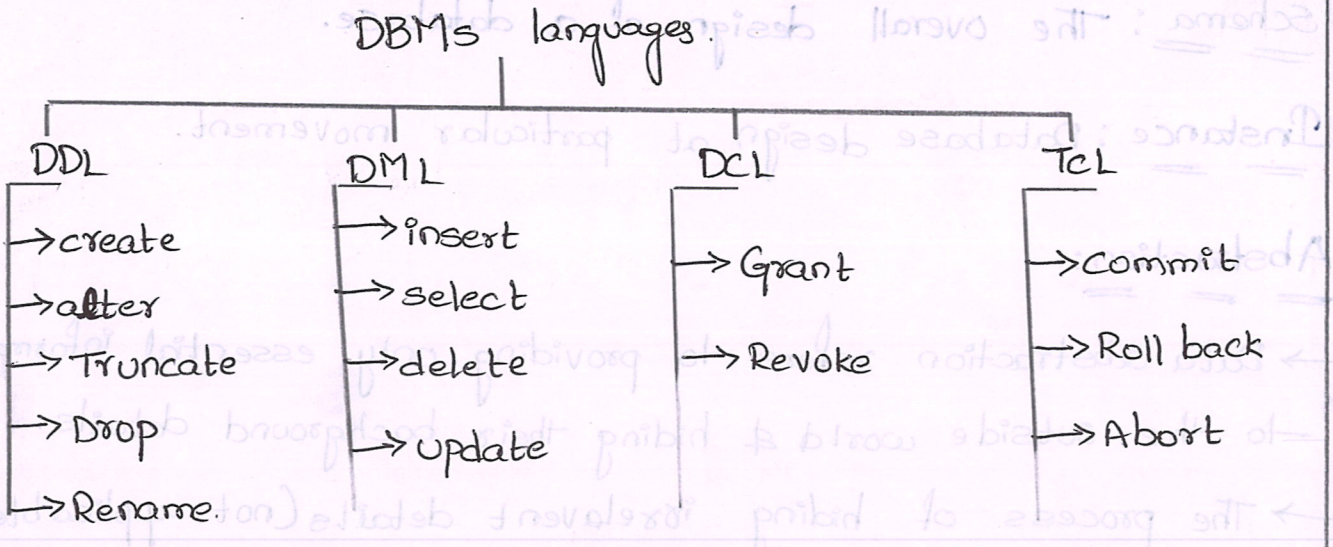
3) Insurences
- Polocy name
- Bonus
- Retailers
- meturity period
- term.

4) Airlines.
- flight schedule
- no. of terminals
- reservation.

5) Enterprise.
- sales
- A/c
- Manufacturing

Data base languages:-

→ It is used to provide communication blw users & DB languages used to read, sture, update the data in the database.



1) DDL (data definition language):-

→ By using this we are defining the data into the DB.

→ to define the database structure or pattern

→ it is used to store the information of metadata like the no. of tables & schemas, their names, indexes, columns in each table, constraints etc----

Ex: create, alter, drop, truncate, Rename---

2) DML (data manipulation language):-

→ It is used for accessing and manipulating data in database. It handles user requests.

Ex: insert, select, delete, update---

3) DCL (data control language):-

→ It is used to retrive the stored or saved data.

→ The DCL execution is transactional.

→ It also has roll back parameters.

Ex: grant

↓

To give user access priviteges to a DB.

Revoke

↓

It is used to take back permissions from the user.

4) TcL (Transaction control language):-

→ TcL is used to run the changes mode by the DML strit.

→ TcL can be graped into a logical transaction.

Ex: commit

↓

It is used to save the transaction on the database.

It is 'T' executed successfully.

Roll back.

↓

(Uncommitted transaction to committed

→ It is used to restore the database to original since the last committ.

Abort:

↓

→ T executed unsuccessfully.

→ to kill a blocking process.

DBMS.

1) create database data_base name;

SQL> Create database HITs;

2) Create table tablename (col name1. datatype1, col2 datatype2...);

SQL) Create table student (sid integer, sname Varchar(10));

3) insert into tablename (col1, col2) values (&col1, &col2);

(or)

insert into tablename values (&col1, &col2);

(or)

insert into tablename values (data1, data2);

SQL) insert into student values (&sid, '&sname');

Enter value for sid: 111 ↵

Enter value for sname: Raj ↵

SQL> /

Enter value for sid: 112

Enter value for sname: Mahi ↵

2 rows created.

4) Select * from tablename;

Select col list from tablename where cdt;

Select * From tablename where cdt;

SQL> select * From student;

SQL> select sname From student where sid = 112;

5) Desc tablename;

SQL> Desc student;

6) Delete From tablename; //it deletes entire data from table:

Delete From tablename where colt;

SQL> Delete from student where sid = 111;

7) Alter table tablename add col-name datatype;

alter table tablename modify col, datatype;

alter table tablename drop col1;

alter table tablename enable/disable colname;

alter table tablename add (colname1 datatype, col2 datatype...);

SQL> alter table student add slept varchar(20);

8) update tablename set colname = value where colt;

SQL> update student set sDept = 'cse' where sid = 112;

SQL> update Emp set Esal = Esal + 1000 where tname = 'Mahi';

9) Drop table tablename;

Drop user username;

SQL> Drop table student; //it destroys the table from DB//

10) Tuncate table tablename;

SQL> Tuncate table student; //it destroys the data from table//

11) Rename <old table> to <new table>;

SQL> Rename Emp to staff;

12) Create user <user name> Identified by <pwd>

SQL> create user 5727 Identified by 1234;

13) Select * from student where sname like A%;

//starts with A.

14) select * from student where sname like '_d%');

//It returns the sname contains the 'd' as the second char.

15) select * from student where sname Like 'b%.U';

//It returns sname starting with b & Ends with U.

16) Select * from student where Sname Like 'c----r';

//It returns sname start with c & End with r contains 5 char b/w c & r//.

17) select Sage From student where sid between 104 and 108;

18) Grant Create session to MCA 12; //It gives permission to create

Grant select an EMP to MCA 12;

Grant select, Insert, Delete on Emp to MCA 12;

Grant all on Emp to MCA 12; //it gives all type of permissions.

19) Revoke all on Emp From MCA18. //cancell all. permissions.

20) select DeptNO, cant(EmpNo) From emp Grap by DEPt NO;

21) select DeptNO, max(sal) From Emp group by DeptrNo Having max(sal) > 2000;

22) Create view Empstu (Eid, sDept) As select Ename, from Emp, Sid stu;

23) select * from empstu;

24) select (srating = srating + 1) As rating from sailors S where Srating > 7;

rating
9
8

⇒

rating
10
9

25) select Sname As CSE from sailors;

Sname
A
B

⇒

CSE
A
B

Syntax : create database databasename;

1) Create database CMREC;

SQL>

2) Syntax:-

Create table tablename (col name1 datatype1, col name2 datatype2 --);

SQL> create table Student (sid integer, sname varchar(10), sage integer);

3) Syntax :- select * from tablename;

SQL> Select * from student;

Sid	Sname	Sage.

4) SQL> Desc tablename;

SQL > Desc student;

Name	Null	Type
sid		integer
sname		Varchar(10)
sage		integer.

5) SQL > Insert into tablename values (&col1, &col2);

SQL> insert into student values (&sid, &sname', &age); ↵

Enter value for sid : 111 ↵
Enter value for sname : xxx ↵
Enter value for sage : 19 ↵

SQL> 1 row created ↵

Enter value for sid : 112 ↵
Enter value for sname : xyz ↵
Enter value for sage : 20 ↵

2 rows created.

SQL> insert into student values (
111, 'xxx', 19) ↵

1 row created.

SQL) insert into student values (
112, 'xyz', 20) ↵

2 rows created.

SQL) select * from student;

sid	sname	sage
111	xxx	19
112	xyz	20

SQL>

6) Delete from tablename;

SQL> Delete from tablename where col1;

SQL> Delete from student where sid = 111; ↵

1 row deleted.

7) SQL> update tablename set colname = value where colt;

SQL> update student set sname = 'Raj' where sid = 112;

Sid	Sname	Sage
111	xxx	19
112	xyz	20

Before update

Sid	Sname	Sage
111	xxx	19
112	Raj	20

After update.

SQL> update student set sage = stage+2 where sname = 'xxx';

1 row updated.

Sid	Sname	Sage
111	xxx	21
112	Raj	20

SQL> update student set sage = sage+10;

2 rows updated.

Sid	Sname	Sage
111	xxx	31
112	Raj	30

8)
SQL> After table tablename add col-name datatype;

SQL> alter table student add sgender char(2);

table altered.

SQL> Select * from student;

sid	sname	sage	sgender
111	xxx	31	NULL
112	Raj	30	NULL

SQL> update student set sgender = 'm' where sage = 31;

1 row updated.

SQL> update Student set sgender = 'FM' where sname = 'Raj';

1 row updated.

SQL> Select * from student;

sid	sname	sage	sgender
111	xxx	31	M
112	Raj	30	FM

MySQL> alter table tablename rename to new name;

MySQL> alter table student rename to student details;

//to change table name//.

MySQL> Alter table table-name change column old-name to new-name;

MySQL> alter table student details change sid to Holl.NO;

MySQL> alter table table.name Rename column old.name to new name;

SQL> alter table student modify sname varchar(8);

//after executing this sname size reduced to 8 previously it is created as 10.

9) SQL> Rename <old table name> to <new table name>;

SQL> Rename table student to CMREC;

10) SQL> Truncate table tablename; //It destroys the data from table

SQL> Truncate table CMREC;

SQL> Select * from CMREC;

But table present in your database but it's a empty table.

Sid	Sname	Sage	Sgender.

SQL> Rename table oldname to new name;

Grant:

SQL> GRANT SELECT ON Employee TO User 1

// This command grants a select permission on Employee table to user 1 //.

REVOKE:

SQL> REVOKE Select an Employee from user1.

// This command will revoke a select privilige an Employee table from User 1 //.

Commit :- to save changes.

⇒ The commit command saves all the transactions to the database since the last commit or Rollback command.

Syntax: COMMIT;

Ex: Student.

Sid	Sname	Sage.
10	X	25
20	Y	24
30	Z	23
40	A	25
50	B	22

Syntax:- SQL>delete from student where sage=25;

SQL> commit;

SQL> select * from student;

Sid	Sname	Sage.
20	Y	24
30	Z	23
50	B	22

Rollback: //used to undo transactions that have not already been saved to the database.

→ This command can only be used to undo transactions since the last commit or Rollback, command was issued.

Syntax:- ROLLBACK;

Ex:- SQL> Delete from student where sage=25;

SQL> ROLLBack; // delete operation would not impact the table //.

SQL> select * from student;

Sid	Sname	Sage.
10	X	25
20	Y	24
30	Z	23
40	A	25
50	B	22

MySQL>create database Raj;

MySQL> show databases;

MySQL> \r ↵

current database

MySQL> use Raji ↵ // database changed . to our own data base.

MySQL> show tables; // it gives test of tables in your current data base //.

MySQL> create table renu (rname char(10), rnum integer, rage integer); ↵

MySQL> insert into renu (rname, rnum, rage) values ('gita', 435, 12);

MySQL> insert into renu (rname, rnum, rage) values ('gitasri', 456, 13);

MySQL> select * from renu;

rname	rnum	rage
gita	435	12
gitasri	436	13

SQL> delete from renu;
// it deletes entire data from table it shows empty table //.

SQL> delete from renu where rnum=435;
// it deletes that row completely.

MySQL> desc renu;

field	Type	NULL	key	default	Extra
rname	char(10)	Yes		NULL	
rnum	int(11)	yes		NULL	
rage.	int (11)	yes		NULL	

3 rows in set

MySQL> update renu set rnum = 333 where rname = 'gita';

MySQL>

rname	rnum	rage
gita	333	12
gitasri	436	13

MySQL> alter table renu add raddress char(15);

MySQL> select * from renu;

	raddress
	NULL
	NULL

iv) Naive users:

→ invoke one of the permanent application programs, that have been written previously.

→ The users without having the programming knowledge they can access the data from the database, just by invoking the previous executed programs.

Ex: people accessing the database over the web, bank tellers. agents, web-users.

DBA:-

→ providing an authorised users to acces the database.

→ coordinates all the activities of the DBS &

→ has a good understanding of the enterprise's information resources and needs.

→ Backuping the data

→ Recover a system from the crashes

→ providing a history of database

→ Monitoring performance and responding to changes.

→ schema and physical organization modification.

→ storage structure and access method definition.

DBMS components:

2) storage manager 1) Query processor.

1) Query processor:-

→ It simplifies and facilitates easy access to the data language, and operates the statement.

In this we have 3, parts

(i) DDL interpreter

ii) DML compiler

iii) Query evaluation engine.

DATABASE SYSTEM STRUCTURE

DBMs Architecture:

→ The people who use database are classified as.

(i) users: use database for general purpose work.

ii) Administrators: who has central control over the database.

↳ users are differentiated by the way they expect to interact with the system is.

(i) Application programmers:-

→ interact with system through DML calls

→ The users who are having the programming knowledge and writting the programs to approaching the database

Ex: s/w Engineers (or) developers.

(ii) Sophisticated users:

→ Form requests in a database query language.

→ The users can approach the database by using Query lang lods → Razor SQL, DataXtractor, dera Rapid SQL

Ex: DBA, Analysts.

iii) specialized users:

→ write specialized database applications that donot fit into the traditional data processing framework.

→ The users having the programming knowledge. but theydonot work with the query language tods.

→ Theye specialized users are unfit for the database traditionally. Ex: DB designers.

(i) DDL interpreter: It interprets DDL statements into low level language understood by the system.

ii) DML compiler: It translates DML statements into an evaluation plan consisting low level instructions that the query evaluation engine understands.

iii) Query Evaluation Engine: It executes low level statements generated by the DML compiler.

2) Storage manager :-

→ Storage manager stores, retrives & update data in the data base.

→ It translates DML statements into low level statements that the database understands.

it's components are :

(i) Authorization and Integrity manager:

→ It checks for the correctness of the constraints specified and also checks the authenticity of the users while entering the database.

ii) Transaction manager: It ensures that the database follows the ACID property.

iii) File manager :- It responsible to manage allocation of space on the disk storage.

iv) Buffer manager :

It is responsible for fetching data from the disk storage and to decide what data to cache to caache in the main memory.

→ It enables to handle data sizes which are lager than the size of the disk.

Disk storage:

→ It's responsible for allocating the disk space and also for maintaining the data in the database.

It's components are:

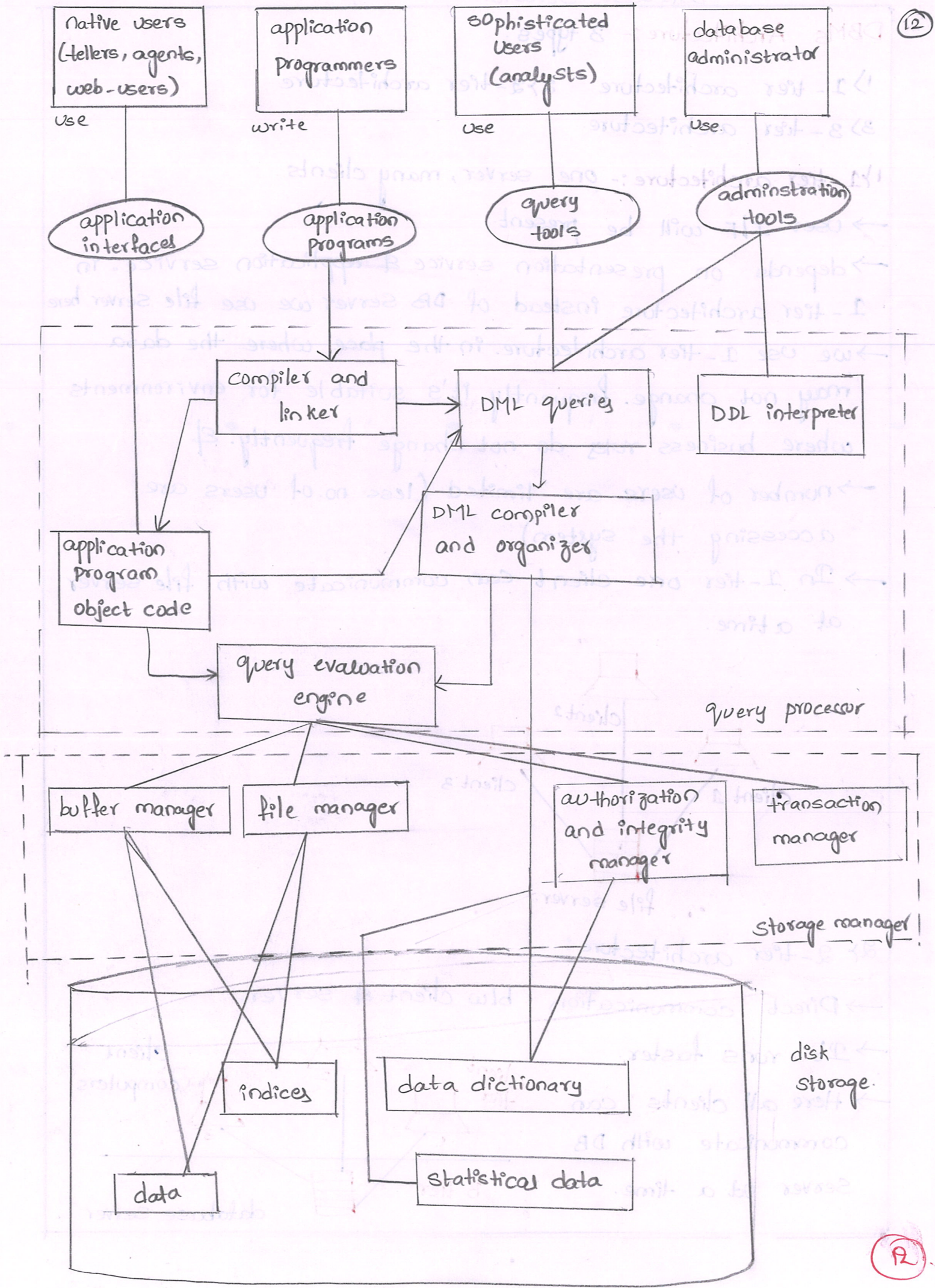
(i) data files :- it is the place where database is stored.

ii) data dictionary :- it stores metadata about the schema of the database.

iii) indices :- It provides Quick access to the data items that hold particular values.

iv) statistical data : It stores all numerical data using the theory of probability. This data dealing with the mathematical science.

Ex: Any financial (economics data, Bank transactions medical history, price of product, ---


native users (tellers, agents, web-users)
application programmers
sophisticated users (analysts)
database administrator
use
write
use
use
application interfaces
application programs
query tools
adminstration tools
compiler and linker
DML queries
DDL interpreter
DML compiler and organizer
application program object code
query evaluation engine
query processor
buffer manager
file manager
authorization and integrity manager
transaction manager
storage manager
indices
data dictionary
disk storage
data
statistical data

Data base structure :

DBMs Architecture :- 3 types.

1) 1-tier architecture 2) 2-tier architecture

3) 3-tier architecture.

1) 1-tier architecture :- one server, many clients.

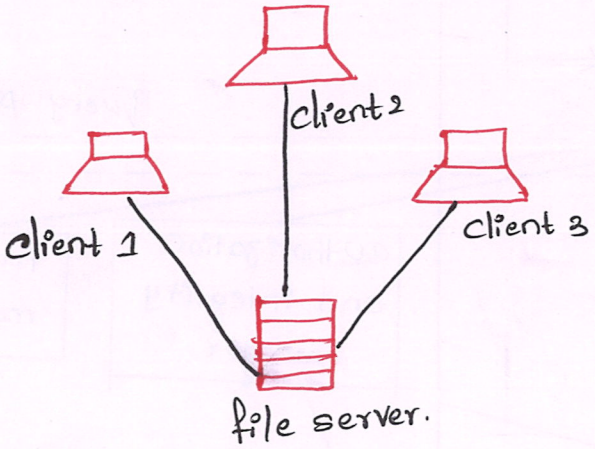
→ User I/F will be present.

→ depends on presentation service & application service. In 1-tier architecture instead of DB Server we use file server here.

→ we use 1-tier architecture. in the place where the data may not change. frequently, it's suitable for environments where business rules do not change frequently. Ex

→ number of users are limited (less no. of users are accessing the system).

→ In 1-tier one client can communicate with file server at a time.

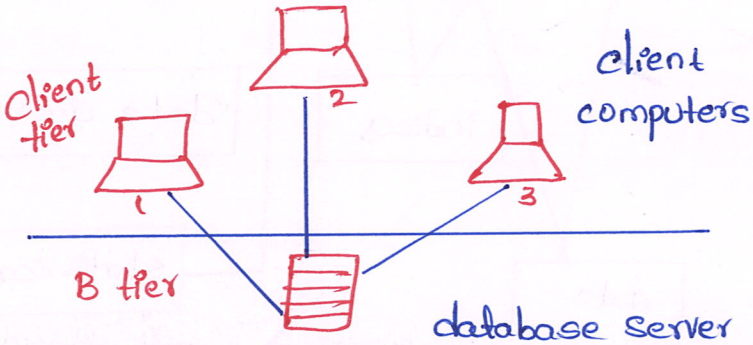


2) 2-tier architecture :

→ Direct communication blw client & server.

→ It runs faster.

→ Here all clients can communicate with DB Server at a time.



2-tier Architecture Adv & disAdv :-

Advantages	disadvantages
→ suitable for environments where business rules do not change frequently. → number of users are limited.	→ useless for large scale organizations. → only a limited number of clients can access.

3-tier Architecture :- completely depends upon nlwing & Internet. (cloud)

→ It's a webbased application

→ It separates tiers from eachother based on complexity of the users and how they use the data present in DB by the user.

→ 3 layers are

(i) client layer (ii) Business layer (iii) Data layer.

1) Database / data tier :-

→ It resides along with is it's query processing languages. Data & constrains will stored here.

2. Application tier :-

→ It present an abstract view of DB for user

→ It acts as mediator blw user & DB.

→ the DB server don't know about clients beyond the application server. so the app. tier completely knows about each clients.

3) User / presentation tier :

→ Here Enduser operates on this tier and they know nothing about any existence of DB beyond this layer.

Scalability: The capacity to be changed in size or scale

Flexibility: The ability to be easily modified

Application program: primary purpose is entering & retrieving information from a database.

Subschema: It's a part of the schema which inherits the properties of schema.

Isolation: The ability of many users to access the same data at the same time. [it increase the no. of concurrency effects].

Schema: A description of data interms of a data model is called as schema.

→ In a relational model, the schema for a relation specifies its name, the name of each field & the type of each field.

Ex: student information in a univerity database Student (sid : integer, name : string, age : integer)

Data Abstraction:

Data Abstraction refers to providing only essential information to the outside world & hiding their background details.

→ The process of hiding irrelevant details from user is called data abstraction.

Sid	sname	sage
726	X	19
605	Y	18
550	Z	20
403	A	17

Here.

Degree = no. of column heads / attributes = 3

Cardinality of a relation = no. of tuples / rows

records = 4.

Grant:

GRANT SELECT ON employee To user1

// This command grants a select permission an employee table to user1. //.

REVOKE SELECT ON employee FROM. User1

// This command will Revoke a select privilige an employee table from user1 //.

Instances and schemas: If table have 100 records so the instance of the DB has 100 records.

I: collection of information stored in DB at a particular moment

S: The overall Design of DB.

Database schemas are various types depends on the level of abstraction.

→ physical schema - database design at physical level.

→ Logical schema - " logical level

→ sub schema - " view level.

Data models:- A model is basically a conceptualization between attributes & entities.

→ Data model is a combination of conceptual tools that are used for describing the data. Data symantics data

relationships.

* Constraints → Data models show that how the data is connected & stored in the system. It shows the relationship between data.

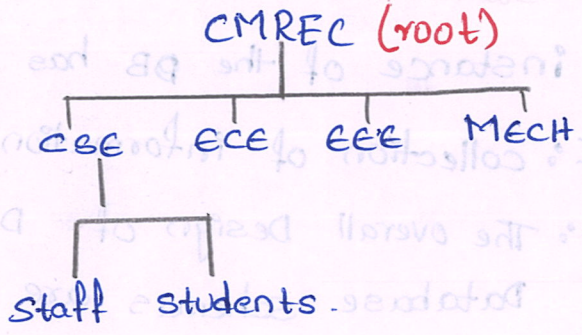
→ Different data models are.

1) Relational data model - Consists of tables and relationships

2. Entity Relationship model (ER model) - consist of entities & Relationships.

→ In this model data represented by using ER diagrams.

3. Hierarchical data model:- It has one parent entity with several children entity but at the top we should have only one entity entity called root.



4. object-oriented data model:-

It is one of the developed data model. & this can hold the audio, video & graphic files.

These consist of data piece &

the methods which are the DBMS instructions.

Attribute : The name of each field is an attribute / column

→ attribute is a characteristic.

→ attribute refers to a database component such as a table

→ attribute refers to a database field.

→ attribute describe the instances in the row of a

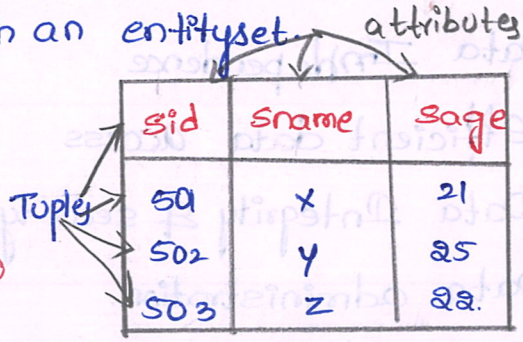
database.

→ Each attribute associated with an entityset.

→ Attribute is a property that describes an entity.

→ attribute represents a column

→ attribute represents a fieldname.



Entity :-

→ Entity is a real world object that represents data in RDBMS.

→ Entity has a no.of attributes (columns)

→ Entity represents a row

→ Entities have attributes.

→ Entity defines database.

Entity - ⬭

Entity set - ▭

Entity set - student

attributes - sid, sname, sage.

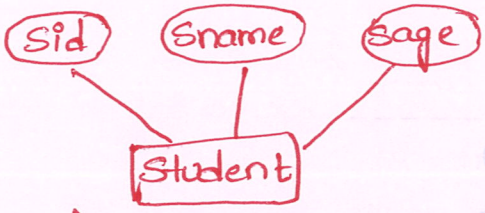


fig: The Student Entity set.

→ An entity is described using a set of attributes

→ All entities in a given entity set have the same attributes

Employee.

Eid	Ename	Esal	Eaddress.
2074	XXX	25000	Nagaram
2076	YYY	28000	ECIL
2078	ZZZ	32000	Uppal
2080	AAA	35000	Sec

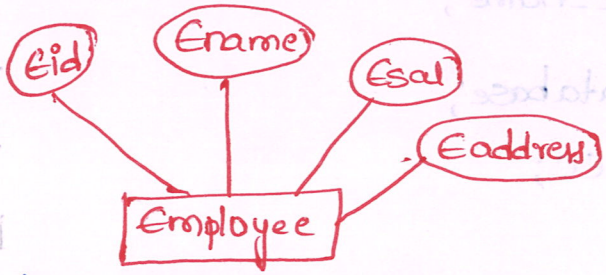


fig: Employee entity set.

Entity set - Employee

attributes - Eid, Ename, Esal, Eaddress

Entities - 2074, XXX, 25000, Nagaram.

Advantages of a DBMs:

→ Data Imdependence
→ Efficient data access
→ Data Integrity & security.
→ Data administration.
→ concurrent Access and crash Recovery.
→ Reduced application development time.

Describe :- DESCRIBE tablename;

(or)

DESC tablename;

→ structure of a table it includes, name of the column, data-type of column and the nullability which means that column can contain null values or not.

Ex: DESC table:

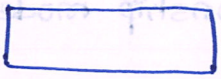
Name	NULL	Type
First-name		char(25)
Last-name		char(25)
Salary		Number(6)

Create database :-

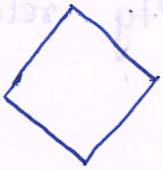
SQL > create database database_name;

Ex: create database mydatabase;
create database CHREC;

Components of a ER diagram:-

Rectangle  - Entity set

Elliples  - attribute

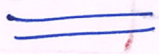
Diamond  - Relationship set

Line - Relationship [attribute to Entity set & Entity set to relationship set]

double Ellipses  - Multivalued attributes [any attribute we can define some more attributes] it holds multiple values Ex: ph.no

double Rectangle  - weak - Entity sets [does not contain any primary key attributes] (Entity set.)

Dased Ellipses  - Derived attributes [any attribute derived from other entity].

Double lines  - total participation of an entity in a relationship set.

ER Model

- **Entity**
  - weak Entity set
- **Attribute**
  - key
  - composite
  - Multivalued
  - Derived
- **Relationship set**
  - one to one
  - one to many
  - many to one
  - many to many

ER - Datamodel:- It's a design / blue print of a DB.

→ ER model stands for an entity - Relationship model

→ It's a high-level datamodel

→ ER model describes the structure of a database with the help of diagram, which knows as entity relationship diagram.

→ ER diagram shows the relationship among entity sets. An Entity set is a group of similar entities and these entities can have attributes.

→ An entity is a table an attribute of a table in DB.

→ The main components of E-R model are entity set and Relationship set.

→ ER diagram shows the complete logical structure of a DB.

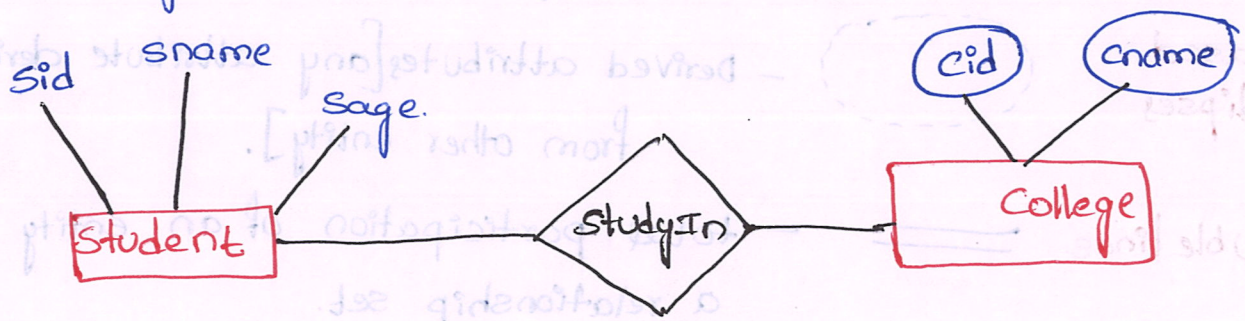


fig :- Sample E-R diagram.

→ The relationship between Student and college in many to 1 as a college can have many students however a student cannot study in multiple colleges at the same time.

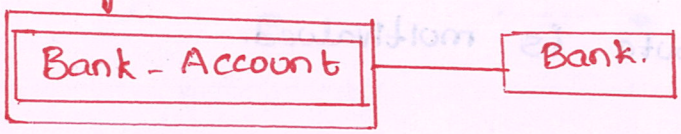
Record - Collection of interrelated data.

1) Entity :- object / component of data

Weak-entity set :- An entity set that can not be uniquely identified by it's own attributes & relies on the relationship with other Entity set is called weak entity.

Ex:- A bank a/c cannot be uniquely identified without knowing the bank details to which the a/c belongs to so bank a/c is a weak entity.

*Weak entity set : which cannot hold the primary key.

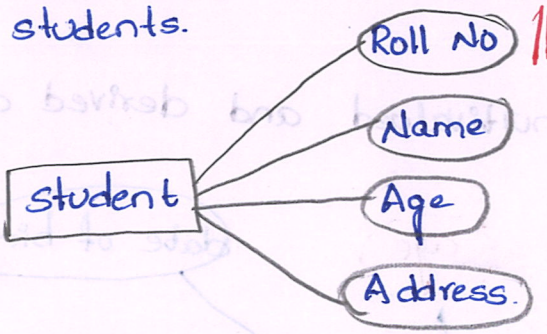


2) Attribute :- An attribute represents the property of an entity set. They are 4 types.

(i) key- attribute :-

→ A key-attribute can uniquely identify an entity from an entity set.

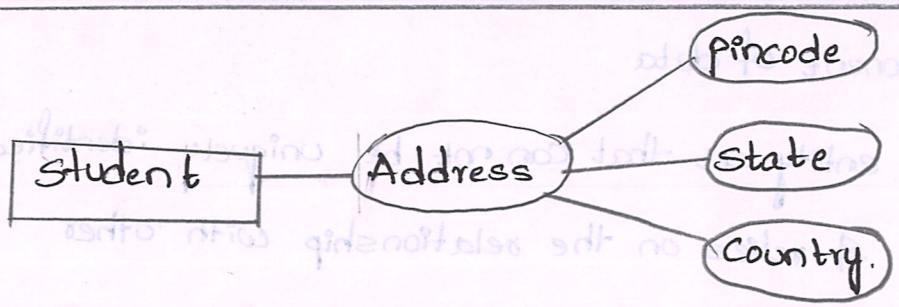
Ex:- student roll number. can uniquely identify a student from a set of students.



ii) Composite Attribute :

→ An attribute that is a combination of other attributes is known as composite attribute.

Ex:- in student entity set, the student address is a composite attribute (as pincode, & address is composed of other attributes as pincode, State, Country



(iii) Multivalued attribute:-

→ An attribute that can hold multiple values is known as multivalued attribute.

→ It is represented with double ovals in ER diagram.

Ex: A person can have more than one phone number. So, phone number attribute is multivalued.

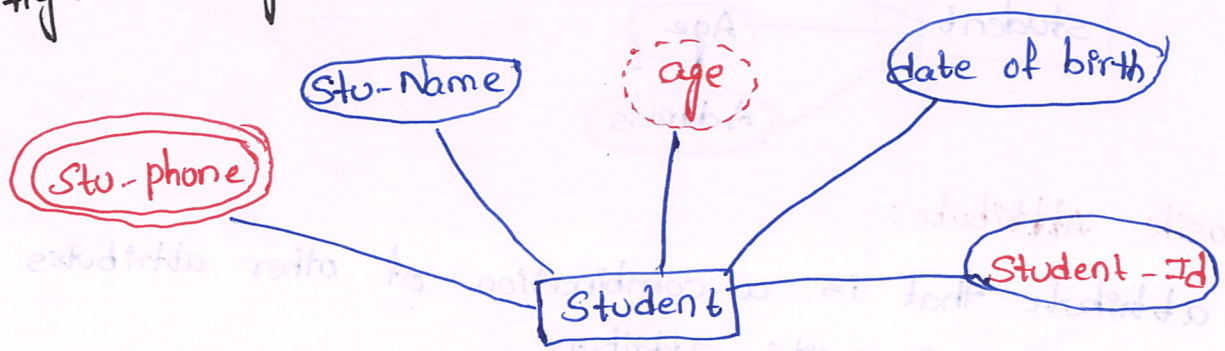
iv) Derived attribute:

→ A derived attribute is one whose value is dynamic and derived from another attribute.

→ It is represented by dashed oval in an ER-diagram

Ex:- person age is a derived attribute as it changes over time and can be derived from another attribute (date of birth)

fig: ER-diagram with multivalued and derived attributes:



3. Relationship set :-

A relationship is represented by diamond shape in ER diagram. It shows the relationship among entities.

4 types of relationships.

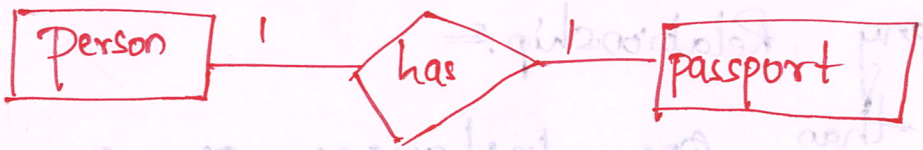
Cardinality :- The no. of times an entity of

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

1) One to One relationship :-

→ When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship.

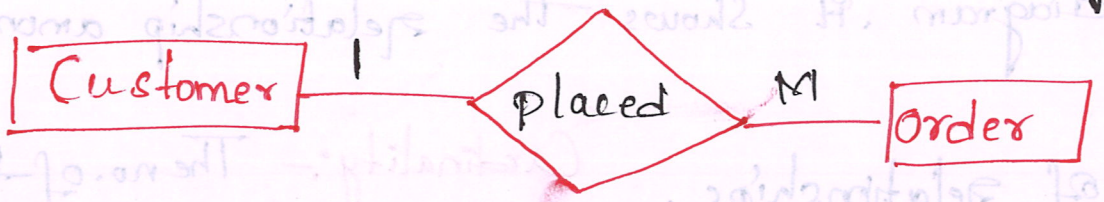
Ex :- A person has only one passport and passport is given to one person.



2) One to many Relationships :-

→ When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship

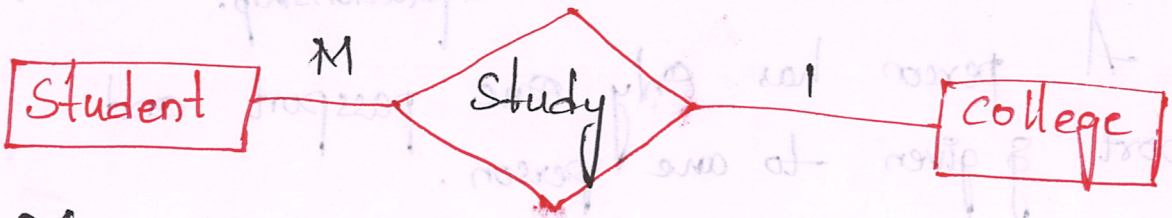
Ex:- A customer can place many orders but a order cannot be placed by many customers



3) Many to One Relationship :-

→ When more than a instances of an entity is associated with a single instance of another entity then it is called many to one relationship.

Ex:- Many students can study in a single college but a student cannot study in many colleges at the same time



4) Many to Many Relationship :-

→ When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship.

Ex:- A student can be assigned to many

Projects of a project can be assigned to many students.



Total Participation of an Entity set :—

⇒ It represents that each entity in entity set must have at least one relationship in a relation-ship set.

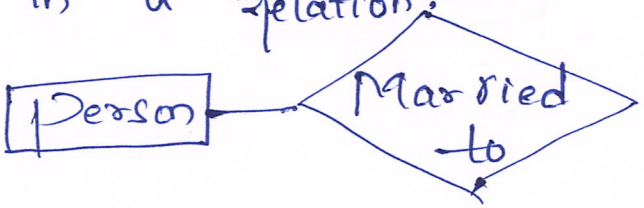
Ex: each college must have at least one associ-ated student

fig: // ER diagram with total participation of college entity set in Study Relationship set. This indicates

Degree of a relationship set :—

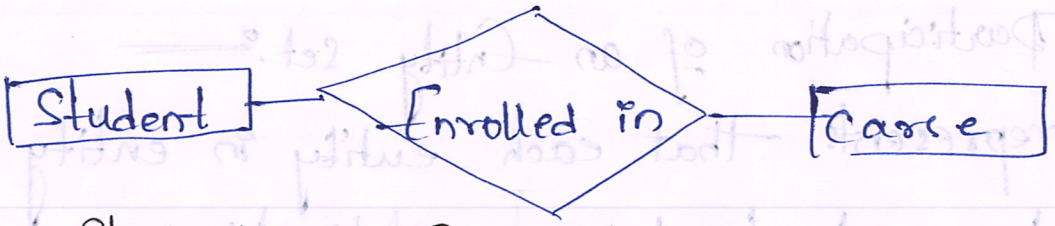
→ It's defined as the no. of different entity set participating in a relationship set is called as degree of a relationship set.

1) Unary Relationship :— Only 1 Entity set parti-cipating in a relation.



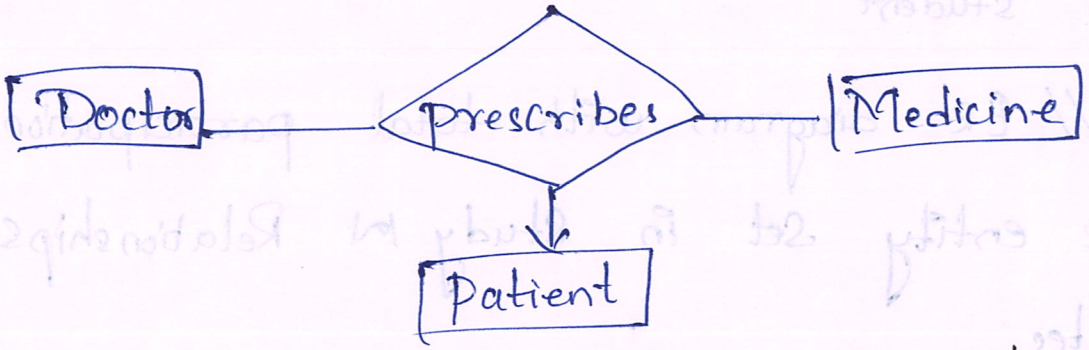
Ex:- One person is married to only one person.

2) Binary Relationship :- When 2 entity sets participating in a relation.



Ex:- Student as Enrolled in course

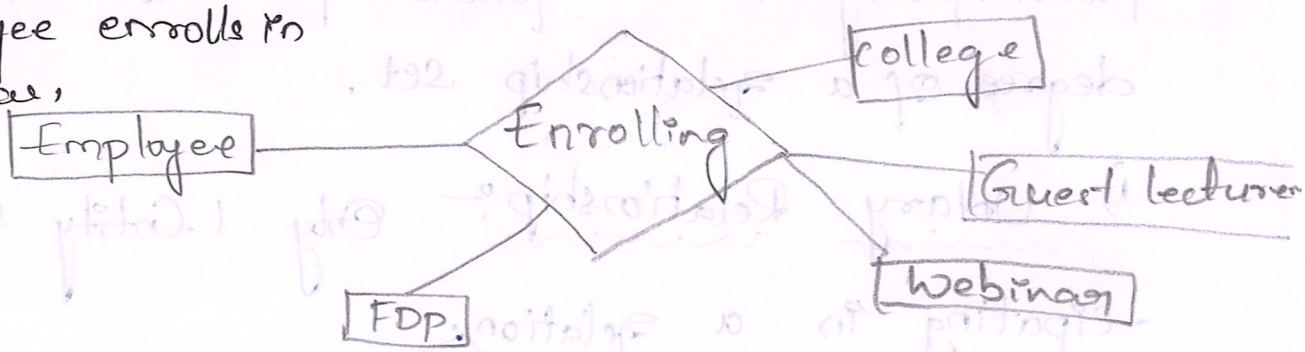
3) Ternary Relationship :- 3 entity sets participating in a relation.



Ex:- Doctor prescribes medicine to patient.

4) N-ary Relationship :- n-entity sets participating in a R.

Ex:- Employee enrolls in FDP, webinar, G-L,



Ex:- ER diagrams

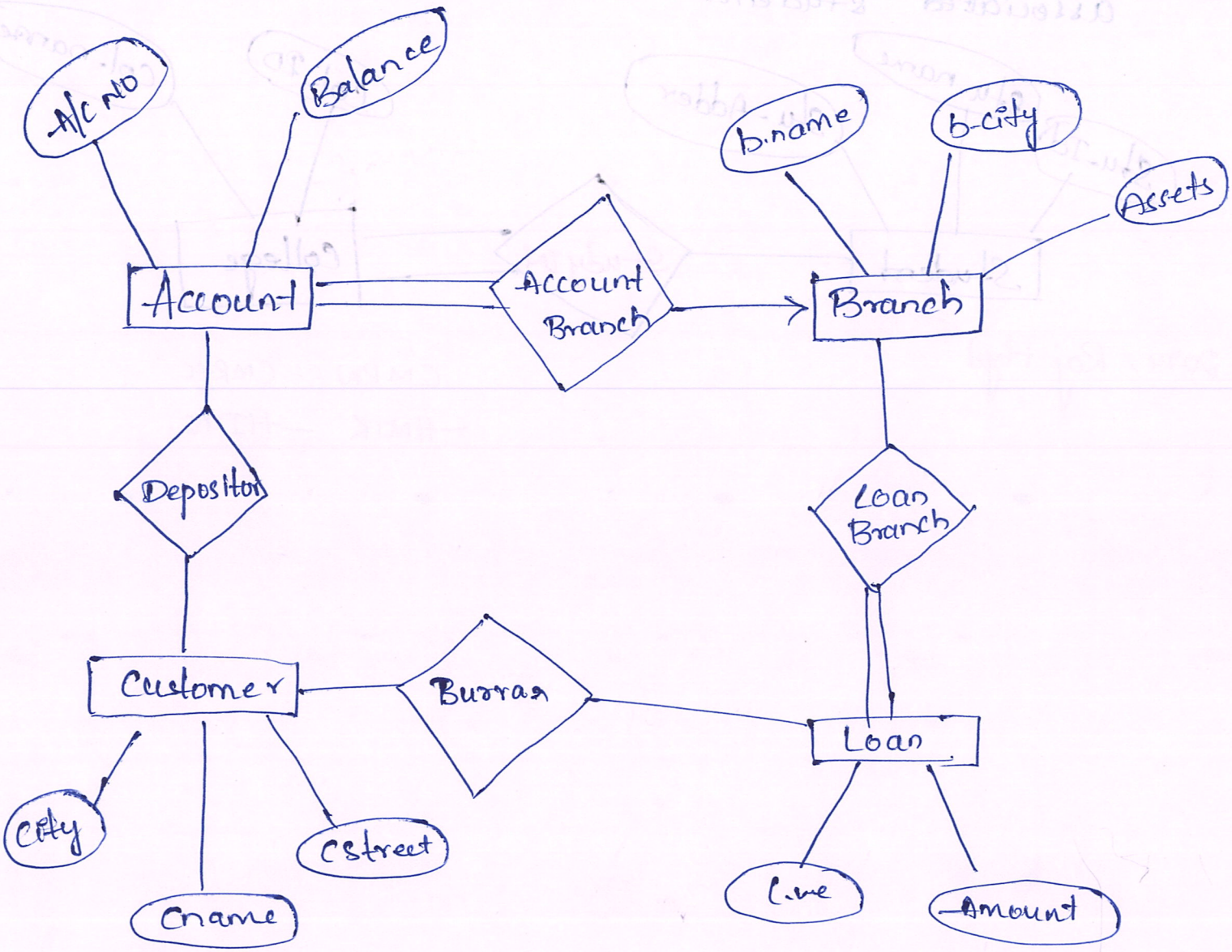
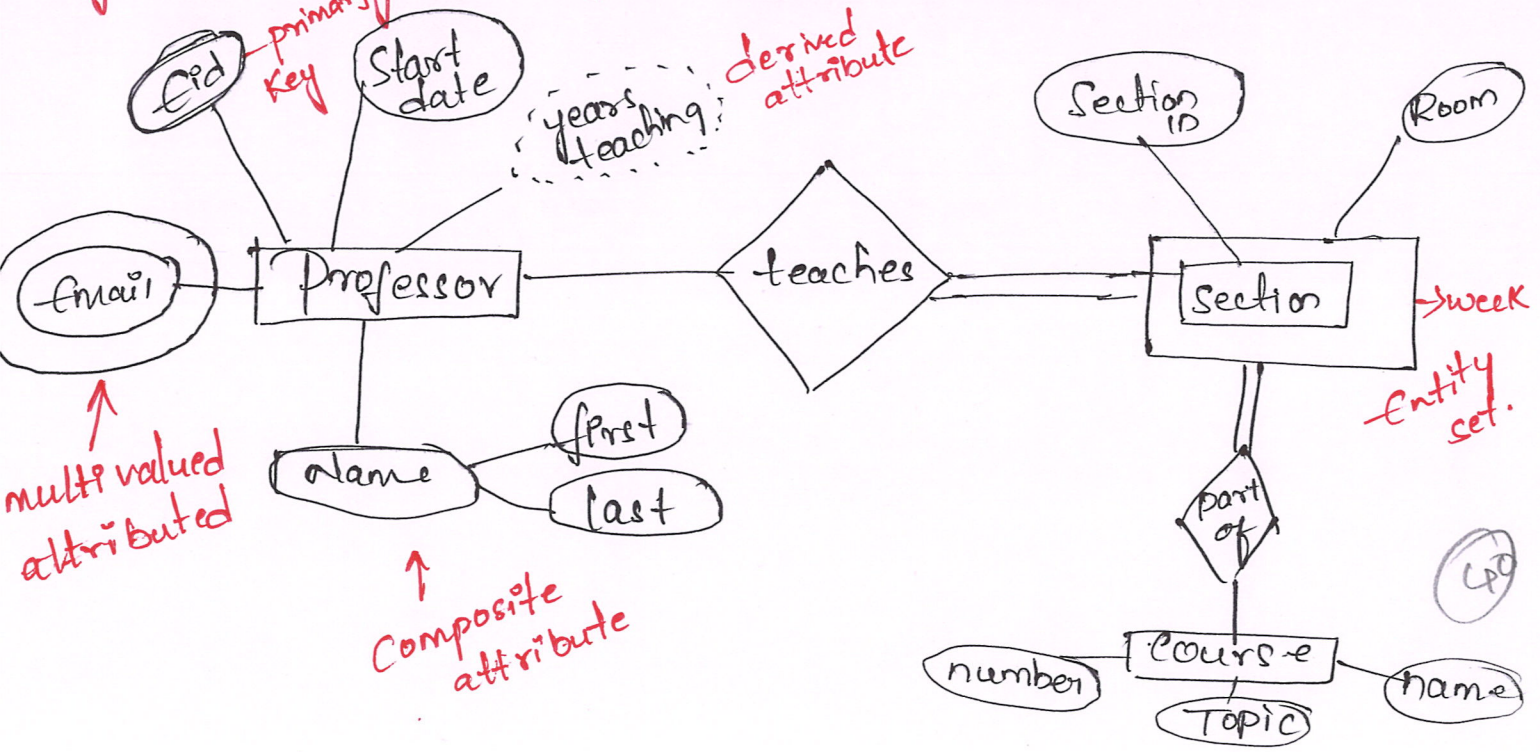
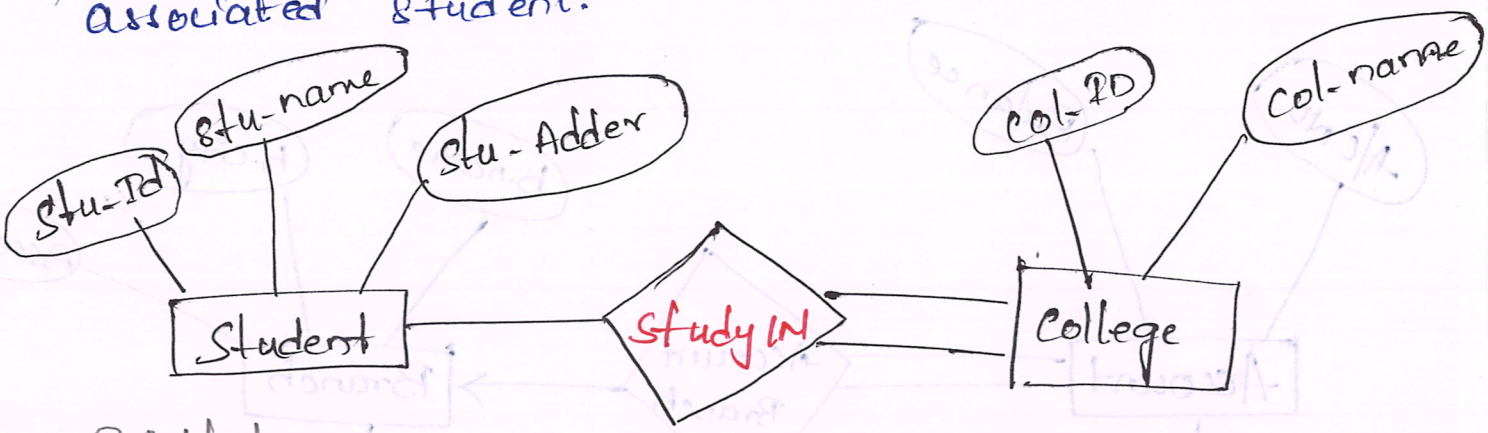


fig ER diagram for Bank details.



that each college must have atleast one associated student.



2074, Raj Hyd

CMRN - CMREC

HMTK - HITS.

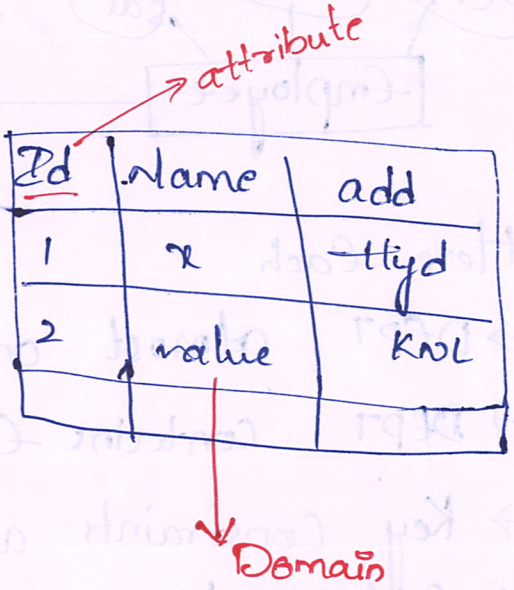
Additional Features of an E-R model :-

1. Key constraints
2. participation Constraints
3. Weak entitieset
4. Class Hierarchies
5. Aggregation.

Constraint = condition.

1. Key Constraints :- / Entity Integrity x Constraint.

→ Every relation has some condition that must hold for it to be a valid relation (table), called Relational Integrity Constraints. (F.K)



→ If the table (relation) satisfied the Constraints then it is said to be valid relation.

* In Key attribute no 2 tuples can have same (identical values)

* (Key attribute no 2 tuples)

Id → Key attribute

Domain (value) Constraint :-

* Every value of an attribute band to have a specific range of values.

1x Hyd
1Y Hyd

x

1 x Hyd
2 x Hyd

✓

Eg: Ph.No - always b/w the 0 to 9

Name - should be only a char value

Id - " " " an Integer

Null	x	Hyd
2	Y	KNL

x

age - should not be -ve or Zero.

→ Key constraint is a condition based on which our ER diagram is constructed

→ For implementing the K.C on binary relationship sets we can represent a thick line from one entity set towards the relationship sets by having the arrow mark.

Ex:- Consider a "manager" relationships with Employee Entityset & Dept entity set.

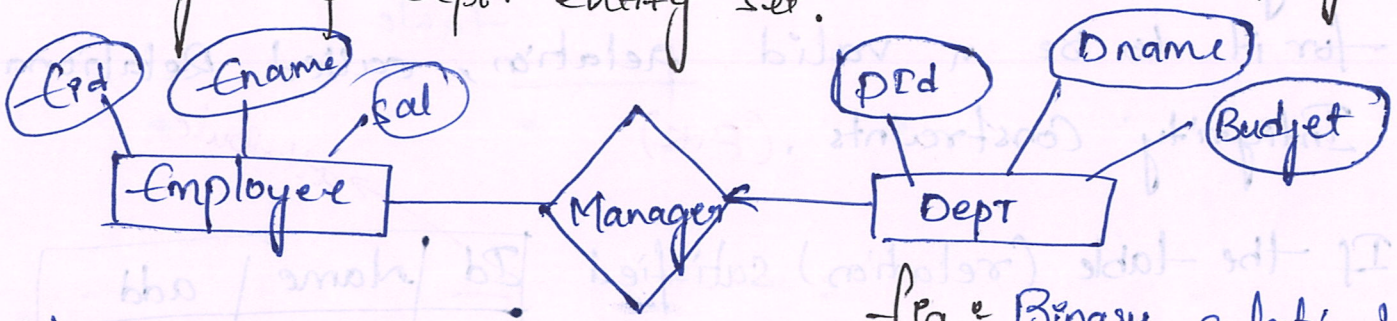


Fig:- Binary relationshipset

Here each

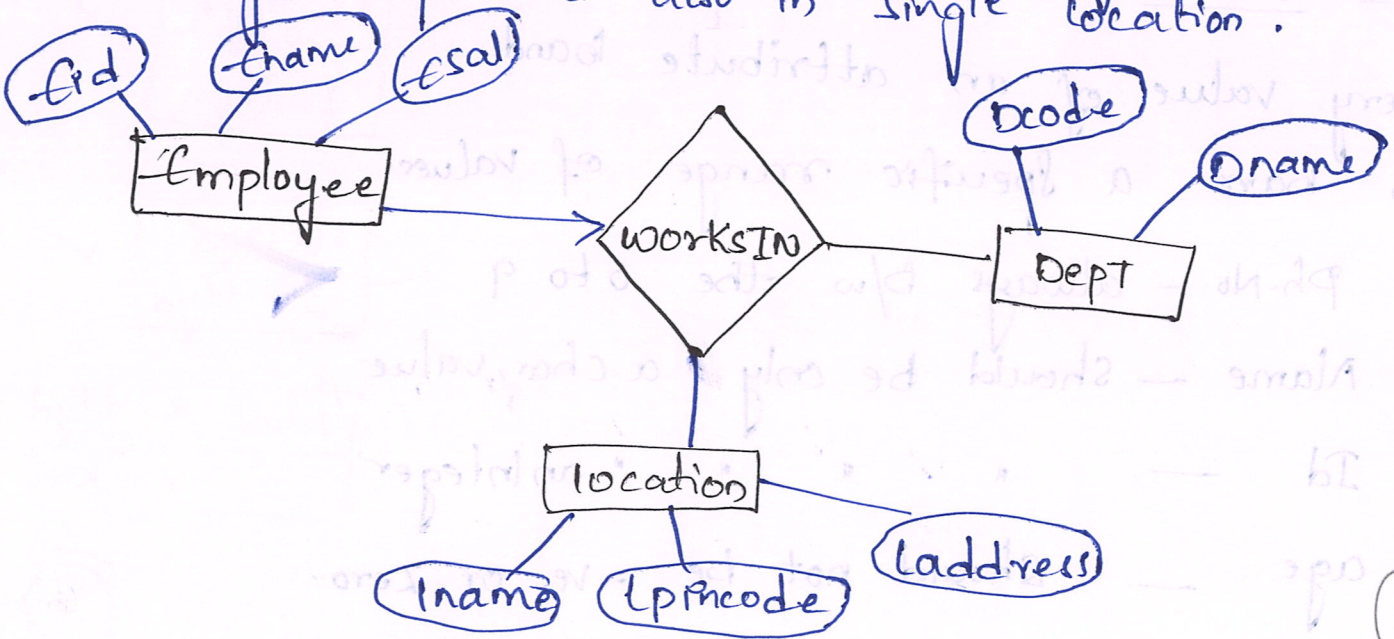
→ DEPT almost one manager an an employee

→ DEPT contains Employee & they manager the Dept

→ Key Constraints an only one entity set

→ Each and every dept has atmost one manager.

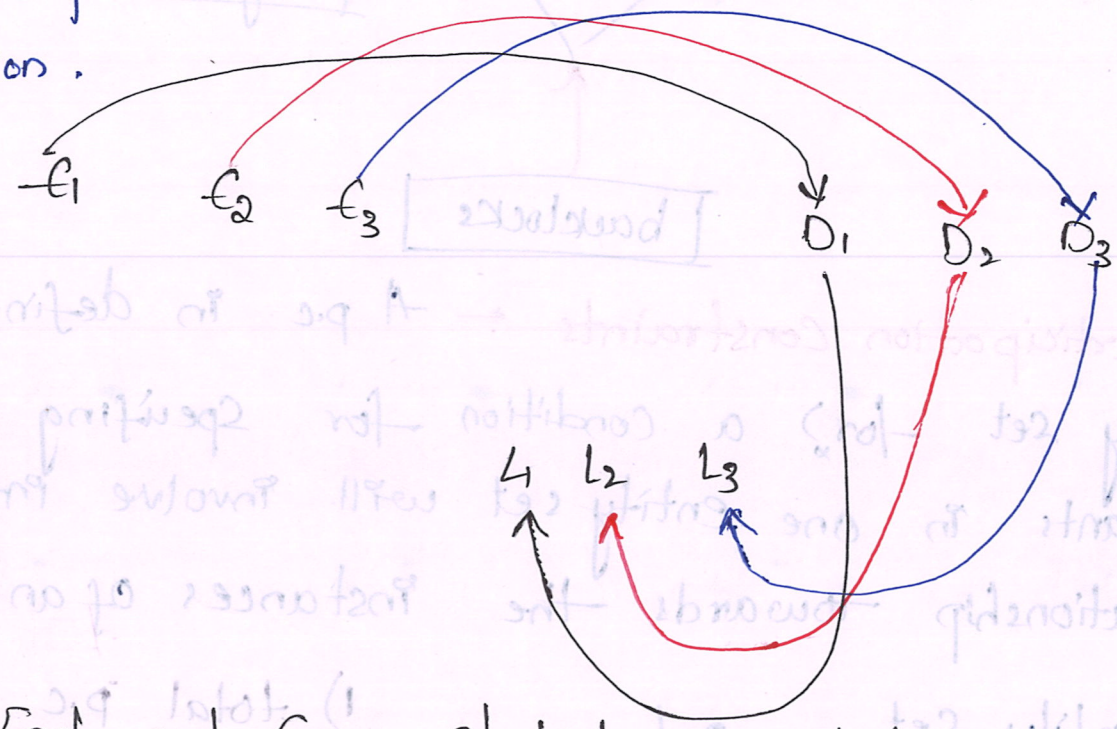
Ex:- Ternary Relationshipset :- Each employee can works in Single dept and also in single location.



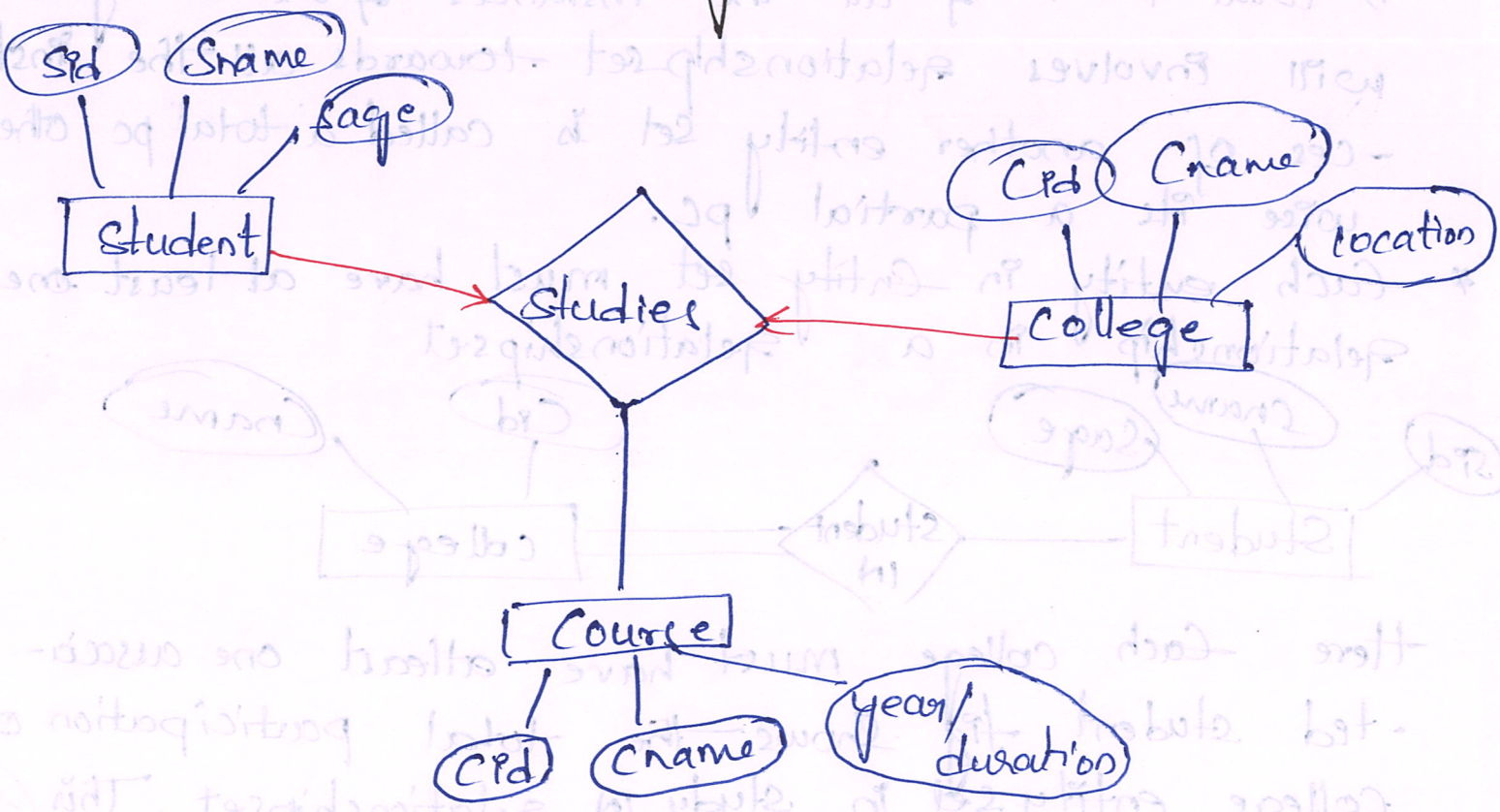
Here

→ Key Constraints is any one of the entity set that is common to all other 2 entity sets.

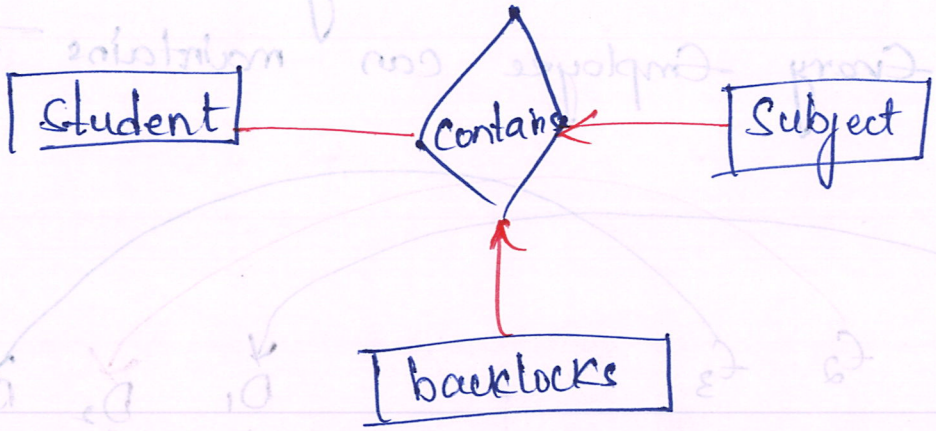
→ Each and Every Employee can maintains the single Dept of location.



Ex: Each and Every student can studies the same Course in various colleges.



Ex:- Each and every student can maintain different subjects with different backlocks.

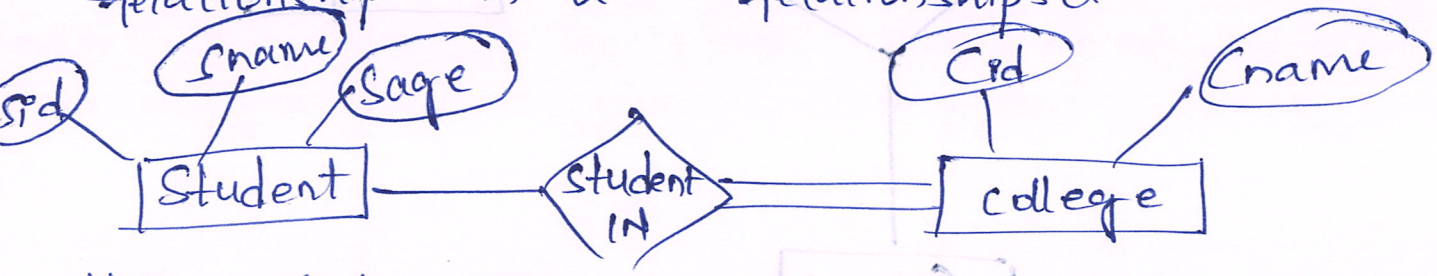


2) Participation Constraints :- A p.c in defined as entity set for a condition for specifing the instants in one entity set will involve into the relationship towards the instances of an another entity set. 2 types.

1) total p.c
2) partial p.c

1) total p.c :- If all the instances of one entity set will involves relationship set towards all the instances of another entity set is called a total pc otherwise its a partial pc.

* Each entity in entity set must have at least one relationship in a relationshipset



Here each college must have atleast one associated student Sid shows the total participation of college entity set in study IN relationshipset. This indicates that each college must have atleast one associated student.

Ex:

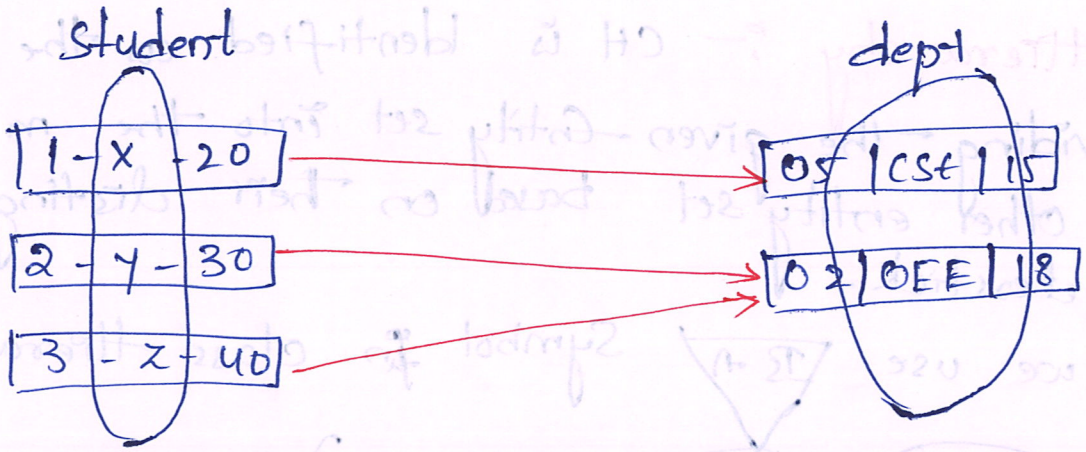


fig :- total participation Constraints.

Ex:

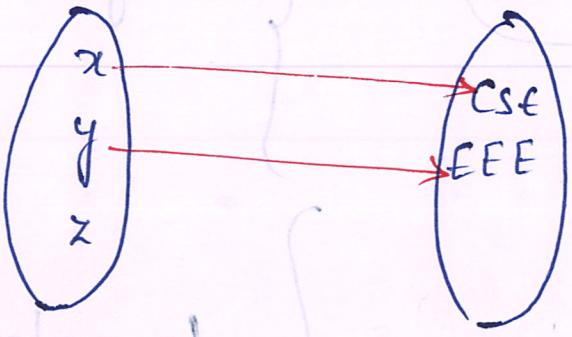


fig :- Partial participation Constraints.

3. Weak Entity set :- Any Entity set that does not contain any primary key then it's said to be weak entity set otherwise it's a strong Entity set

Primary Key :- Minimal set of attributes that are uniquely identifying a tuple but which are not maintaining the same information & it can't allows null values

→ Primary key is unique (1) for each table

Ex:- Driver license number, pan number, adhar number, vehicle identification number (VIN)

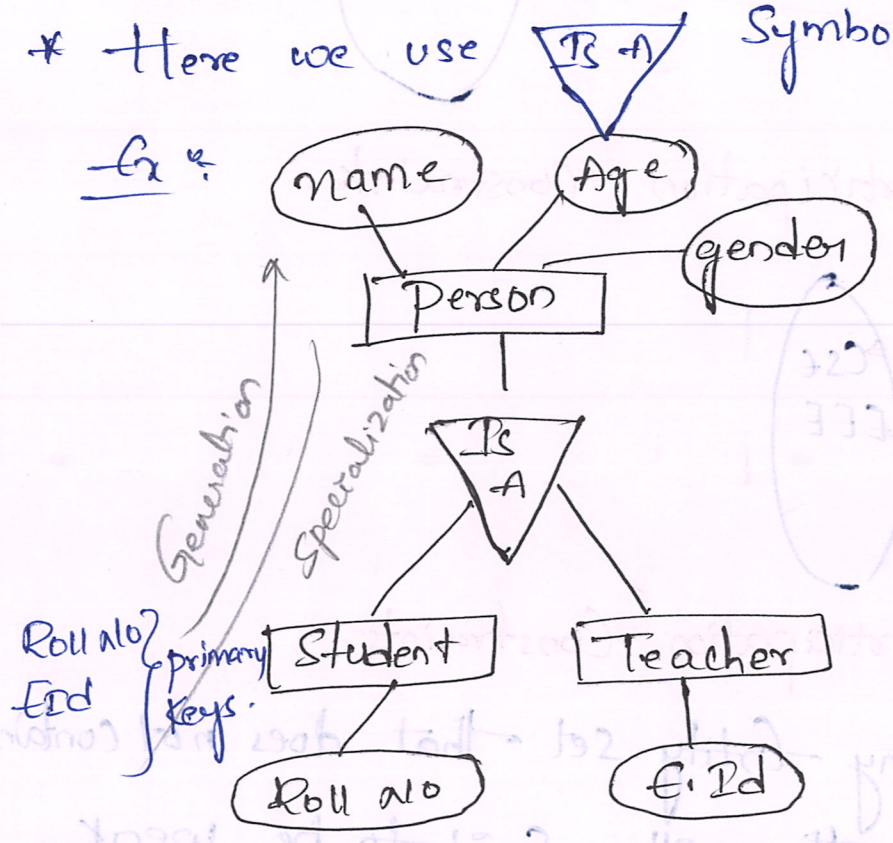
Ex:- A bank a/c cannot be uniquely identified without knowing the Bank details to which it belongs to



4) Class Hierarchy :- CH is identified as the process of dividing the given entity set into the multiple no. of other entity set based on their distinguish chareeteristick.

* Here we use Is A Symbol in class Hierarchy

Ex :-



Here

A person is a student having Roll no & A person is a Teacher having Id.

A Student is a person having name, age, & gender

A Teacher " " " name, age, gender.

1) Generalization property

→ It is a process in which the common attributes of more than one entities set form a new entity set. This viewly formed entity is called generalized entity set.

→ Its a buttom up approch.

2) Specialization property

→ Its is an process in which an entity set is devided into sub entities set based on their Chareeteristics

→ Reverse process of generalization

→ In this two entities combine together to form a new higher level entity set)

→ It a stopdown process.

1) Generalization example :-

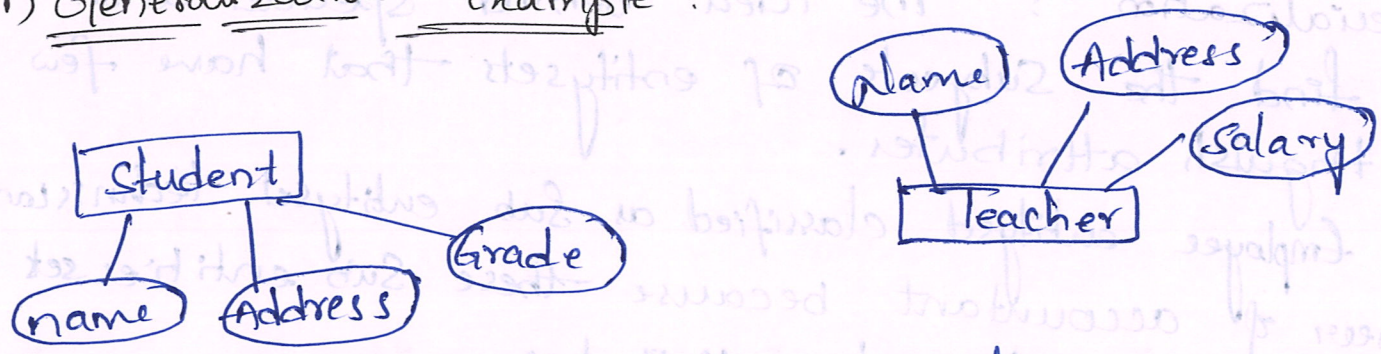


fig : Before generalization.

These two entity sets have two common attributes name & Address. we can make a generalized entity with these common attributes.

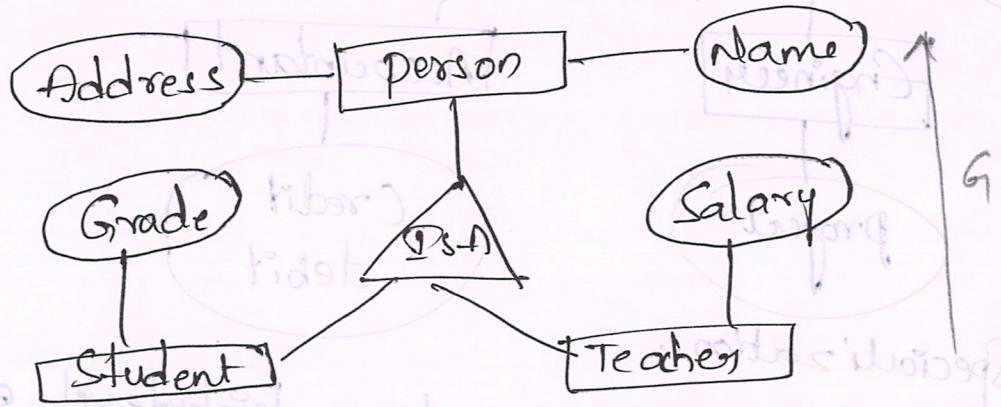


fig : After generalization.

* Generalization uses bottom-up approach where two or more lower level entities sets consume together to form a higher level new entity set.
* the new generalized entity set can further combine together with lower level entity set to create a further high level generalized entity set.

2) Specialization :- The idea behind specialization is to find the subsets of entitysets that have few distinguish attributes.

Ex:- Employee entityset classified as sub entityset technician engineer & accountant because these sub entities set have same distinguish attributes.

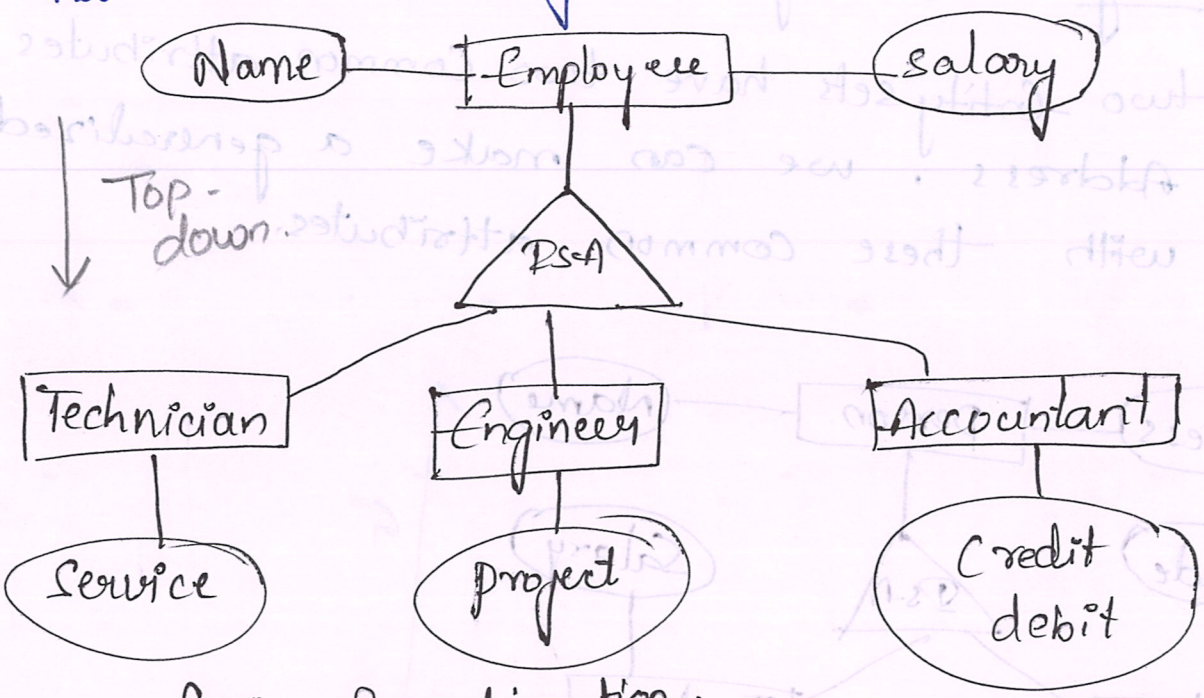


fig :- Specialization.

→ In the above diagram, we have higher level entity set. "Employee" which we have divided in sub entityset "Technician", "Engineer" "Accountant".

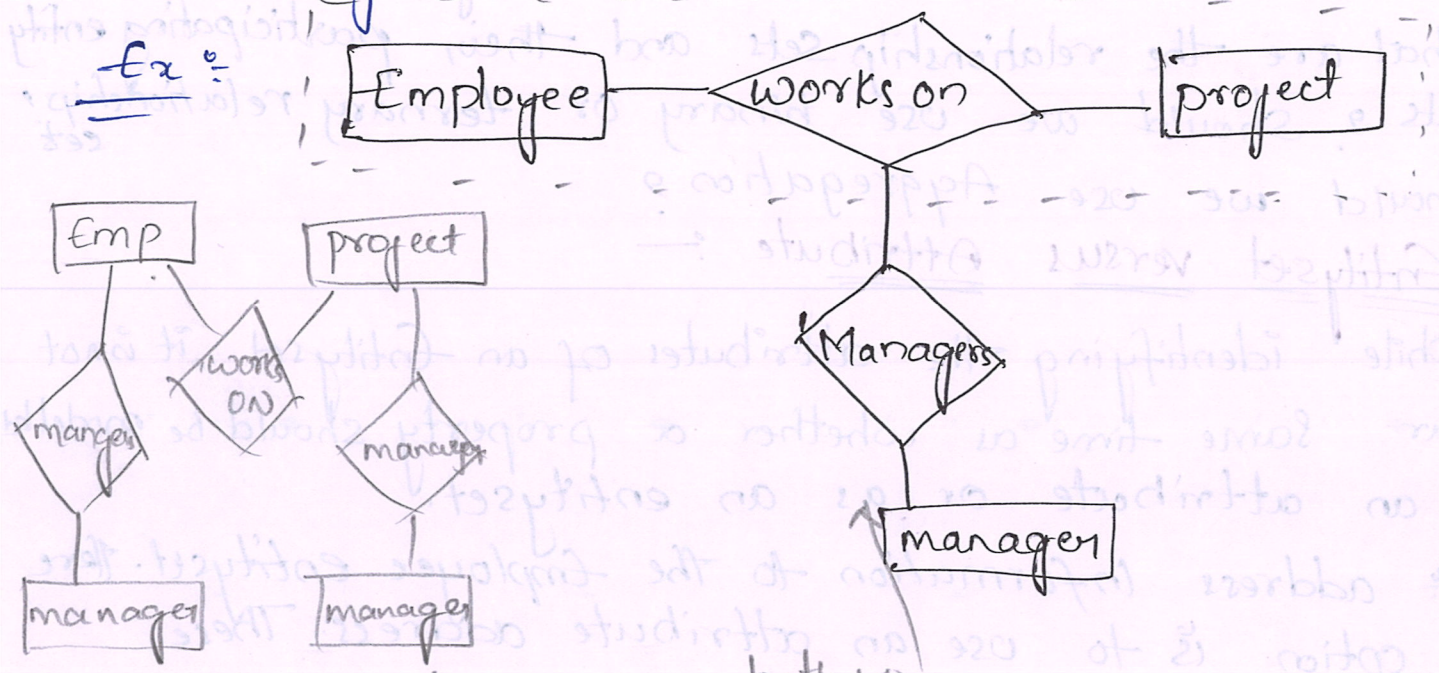
→ All of these are just an employee of a company however their role is completely different & they have few different attributes.

→ Technician handles a service requests Engineer works on a project & Accountant handles the credit & debit details

→ All these 3 employee types have few attributes common such as name & salary & parent entityset employee.

5) Aggregation :- is a process in which a single Entityset alone is not able to make Sense in a relationship so the relation ship of 2 entitysets acts as one entityset

Ex :-



→ Manager has to manage both so

→ Here the manager not only manages the employee working under them but he has to manage the project as well

→ In such scenerio if entity set manager makes a manager relationship with either employee or project entity set alone then it will not make any sense bcz he has to manage both.

→ In this case the relationshipset of two entity sets acts as one Entity set

→ In our employee the relationshipset "works on" b/w "Employee" & "project" acts as one entityset that has a relationship set "Manages" with the entityset "manager".

* The process of combining 2 or more entity sets and the combination can be treated as single entity set.

→ before aggregation it is ternary relationship set

→ after " " Binary " " "

Conceptual design with ER model :- Developing an ER diagram presents several choices, including the following

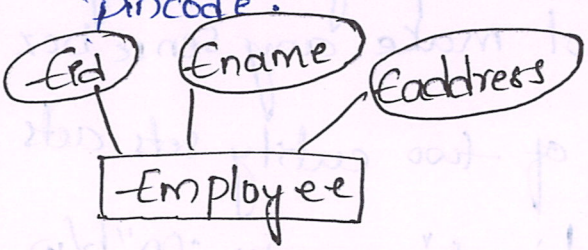
1) Should a concept be modelled as an entityset or an attribute
2) " " " " " entity set or a relationship set
3) What are the relationship sets and their participating entity sets ? Should we use binary or ternary relationship set
4) Should we use Aggregation ?

i) **Entityset versus Attribute** :-

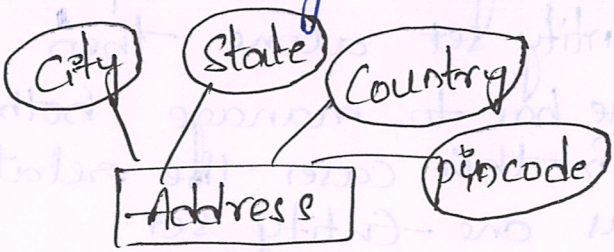
→ while identifying the attributes of an entityset, it is not clear same time as whether a property should be modelled as an attribute or as an entityset.

Ex : address information to the employee entityset. Here one option is to use an attribute address. These option is appropriate if we need to record one address pr employee & it think of an address as a string.

→ An alternative is to create an entity set called address & relationshipset "Has", attributes city, state, country, Pincode.

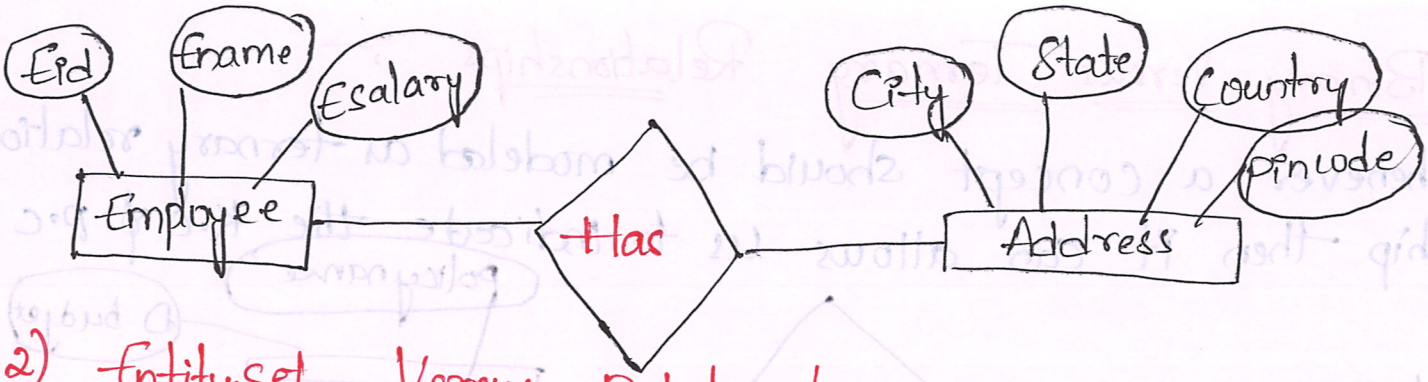


⇒



Eid	Ename	Eaddress
2074	Ray	Hyd
2086	Laxmi	Kurnool
2094	mahi	ECIL

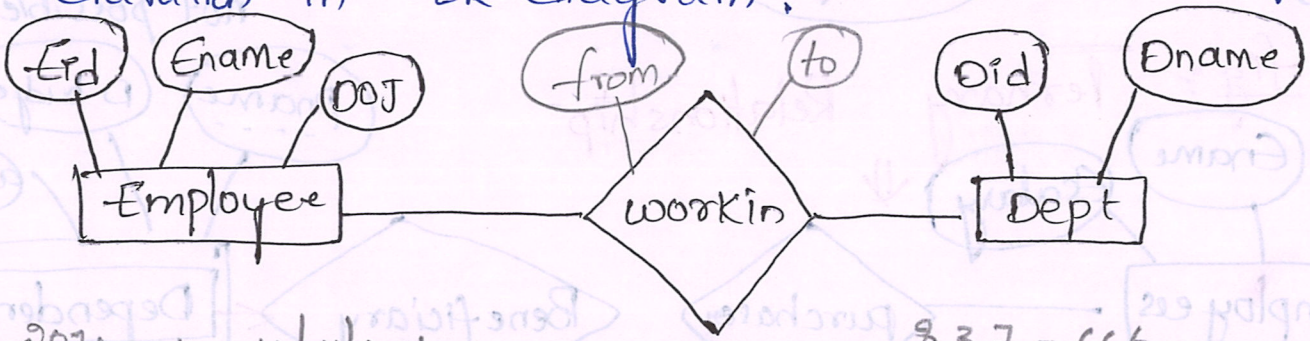
city	state	country	pincode
Hyd	Telangana	Indian	500083
Knl	AP	India	518401



2) Entityset Versus Relationshipset :-

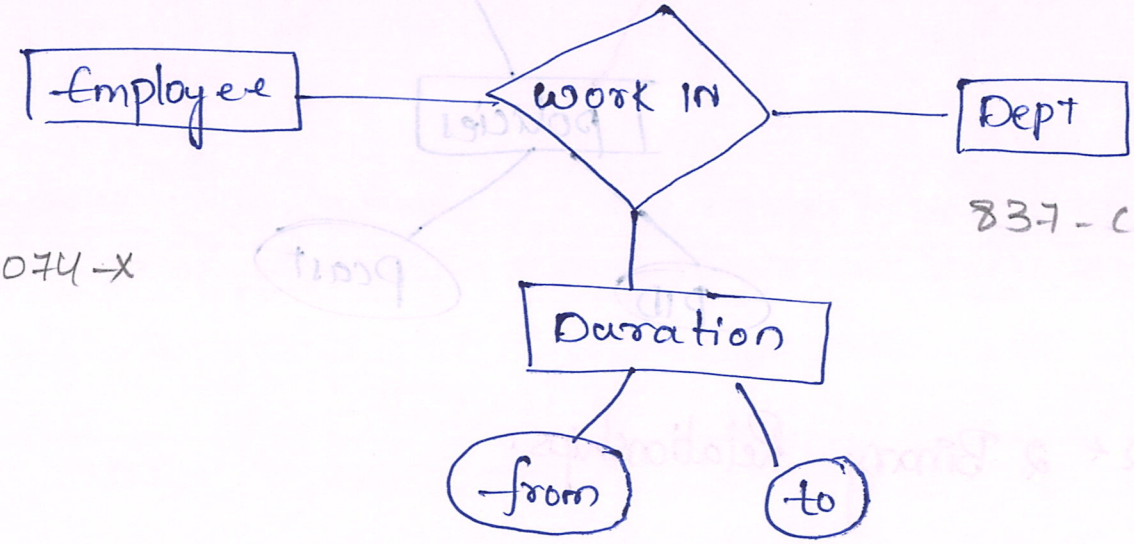
→ When ever a given concept can be modelled as relationship set then it cannot allows the multiple duration for the same dept in which the same employee. So

→ for allowing the multiple durations to the employee who works in the same dept. there instead of relationshipset we can introduce a new entity set called 'duration' in ER diagram.



2074- X- 11/11/2014

837 - CSE



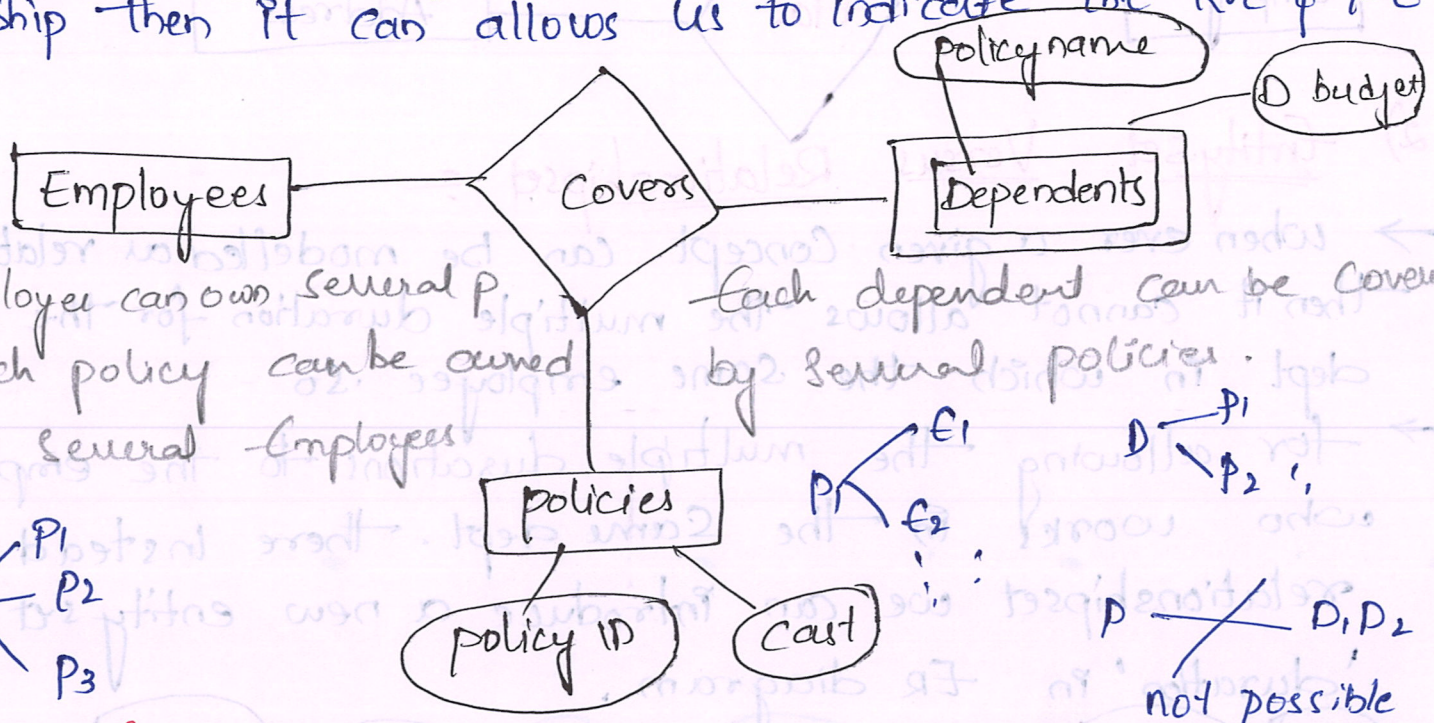
2074-X

837 - CSE

11/11/2014 30/5/2019

3) Binary Versus Ternary Relationships :-

→ Whenever a concept should be modeled as ternary relation -ship then it can allows us to indicate the K.C & P.C



Employee can own several P. & each policy can be owned by several Employees

Each dependent can be covered by several policies.

fig :- Ternary Relationship

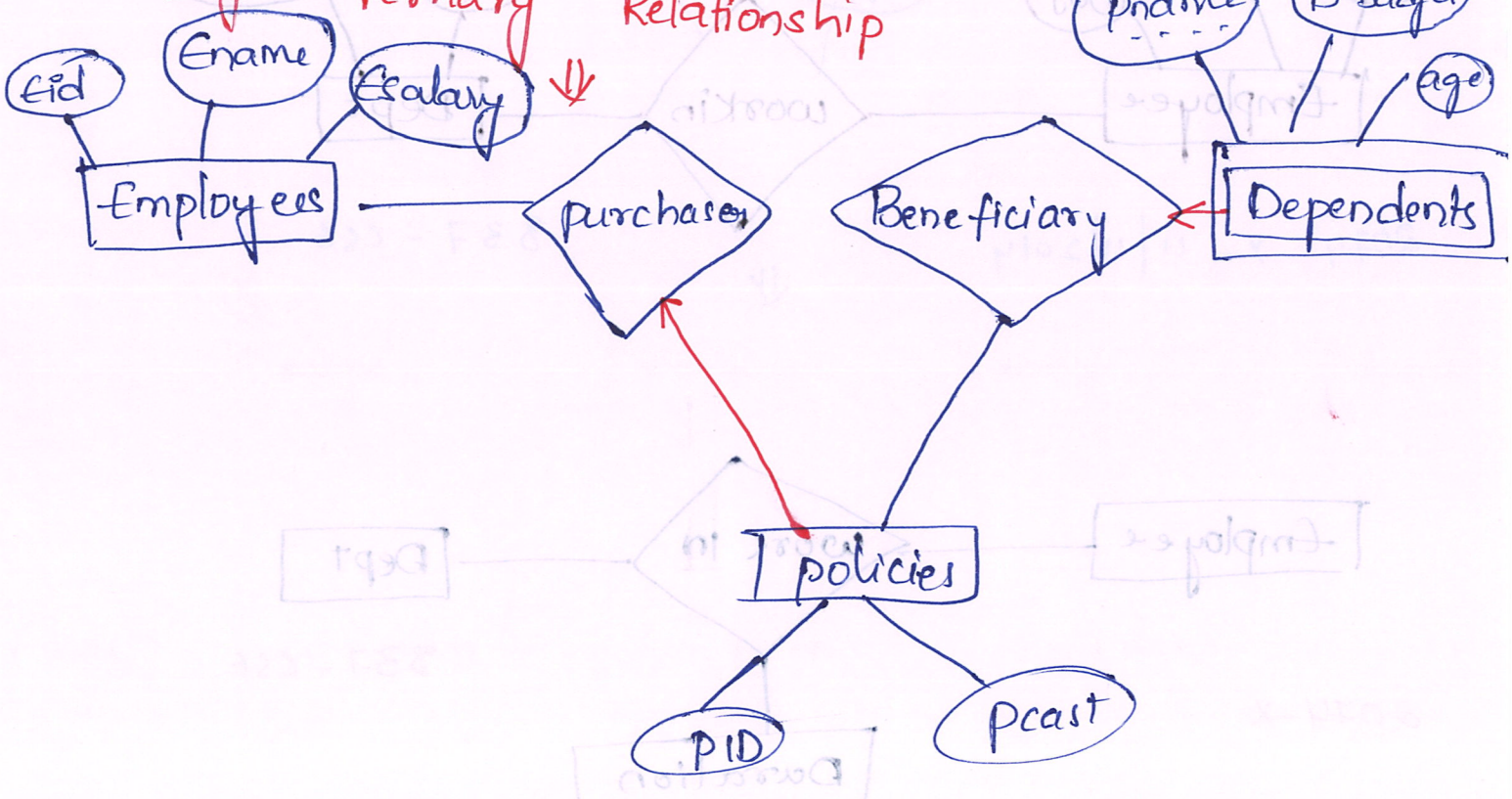


fig 2 :- 2 Binary Relationships.

In fig employee can own several policies, each policy can be owned by several employees. & each dependent can be covered by several policies.

Suppose that we have the following additional features / requirements:

i) A policy cannot be owned jointly by 2 or more employees
ii) Every policy must be owned by some employee
iii) Dependents is a weak entity set, & each dependent entity, sets is uniquely identified by taking p name in conjugation with the policy ID of an policy entity set.

5. Aggregation Versus Ternary Relationships :-

→ when ever the concept should be modeled as an aggregation in which if we can identify the unnecessary relationship then we can eliminate them and translate the given ER diagram with an aggregation to without aggregation.

→ use a ternary relationship instead of aggregation

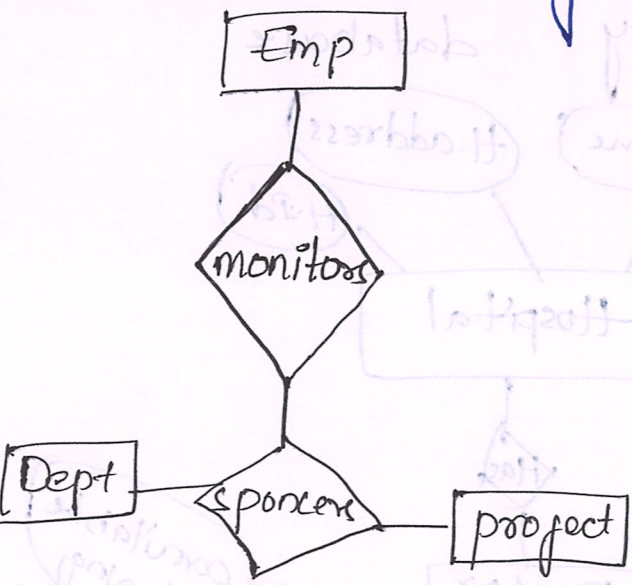


fig : with aggregation.

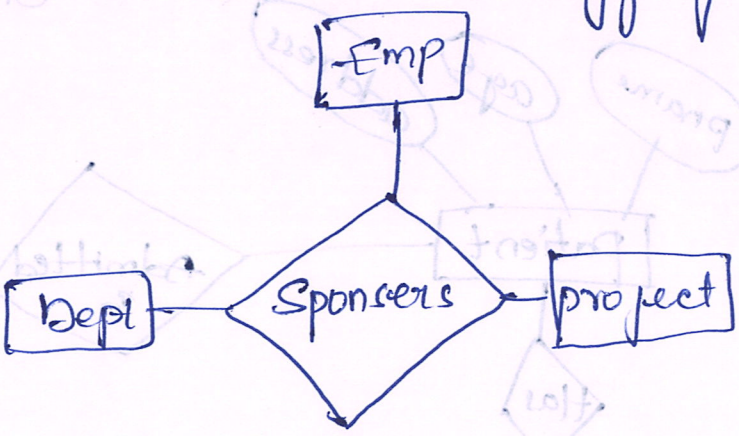


fig : without aggregation ternary relationship

with (Binary)

Conceptual design for large Enterprises :-

→ The process of describing small fragments of application code interms of ER diagram is said to be a conceptual design.

→ C.D for large enterprises can require more than one designer and span the data & specify application code for various different user group.

→ The important aspect in C.D for large enterprises is too define the methodology to structure overall design.

→ The design process can covers all the requirements or conflicts can be reserved and move towards the Conceptual phase & logical phase.

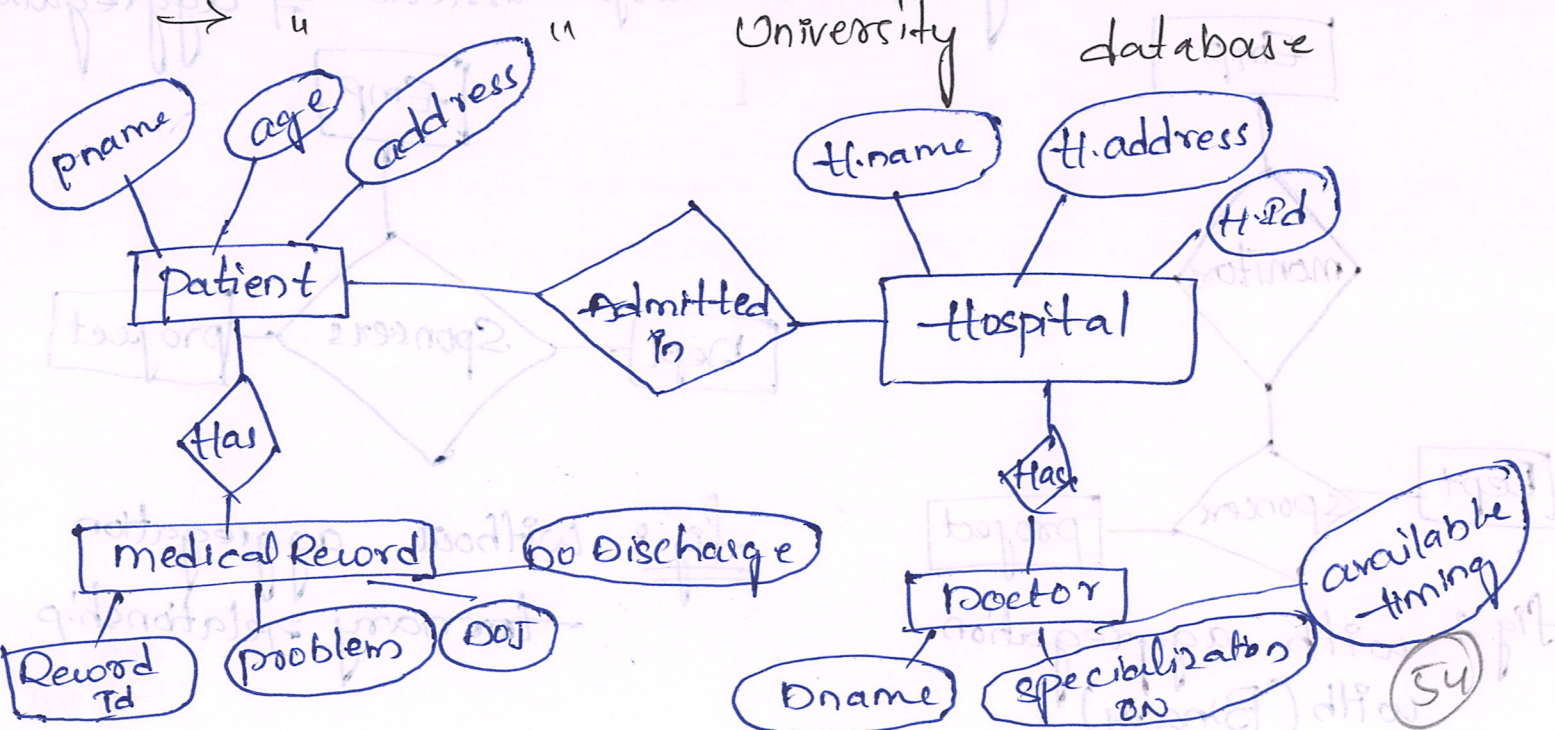
→ The alternative way of representing the conceptual design for large Enterprises is schema integration.

Ex:- ER diagram of school management system.

→ ER diagram of school management system.

→ ER " " hospital "

→ " " University database



Types of Relationship sets :- Based on no. of Entity sets.

1) Unary — one Entity set
2) Binary — two " "
3) Ternary — 3 "
4) n-ary — 3+ "

History of DBS :-

1) Early 1950's and late 1960's "Magestic tapes" (punch cards)
2) Early 1960's and late 1970's "Hard disk"
3) In 1970's — Relational model (Codd Invented data in the form of tables)
4) In 1980's — "System-R-design"
5) In 1990's — SQL (Queries)
6) "www" late 1990's (to maintain large amount of data bases)

Integrity Constraints over relations :-

1) Entity Integrity Constraints / Key Constraints
2) Referential Integrity Constraints / Foreign Key Constraints
3) Domain Constraints
4)

Key = attribute / 1 or 2 attributes.

Super Key :- Single or Combine of attributes we can identify a read in a table

→ set of attributes within a table that can uniquely identify each record with in a table

Here Id & ph.No are called as Super key. bcz by using this Id, we can identify each record in a table. & No student can have same Id as well as ph.No. So we can easily find any information about student.

Student

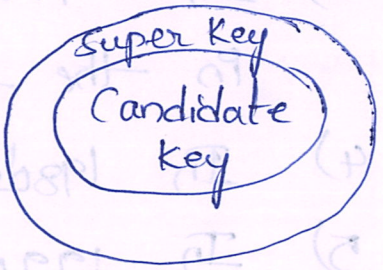
Id	Name	ph.No	Addre
1	X	94321	KNL
2	X	8686..	Hyd
3	Y	930...	ECIL

→ Super key is a Superset of Candidate Key.

Here {Id} as Super Key

{Id, name} is Super key → Name same but Id can't be same

{ph.No}



Candidate Key :- 1 or 2 attributes.

→ A Super key without Redundancy is called candidate key

→ It is not reducible (devide) further

→ Minimal set of attributes used to uniquely differentiate record of the table (take Single (or) Combination of 2 attributes to identify a record in a table)

→ C.K is a Subset of SK

SId	SName	ph.No	Address
1	X	94348..	KNL
2	X	756234..	Pune
3	Y	652720..	Hyd
2	Y	830402..	mumbai

Here
{sid, Sname}
Candidate Key
{ph.No, Address}

SId	SName	ph.No	Address
1	X	9493 48..	Hyd
2	Y	756234..	KNL
3	X	652720..	KNL
4	Z	830402..	Chennai

SId → Candidate Key
ph.No → " "
{Name, Address} → Note a Candidate Key bcz redundancy

→ Candidate Key can not be null /empty & value should be unique

→ more than one (2) C.K for a table (single)

Relational Model

→ A Relation is a table with columns & Rows.
→ It is based on the mathematical concept of relation which is represented physically as table
→ It is used for data storage & processing.

Student

	SID	&Name	DOB
t_1	1	X	9-6-1996
t_2	2	Y	8-6-1980
t_3	3	A	4-6-1976
t_4	4	B	4-6-2006

Here
→ table is a Relation
→ Student is a table name (or) Relation name
→ SId, Sname, DOB → attributes / Fields
→ Each row called as a tuple
→ Each value called as a Domain.

Degree = 3
Cardinality = 4

Informal terms — Formal terms (DBMs)

table → Relation
Column header / Field → Attribute
All possible column values → Domains
Row → Tuple / Entity / Instance
table definition → Schema of Relation.

Degree of a Relation : 3
→ The no. of attributes / no. of column heads in the R.

Cardinality of a Relation : 4
→ The no. of tuples (or) Records (or) entity in a R.

Properties of Relation :
→ tablenames distinct (different) from all the names in the relationship schema.

→ Each cell of the Relation contain Exactly one value
→ Each attribute name is distinct.
→ Each tuple (row) is distinct. There are no duplicate tuple [Suppose a col may be same but Complete row should not be same].

1 x Hyd
2 x Hyd

→ The order of attributes has no significance.
→ " " " tuples has no ",

Referential Integrity Constraints :- / Foreign Key Constraints :-

→ Every Relation has same condition that must hold fruit to be valid relation then it is said to be a R.I.C. → It checks the conditions to satisfied in more than one relation (2 or more tables) connected with same relationship.
→ If the table satisfies the constraints then it is used said to be a valid relation.
→ It is a condition based on which the Inserted values in the referencing relation must be referal from the other referenced relation or not
→ R.I refers to the accuracy and Consistency of data with in a relationship set.
→ In a relationship set data is linked b/w 2 or more tables.
→ This is achieved by having the foreign Key (in the associate table) reference a primary key value (in the primary or parent table)

→ So the R.I requires that whenever a Foreign Key value is used it must reference a valid existing Primary Key in the parent table.

→ Adding records to a related will prevent users from:

1) Adding records to a related table, if there is no associated records in the primary table.
2) Changing values in primary table that results in orphanned records in a related table.
3) Deleting records from a primary table if there are matching related records.

* Referential Intigrity is a Subset of data Integrity which is Concerned with the accuracy and Consistency of all the data)

Primary Key :-

→ minimized set of attributes that are uniquely Identifing a tuple in a given table. but which are not maintaing the same information & it cannot allows null values.

→ Primary Key is unique(1) for each table

Syntax:- Create table student (sno int, sname varchar(10), age real, constraint Student Key Primary Key (s.no));

Name of constraint

SQL > desc student;

Name	Null ?	type
S.no	not Null	int
Sname		varchar(10)
age		real

SId	Name	age
123	X	15
356	Y	16
473	Z	17

123 Null 15 X

123 356 not P.K (59) X

Foreign Key :- A.F.K is defined as the minimum set of attributes, which are referencing from the other referenced relation and there " " attributes must be the Primary Key attributes in referenced relation.

Syntax :- Create table MECH (Roll NO number, marks Integer, Grades char(5) Constraint MECH key primary Key(S NO) Foreign Key (Roll NO) References student(S NO));

Student

S.NO	SName	age
123	X	15
356	Y	16
473	Z	17

MECH

Roll NO	Marks	grade
123	25	A
356	24	A
473	23	A

S.NO → Primary Key in Referenced relation

Roll NO → F.K (referenced form student (S NO) Referencing relation.

Sailors table :- Create table sailors (sid int, sname char(10) age real , rating int, Constraint MECH primary key sid);

Boats table :- Create table boats (bid int , bname char(10), bcolor char (6), Constraint MECH primary key bid);

Reserves table :- Create table Reserves (Rsid int , Rbid, int day date , constraint CMREC primary key (Rsid , Rbid ,day), Foreign key (Rsid) References , Sailors (sid) , Foreign key (Rbid) references .boats (bid));

Sailors

Sid	Sname	age	rating
22	Dustin	45.0	7
29	Brutus	33.5	1
85	Art	25.5	3

MECH → Constraint name

Sid → primary key

Boats

bid	bname	bcolor
101	Interlake	blue
102	Interlake	red
103	clipper	green
104	marine	red

MECH → C.Name

bid → primary key

Reserves

Rsid	Rbid	day
22	101	10/10/98
29	102	4/6/98
85	103	9/5/98

CMREC → Constraint name

Rsid , Rbid → Foreign keys.

⇒ cause / rules

Enforcing Integrity Constraints :-

→ Enforcing IC is defined as the command that are executed for violating the given IC. an relation scheme they are 2 ways for Enforcing primary key constraints as follows

1) whenever we have Inserting row into the given relation after insertion, the same row is trying to insert one or more time into the given R.

2) when ever we are Inserting only value for the given P.K attribute then the instance can contains a Null value for Non Inserted attribute

1	x	20
1	x	20

Various referential Integrity Enforcement Steps :-

1	x	20
2	Null	30

1) At the time of Inserting a new row in the referencing relation we must verify that the Inserted values of a foreign key attribute is presented in the referenced relation or not ?

2) There is no alternative way we have for performing an update command an referenced relation.

3) There is no alternative way we have for executing the drop & delete command on referenced relation.

Table constraint :- Its a condition on specific attribute can allows only specific range of values check (condition)

→ It specifies the condition over a single table. (62)

Ex:- Create table student (sid int, sname chars) age real check age < 20) ;

Assertion :- Implementation of are over a multiple no. of tables.

logical database design :- Here we convert the given ER diagrams into relations or tables.

1) Entity sets to tables
2) Translating the relationshipsets without Key Constraints
3) " " " with "
4) " " " with participation Constraints
5) Translating the weak entity set
6) " " class Hierarchies
7) " " ERdiagrams with Aggregation.

Views :- It is a table in which all rows explicity stored in the database.

Syntax :- Create view viewname as Select < list of attributes > from < list of tables >;

Ex :- Create view CSE as Select Sid, bid from Sailors Boats;

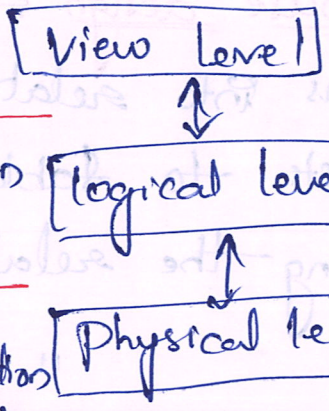
Data Independance :-

→ Capacity of Changing the Schema at one level without affecting the other levels.

→ Mapping b/w any 2 abstractions.

→ All DB systems are designed in multilayer

→ DBS stores data and data (meta data).



logical data Independence
(mapping b/w view level abstraction & logical level Abstraction)

Physical data Independence.
(mapping b/w logical level abstraction & physical level " ")

fig 1

It has the ability to change the physical data without affencing the h.. schema and external schema.

Database design at physical level → physical schema
" " logical " → logical "
" " view " → External "
" " External level (Application program)

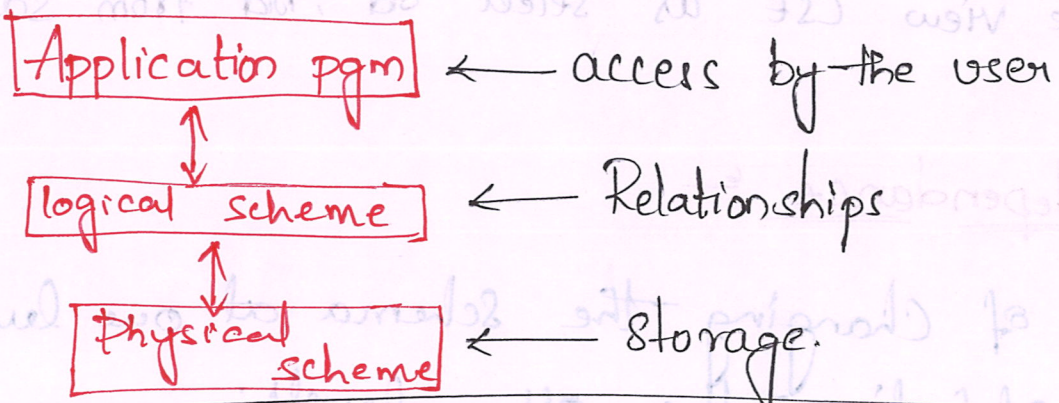
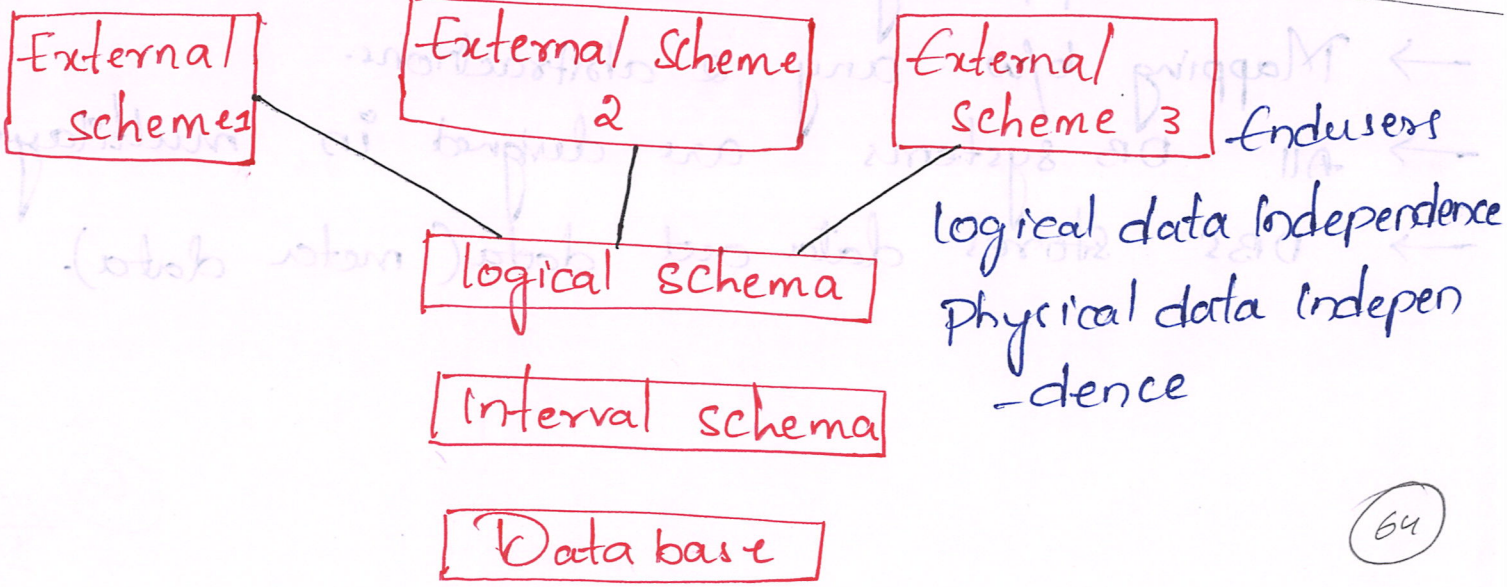


fig 2



logical data Independence
Physical data Indepen-dence

View :-

→ The dynamic result of one or more relational operating on the base relations to produce another relation is called view.

→ A view is a virtual relation that doesn't necessarily exist in the database, but can be produced upon request by a particular user at the time of request.

→ A view is object that gives the user a logical view of data from an underlying table or tables. you can restrict what users can view by allowing them to see only a few columns from a table.

Purpose of view : It is desirable for several reasons

→ It simplifies queries

→ It can be queried as a base table

→ It provides a powerful and flexible security mechanism by hiding parts of the database from certain users

→ It permits users to access data in a way that is customized to their needs, so that the same data can be seen by different va users in different ways at the same time.

Syntax : SQL> create view v_book as select title, author_name from book;

View created

SQL > select * from v_book;

title	author_name
Oracle	Arora
DBMS	Basu
Dos	Sinha

Syntax for selecting data from view

SQL > Select title from v_book where author_name = "Basu";

o/p

title
DBMS
ADBMS

Views :-

→ The adjective result of one or more selectioned operations on the base relations to produce another relation is called views.

→ A view is a virtual relation that doesn't necessarily exist in the database, but can be produced upon request by a particular user at the time of request.

→ A view is object that gives the user a logical view of data from one underlying table or table. Upon restrict user's data access by allowing them to see only a few columns from a table.

Purpose of views :- It is desirable for several reasons

→ It simplifies queries

→ It can be queried as a base table

→ It provides a powerful and flexible security mechanism by hiding parts of the database from certain users.

→ It permits users to access data in a way that is customized to their needs so that the same data can be seen by different users in different ways at the same time.

SQL> create view V_book as select title, author_name from books;

View Created.

SQL> select * from V_books;

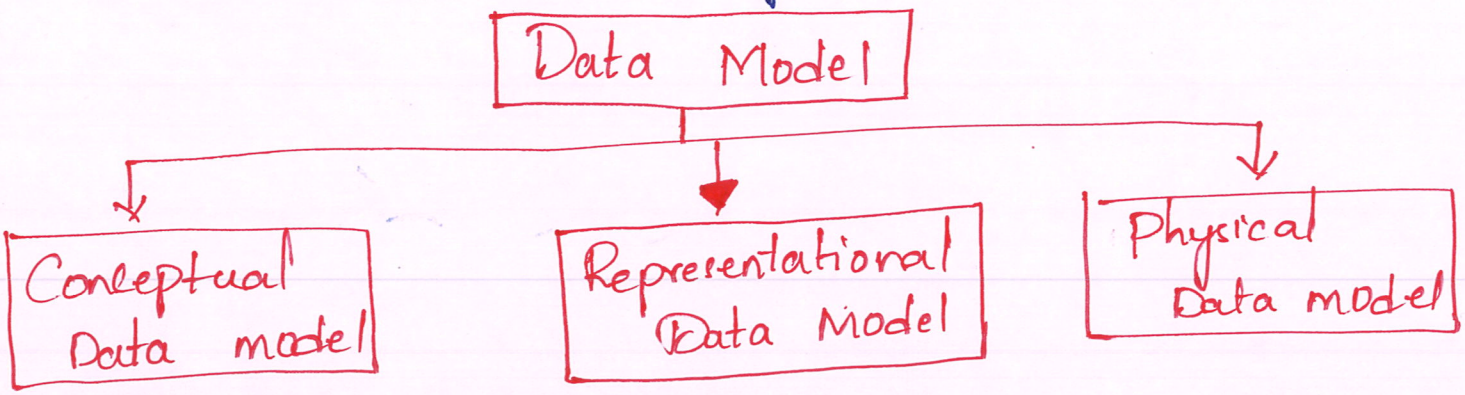
title	author_name
Oracle	Arora
Dbms	Gupta
Dcs	Gupta

SQL> select title from V_books where author_name = 'Gupta';

title
Dbms
Dcs

Data model :- Data model in DBMS as the Concept of tools that are developed to summarize the description of the database.

It classifies into 3 types



I Conceptual Data model :-

* It describes the database at a very high level
* Its useful to understand the needs or requirements gathering process. (ER model)

II Represented Data Model :-

* Its used to represent only the logical part of the Database and doesn't represent the physical structure of the Database
* It focus primarily on the design part of the database (Relational model).

III Physical Data model :-

* Ultimately all the data in a database is stored physically on a secondary storage device such as discs and tapes.
* This is stored in the form of files, records and certain other data structures.

Data Model :- Data model in DBMS or the Concept of tools that are developed to Summarize the description of the database.

It classifies into 3 types



I Conceptual Data model :-

* It describes the database at a very high level
* Its useful to understand the needs or requirements
- ents gathering process. (Ex model)

II Representated Data Model :-

* Its used to represent only the logical part of the Data
how and doesn't represent the physical structure of the Data base.

* It focus primarily on the design part of the databases (Relational model).

III Physical Data model :-

* It finally at all the data in a database is stored physically on a Secondary storage devices such as discs and tapes.

* This is stored in the form of files, records and certain other data structures.

History of DBMS

→ Charles Bachman (1960) – Integrated data store (based on New data model)

→ In late 1960's IBM (International Business machines Corporation) developed the Integrated management systems which is the standard database system used till data in many places. It has developed based on the hierarchical database model

→ During 1970 the relational database model was developed by Edgar Codd many of the data base models we are today are relational based. It was considered the standardized data base model.

→ later during the same decade (1980's), IBM Developed the structural Query language (SQL), It was declared as standard language for the quires by ISO and ANSI

Date & Datatype :- In my sql the default date functions are

1) now() → Current date & time

Ex :- Select now();

O/p :- 2017-01-20 08:03:52

2) curDate() → Current date

Ex :- Select curdate();

O/p : 2017-01-13

3) Curtime() → Current time

Ex :- Select Curtime();

O/p :- 08:05:15

4) Date() → to Insert date manually

SQL> Create table student (sid Int,
sname varchar(10),
sDOB Date format 'dd.mm.yyyy');

SQL> Insert into student (sid, sname, sDOB) values
(101, x, '07.06.1992');

SQL * from student;

Student

Sid	Sname	SDOB
101	x	07.06.1992

1) How to Insert -ve values & Decimal values into data base by using SQL ?

Ans The decimal datatype can store negative numbers as well as decimal values.

Ex: Create table ECE (marks decimal (5,2));

SQL>

SQL> Insert into ece (marks) values (22.6);

SQL> Insert into ECE (marks) values (-4.36);

SQL> Select * from ECE;

ECE

marks
22.6
-4.36

Syntax: decimal [(p[,s])]

(or) (decimal (size, d)

always s ≤ p

s - scale

p - precision

→ p stands for precision, the total no. of digits in the value, i.e, on both sides of the decimal point.

→ s stands for scale, no of digits after the decimal point

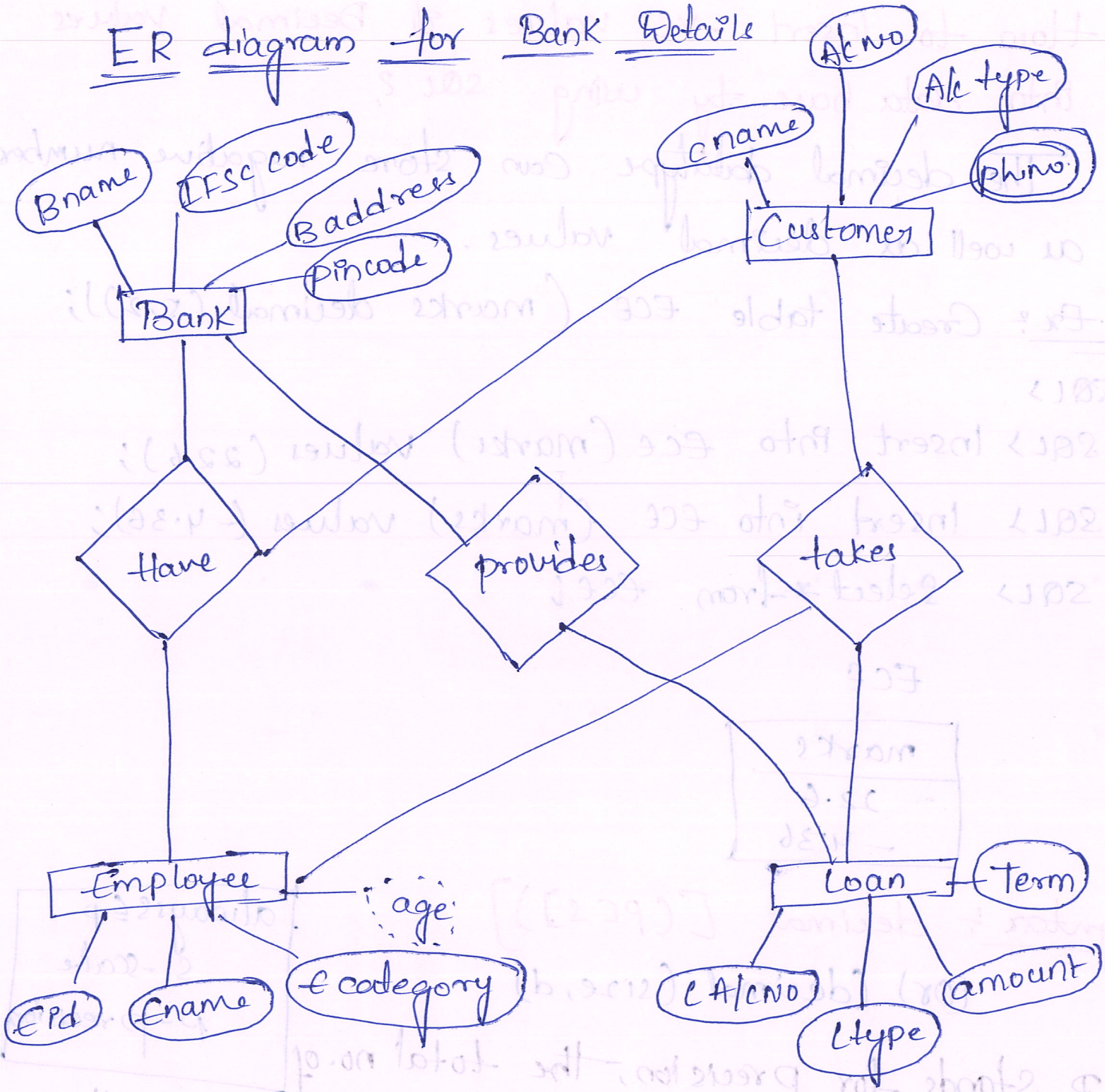
Ex: decimal (4,2) Indicates that the number will have 2 digits before the decimal point and 2 digits after the decimal point like 23.42, 4.

* To Insert binary values we use Binary (size)

* " " float " " float (size, d)

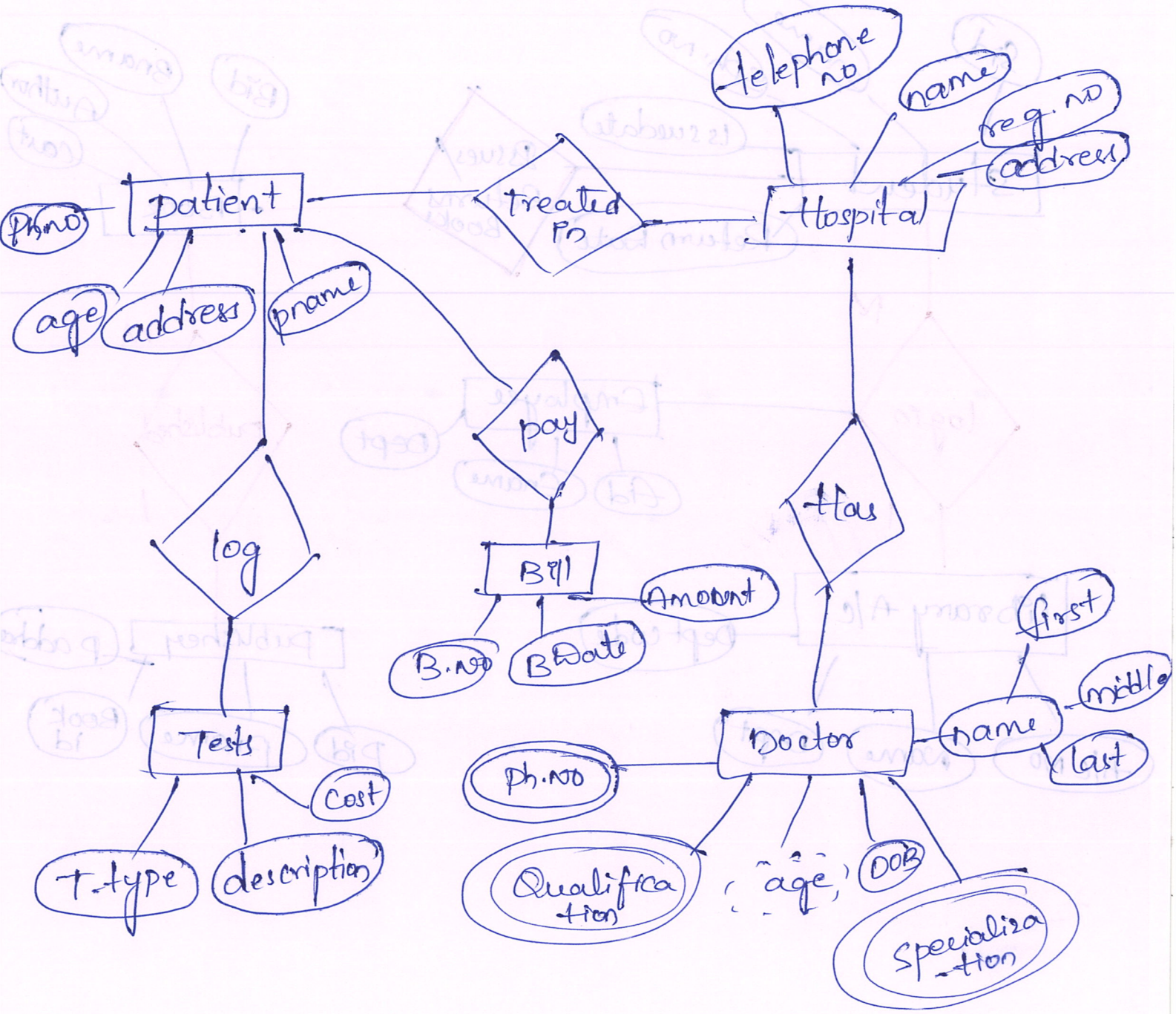
size → total no. of digits, d → no. of digits after decimal point

ER diagram for Bank Details

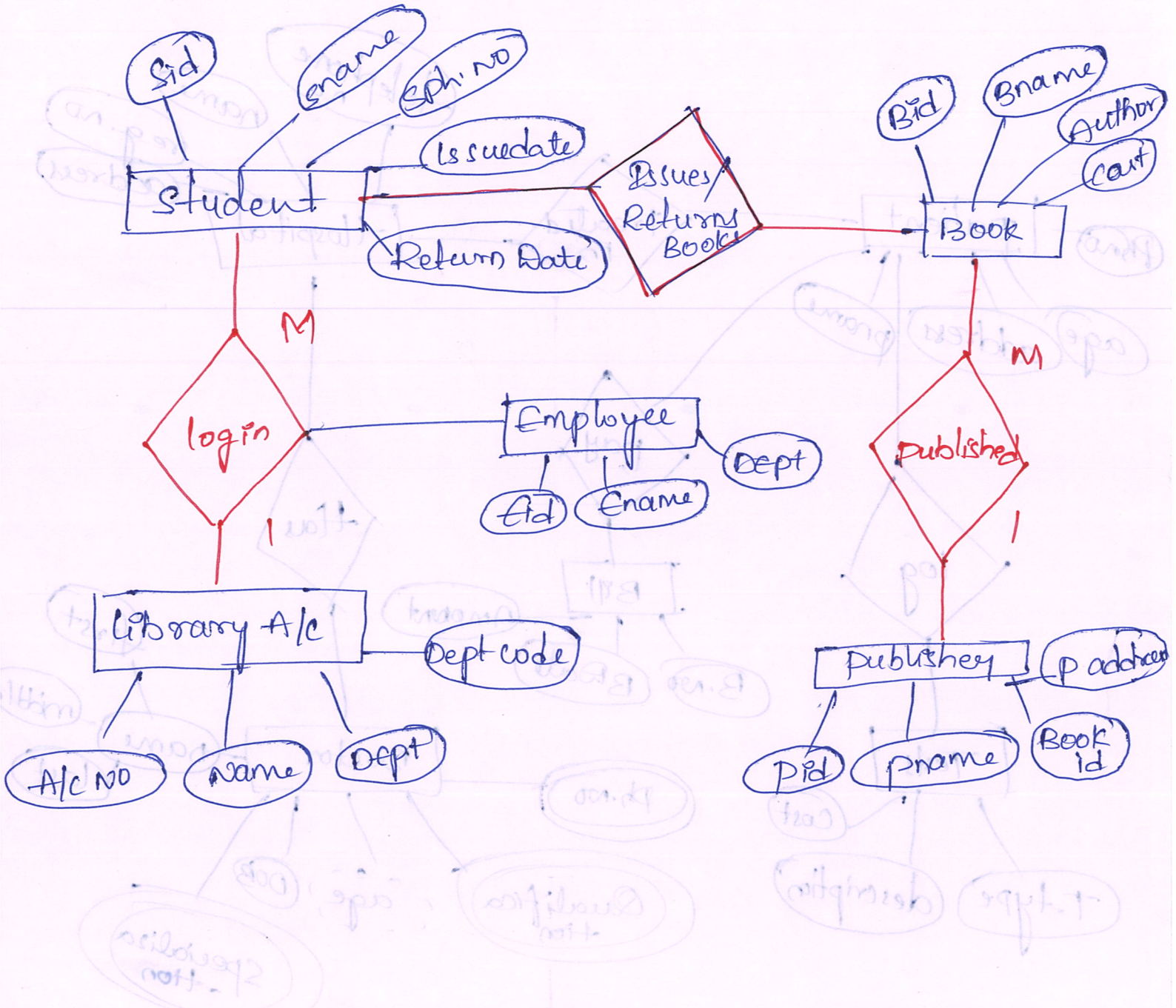


* Bank Have customers as well as Employees
* Bank provides loans
* Customers / Employees takes loans.

Ex: ER diagram for Hospital Management System.



ER - Diagram for Library Management System:-



UNIT-II

Relational Algebra

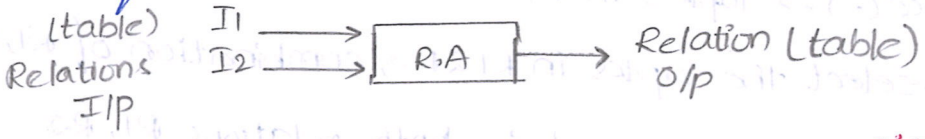
→ It is used in the design of transactions & forms the conceptual basics for SQL

→ It consists of operators and manipulators which apply to relations and give new relation as a result. But do not change the actual relation in Database.

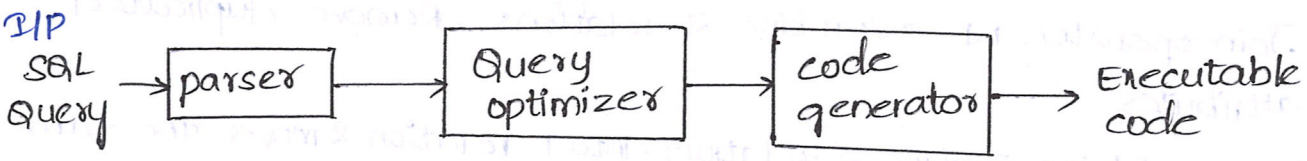
→ It is a procedural Intermediate language used within the DBMS

→ The Relational Algebra (R.A) is a procedural Query language

→ It contains set of operations that take one or two relations as input and produce a new relation as op depends on user request.



→ R.A collects instances of relations as I/p & gives occurances of Relations as O/p → R.A operations are performed recursively on R.



parser: It's a compiler or Interpreter, that breaks data into smaller pieces elements for easy translation into another language.

SQL Query → Relational algebra expression

Query optimizer: converts relational algebra expression into optimized relational algebra expression gives most efficient querys.

Code generator: It generates Executable case by using Query Execution plan.

Advantages of R.A:

1. It's one of the mathematical background (basics of many interesting developments & theorems
2. It gives most efficient querys & forms.

Basic operations: R.A contains several fundamental operations

1] selection operation(σ) → select a subset of rocus from relation

2] projection operator(π) → used to choose all columns from relation deletes unwanted columns from relation

3]. Cross product ($\times$) → Allows us to combine 2 relations in 1 relation

4] set difference ($-$) → Tuples in R1 but not in R2

5] Union ($\cup$) → select the tuples in R1 & R2, combination of R1, R2

6] Intersection ($\wedge$) → tuples present in both relations R1, R2

7] Rename (ρ) → give new names to the R.A. Expressions

8] set operations → union, intersection, cross product, difference

9] Join operation $\bowtie$ → combine 2 relations, Remove duplicated attributes

i) Natural join: Combine 2 relations into 1 relation & moves the same columns in 2 relations into only 1 column in the resulting R

ii) Equi join: Equalities b/w two fields connected by logical And($\wedge$)

iii) Conditional join / Theta join: it is identical to a selection condition in form. This operation is defined a cross product followed by a selection.

→ It uses any condition in the selection criteria

10] Division operation: ($\div$) → It is suited to query that include the phrase for all.

Relational Algebra operations

- Unary Relational operations
  - 1] select (σ)
  - 2] project (π)
  - 3] Rename (ρ)
- Binary Relational operations
  - 1] Join
    - i) Inner Joins
      - ⋈ Conditional join/Theta join
      - Equi join
      - natural join
    - ii) outer joins
      - left outer join (⟕)
      - Right outer join (⟖)
      - Full outer join (⟗)
  - 2] Division ÷ /
- Set operations
  - Union (∪)
  - Intersection (∩)
  - Difference (-)
  - Cartesian product (X)

I. Unary Relational operations:

1] Select operation (σ):

→ It select tuples (row) that satisfy the given predicate (Exp) from a relation

→ Notation : $\sigma_p(r)$

[sigma = σ
p = prepositional logic and, or, not
r = relation

→ select uses σ condition to select rows from a relation

→ A new relation is created from another existing relation by selecting only those rows requested by the user that satisfy a specified condition.

syntax: $\sigma_{\text{selection_condition}}(\text{Relation_name})$

→ The select operation uses the column names in specifying the selection condition

→ Actually the selection conditions are same as the conditions used in · If statements of programming languages

→ The selection condition uses the relational operations

$<, >, <=, >=$

→ we can combine several conditions into a large condition using the logical connectivities 'AND' represented by '∧', OR represented by '∨'

① Ex: Find the customer details who are living in Hyd city from customer relation

$\sigma_{city=Hyd}$ (customer)

② Ex: Find the customer details who are living in hyd city and whose customer_id is greater than 1000 in customer relation:

$\sigma_{city='Hyd' \wedge customer_id > 1000}$ (customer)

Ex: ③ $\sigma_{subject='database'}$ (BOOKS) // It select tuples from books where subject is database //

④ $\sigma_{sage>20}$ (student)

⑤ $\sigma_{Esalary>20,000}$ (Employee)

⑥ $\sigma_{subject='database' \text{ and } price=450}$ (BOOKS)

// It select tuples from books table where subject is database ∧ price is 450 //

Books

subject	year of publishing	price
2009	Datastructures	400
2010	Database	450
2011	DBMS	550

Ex:3 O/P:

subject	Year of publishing	price
2010	Database	450

Ex: 6 O/P

subject	Year of publishing	price
2010	Database	450

Ex ⑦ ⌐ subject = "database" and price = 450 or year > 2009 (Books)

// It select tuples (rows) from books table where subject is database & price is 450 or those books published after 2009 //

Subject	Year of Publishing	price
Database	2010	450

Student

SID	SNAME	SAge	SAddress
A0	Rahul	20	Hyd
A1	pinky	21	KNL
A2	Reva	20	chennai
A3	Adhvik	22	Delhi

Employee

EID	EName	ESalary	EAddress
2074	Upender	20,000	Kerala
2086	Harika	21,000	Banglore
2094	Tulasi	20,000	Kadapa
2098	Vimala	25,000	Vizag

4 O/p:-

A1	pinky	21	KNL
A3	Adhvik	22	Delhi

5 O/p:

2086	Harika	21,000	Banglore
2098	Vimala	25,000	Vizag

⑧ ⌐ SNAME = "Reva" (student) ⇒

A2	Reva	20	Chennai

⑨ ⌐ SAge >= 20 (student) ⇒

A0	Rahul	20	Hyd
A2	Reva	20	Chennai

⑩ ⌐ EAddress = "Kadapa (Employee) ⇒

2094	Tulasi	20,500	Kadapa

⑪ ⌐ Esalary <= 20,000 (Employee)

2074	Upender	20,000	Kerala
2094	Tulasi	20,000	Kadapa

⑫ ⌐ STQ = A2 and sAddress = "chennai" or sAge = 20 (student)

A2	Reva	20	Chennai

Eg: person

Name	Age	weight
Harry	34	80
Sally	28	64
George	29	70
Helena	54	54
Peter	34	80

$\sigma_{Age \geq 34}$ (person)

Name	Age	weight
Harry	34	80
Helena	54	54
Peter	34	80

$\sigma_{Age = weight}$ (person)

Name	Age	weight
Helena	54	54

Eg: Find the customer details who are living in hyderabad city from customer relation

$\sigma_{city = "Hyd"}$ (customer)

Eg: Find the customer details who are living in hyd city and whose customer_id is greater than 1000 in customer relation

$\sigma_{city = "Hyd" \wedge customer_id > 1000}$ (customer)

② project / projection operation :- (π)

→ The project is a relational algebra operation that creates a new relation by deleting columns from an existing relation by selecting only those columns requested by the user from projection is denoted by Greek letter pi (π)

→ In simple words the project operation extracts specified columns from table & it eliminates unwanted columns & eliminates duplicates.

→ The projection method defines a relation that contains a vertical subset of relation.

Syntax: $\pi_{A1, A2 \ldots An}(r)$

π - projection symbol

r - relation (table name)

A1, A2 ... An - attribute names of relation r

Ex:- Select column SNAME from student

π_{SNAME} (student) ⇒

SNAME
Rahul
pinky
Reva
Adhvik

Ex: Select column sid, saddress from student

$\pi_{sid, saddress}$ (student) ⇒

sid	saddress
A0	Hyd
A1	KNL
A2	Chennai
A3	Delhi

Ex: Select Ename, Esalary from Employee

$\pi_{Ename, Esalary}$ (Employee) ⇒

Ename	Esalary
Upendra	20,000
Harika	21,000
Tulasi	20,000
Vimala	25,000

Ex: Find the customer_name (not customer details) who are living in Hyd city from customer relation.

$$\Pi_{\text{customer name}} \left(\sigma_{\text{city} = \text{"Hyd"}} (\text{customer}) \right)$$

Ex: Find the Employee name who are living in Banglore from Employee table

$$\Pi_{\text{Ename}} \left(\sigma_{\text{EAddress} = \text{"Banglore"}} (\text{Employee}) \right)$$

Ename
Hanika

Ex: Find the Eid who are having their salary greater than 20,000 from Employee table

$$\Pi_{\text{Eid}} \left(\sigma_{\text{Esalary} > 20{,}000} (\text{Employee}) \right)$$

Eid
2086
2098

It will execute the selection operation (σ) first the projection of the resulting relation on the Eid column is carried out.

NOTE: Selection, projection operations allows us to manipulate the data in single Relation.

3. Rename (ρ) operation:

→ The Rename operation denoted by rhd(ρ) is a relational Algebra which is used to give the new names to the relation Algebra Expression.

→ we can give list of terms having the form
old name → new name

→ For ρ to be well-defined, we no 2 fields in the result have the same name.

Ex: ρ (Temp1, σ bid=103 Reserves) ⇒ Temp1 denotes the intermediate relation that identifies reservations of boat 103.

Ex: ρ (Temp2, Temp1 ⋈ sailors) ⇒ Temp2 is another intermediate relation & it denotes sailors who have made a reservation in the set temp1

syntax: $\rho_{x}(E)$ $\rho_{newname}$ (old name)

where ρ → Rename operator
x → Rename of a Relation (new name)
E → Relation name (old name)

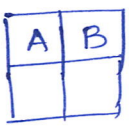
syntax: $\rho_{X(A1, A2 \ldots An)}{}^{E}$

A1, A2, ... An → list of attribute names

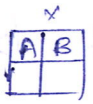
Ex: $\rho_{Faculty}$ (Employee) ⇒ Employee table changed as a faculty table.

Ex: $\rho_{Faculty(Fid, Fname, Fsalary, Faddress)}$ (Employee) ⇒ Employee table changed as faculty table and its attributes changed as Fid, Fname, Fsalary, Faddress

(oldname)

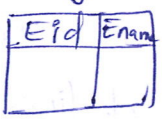
Ex: ρ_{x} (Emp) ⇒ 

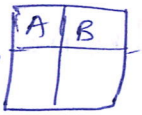
|| table name changed $\rho_{newname}$

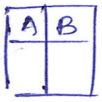
$\rho_{x(A,B)}$ (Emp) ⇒ 

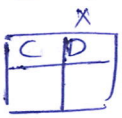
//attributes changed

Employee

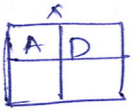


⇒ 

$\rho_{(A,B)}$ (Emp) ⇒ 

$\rho_{x(C,D)}$ (Emp) ⇒ 

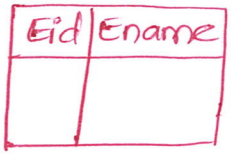
→ //to change attribute names

$\rho_{x(A,D)}$ (Emp) ⇒ 

$\rho_{x(C,B)}$ (Emp) ⇒ 

old Table

Employee

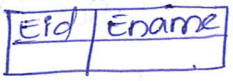


⇒

Rename table Employee as staff

ρ_{staff} (Employee)

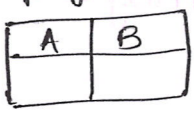
staff



Rename attributes in Employee

$\rho_{Employee(A,B)}$ (Employee)

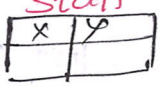
Employee



Rename attributes in staff

$\rho_{staff(x,y)}$ (staff)

staff

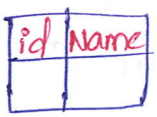


Rename single attributes in Employee

$\rho_{Employee(C,B)}$ (Employee)

Rename attributes & table from x(A,B) to CSE(id, name)

$\rho_{CSE(id,name)}$ (x)



II Binary Relational operations:

1] Join ($\bowtie$) operation :- to combine two relations like cartesian product, but finally removes duplicated attributes and makes the operations very simple

→ Join connects relations on columns containing comparable information

→ join operation is essentially a cartesian product followed by a selection criterian.

→ join operation also allows joining variously related tuples from different relations.

I] inner joins

→ In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded.

3 types (i) Theta join (ii) Equi join (iii) Natural join

i) Theta join :- / condition join :

→ The join operation accepts a join condition c and a pair of relation instances as arguments and returns a relation instance.

→ The join condition is idential to a selection condition in form. The operation is defined a cross product followed by a selection as given below.

$$R \bowtie_c S = \sigma_c (R \times S)$$ where c = condition

→ The general case of join operation is called Theta (θ) join

→ It use any conditions in the selection criteria.

Ex: Consider the following two relations car and boat

Car

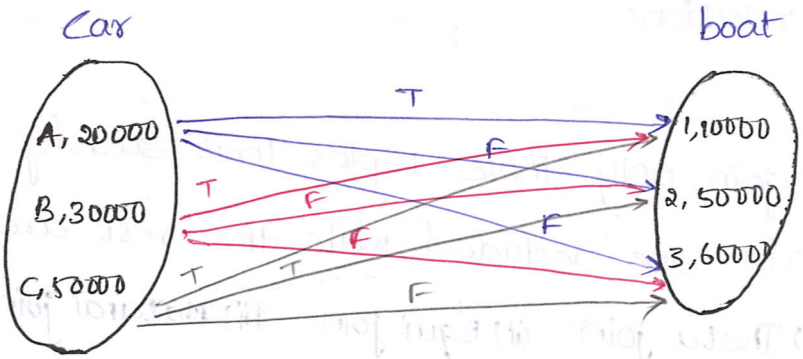
Car-model	Car-price
A	20000
B	30000
C	50000

Boat

Boat_model	Boat_price
1	10,000
2	50,000
3	60,000

The condition join

$$Car \bowtie_{carprice >= boatprice} Boat$$



Result:

Car_model	car_price	Boat_model	Boat_price
A	20000	1	10000
B	30000	1	10000
C	50000	1	10000
C	50000	2	50000

(iii) equi-join: join condition consists solely of equalities b/w two fields on R and S connected by logical and '∧' operator (∧)

→ when a theta join used only equivalence condition it becomes a equijoin.

Ex: consider the same car & boat tables

$$car \bowtie_{carprice = boatprice} boat$$

Result:

Car_model	Car_price	Boat_model	Boat - price
c	50000	2	50000

(iii) Natural join:

→ Natural join can only be performed of there is a common attribute (column) b/w the relations. The name and type of the attribute must be same.

→ Natural join combine two different relations into one relation and makes the same columns in two different relations into only one column in the resulting relations.

Ex:

C

Num	Square
2	4
3	9

D

Num	Cube
2	8
3	27

C ⋈ D

i) Identify common attributes
ii) Create a newtable with all attributes
iii) if 2 tables have same attribute then take one in result table
iv) join the 2 tables based on their values.

Result:

Num	Square	Cube
2	4	8
3	9	27

2 Ex:-

Ename	street	city
cat	c-street	c-city
Rabbit	r-street	r-city
sad	s-street	s-city
mad	m-street	m-city

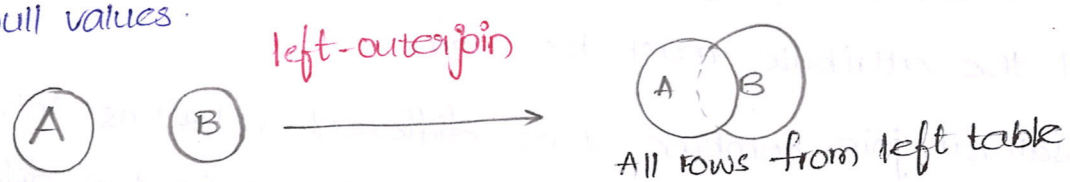
Ename	Branch name	Salary
cat	m-branch	1500
Rabbit	m-branch	1300
Good	n-branch	5300
mad	n-branch	1500

II Outer-join:- In outerjoin, along with tuples that satisfy the matching criteria, we also include same or all tuples that donot match the criteria.

(i) left outerjoin ($\bowtie$):-

→ In the left outer join, operation allows keeping all tuples in the left relation.

→ However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.



Eg:

A

Num	Square
2	4
3	9
4	16

B

Num	Cube
2	8
3	27
5	125

A $\bowtie$ B

Num	Square	Cube
2	4	8
3	9	27
4	16	NULL

(ii) Right outer join ($\bowtie$):

→ In the right outer join, operation allows keeping all tuples in the right relation.

→ However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values



Eg:

A $\bowtie$ B

Num	Square	Cube
2	4	8
3	9	27
5	NULL	125

Num	Cube	Square
2	8	4
3	27	9
5	125	NULL

(iii) Full outer join (⟗):

→ In full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

A ⟗ B

A

Num	Square
2	4
3	9
4	16

B

Num	Cube
2	8
3	27
5	125

Result:

Num	Square	Cube
2	4	8
3	9	27
4	16	NULL
5	NULL	125

2] Division operation (÷, /):

→ It is suited to queries, that include the phrase for all.

A

Sno	Pno
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

B1

Pno
P2

B2

Pno
P2
P4

B3

Pno
P1
P2
P4

A/B1

Sno
S1
S2
S3
S4

S1 P2
S2 P2
S3 P2
S4 P2

A/B2

Sno
S1
S4

S1 P2
S1 P4
S2 P2
S3 P2
S4 P2
S4 P4

A/B3

Sno
S1

S1 P1
S1 P2
S1 P4
S2 P1
S2 P2
S3 P2
S4 P2
S4 P4

→ In the above example consider two relation instance A and B in which A has exactly two fields sno, pno and B has just only one field sno, with the same domain as in A.

→ we define the division operation A/B as set of all sno values such that for every pno value in B, there is a tyo tuple ⟨sno, pno⟩ in A.

→ Another way to understand division is as follows

For each sno value in (the first column of) A, consider the set of pno values that appear in (the second field of) tuples of A

with that sno value. If this set contains (all pno values in) B, the sno value is in the result of A/B.

In fig it helps to think of A as a relation listing the parts supplied by suppliers and of the B relations as listing parts. A/Bi computes suppliers who supply as parts listed in relation instance Bi.

III Set operations :

→ The set operators are availed to combine information of similar type from one or more tables.
→ The set operators look similar to SQL joins although there is a big difference.
→ SQL joins tends to combine columns from different tables. where SQL set operators combine rows from distinct queries
→ There are distinct type of set operators in SQL like union, Intersection, difference, cartesian product.

1. Union(U):

→ It's a combination of 2 relations R1 UR2
→ It removes duplicate from the result i.e. just one row is listed for every duplicated result.
→ The result of this operation by R1UR2 is a relation that include all tuples that all are either in R1 or in R2 or in both R1 and R2.

Ex:

R1

X	Y
1	A
2	B

R2

X	Y
4	C
2	B
3	B

⇒

R1 UR2

X	Y
1	A
2	B
3	B
4	C



Divide operation:

E1:

Student S

Name	age
A	20
B	21
C	22
A	22
B	20
C	21
A	21
B	20

Employee E

G

Name
A

E2

Name
B
C

E3

Name
C

E4

Name
A
C

S/E1 ⇒ (A, 20), (A, 22), (A, 21) = | age |
|---|
| 20 |
| 22 |
| 21 |

S/E2 ⇒ (B, 21), (C, 22), (B, 20), (C, 21), (B, 20) ⇒ (B, 21), (C, 21) ⇒ | age |
|---|
| 21 |

S/E3 ⇒ (C, 22), (C, 21) ⇒ | age |
|---|
| 22 |
| 21 |

S/E4 ⇒ (A, 20), (C, 22), (A, 22), (C, 21), (A, 21) ⇒ (A, 22), (C, 22) and (A, 21), (C, 21) ⇒ | age |
|---|
| 22 |
| 21 |

1] Eg: find the customer-id of all customer in the bank who have either an account (Depositors) or alean (Borrower) or both

$\Pi_{customer_id}$ (Depositer) $\cup$ $\Pi_{customer_id}$ (Borrower)

A/c Depositor

C-id	name
2219	X
2496	Y
2376	Z

Borrower_(loan)

2219	X
2499	B
2478	C

$\Rightarrow$

cid
2219
2496
2499
24376
2478

2] Eg: student

id	name	age
1	X	20
2	Y	21
3	Z	22

Employee

id	name	salary	age
11	A	15000	20
12	B	11000	22
13	C	12000	25

$\Rightarrow$

name
X
Y
Z
A
B
C

Π_{name} (student) $\cup$ Π_{name} (Employee)

3] Eg: project the names of author who have either written books or an articles or both

SQL > select authorname from books union select authorname from Article;

$\Pi_{authorname}$ (Books) $\cup$ $\Pi_{authorname}$ (Article)

AId	AName
9999	Raju
8888	Ramya
7777	Rahul

AId	AName
9999	Raju
6666	Anju
5555	Devi

Author name
Raju
Ramya
Rahul
Anju
Devi

4) Eg:

R

Name	Num
A	1
B	2
D	3
F	4
E	5

S

Name	Num
A	1
C	2
D	3
E	4

$\rightarrow$

RUS

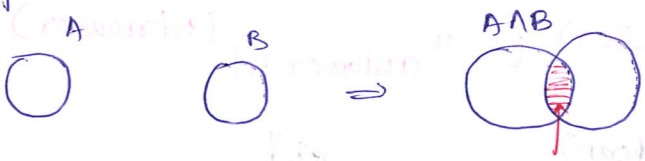
A	1
B	2
C	2
D	3
F	4
E	4
E	5

SQL> select NAME from R union select NAME from S; o/p $\rightarrow$

A
B
C
D
E
F

17

2. Intersection ($\cap$): It defines a relation consisting of a set of all tuple that are in both A and B



Ex: lets consider the same tables from previous

Example (refer tables in union)

1Ex:- Π_{c_id} (Depositor) $\cap$ Π_{cid} (Borrower) $\Rightarrow$

cid
2219

2Ex: Π_{age} (student) $\cap$ Π_{age} (Employee) $\Rightarrow$

age
20
22

Π_{name} ($\sigma_{age \geq 22}$ (student) $\cap$ $\sigma_{age \geq 22}$ (Employee) $\Rightarrow$

name
Z
B
C

3Ex: $\Pi_{authorname}$ (BOOKS) $\wedge$ $\Pi_{authorname}$ (Article) $\Rightarrow$

Aname
Raju

4Ex: R $\cap$ S $\Rightarrow$

R∩S	
A	1
P	3

SQL> select age from student Intersect
select age from Employee;

SQL> select name from student where age ≥22
Intersect select name from Employee where age ≥22;

3. difference / minus / set Difference (-) :-

→ A-B is a relation which includes all tuples that are in A but not in B.

→ This operator is availed to retrive the information of one table which is not available in another table.

Ex:

R

Name	NO
A	1
B	2
D	3
F	4
E	5

S

Name	NO
A	1
C	2
D	3
E	4

R - S

Name	NO
B	2
F	4
E	5

S - R

Name	NO
C	2
E	4

SQL> select Name, NO from R
minus select Name, NO from S;

SQL> select NO from R minus
select NO from S;

NO
2
4

Ex: $\Pi_{customer_id}(Depositor) - \Pi_{customer_id}(Borrower)$

$\Rightarrow$

Customer-id
2496
2376

2Ex: $\Pi_{age}(student) - \Pi_{age}(Employee) \Rightarrow$

age
21

3Ex: $\Pi_{AName}(BOOK) - \Pi_{Aname}(Article) \Rightarrow$

AName
Ramya
Rahul

$\Pi_{Aname}(Article) - \Pi_{Aname}(BOOKS) \Rightarrow$

AName
Anju
Devi

(4) Cartesian product (X)

→ This type of operation is helpful to merge columns from 2 relations

→ combines information of 2 different relations into one.

Ex:

R

A	1
B	2
D	3
F	4

S

A	1
C	2
D	3

R × S

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

A

x	y
1	1
1	2

B

x	y
1	1
1	3

$\sigma_{cd2=1}(A \times B)$

x	y
1	1
1	1

A × B

1	1	1	1
1	1	1	3
1	2	1	1
1	2	1	3

Ex: borrower (b.loan no, c-id)

loan (l.loan.no, amount)

① Find the customer-id of all customer who have loan >10,000 and customer have both loan no and A/c no same.

$\pi_{c\text{-}id}(\sigma_{b.loan.no = L.loan\,no}(\sigma_{amount > 10000}(borrower \times loan)))$

borrower

b.l.no	c-id
12345	111
14567	222

loan

L.Lno	amount
12345	15000
15260	9000

borrower × loan

12345	111	12345	15000
12345	111	15260	9000
14567	222	12345	15000
14567	222	15260	9000

o/p:

cid
111

Union compatible: 2 tables have same attributes, same domain, same degree(no.of attributes)

Imp notes: for performing union, Intersection, minus operations the 2 relations are union compatible

→ attribute domains need to be compatiable

→ The 2 relations must contain the same no.of columns

→ Each column of the first relation must be either the same datatype as the corresponding column of the second e.

→ Attributes name & type must be matchin both relations then its said to be a union compatible & same no.of attributes.

Relational Calculus

→ Relational calculus (R.C) is one of the formal Query language and it is an alternate to R.A.

→ R.C is used for constructing the expressions for defining Queries.

→ In R.C, A Query is expressed as a formula consisting of a no. of variables and an expression involving these variables.

→ The formula describes the properties of the result relation to be obtained.

→ There is no mechanism to specify how the formula should be evaluated it is upto the RDBMS to convert these non-procedural Queries to equivalent, efficient procedural Queries. Relational calculus come in 2 types.

1. Tuple Relational Calculus (TRC): variables range over tuples of a relation
2. Domain Relational Calculus (DRC): " " " elements of a Domain

1. Tuple Relational Calculus (TRC):

→ It describes the desired information without giving a specific procedure for obtaining that information.

A Query in the tuple relational calculus is expressed as,

$\{t \mid p(t)\}$ // it is the set of all tuples t such that predicate p is true for t.

T → Tuple variable

p(t) → formula or condition for getting the T.

Ex: Find all the tuples from the depositor relation

⇒ $\{t \mid t \in \text{Depositor}\}$ $\{t \mid t \in \text{loan}\}$ loan

Ex: Find the loan number, branchname and amounts for loans of over 1200

⇒ $\{t \mid t \in \text{loan} \wedge t[\text{amount}] > 1200\}$

loan

loan_no	Branchname	Amount
11	ECIL	900
14	Dilsukhnagar	1500
17	Malakpet	500

Ex: Find the sailors whose rating > 7 ?

$\{S \mid S \in Sailors \wedge S.rating > 7\}$

Ex: Find the Employees whose salary > 20000

$\{E \mid E \in employee \wedge E.salary > 20000\}$

Ex: Find the names of sailors who have reserved boat 103

$\{P \mid \exists S \in Sailors \exists R \in Reserves (R.sid = S.sid \wedge R.bid = 103) \wedge P.sname = S.sname\}$

Ex: Find the names and ages of sailors who have reserved a red boat ?

$\{Q \mid \exists S \in Sailors \exists R \in Reserves \exists B \in boats (R.sid = S.sid \wedge R.bid = B.bid \wedge B.bcolor = 'red' \wedge Q.name = S.sname \wedge Q.age = S.sage)\}$

Ex: Find the names of sailors who have reserved all boats?

$\{P \mid \exists S \in Sailors \forall R \in Reserves \forall B \in boats (S.sid = R.sid \wedge R.bid = B.bid \wedge P.name = S.sname)\}$

$\wedge$ - conjunction
$\vee$ - disjunction
$\Rightarrow$ Implication
$\Leftrightarrow$ Equivalence

predicate Formula contains

1. Attributes & constants
2. Comparison operators ($<, \leq, =, >, \geq, \neq$)
3. Connectives : $\wedge, \vee, \neg$
4. implications ($\Rightarrow$)

$X \Rightarrow Y$, if X is true then Y is true $[X \Rightarrow Y \equiv \neg X \vee Y]$

5. Quantifiers:

$\exists, \forall$

There exists, forall $\exists t \in r(Q(t)) \Rightarrow$ $\exists$ a tuple in r such that Q(t) is true

$\forall t \in r(Q(t)) \equiv Q$ is true for all tuples in a relation r.

P	Q	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$	$\neg P$	$\neg Q$
T	T	T	T	T	T	F	F
T	F	F	T	F	F	F	T
F	T	F	T	T	F	T	F
F	F	F	F	T	T	T	T

2. Domain Relational Calculus (DRC):

→ Domain Relational calculus uses domain variables that take on values from an attributes domain, rather than for an entire tuple.

syntax: $\{\langle x_1, x_2, \ldots x_n \rangle \mid P(x_1, x_2 \ldots x_n)\}$ P-formula

$x_1, x_2, x_3 \ldots x_n$ - represents domain variables

Ex: Find the loan number, Branch name and amount for loans of over 1200

$\Rightarrow \{\langle L, b, a \rangle \mid \langle t, b, a \rangle \in loan \wedge a > 1200\}$

Ex: Find all loan numbers for loans with an amount >1200

$\Rightarrow \{\langle L \rangle \mid \exists (b,a)(\langle L, b, a \rangle \in loan \wedge a > 1200)\}$

Ex: Find the sailors whose rating > 7

$\Rightarrow \{\langle I, N, R, A \rangle \mid \forall \langle I, N, R, A \rangle \in sailors\ (R > 7)\}$

Ex: Find the Employees whose salary >20000

$\Rightarrow \{\langle I, N, S \rangle (\forall \langle I, N, S \rangle \in Employee\ (S > 20000)\}$

Ex: Find the names of sailors who have reserved boat no 103

$\Rightarrow \{\langle N \rangle \mid \exists \langle SI, R, A (\exists \langle I, N, R, A \rangle \in sailors\ (\langle I_S, I_B, D \rangle \in$ Reserves $(I_S = SI \wedge I_B = 103)))\}$

Ex: Find the sid & names of sailors who have Reserved a red boat

$\Rightarrow \{\langle SI, N \rangle \mid \exists \langle R, A \rangle (\langle I, N, R, A \rangle \in sailors\ (\langle I_S, I_B, D \rangle \in Reserves$

$(\langle BI, BN, C \rangle \in Boats\ (SI = BI = I_B \wedge C = \text{"Red"})))\}$

Ex: Find names of all sailors who have reserved all boats

→ $\{\langle N \rangle \mid \exists \langle SI, R, A \rangle (\langle SI, N, R, A \rangle \in sailors\ (\langle I_S, I_B, D \rangle \in Reserves$

$(\langle BI, BN, C \rangle \in Boats\ (SI = IS \cap BI = IB)))\}$

Expressive power of the language:

→ TRC restrict to safe Expressions and is equal in Expressive power to relational algebra.

→ Thus for every relational-algebra Expression, there is an equivalent Expression in TRC and for TRC Expression there is an equivalent

Relation algebra Expression

TRC = RA
RA = TRC

A safe TRC formula Q is a formula such that:

i) for any given I (Relation), the set of answer for Q contains only values that are in dom (Q, I)

ii) For each sub expression of the form ∃R (P(R)) in Q, if a tuple r (assigned to variable R) makes the formula true, then r contains only constraints in Dom (Q, I)

iii) For each sub expression of the form ∀R (P(R)) in Q, if a tuple r (assigned to variable R) contains a constant that is not in Dom(Q, T) then R must make the formula T.(True)

Ex:- Unsafe Exp: {S/¬(S∈sailors)}

→ it is possible to write syntatically correct calculus queries that have an infinite number of answers such queries are called unsafe.

Some Examples for TRC & DRC

1. Find sailors names who have rating more than 7?

SQL> select sname from sailors where srating > 7;

TRC: {T/∃T∈sailors ∧ S.rating > 7 ∧ T.sname = S.sname}

DRC: {<SN> / ∃<SI, SN, SA, SR> ∈ sailors (SR > 7)}

2. Find the names of sailors who reserved boats no 103

TRC: {P/ ∃S∈sailors ∃R∈Reserves (R.sid = S.sid ∧ R.bid = 103) ∧ P.sname = S.sname}

DRC: {<SN> / ∃<SI, SR, SA> (<SI, SN, SA, SR> ∈ sailors (<IS, IB, D> ∈ Reserves (IS = SI ∧ IB = 103)))}

3. Find the names & ages of sailors who reserved a red boat

TRC: {Q/ ∃S∈sailors ∃R∈Reserves ∃B∈boats (R.sid = S.sid ∧ R.bid = B.bid ∧ B.bcolor = 'Red' ∧ Q.name = S.name ∧ Q.age = S.sage)}

DRC: $\{\langle SA, SN\rangle \mid \exists \langle SI, SR\rangle (\langle SI, SN, SA, SR\rangle \in sailors \wedge (\langle BI, BN, BC\rangle \in boats \wedge (\langle Is, Ib\rangle D\rangle \in Reserves \wedge (Is = SI, Ib = BI \wedge BC = \text{"Red"})))\}$

4. Find the names of sailors who reserved all boats;

TRC:- $\{P \mid \exists S \in sailors \forall R \in Reserves \forall B \in Boats (S.sid = R.Rsid \wedge B.bid = R.Rbid \wedge P.name = S.sname)\}$

DRC: $\{\langle SN\rangle \mid \exists \langle SI, SR, SA\rangle (\langle SI, SN, SA, SR\rangle \in sailors (\langle Is, IB, D\rangle \in Reserves (\langle BI, BN, BC\rangle \in Boats (SI = Is \wedge BI = IB))))\}$

Examples for TRC & DRC

① find saddress of a student who have sid = 101

SQL > select sadd from student where sid = 101;

TRC: $\{S1 \mid \exists S \in student \wedge sid = 101 \wedge S1.sadd = S.saddress\}$

DRC: $\{\langle C\rangle \mid \exists \langle A, B\rangle (\langle A, B, C\rangle \in student (sid = 101))\}$

sid	sname	saddress
A	B	C
101	X	Hyd
102	Y	KNL
103	Z	ECIL
104	M	KNL

R.A: $\Pi_{sadd}(\sigma_{sid=101}(student))$

table: student - 's'

R.A: $\Pi_{sname}(\sigma_{sadd='KNL'}(student))$

② Find student names who are living in KNL

SQL > select sname from student where sadd = 'KNL';

TRC: $\{O \mid \exists O \in student \wedge sadd = 'KNL' \wedge O.sname = S.sname\}$

DRC: $\{\langle B\rangle \mid \exists A, C\rangle (\langle A, B, C\rangle \in student (sadd = 'KNL'))\}$

③. Find students who are from kurnool

SQL > select * from student where saddress = 'KNL';

TRC: $\{N \mid N \in student \wedge sadd = 'KNL'\}$

DRC: $\{\langle A, B, C\rangle \mid \langle A, B, C\rangle \in student (sadd = 'KNL')\}$

R.A: $\sigma_{sadd='KNL'}(student)$

Student S

sid X	sname Y	saddress Z
565	Anil	Hyd
575	Raju	KNL
585	Laxman	KPHB
595	Anil	ECIL

Employee E

Eid A	Ename B	Eaddress C
2074	Anil	KPHP
2086	Kavita	Uppal
2092	Raji	Hyd
2010	Laxman	KNL

DEPT D

Dsid M	DEid N	Dname O
565	2074	CSE
575	2086	MECH
585	2092	IT
595	2010	ECE

S.sid = D.Dsid
X = M

E.Eid = D.Eid
A = N

$\langle X, Y, Z \rangle \in$ student

$\langle A, B, C \rangle \in$ Employee

$\langle M, N, O \rangle \in$ DEPT

1. Find the student names who are belongs to MECH dept?
2. Find the student ids who are having faculty Kavita from MECH dept.
3. Find the list of students of Employees comming from KNL?

Basic SQL : structured Query Language

→ it's a standard language for accessing and manipulating DB.

→ Its a Technology - It's a traditional file processing system.

→ In 1970 "codd" Introduce a concept of Relational model.

→ It's a language influenced by the concept tuple relational calculus

→ Its a non procedural language

Form Ba) → SQL evolved from the Relational model

→ SQL used for constructing the Queries

Form Basic SQL Query:

Every Query consists of 3 clauses

Syntax:

select [distinct] Target list FROM Relation-list where Qualification

Relation-list → A list of relation / table names.

Target-list → A list of attributes

Qualification → Comparisons combined using AND, OR, NOT

Distinct → is an optional keyword to avoid duplicates.

Ex: select distinct sid from student where sage >25;

Ex:- select sid from sailors where srating > 7;

Here

select → It selects the list of attributes

from → It gives list of Relation names

where → It's a condition

Ex: select * from student;

// It displays all rows from student relation

Ex: select sname, bname from sailors, Boats where sid > 11, bid > 103;

Conceptual Evaluation strategy: It's a way of evaluating Query that tends to easily understand rather than the effecient 4 steps in this

1. Find the cross product of the list of tables.
2. Eliminating the list of rows/tuples in cross product based on Qualification
3. Eliminating the list of attributes in the result
4. Eliminating the duplicates into the result by using distinct keyword.

Ex1: Find the Names of sailors who have reserved a boat NO 103. Identify the various steps considered to evaluating the above query by using conceptual Evaluation strategy

SQL> select s.sname from sailors s, Reserves R where s.sid = R.Rsid and R.Rbid=103;

boats

bid	bname	bcolor
101	dippu	Red
103	Intelake	blue

sol: step1: take 2 tables sailors & Reserves as shown below

Sailors

sid	sname	srating	sage
21	Dustin	9	25.0
34	Horatio	8	39.7

Rsid	Rbid	day
21	103	21-8-2008
21	101	27-9-2007
34	103	2-1-2007

sid	sname	srating	sage	Rsid	Rbid	day
21	Dustin	9	25.0	21	103	21-8-2008
21	Dustin	9	25.0	21	101	27-9-2007
21	Dustin	9	25.0	34	103	2-1-2007
34	Horatio	8	39.7	21	103	21-8-2008
34	Horatio	8	39.7	21	101	27-9-2007
34	Horatio	8	39.7	34	103	2-1-2007

Sailors

sid SI	sname SN	age SA	rating SR
22	Dustin	45.0	7
29	Brutus	33.5	1
85	Art	25.5	3

Boats

bid BI	bname BN	bcolor BC
101	Interlake	blue
102	Interlake	red
103	Interlake	green
104	Interlake	red

Reserves

Rsid Is	Rbid Ib	day D
22	101	10/10/98
29	102	11/6/98
85	103	9/5/98

Step 2: Eleminating the list of rows based on Qualification

boatno = 103

sid	sname	rating	age	Rsid	Rbid	day
21	Dustin	9	25.0	21	103	21-08-2008
21	Dustin	9	25.0	21	101	27-9-2009
34	Horatio	8	39.7	34	103	2-1-2007
21	Dustin	9	25.0	21	103	21-8-2008
34	Horatio	8	39.7	34	103	2-1-2007

step3: Eliminating the list of attributes in the result from select list.

select sname so

sname
Dustin
Dustin
Horatio
Dustin
Horatio

step4: Eliminating the duplicates by using the "Distinct" keyword

sname
Dustin
Horatio

Ex2: Consider 2 Relations

Student

SName	SAddress
A	ECIL
B	Uppal
C	KPHB

Employee

Ename	EAddress
x	Nagole
y	Tarnaka
z	Uppal
M	ECIL

DEPT

Dname	Sname	Ename
CSE	A	M
EEE	B	z
IT	C	y
MECH	B	x

degree = 2

cardinality = 3

2

4

3

4

Q] Find the list of sname and Enames from student & Employee Relation who are comming from Uppal & ECIL

o/p:

Sname	Ename

SXE
3x4 = 12 tuples

Q2: Find the snames who are belongs to MECH dept.

o/p:

sname

SXD
3x4 = 12 tuples

Q3: Find the Dnames who are comming from Nagole

o/p:

Dname

SXEXD
3x4x4 = 48 tuples

Expressions and strings in the select command:

AS → Aliasing the names, Rename the old name with new name

BETWEEN → used for identifying the Range of tuples within the specific range of values

LIKE → it's used for comparision of strings from the various relations and
'_' stands for any one character &
% stands for 0 (zero) or 1 or more arbitary characters (string)

NOTE: % → The % character matches any substring
_ → The _ " " character

AS

Ex1: Find srating from sailors where srating = srating+1?

SQL> select (s.rating = s.rating+1) As rating from sailors.

Ex2: Find sailors names & modify sname as CSE in sailors table

SQL> select s.sname As CSE from sailors.

o/p1:

srating
9
8

⇒

srating
10
9

// srating = srating+1

o/p2:

sname
Dustin
Horatio

⇒

CSE
Dustin
Horatio

o/p3:

rating
9
8

MECH
27
24

// srating = srating x3

Ex3: Find the rating of a sailors with a multiple of 3 for every sailor and make the result renamed as MECH

SQL> select s.rating = s.rating x3 As MECH from sailors s;

INBETWEEN

Ex1: Find the names of sailors whose rating is in range of 2 and 8

SQL> select s.sname from sailors s where s.rating INBETWEEN 2AND8;

sname
Horatio

//refer tables in page 26.

Ex: Find the colors of boats whose bid is in range of 102 And 105;

SQL> select bcolor from boats b where b.bid inbetween 102 And 105;

bcolor
blue

//refer table from page 26

Ex: find the Enames from Employee relation whose salary is INBETWEEN 10000 AND 25000

SQL> select Ename from Employee E where E.E salary INBETWEEN 10000 And 25000

Eid	Ename	Esalary
2076	X	9000
2086	Y	18000
2092	Z	15000

o/p:

Ename
Y
Z

Ex: Find the age of students range of 25 And 30

SQL> select sage from student s
where s.sage INBETWEEN 25
And 30.

o/p:

sage
26
25
28

sid	sname	sage
1	A	19
2	B	20
3	C	26
4	D	20
5	E	25
6	F	28

LIKE:

Ex1: Find the names of sailors whose name starts with 'D' and Ends with 'N' and also name contain exactly 6 characters?

SQL> select s.sname from sailors s where s.sname LIKE
'D_ _ _ _N'; 'D - - - - - N o/p: DUSTIN

Ex2: Find the names of sailors whose name contains 6 characters?

SQL> select s.sname from sailors s where s.sname LIKE
'_ _ _ _ _ _'; o/p: Dustin Luber
Brutus

Ex3: Find the names of sailors whose name status with A and Ends with T and name contains atleast 3 characters?

SQL> select s.sname from sailors s where s.sname LIKE
'A _ %T' ; o/p: ART

Ex: LIKE operator

		Description
① where customer name		
ant LIKE 'a%'	-	finds any values that starts with 'a'
Raja '%a'	-	Ends with 'a'
Orange, more '%or%'	-	"or" in any position Ex: honor
Orange '_r%'	-	'r' in second position
ant, angry 'a_ _%'	-	starts with 'a' are atleast 3 characters in length
auto 'a%o'	-	starts with 'a' ends with o
'a_ _%'	-	starts with 'a' and atleast 3 characters in length.

SQL> select * from customers where customer name NOT LIKE 'a%'; // customer name that does NOT starts with "a".

Set operators: Union (U), Intersection (∩), MINUS (-) EXCEPT (-)

UNION ALL, INTERSECTION ALL, MINUS ALL.

set comparision operators: IN, NOT IN, EXIST, NOT EXIST, OP ANY, OP ALL

→ We are comparing the values of one Query with the value of another Query

→ The set comparison operators are used in Nested Queries (or) correlated nested queries.

→ Here op denotes set of comparision operators

{ <, >, <=, >=, ==, != }

NOTE:

By using Union, Intersect, minus / Except operators the result cannot allow duplicates tuples into the result. So for allowing the duplicate tuples in the result the other operators are Union ALL, INTERSECTION ALL, MINUS ALL.

→ Nested Query: A Query within the Query we are implementing one more Query. IN, NOT IN operations used here.

Query 1 (Query 2) // they depends on each other

↑ outer Q ↓ Inner Q

→ Correlated Nested Query: EXIST, NOT EXIST, OP ANY, OP ALL

A Query within the Query we are implementing the one more Query but these Queries are not related with each other.

Query 1 (Query 2) // They are not depends on each other.

* In correlated Nested Query (C.N.Q) the inner Query must be independent from outer Query.

* The inner Query may retrives all the tuples based on the inner

Query condition. The outer Q can picks the matched tuples from the resultant of an inner Query.

1. Find the names of sailors who have Reserved bid = 103?

Basic Query: select s.sname from sailors S, Reserves R where s.sid = R.Rsid and R.Rbid = 103;

Nested Query: select s.sname from sailors s where s.sid IN (select R.Rsid from Reserves R where R.Rbid = 103);

Nested correlated Query: select s.sname from sailors s where s.sid Exists (select * from Reserves R where s.sid = R.Rsid and R.Rbid = 103);

Op ANY & Op ALL: <, <=, >, >=

worst → < ANY

Younger → < ANY

betterthan → > ANY

Highest → > ALL

Oldest → > ALL

Ex: Find the ratings of sailors whose rating is worst than the rating of some of the sailors called 'dustin'.

SQL> select s.srating from sailors s where s.srating < ANY (select s.srating from sailors s where s.sname = 'Dustin');

Ex: Find the sid of sailors whose reservations must be better than the reservations made by 'Lubber'

SQL> select s.sid from sailors s where s.sid > ANY (select R.Rsid from reserves R where s.sid = R.Rsid and s.sname = 'Lubber');

Aggregate operators: sum() - it prints sum of all values in column

Avg(), MIN() → It prints minimum value in a column, MAX(),

COUNT() → it ~~cant~~ no of values in column.

oldest - max()

sum - sum()

Avg - Avg( )
least - min( )
count - count(*)

// Aggregate operators can apply an any column.

Ex: Find the oldest age of sailor

SQL> select max(s.sage) from sailors s;

Ex: Find the sum of all the ratings which are equivalent to the sid = 95

SQL> select sum(s.rating) from sailors s where s.sid = 95;

Ex: Find the Average sage of sailors who have reserve a red boat and the result have been renamed as "medium".

SQL> select Avg(s.sage) As medium from sailors s, Reserves R, Boats B where s.sid = R.Rsid and B.bid = R.Rbid and B.bcolor = 'red';

Ex: Find the least sid of sailors who have reserved a boat No 103.

SQL> select min(s.sid) from sailors s, Reserves R where s.sid = R.Rsid and R.Rbid = 103;

Ex: Count all the sailors in sailors Relation

SQL> select count(*) from sailors s;

Ex: Find the distinct names of sailors who have Reserved a boat called "marine" and count the names of sailors.

SQL> select count (distinct s.sname) from sailors s, Reserves R, boats B where s.sid = R.Rsid and B.bid = R.Rbid and b.bname = 'marine';

Active databases: The databases which associates an Triggers is called active database → A Datatbe that has a set of associated triggers is called an active database.

Various clauses in SQL:- 3 clauses

1. Order By - Ascending / Descending order
2. Group By - Result in the form of Groups
3. Having - define group Qualification / edit

Ex: find the youngest sailors for each rating level

SQL> select s.srating, MIN(s.sage) from sailors s. Group by s.rating;

Ex: find the names of sailors who have reserved atleast 2 boats on 10/oct/98 and display the result in ascending order.

SQL> select s.sname from sailors s, Reserves R where s.sid = R.Resid and R.day = '10/oct/98' Groupby S.sname Having count(*) > 1 order by s.sname (asce);

NULL values: - 2 ways

1. Any one of attribute cannot assign any value then that attribute is null attribute
2. Directly we can assign 'NULL' for a particular attribute

Intigrity constraints in SQL:

1. Table constraint - CHECH
2. Domain constraint and distinct types
3. Assertion: It's a general constraint based on which we are implement table constraint over multiple no. of tables

1.2 Ex:-

Create Domain srating integer CHECH (srating >= 1 and srating <= 10); // srating Attribute created with Domain range >= 1 & <= 10

3 Ex:- syntax:- create assertion assertion name (group of tables) (CHECK constraint);

SQL> create assertion (MECH (sailors primary key (sid) and Reserves primary key (Rsid, Rbid, day) and Boats primary key (bid));

Ex: create assertion CMR CHECK ((select count (s.sid) from sailors s) + (select count (b.bid) from Boats B) < 100);

TRIGGERS and Active databases:

* Trigger is a procedure that is automatically invoked by DBMS to do the specific changes into the database.
* DBA changes the triggers

3 parts in Trigger

1. Event: An event is specified as where we are excepting the changes in the DB. A change to the DB that activates the T.
2. Condition: At what time we have to perform an action or condition to take an action. A Query / test that is run when the trigger is activated.
3. Action: Used to perform the specific changes specified in trigger A procedure that is executed when the trigger is activated and it's condition is true.

Based on the level of approach in the database there are 2 types of triggers

I Row-level Triggers:- used to identify the no. of rows (count) in a table.

→ Rowlevel trigger can be performed as INSERT, DELETE, UPDATE operations.

→ Triggers also be specified either before or after performing an Operation

→ The Rows can be indentified by using the stmt called "for each Row".

II - stmt - level triggers:

→ statement level triggers is used for identifying the no.of stmt count in a file

→ stmt - level trigger can contains the stmt called 'Forevery stmt';

Ex: Consider the sailors schema and create a sailors table at initial the no.of rows in a sailors table is initilized as zero. after performing an operation Insert Every row is incremented into the table

```
SQL> create Trigger (MECH) Before Insert ON sailors
    Declare
        count number;
    Begin
        count =0;
    End;
SQL> create Trigger (MECH 2) After Insert ON sailor for each row
  Begin
      count =count+1;        // Declaration part & exception handling
                                part are optional.
  end;
```

Syntax: create trigger [trigger_name]
[before | after | INSTEAD OF] // Trigger Timing
{ insert | update | delete} // event
on [table_name]
[for each row]
[trigger_body] [when<condition for trigger to get exe>]
DECLARE <Declaration part>

```
BEGIN
        < Execution part >

Exception < Exception handling part>
END;
```

Ex: create trigger stu-marks
before INSERT
ON
student
for each row
set new.total = new.sub1 + new.sub2 + new.sub3,
new.per = new.total * 80/100;

// The above SQL stmt will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database trigger will compute those two values and insert with the entered values ie,

SQL > insert into student values (0, "ABCDE", 20, 20, 20, 0, 0);

SQL > select * from student;

tid	name	sub1	sub2	sub3	total	percentage
100	ABCDE	20	20	20	60	36

In this way trigger can be creates and executed in the DBS.

Ex: Create Trigger Emp-view-modify-trigger

```
INSTEAD OF UPDATE
ON EMP_CSE_View
FOR EACH ROW
BEGIN
UPDATE dept
SET location = : new.location
WHERE dept_name =: old.dept_name;
END;
/
```

o/p: Table created,

Ex: create Trigger init-count BEFORE INSERT ON students (Event)

```
DECLARE
    count integer;                    (Action)
BEGIN
    count := 0;        // The init count trigger is executed just once per
                       insert statement, regardless of the no. of records
                       inserted → stmt level T
END
```

```
create Trigger increment-count AFTER INSERT ON students (Event)
when (new.age < 18)  (condition)
for each Row         // increment the count for each modified
                        record → so Rowlevel Trigger //
Begin                (Action)
  count := count + 1;
  End.
```

Instead of Trigger: is a special type of trigger. It is used only in DML triggers. It is used when any DML event is going to occur on the complex view.

consider an example in which a view is made from 3 base tables. when any DML event is issued over this view, that will became invalid because the data is taken from 3 different tables. so in this INSTEAD OF TRIGGER is used.

→ This instead of trigger is used to modify the base tables directly instead of modifying the view for the given event.

Ex: create Trigger Emp_view_modify_trigger

① create Emp & DEPT tables
② create view for above tables
③ Update of view before the instead of trigger
④ Creation of the instead of trigger
⑤ Update of view after instead of trigger.

Unit - III

Schema Refinement And Normal Forms

S.No	Topic	Pg. No	classes
1)	Introduction to schema Refinement		
2)	Functional dependencies		
3)	Normal forms Introduction		
4)	Decomposition & types		
5)	1NF, 2NF		
6)	3NF, BCNF		
7)	4NF, 5NF, DCNF		
8)	Schema Refinement in DB Design		

Unit - III

Schema Refinement and Normal Forms

I. Schema Refinement :-

Schema: The overall design (description) of the database.

Refinement : Making the modifications to the Schema.

Schema Refinement is used to do changes in the database by using functional dependencies and normalforms.

→ The DB design issue for performing the Schema refinement is for reducing the redundance

→ Redundancy is defined as the process of repeating the same information at various locations.

Problems caused by Redundancy:-

The Redundancy can arise various problems of insertion, deletion & updation. 4 problems.

i) Redundant Storage : Same data at multiple locations so memory is wasted.

ii) Insert Anamolies : without verifying the previous tuples the insertion process is carriedout.

iii) Delete Anamolies : While deleting the contents of a DB can have the common conditions. So all the contents including the redundant tuples may also be deleted

from the database.

iv) Update Anamolies: Whenever update operation is carried out with the perevious contents, then we have the chance for arising the redundancy. So schema refinement is used for reducing the redundancy.

→ Schema Refinement is a process of decomposing a relation into multiple no. of smaller relations for receiving the redundancy.

Decomposition:-

→ A decomposition is the process of dividing (or) decomposing the original relation schema into multiple no. of smaller relations.

Problems Related to Decomposition:-

I. Do we need to decompose a relation?

II. What are the various problems can a decomPose will arise?

I. Do we need to decompose a Relation?

→ For decomposing a relation, first we may verify either the relation schema can have the redundancy?

→ If the relation can have the redundancy then we may prefer decomposition.

→ So the solution for this problem, may suggest by "Normal Forms."

II. What are the various problems can a decompose will arise?

→ For knowing the list of problems caused by decomposition we may prefer various Properties of decomposition & properties.

1) Loss - less join decomposition.

2) Dependency preserving decomposition.

Disadvantages of decomposition: Some information is last during retrieval of original relation or table.

1) Loss - less join decomposition :- (Non - additive join):-

→ It can states that the combination of the Smaller relations can recover the original relation or not.

→ If we decompose a relation R into relations R1 & R2.

* Decomposition is lossy if $R1 \bowtie R2 \supset R (R1 \bowtie R2 \neq R)$
* Decomposition is lossless if $R1 \bowtie R2 = R$

→ To check if for loss - less join decomposition using FD set, following conditions must hold.

① $Att(R1) \cup Att(R2) = Att(R)$
Each attribute of R must be either in R1 or in R2.

② Intersection of attributes of R1 and R2 must not be null.

Attri (R1) ∧ Attri (R2) ≠ 0

3) Common attribute must be a key for at least one relation (R1 or R2)

Att (R1) ∩ Att (R2) → Attri (R1) or Attri (R1) ∧ Att (R2) → Attri (R2)

Ex:- R (A, B, C, D) with FD set {A → BC} is decomposed into R1 (ABC) and R2 (AD) which is a lossless join decomposition as:

1) (ABC) ∪ (AD) = (ABCD)

2) (ABC) ∧ (AD) ≠ 0 (ABC ∧ AD = A)

3) Attri (R1) ∧ Attri (R2) = A is a key of R1 (ABC)

bc2 A → BC is given.

Lossy join decomposition: The decomposition of relation R into R1 and R2 is lossy when the join of R1 and R2 does not yield the same relation as in R. R⊂R1 ⋈ R2

Loss-less join D:- R1 ⋈ R2 = R

2) Dependency Preserving Decomposition:

→ It can state that for enforcing the constraints an small relations we may enforce the constraints on original relation schema.

→ If we decompose a relation R into relations R1 and R2, All dependencies of R either must be a part of R1 or R2 or must be desirable from combination of FD's of R1 and R2.

Ex:- R(A, B, C, D) with FD set {A → BC} is decomposed into R(ABC) and R2(AD) which is dependency preserving bc2 FD A → BC is a part of R1(ABC).

Ex:- Consider the schema R(A, B, C, D) and FD Set {A → B, C → D}, then the decomposition of R into R1(AB) and R2(CD) is ?

Sol For Loss-less join decomposition 3 conditions must be true.

1) Attri(R1) ∪ Attri(R2) = Attri(R) = ABCD.

2) Attri(R1) ∧ Attri(R2) = 0, which violates the condition of lossless join decomposition. Hence the decomposition is not loss-less.

For dependency preserving decomposition.

→ A → B can be ensured in R1(AB) and C → D can be ensured in R2(CD). Hence it is D.P.D.

Ex:-

R

A	B	C
a1	d1	c1
a2	d2	c2
a3	d3	c3
a4	d4	c4

R1(AB)

A	B
a1	d1
a2	d2
a3	d3
a4	d1

R2(BC)

B	C
d1	c1
d2	c2
d3	c3
d1	c4

$\Pi(R1(r) \bowtie \Pi R2(r))$ / $R1 \bowtie R2$ (Natural Join)

A	B	C
a1	d1	c1
a1	d1	c4
a2	d2	c2
a3	d3	c3
a4	d1	c1
a4	d1	c4

Here $R \subseteq R1 \bowtie R2$ So it is a lossy decomposition

Ex:- R

A	B	C	D
a1	d1	c1	d1
a2	d2	c2	d2
a3	d3	c3	d3
a4	d1	c4	d1

R1(AB)

A	B
a1	d1
a2	d2
a3	d3
a4	d1

R2(BC)

B	C
d1	c1
d2	c2
d3	c3
d1	c4

R3(CD)

C	D
c1	d1
c2	d2
c3	d3
c4	d1

$\Pi R1(r) \bowtie \Pi R2(r) \bowtie \Pi R3(r)$

A	B	C	D
a1	d1	c1	d1
a1	d1	c4	d1
a2	d2	c2	d2
a3	d3	c3	d3
a4	d1	c1	d1
a4	d1	c4	d1

Here $R \subseteq \Pi R1(r) \bowtie \Pi R2(r) \bowtie \Pi R3(r)$

So it's a lossy decomposition.

Ex:-

S	P	d
s1	p1	d1
s2	p2	d2
s3	p1	d3

Instance r

S	P
s1	p1
s2	p2
s3	p1

$\Pi SP(r)$

P	D
p1	d1
p2	d2
p1	d3

$\Pi PD(r)$

S	P	D
s1	p1	d1
s1	p1	d3
s2	p2	d2
s3	p1	d1
s3	p1	d3

$\Pi SP(r) \bowtie \Pi PD(r)$

Here $r \subseteq \Pi SP(r) \bowtie \Pi PD(r)$ So it's lossy decomposition

Ex:- R

A	B	C
a1	d1	c1
a2	d2	c2
a3	d3	c3
a4	d4	c1

R1(AB)

A	B
a1	d1
a2	d2
a3	d3
a4	d4

R2(BC)

B	C
d1	c1
d2	c2
d3	c3
d4	c1

R1 ⋈ R2

A	B	C
a1	d1	c1
a2	d2	c2
a3	d3	c3
a4	d4	c1

Here $R = R1 \bowtie R2$ So it's a loss-less Join decomposition

Ex:- R(ABC), R1(AB), R2(BC) $F.D = \{A \to B, B \to C, A \to C\}$

Here $A \to B$, $B \to C$ $F.D^+ = \{A \to B, B \to C, A \to C, AB \to C, AC \to BC\}$

So $A \to C$

Here $F1 = \{A \to B\}$ $F2 = \{B \to C\}$

$F1 \cup F2 = \{A \to B, B \to C\}$

$(F1 \cup F2)^+ = \{A \to B, B \to C, A \to C, AB \to C, AC \to BC, A \to BC\}$

So $(F1 \cup F2)^+ = FD^+$

Hence it's a Dependency preserving decomposition

Ex:- R(ABCD), R1(AB), R2(BC), R3(CD) F.D =

$\{A \to B, A \to CD, B \to C\}$

Sol

$F.D^+ = \{A \to B, A \to CD, B \to C, A \to C, A \to D, A \to BC, AC \to BC, A \to BCD\}$

$F1 = \{A \to B\}$ $F2 = \{B \to C\}$

$F1 \cup F2 \cup F3 = F1 \cup F2$

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

$(F1 \cup F2 \cup F3)^+ = \{A \to B, B \to C, A \to C, AC \to BC, A \to BC\}$

Here we observe that

$(F1 \cup F2 \cup F3)^+ \neq F^+$ So it's not a dependency Preserving decomposition.

Note: F = Set of functional dependency

F^+ = closure set of F.D / additional FD's.

Inference Rules :- 5

1) $Y \subseteq X$ then $X \to Y$ [R.H.S $\subseteq$ L.H.S] Reflexive Rule
2) $A \to B$ and C then $AC \to BC$ Augmentation Rule
3) $A \to B$, $B \to C$ then $A \to C$ Transitivity Rule
4) $A \to B$, $A \to C$ then $A \to BC$ Union Rule
5) $A \to BC$, then $A \to B$, $A \to C$ Decomposition Rule

Armstrong's Axiom : Reflexive + Augmentation + Transitivity Rule

Functional Dependency

Ex :-

A	B	C
a1	b1	c3
a2	b4	c2
a3	b2	c1
a1	b1	c3

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

$A \rightarrow B$

a_1	b_1
a_2	b_4
a_3	b_2
a_1	b_1

$B \rightarrow C$

b_1	c_3
b_4	c_2
b_2	c_1
b_1	c_3

$A \rightarrow C$

a_1	c_3
a_2	c_2
a_3	c_1
a_1	c_3

$AB \rightarrow C$

a_1	b_1	c_3
a_2	b_4	c_2
a_3	b_2	c_1
a_1	b_1	c_3

$Ac \rightarrow B$

a_1	c_3	b_1
a_2	c_2	b_4
a_3	c_1	b_2
a_1	c_3	b_1

$BC \rightarrow A$

b_1	c_3	a_1
b_4	c_2	a_2
b_2	c_1	a_3
b_1	c_3	a_1

Ex:-

A	B	C	D
a_1	b_3	c_1	d_4
a_2	b_4	c_2	d_3
a_4	b_2	c_3	d_1
a_3	b_1	c_4	d_2
a_1	b_3	c_4	d_1

F.D = $\{A \rightarrow B, B \rightarrow C, A \rightarrow C, AB \rightarrow C, AD \rightarrow B, DB \rightarrow C, AC \rightarrow B,$
$BC \rightarrow A, CD \rightarrow A, CD \rightarrow B, BCD \rightarrow A, ADB \rightarrow C, ACD \rightarrow B\}$

Ex:-

A	B	C	D
a_1	b_3	c_3	d_1
a_2	b_2	c_2	d_2
a_3	b_4	c_1	d_3
a_1	b_3	c_4	d_1
a_4	b_2	c_1	d_4

F.D = $\{A \rightarrow D, A \rightarrow B, AC \rightarrow B, BD \rightarrow A, AB \rightarrow D, B \rightarrow D, AC \rightarrow D$
$AD \rightarrow B, BC \rightarrow D, BC \rightarrow A, CD \rightarrow A, CD \rightarrow B, AC \rightarrow BD,$
$BC \rightarrow AD, ABC \rightarrow D, ACD \rightarrow B, BCD \rightarrow A, CD \rightarrow AB\}$

Functional Dependency

Ex 1:-

	A	B	C
t1	a4	b1	c3
t2	a2	b4	c2
t3	a2	b2	c4
t4	a1	b1	c3

$\Rightarrow$ Functional Dependencies

$B \rightarrow C$

$AB \rightarrow C$

$Ac \rightarrow B$

$A \rightarrow B$

a4	b1
a2	b4
a2	b2
a1	b1

For a2 we are getting 2 outputs so it's not come under function dependency

---

$B \rightarrow C$

b1	c3
b4	c2
b2	c4
b1	c3

Here b1 c3
b1 c3

for single b1 we are getting single output that is c3 so it is a F.D

$A \rightarrow C$

a4	c3
a2	c2
a2	c4
a1	c3

Not F.D

$AB \rightarrow C$

a4 b1 $\rightarrow$ c3

a2 b4 $\rightarrow$ c2

a2 b2 $\rightarrow$ c4

a1 b1 $\rightarrow$ c3

all are different

So it's F.D

$Ac \rightarrow B$

a4 b c3 $\rightarrow$ b1

a2 c2 $\rightarrow$ b4

a2 c4 $\rightarrow$ b2

a1 c3 $\rightarrow$ b1

all are different

So it's F.D

$BC \rightarrow A$

b1 c3 $\rightarrow$ a4

b4 c2 $\rightarrow$ a2

b2 c4 $\rightarrow$ a2

b1 c3 $\rightarrow$ a1

Here for

b1 c3 $\rightleftarrows$ a4, a1

So it's not F.D

---

each student have name

Sid1 — Name1

Sid2 — Name2

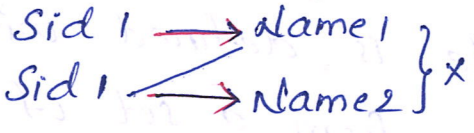
2 students have same name

Sid1 — Name1

Sid2 /



Student can't have 2 id's



1 id can't given for 2 students

Functional Dependency: Functional dependency is a relationship that arises / exists when one attribute uniquely determines another attribute.

→ F.D in a database serves as a constraint between two sets of attributes.

→ F.D is denoted by an arrow ⟶

→ F.D of x on Y is represented by $X \rightarrow Y$

→ Consider R be a relation schema t1 & t2 are tuples in R, A & B are 2 attributes in R then $t1.A = t2.A$ & $t1.B = t2.B$ then only we can define A functionally determines B as $A \rightarrow B$.

Based on the key attribute F.D are classified into 3 types.

1) Partial dependency: Non key attributes are determines from (partial) key attributes.

Ex: ABCD $AB \rightarrow$ key, $CD \rightarrow$ Numkey then $A \rightarrow CD$, $B \rightarrow CD$, $A \rightarrow C$, $B \rightarrow D$

2) Transitive dependency:

A F.D which may define over a set of non-key attributes is known as transitive

D.

$C \longrightarrow D$ is transitive D.

3) Full Functional dependency: A F.D in which the right hand side is defined with a set of non key attributes from a set of key attributes on L.H.S then it's known as full F.D

$$\underset{\text{Key}}{AB} \longrightarrow \underset{\text{Non key}}{CD}$$

Attribute Closure: It is defined as the process of identifying the list of attributes we may cover from the set of F.D. 'F'.

Ex:- $F = \{A \rightarrow B,\ B \rightarrow CD,\ C \rightarrow D\}$

Attribute Closure:

$A^+ = ABCD$	$AB^+ = ABCD$	$BC^+ = BCD$	$ABC^+ = ABCD$
$B^+ = BCD$	$AC^+ = ACDB$	$BD^+ = BDC$	$ABD^+ = ABCD$
$C^+ = CD$	$AD^+ = ADBC$	$CD^+ = CD$	$ACD^+ = ACDB$
$D^+ = D$			$BCD^+ = BCD$
			$ABCD^+ = ABCD$

Key attributes = $\{A^+, AB^+, AC^+, AD^+, ABC^+, ABD^+, ABCD^+\}$

Ex:- $F = \{A \rightarrow B,\ A \rightarrow C,\ B \rightarrow C\}$

Attribute closures:

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

Key attributes = $\{A, AB, ABC\}$.

Join Dependency: J.D is to say that the relationships in the join dependency are independent of each other.

Let R be a relation schema and let R1, R2, ---, Rn be a decomposition of R.

The relation r(R) satisfies the join dependency.

(R1, R2 --- Rn) if $\bowtie_{i=1}^{n} \Pi R_i(r) = r$

→ A J.D is defined as a further generalization of MVD in which the list of decomposition relation can satisfy loss-less join decomposition.

JD $\bowtie$ {R1, R2 --- Rn} i) $R_i = R$ $(i = 1, 2, ---, n)$

Dependency: The process of identifying a list of attributes which may arise the redundancy on the relation is called dependency.

Types of Functional Dependencies:

A F.D is an association b/w two attributes of same relational database table. One attribute is called determinant and another one is called determined.

A ⟶ B

↓ determinant ↓ determined

Eid	Ename	Salary
1	Raj	20000
2	Rohith	15000
3	Anitha	18000
4	Vinay	19000

Here
Eid → Name
Eid → Salary
Eid → (Name, Salary)

Here
Eid is determinant

Types of Functional Dependencies:

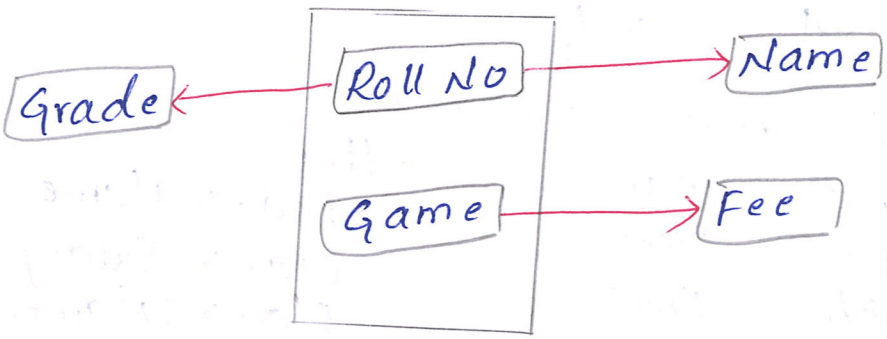
1) Partial dependency and Fully Functional dependency.
2) Transitive dependency and Non-Transitive "
3) Single valued " and Multivalued "
4) Trivial dependency and Non-Trivial ".

1) Partial dependency and Full Functional dependency:

→ Suppose we have more than one attribute in primary key let A be the non prime key attribute, if A is not dependent upon all Prime key attributes then partial dependency exist.

→ Let A be the non prime key attribute and value of A is dependent upon all primary key attribute, then A is said to be fully functional dependent.

Ex: Consider a relation student having prime Key attributes (Roll no and Game) and non prime Key attributes (Grade, name and Fee).



Here
→ Name and fee are partially dependent because

You can find the name of the student by his Roll no and fee of any game by name of game.

→ Grade is fully functionally dependent because you can find the grade of any student in a Particular game, if you know roll no and game of the student.

→ Partial dependency is due to more than one Prime key attribute.

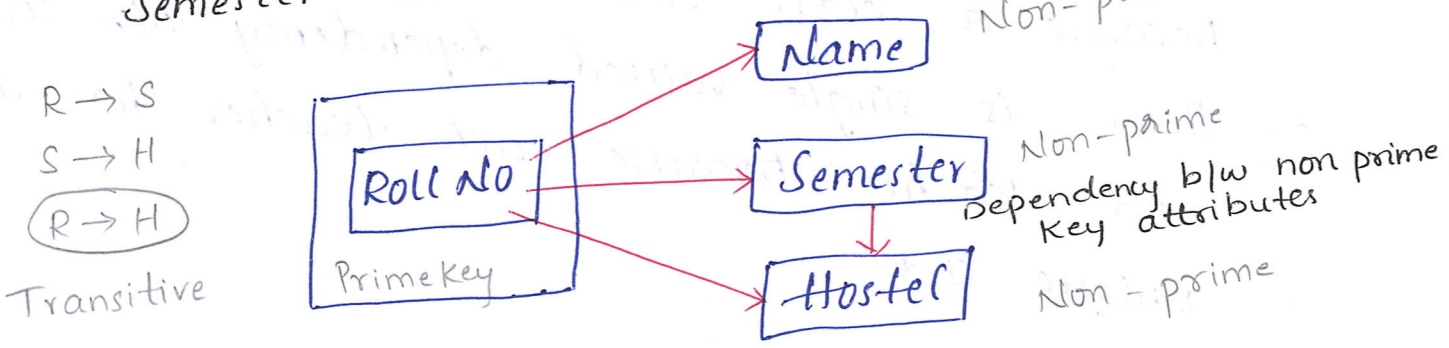
2) Transitive & Non-transitive dependency :-

→ Transitive D is due to dependency b/w non Prime key attributes. Suppose in a relation R, $x \to y$ (Y depends on X), $Y \to z$ (z depends on Y) then Y $x \to z$ (z depends on x). Therfore z is said to be transitively dependent on x.

→ Any F.D which is not transitive is known as a non-transitive dependency.

→ Non transitive dependency exists, if there is no dependency between non prime key attributes.

Ex:- Consider a relation student, having prime key attribute (Roll no) and non prime key attribute (name, semester, hostel). For each Semester there is different level.



→ Here hostel is transitively dependent upon roll no.

→ Name is non transitively dependent on Roll no.

3) Single valued and multivalued dependency:

→ In any relation R, if for a particular value of x, y has single value then it is known as Single valued dependency. $x \rightarrow y$

→ In any relation R, if for a particular value of x, y has more than one value, then it is known as multivalued dependency. it is denoted by $x \rightarrow\rightarrow y$.

Ex:- Consider a relation Teacher

Id	Teacher	class	Days
12	Sam	Computer	1
22	John	Computer	6
12	Sam	Electronics	3
22	John	Mechanical	5
32	Nick	Mechanical	2

→ There is MVD b/w Teacher and class because a teacher can take more than one class.

→ There is another MVD b/w class and days because a class can be more than one day.

→ There is single valued dependency b/w Id and teacher because each teacher has an unique Id.

A	B	C
a1	b1	c1
a1	b2	c1
a2	b3	c2
a2	b3	c2

Here $a1 \rightarrow b1$, $a1 \rightarrow b2$ So $A \rightarrow\rightarrow B$

$b1 \rightarrow c1$
$b3 \rightarrow c2$
$b3 \rightarrow c2$
} So $B \rightarrow C$

$A < {}^{1}_{2}$

ID	Name	Branch
1	A	CSE
2	A	ECE

Here
$A \rightarrow CSE$, $A \rightarrow ECE$ So Name $\rightarrow\rightarrow$ Branch
Name $\rightarrow\rightarrow$ Id

4) Trivial and Non-trivial dependency:

→ In any relation R, $x \rightarrow y$ is trivial, if $y \subseteq x$

→ In any relation R, $x \rightarrow y$ is non-trivial if $y \nsubseteq x$.

Ex:- $A \rightarrow B$
$Bc \rightarrow D$
$D \rightarrow E$
$E \rightarrow A$

$R: \{A\ B\ C\ D\ E\}$

$Bc \rightarrow BcD$
$\rightarrow BcDE$
$\rightarrow BcDEA$

$\boxed{Bc \rightarrow ABCDE}$ So it's a non-trivial

$A \rightarrow B$
$D \rightarrow E$
$E \rightarrow A$
} Non-trivial

N.T.F.D: If atleast one attribute that is not part of L.H.S then it is said to be non-trivial D.

Ex:- $AB \rightarrow A$
$AB \rightarrow B$
} Trivial
$AB \rightarrow AB$
$AB \rightarrow C$
$AB \rightarrow AC$
} Non-trivial

L.H.S $\rightarrow$ R.H.S
$BC \rightarrow ABCDE$

Here BC only present LHS
ADE not present in L.H.S
So it's a non-trivial dependency

L.H.S $\rightarrow$ R.H.S
$ABCDE \rightarrow BC$

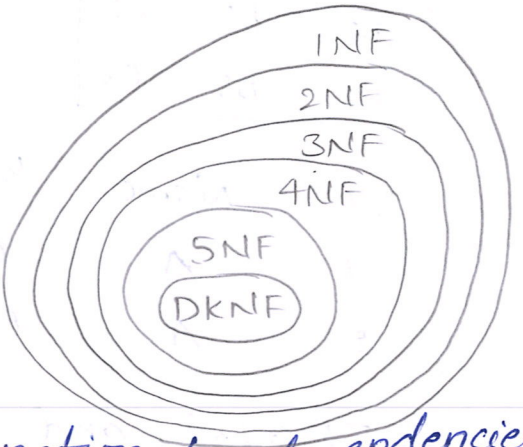
The complete R.H.S BC presents in L.H.S
So it's a trivial dependency

Note:

→ MVD is a generalization of F.D

→ JD is further generalization of MVD.

Levels of Normalization:



→ Each level is subset of the lower level.

1NF - Atomic values

2NF - must be 1NF, fully functional dependencies.

3NF - Must be in 2NF, Non transitive dependencies.

BCNF - Must be in 3NF, All attributes (L.H.S) must be the candidate keys.

4NF - Must be in BCNF & atleast 1 of the candidate key can have a single attribute.

5NF - Must be in 3NF & every candidate key can have a single attribute.

R(ABC) AB - key

Fully Functional dependency: Primekey ← Non prime key (Completely depends on all)

$AB \rightarrow C$

Partial type of F.D:- Parts of Prime key $\begin{cases} A \rightarrow C \\ B \rightarrow C \end{cases}$ Non prime

Transitive dependency: If there is a dependency between non-prime key attributes.

$\left.\begin{matrix} A \rightarrow B \\ B \rightarrow C \\ \Downarrow \\ A \rightarrow C \end{matrix}\right\}$ Transitive

$\left.\begin{matrix} A \rightarrow B \\ A \rightarrow C \end{matrix}\right\}$ Non-transitive

Normal Forms

Normal Forms: N.F can only provides a suggestion to decompose a relation into multiple number of smaller relation or not, and to check whether the database design is to be good design or not.

→ To reduce data redundancy.

→ Based on the functional dependencies we can define 1NF, 2NF, 3NF & BCNF (Boyce codd NF)

→ Based on the multivalued dependency we can define 4NF.

→ Based on the join dependency we can define JNF and DKNF (Domain key NF)

1NF - A Relation is in 1NF if domains of each attribute contains only atomic values. It means atomicity must be present in relation.

First Normal Form :- (1NF) :-

→ A table is said to be in 1NF then the table can allows atomic values only but it cannot allows multiple no. of attribute values in a single tuple into the table.

→ Every domain in R can allows single attribute value is called automic values.

→

Sid	Sname	Saddress
1	X	HYD, BGLR
2	Y	KNL, KDP
3	Z	ATP, CTR

→ 1NF was defined to disallow multivalued attributes, composite attributes and their combinations.

The above table may not be in INF because the table can allows a set of tuples with multiple no. of attribute values for making the above table into the INF we can define the above table as follows.

Sid	Sname	Saddress
1	X	HYD
1	X	BGLR
2	Y	KNL
2	Y	KDP
3	Z	ATP
3	Z	CTR

Hence the above table have been translated from the previous table which do not have atomic values.

INF Rules:

1) Table should only have single (atomic) valued attributes / columns.
2) Values stored in a column should be of the same domain.
3) All the columns in a table should have unique names.
4) And the order in which data is stored, does not matter.

Ex:-

RollNo	Name	Subject
101	Ashok	OS, CN
103	Rajesh	Java
102	Bharath	C, C++

⇒

RollNo	Name	Subject
101	Ashok	OS
101	Ashok	CN
103	Rajesh	Java
102	Barath	C
102	Barath	C++

The above table satisfies 3 rules out of the 4 rules. Here 3 different students in our table, 2 have opted for more than 1 subject, and we have stored the subject names in a single column. But as per the 1st NF each column must contain atomic value.

So we have to break the values into atomic values by doing so, although a few values are getting repeated but values for the subject column are now atomic in each row/record. *Using the 1NF, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

2 NF:

→ It is based on concept of full functional dependency. A relation is in 2NF if (i) It is in first normal form.

ii) Every non-prime attributes in R are fully functionally dependent on primary key of R.

→ A table is said to be in 2NF then the table must be in 1NF & it should be free from partial type F.D

Ex:- Consider a Relation schema R(ABC) with F = {A → B, A → C, B → C} and AB is a superkey in R identify the relation schema R is in 2NF or not.

Step 1 : Find R is in 1NF or not.

Step 2 : Identify the partial functional dependencies.

$A \rightarrow C$

$B \rightarrow C$

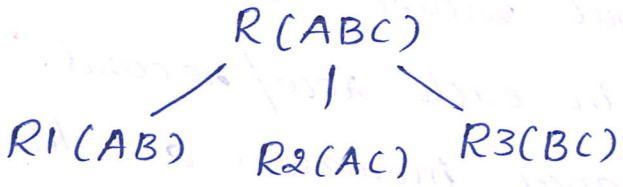
Step 3 : Identify attribute closure for above F.D's.

$A^+ \longrightarrow Ac$

$B^+ \longrightarrow BC$

$AB \rightarrow$ Primary Key

Step 4 : Decompose R into R1, R2, R3



2NF, we identify the partial type of F.D and we have to seperate them from given relation schema.

Ex:- Student (SId, Sname, EId, Ename, Grade)

Student

Sid	SName	Eid	Ename	Grade
1	x	3	A	5
2	y	2	B	4
3	z	1	C	6

2NF $\Longrightarrow$

Student

Sid	Sname
1	X
2	Y
3	Z

Employee

Eid	Ename
3	A
2	B
1	C

$Sid \rightarrow sname$

$Eid \rightarrow Ename$

$SidEid \rightarrow Grade$

Grade

Sid	Eid	Grade
1	3	5
2	2	4
3	1	6

→ Now the table is in 1NF bcz all attributes are single valued. But it is not yet in 2NF. If student leaves university and the type is deleted, then we loose all information about Employee A; Since this attribute is fully functionally dependent on the primarykey sid.

→ To solve this problem, we must create a new table Employee with the attribute Eid, Ename.

→ The third table grade is necessary for combining the two relations student & employee and to manage the grades. Besides the grade it contains only the two ids of the student and the employee.

→ If now a student is deleted, we do not loose the information about the employee.

Ex:- Schema = {city, Street, H.No, Hcolor, citypopulation}

1) Key → {city, street, H.No}

2) {city, street, H.No} → {H.color}

3) {city} → {city population}

4) citypopulation not belongs to any key, city Population is F.D on city which is a proper Subset of the key.

Convert to 2NF:

Old Schema → {city, street, H.No, Hcolor, citypopulation}

⇓

1) new schema → {city, street, HNo, Hcolor}

2) new schema → {city, city population}

Ex:- Schema → {studio, movie, budget, city}

1) Key → {studio, movie}

2) {studio, movie} → {budget}

3) {Studio} → {city}

4) City is not a part of key.

5) City FD on studio which is a proper subset of the key.

Convert to 2NF:

old schema → {Studio, movie, budget, city}

⇓

New schema → {movie, studio, budget}

new schema → {Studio, city}

2NF example: Remove Partial dependencies

Table Purchase details

Customer Id	Store Id	Purchase Location
1	1	ECIL
1	3	Uppal
2	1	ECIL
3	2	Ramantapur
4	3	Uppal

This table has composite Primary key [Customer Id, Store Id]

Purchase Location - Non key

Here Partial F.D: SId → Purchase location. Plocation is only depends on storeid, which is only part of the primary key. So this table is not in 2NF. So break the table into 2. Find partial F.D, Store id → P.l

Store id → Purchase location ↓ fully F.D

Purchase

Customer Id	Store Id
1	1
1	3
2	1
3	2
4	3

Store

StoreId	Purchase location
1	ECIL
2	Ramantapur
3	Uppal

What we have done is to remove partial F.D that we initially had. Now store id → Purchase location is fully dependent on primary key.

2NF Ex:- R(ABCDEF)

EC → Primary Key

C → F, E → A, EC → D, A → B

Note:

* Non key attribute derived from R(ABC) all Key attributes then it is a fully F.D AB — key, AB → C
* Non key attribute derived from only part of a key attribute then it is a partial type of F.D

Ex:- A → C
B → C

C → F
E → A
EC → D
A → B

* How to find a key attribute?

First write all R.H.S attributes FADB in this which attributes are missing find those.

Here EC missing. So EC is key attribute.

EC^+ = ECFABD // it covers all attributes so it's a key.

1) It is in 1NF.

2) Find partial dependencies

C → F → Partial

E → A → Partial

EC → D → fully F.D

A → B → Both nonkey

R(ABCDEF)

R1(CF) R2(EAB) R3(ECD)

Third Normal Form (3NF):- No Transitive

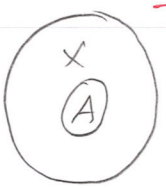
→ Third normal form is based on the concept of non-transitive dependency. A relation R is in 3NF if i) It is in 2NF.

ii) There should be no transitive dependency of a non prime attribute on the primary key.

iii) If non trivial FD $X \rightarrow A$ holds then either

a) X is a superkey or

b) A is primekey



→ If transitive dependency is found then we have to remove them from R.

→ No non prime (non key) attribute is transitively depends on any key i.e no non-prime attribute depends on other non prime attributes. All non-Prime attributes must depends only on the key attributes.

Ex:- $A \rightarrow B$
$BC \rightarrow D$
$D \rightarrow E$
$E \rightarrow A$
$BC \rightarrow ABCDE$

$BC \rightarrow D$
$D \notin BC$ } non trivial



Ex:- Vendor

Id	Name	A/C No	Bank code No	Bank Name

Id → Name, A/c, Bank name

⇓

Bank name → Bank code

Here there is a transitive dependency b/w bank code No and bank name bcz bank code no is not the Primary key

⇓

To get to the 3NF, we have to put the bankname in a seperate table together with the clearing number to identify it.

Vendor

Id	Name	A/c No	Bank name

Bank

Bank code No	Bank name

Ex:- R(A,B,C,D,E,F)

$AB \rightarrow CDE$ AB is candidate key

$D \rightarrow F$ (No candidate key) => If D is null then we can never find/search the value of F with the help of AB. So we remove Transitive D

Ex:- R(ABCD) $F = \{A \rightarrow CD, B \rightarrow C, C \rightarrow D\}$

Convert the given relation into 3NF.

$AB \rightarrow$ Key

$AB^+ = ABCD$

A	B	C	D

Sol)

① R is already in 1NF

② Identify dependencies partial-functional

$A \rightarrow CD$

$B \rightarrow C$

3) Find closures for P.F.D

$A^+ \rightarrow ACD$; $B^+ \rightarrow BCD$

4) Decompose given R(ABCD)

R1(AB) R2(ACD) R3(BCD)

5) Identify Transitive dependencies in F

$C \rightarrow D$

6) Attribute closures of $C \rightarrow D$

$c^+ \rightarrow CD$

7) Decompose R(ABCD)

R1(AB) R2(ACD) R3(CBCD)

R'3(BC) R3''(CD)

8) Remove / decompose (or) transitive dependencies from decomposition

R(ABCD)

R1(AB) R2(ACD) R3(BC) (2NF)

$A \rightarrow B$ $A \rightarrow C$ (1) $B \rightarrow C$

R4(AC) R5(AD) (3NF)

Ex:- 3NF R(ABCDEFGHIJ) AB – Candidate key.

F.D : F = { AB → C All key → non key (F.F.D)
A → DE Part of key → non key (P.F.D)
B → F " " (P.F.D)
F → GH non key → non key (Transitive)
D → IJ } " " "

1) It is in 1NF

2) Find partial type of F.D & closures of them.

A → DE A^+ = ADEIJ

B → F B^+ = BFGH

3) Decompose R(ABCDEFGHIJ)

R1(ABC) R2(ADEIJ) R3(BFGH)

4) Find transitive F.D

F → GH F^+ = FGH

D → IJ D^+ = DIJ

5) Decompose R(ABCDEFGHIJ)

2NF

R1(ABC) R2(ADEIJ) R3(BFGH)

3NF

R'_2(ADE) R''_2(DIJ) R'_3(BF) R''_3(FGH)

Ex: 3NF R(ABCDEFGHIJ)

F = AB → C
BD → EF
AD → GH
A → I
H → J

Part of key → non key ⇒ P.F.D
" → " ⇒ "
" → " ⇒ P.F.D
" → " ⇒ P.F.D
non key → non key ⇒ T.F.D

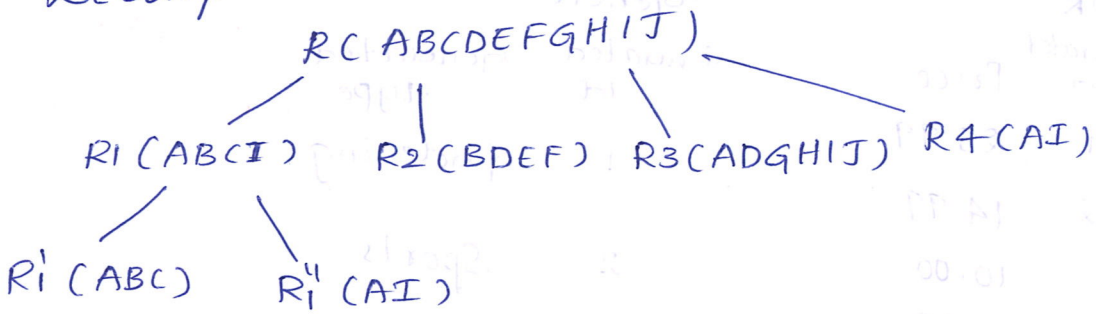
ABD missing

1) It is in 1 NF.

2) Partial dependencies

AB → C $AB^+ = ABCI$
BD → EF $BD^+ = BDEF$
AD → GH $AD^+ = ADGHIJ$
A → I $A^+ = AI$

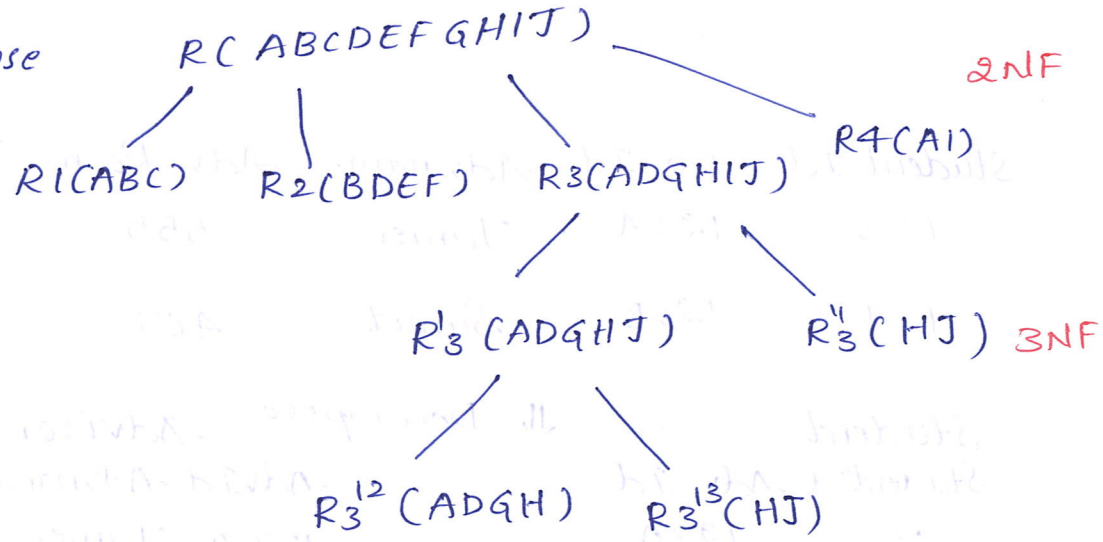
3) Decompose



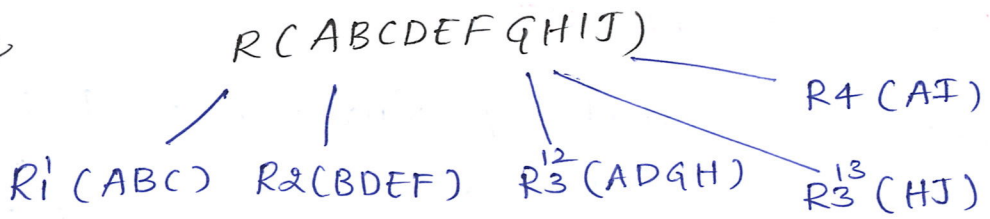
4) Transitive dependencies

H → J $H^+ = HJ$

5) Decompose



Finally



Ex:- Book details (3NF)

Book Id	Generated Id	Generate type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

BookId → Primary key

Book Id → price

Book Id → Gener Id

Genere Id → Gener Type

⇓

Book Id → Gener Type via Gener Id

All are unique

So it's a primarykey ⇓ Decompose (We plot table into 2 tables)

Book

Book Id	Generated Id	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

Genere

Generated Id	Generated type
1	Gardening
2	Sports
3	Travel

BookId → generId

BookId → price

generId → gtype

Ex:- Student Table:

Student Id	Adv Id	Adv name	Adv Room
123	123A	James	555
124	123B	Sumit	467

SId → AId

AId → AN

SId → AN

} Transitive FD

Student

StudentId	Adv Id
123	123A
124	123B

⇓ Decompose

Advisor Table

AdvId	Advname	Adv Room
123A	James	555
123B	Sumit	467

Boyce Codd Normal Forms:

Basic definition: A table is said to be in BCNF then the table must be 3NF & in which the list of all attributes must be a candidate key & every non-trivial F.D can have it's determinent is a candidate key.

→ BCNF is considered to be the stronger than 3NF.

Formal definition: A F.D $X \rightarrow A$ in R can satisfies any one of the following stmts is to be true then R is said to be in BCNF.

i) X is superkey.

ii) $A \subseteq X$ then it is a trivial F.D.

iii) For every FD, LHS is a superkey.

Ex: Employee table:

Emp-ID	Emp-Country	Emp-dept	Dept-type	Emp-dept no
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table functional dependencies are as follows

EmpId → Emp-Country

Emp dept → {Dept-type, Emp-dept-no}

Candidate key; {EmpId, Emp-dept}

The table is not in BCNF bcz neither emp-dept nor emp-Id alone are keys. To convert the given table into BCNF, we decompose it into 3 tables :-

Emp-country, Emp-dept, Emp-dept-mapping table

Emp-Id	Emp-Country
264	India
364	UK

I Emp-Country table

Emp-dept	Dept-type	Emp-dept-No
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

II Emp-dept table

Emp-Id	Emp-dept
D394	283
D394	300
D283	232
D283	549

III Emp-dept mapping table

Here functional dependencies: L.H.S = R.H.S

EmpId → Emp Country

Emp-dept → {dept-type, Emp-dept-No}

Emp-Id → Emp-dept

Now this is in BCNF b/c Left side part of both the FD is a key.

Differences between 3NF and BCNF:

3NF	BCNF
1) 3NF States that no non-prime attribute must be transitively dependent on the candidate key of the Relation (No-transitive)	1) If a trivial F.D $X \rightarrow Y$ exists for a relation then X must be a Super key.
2) 3NF can be obtained without sacrificing the dependency of relation (it preserves all F.D's)	2) Dependency may not preserved while obtaining BCNF (It may not preserves all F.D's)
3) 3NF can be achieved without loosing any info from the old table.	3) BCNF we may loose some information from the old table.
4) It concentrates only on primary key.	4) It concentrates not only on a primary key but also on candidate key.
5) Redundancy is high	5) Redundancy is low.
6) In F.D of $X \rightarrow A$ must be a part of same key.	6) In every F.D every (L.H.S) attribute is a candidate key.
7) The R must be in 1NF, 2NF	→ The R must be in 1NF, 2NF, 3NF.
8) Lossless decomposition can be achieved in 3NF.	8) Loss-less decomposition is had to achieve in BCNF. → Lossy Join decomposition will be achieved.

Finally BCNF is much restrictive than 3NF. which help in normalizing the table more. The relation in 3NF has minimum d redundancy left which is further removed by the BCNF.

Ex:- R. F.D { $A \to B$, $A \to CD$, $B \to C$, $C \to D$ }

R(ABCD)

① R is in 1NF.

② Partial functional dependencies $A \to CD$, $B \to C$

③ Identify Attribute closures $A^+ = ACD$, $B^+ = BCD$

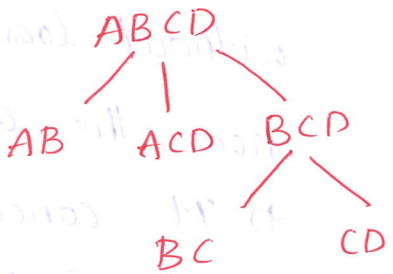
④ Decompose R(ABCD)

R1(AB) R2(ACD) R3(BCD)

⑤ Identify transitive dependencies

$C \to D$

⑥ Find closures $c^+ = CD$



⑦ Decompose (ABCD)

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

R1(AB) R2(ACD) R3(BCD)

$A \to B$

R'2(AC) R''2(AD) R'3(BC)

8) check that all the list of attributes can have the candidate keys or not.

So, Now the R(ABCD) is in BCNF.

BCNF Ex:- R (ABC)

AB → C
C → B

} It is 2NF, 3NF
it is not in BCNF

Candidate key — AB, AC

AB → C
P → NP => F.F.D

C → B
Part of C.K → P

Partial F.D	$\alpha \rightarrow B$	
(Part of C.K) P	NP	2NF
Transitive	$\alpha \rightarrow B$	
NP	NP	3NF
	$\alpha \rightarrow B$	BCNF
P/NP	P	

α Should be super key

BCNF
$\alpha \rightarrow B$
P/NP → P

R (ABC)

R1(ABC) — AB is key, AB → C

R2(CB) — Here C is key, C → B

R (ABCD)

Ex:- F = { A → B, AB → D, B → C }

loss - less Join D

A → B
AB → D
B → C

(A) → BDC
A is candidate key

AB → ABDC

A → B
AB → D — Key → Non key
B → C — part of key → non key

Here A → B, AB → D } Both L.H.S are Super key

B → C Here B is not a super key

R (ABCD)

R1 (ABD) — A → D, A → B, A → BD and A is a key

R2 (BC) — B → C, B is a key

BCNF

Ex:- R (ABCD)

it must be super key (K / NK, P / NP) → (P)

F.D : { AB → C F.F.D Key — NK
AB → D F.F.D K — NK
C → A } C, D are not super key / C.K Part of key — part of key
D → B Part of key — Part of key

Candidate key's are

AB
BC
CD
AD

So C → A, D → B } in 3NF But not in BCNF.

So decompose R (ABCD)

R1(CA) R2(BCD)

R2'(DB) R2''(CD)

Ex:- R(A, B, C) F = { A → B, B → C } key = {A}

Here in B → C, B is not super key

R (ABC)

R1(BC) R2(AB)

Ex:- R(ABCD) F = { A → B, BC → D, A → C } A key

Here in BC → D BC is not super key

So R(ABCD)

R1(AB) R2(AC) R3(BCD)

Ex:- R(ABCDEFGH) AE — Candidate Key

F.D

A → BC part of key → non key (partial F.D)
B → D Non key → non key (Transitive)
E → FG Part of key → non key (partial F.D)
AE → H All key → non key (Fully F.D)

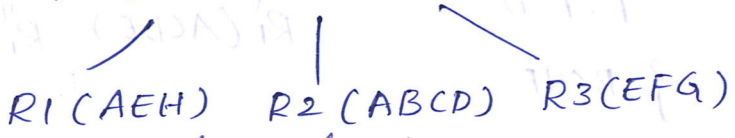
① 1NF

② Partial dependencies

$A \to BC$ $\quad A^+ = ABCD$

$E \to FG$ $\quad E^+ = EFG$

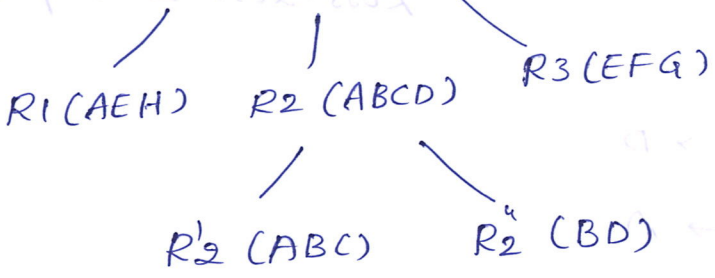
③ Decompose R (ABCDEFGH)



④ Find transitive dependencies

$B \to D$ $\quad B^+ = BD$

⑤ Decompose R (ABCDEFGH)



$A^+ = ABCD$

So decompose D keep only 'B'

BCNF Ex:- R (ABCD) / $F = \{A \to BCD, BC \to AD, D \to B\}$

$A \to BCD$ Here A is superkey

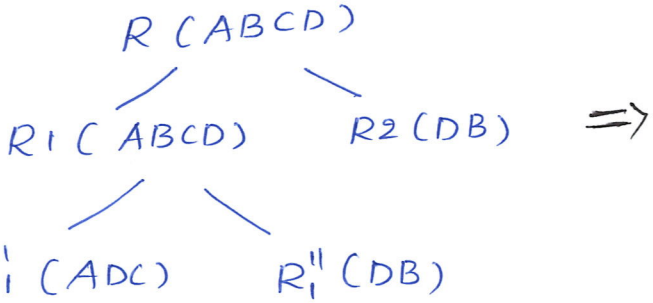
$BC \to AD$ Here BC is superkey but

$D \to B$ Nonkey → key So there is no dependency

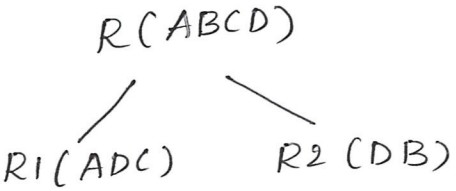
blw them. So we decompose them.

A → Super key

Bc → Superkey



⇒



$AB^+ = ABCD$

Here D gives B So we keep only 'D' we decompose 'B'.

BCNF Ex:- R(ABCDE) AB, CD, C are candidate keys

$F = \{$

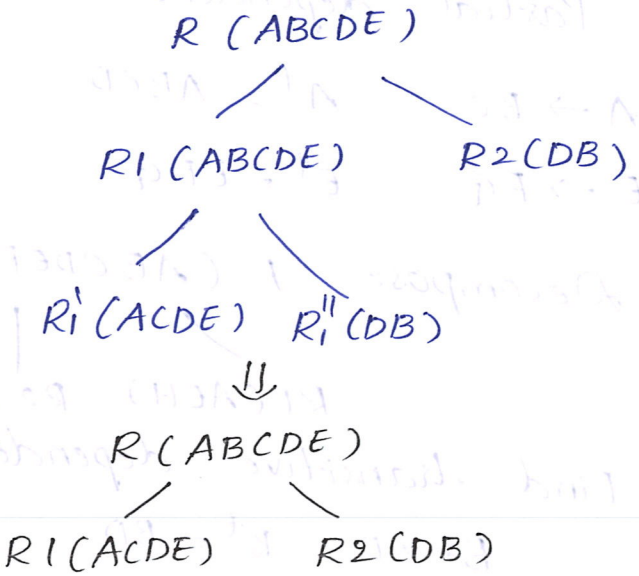
$AB \rightarrow C$ F.F.D

$CD \rightarrow E$ F.F.D

$C \rightarrow A$ F.F.D

$C \rightarrow D$ F.F.D

$D \rightarrow B$ F.F.D } BCNF



Loss-less & dependency preserving

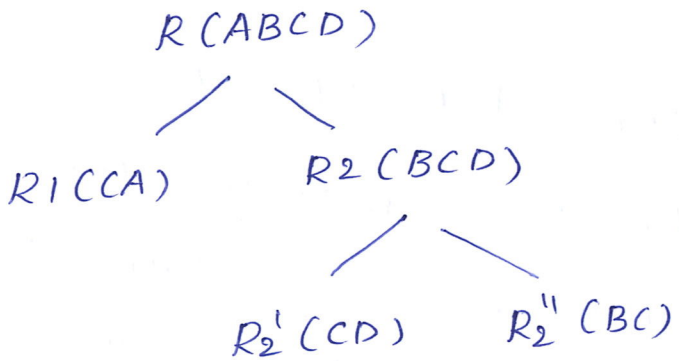
BCNF Ex:- R(ABCD)

$F = C \rightarrow D$

$C \rightarrow A$

$B \rightarrow C$

Candidate key B



4NF (4th Normal Form) : — No multivalued attributes

→ 4NF is a generalization of BCNF. And it must be defined from the BCNF.

Basic definition: A table is said to be in 4NF then it must be in BCNF & atleast one of the candidate key can have a single attribute.

Formal definition: Let us consider R is relation Schema X, Y are attributes in R then the following multivalued dependency (MVD) X multidetermines Y.

$X \rightarrow\rightarrow Y$ can satisfies any one of the following 2 statements is to be true then R is in 4NF.

1) $Y \subseteq X$ or $R = XY$

2) X is a superkey.

→ MVD always requires atleast 3 attributes b/c at consists of atleast 2 attributes that are depends on 3rd attribute. must (3 columns in a table)

→ For a dependency $A \rightarrow B$, if for a single value of A, multiple value of B exists.

* In 4NF, it mustnot contain more than 1 MVD i.e., the table should not have more than 1 MVD.

→ A table with a MVD violates the 4NF bcz it creates unnecessary redundancies and can

Contribute to inconsistent data.

→ To bring this up to 4NF, it is necessary to break this information into 2 tables.

→ 4NF Reduces Redundancy.

Ex:-

Student R1

Sid	Sname
S1	A
S2	B

Course R2

Cid	Cname
C1	C
C2	D

$R = R1 \times R2$

Sid	Sname	Cid	Cname
S1	A	C1	C
S1	A	C2	D
S2	B	C1	C
S2	B	C2	D

MVD are Sid →→ CID

Sid →→ Cname

Sname →→ cname

R ⇓ decompose

R1 R2

Ex:- Student details : R

Primary key : {Sid, Subject, Activity}

Sid	Subject	Activity
100	Music	Swimming
100	Accounting	Swimming
100	Music	Tennis
100	Accounting	Tennis
150	Maths	Jogging

=> Many Students id have same Subject.

=> Many Student id have same activity.

=> this violates (not in) 4NF.

=> So decompose the table to convert into 4NF.

Old Schema → {Sid, Subject, Activity}

New Schema → {Sid, Subject}

New Schema → {Sid, Activity}

R1

Sid	Subject
100	Music
100	Accounting
150	Maths

R2

Sid	Activity
100	Swimming
100	Tennis
150	Jogging

Now this table is in 4NF (No multivalued attributes)

Activity → Sid

Subject → Sid

5NF (5th Normal Form): (or) PJNF (project join NF):

A Relation R is in 5NF if and only if it Satisfy following conditions.

1) R Should be already in 4NF.
2) It cannot be further non loss decomposed (JD)

→ A Relation R is in 5NF if and only if every ID in R is implied by the candidate keys of R.

→ A Relation is decomposed into 2 relations must have loss-less join property, which ensures that no spurious (or) extra types are generated, when relations are returned through a natural join.

Table : ACP

R

Agent	Company	Product
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

R1

Agent	Company
A1	PQR
A1	XYZ
A2	PQR

R2

Agent	Product
A1	Nut
A1	Bolt
A2	Nut

R3

Company	Product
PQR	Nut
PQR	Bolt
XYZ	Nut
XYZ	Bolt

$R1 \bowtie R3 \bowtie R2 \Rightarrow ACP$ B

$R = R1 \bowtie R2 \bowtie R3$

Loss - less Join (

Here the sequence of join is independent in any order $(R1 \bowtie R2) \bowtie R3$ (or) $R1 \bowtie (R2 \bowtie R3) \Rightarrow R$

$\Rightarrow$ Join should be Loss-less.

5NF

Ex:

R1 (AB)

A	B
a1	d1
a2	d2
a3	d3
a4	d1

R2 (BC)

B	C
d1	c1
d2	c2
d3	c3
d1	c4

R1 $\bowtie$ R2

A	B	C
a1	d1	c1
a2	d2	c2
a3	d3	c3
a4	d1	c4

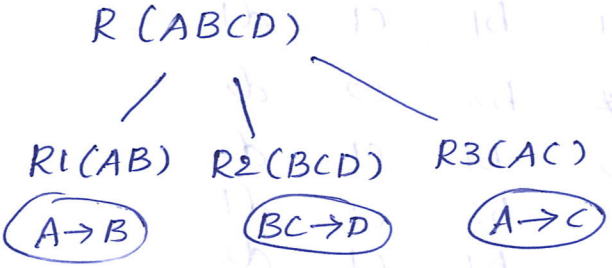
R

A	B	C
a1	d1	c1
a2	d2	c2
a3	d3	c3
a4	d1	c4

Here $R1 \bowtie R2 = R$ So it's in 5NF.

Ex:- R(ABCD) A is Key

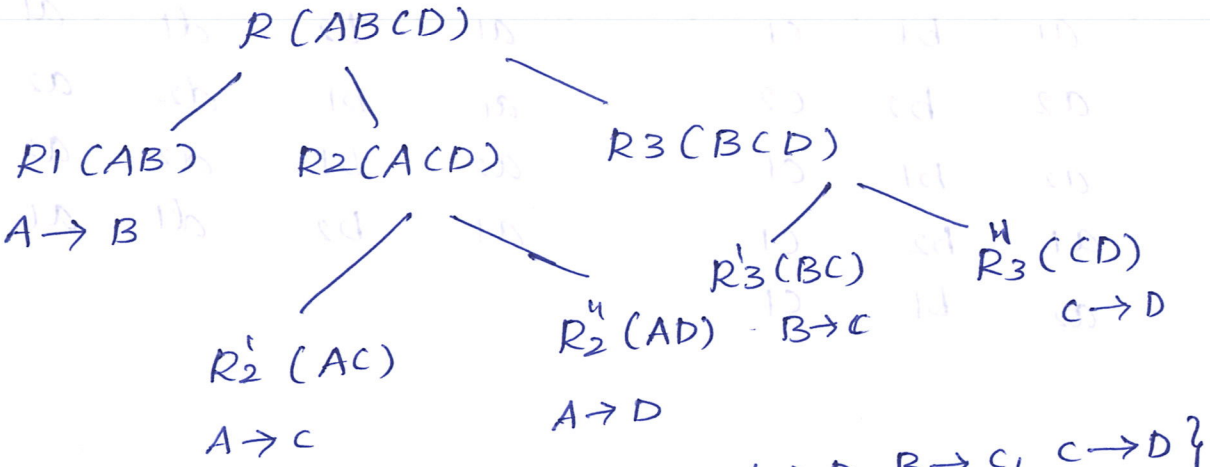
F.D = { A → B
BC → D
A → C }



R1 ⋈ R2 ⋈ R3 = R

5NF loss-less

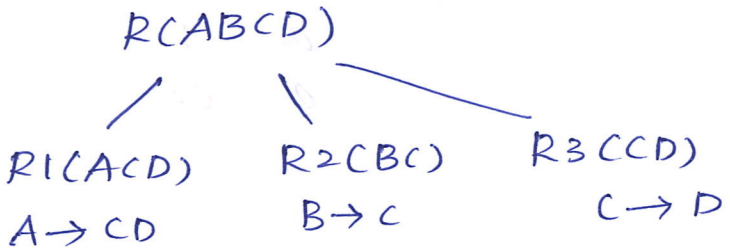
Ex:- R(ABCD) F = { A → CD , B → C , C → D } AB - Key



R1 ⋈ R2 ⋈ R3 = { A → B , A → C, A → D, B → C, C → D }

R ⊆ R1 ⋈ R2 ⋈ R3 So it's a lossy join D.

It's not in 5NF.



Now it is in 5NF.

R1 ⋈ R2 ⋈ R3 = R

{ A → CD , B → C, C → D } = { A → CD , B → C, C → D }

4NF

ex:-

R

A	B	C	D
a1	b1	c1	d1
a2	b2	c2	d2
a2	b1	c1	d2
a1	b2	c2	d1
a2	b1	c1	d2

A	B
a1	b1
a2	b1

R1

A	B	C
a1	b1	c1
a2	b2	c2
a2	b1	c1
a1	b2	c1
a2	b1	c1

R2

A	B	D
a1	b2	d1
a1	b1	d2
a2	b1	d2
a1	b2	d1

a1 b1 d1

a2 b2 d2

a2 b1 d2

a1 b2 d1

Ex:-

R

A	B	C
a	b1	c1
a	b2	c2
a	b1	c2
a	b2	c1

A →→ B

B →→ C

A →→ C

R1

A	B
a	b1
a	b2

B	C
b1	c1
b1	c2
b2	c1
b2	c2

R2

A	C
a	c1
a	c2

Ex:-

R

A	B	C	D
a	b1	c1	d1
a	b2	c2	d2
a	b2	c2	d2
a	b1	c1	d1

A	B	C
a	b1	c1
a	b2	c2

A	B	D
a	b1	d1
a	b2	d2

Schema Refinement in Database Design :-

It is based on 4 steps.

1) Constraints an entity sets
2) Constraints on Relationship sets
3) Identify attributes for entity set
4) Identify the entity set

→ The process of schema refinement is mainly define for reducing the redundancy into the relation, but in the database design the Schema refinement can be performed over a multiple no. of tables with in the database.

1) **Constraints an Entity Set :** The S.R in DB design can performed initially based on list of entity Sets which may arise the redundancy into the relation then we can identify a primary key constraints and the list of attributes which may arise the redundancy.

2) **Constraints on Relationship sets :** In this step we can prefer the (key constraint) decomposition based upon the key constraint if the key constraint can allow the redundancy then we can prefer decomposition otherwise there is no need of decomposition at this step.

3) Identify attributes on entity set : In the Previous step if we may prefer decomposition then the new entity set can requires various Set of attributes from the DB which may not arise the redundancy.

4) Identify the entityset : From the above step with various set of identified attributes can have the chance for arising the redundancy then we can prefer to identify various constraints on new entity set.

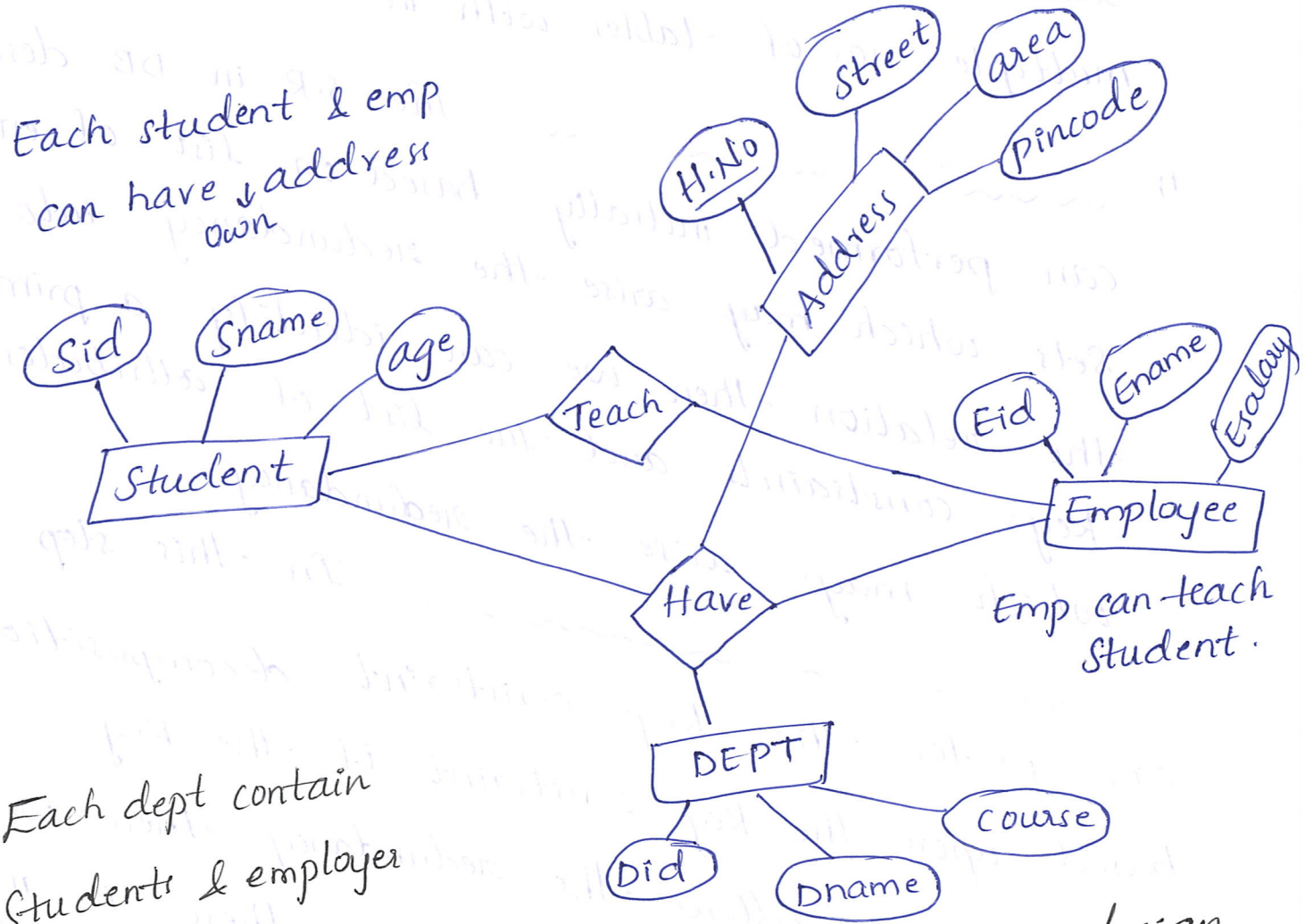


fig: After Schema Refinement into DB design

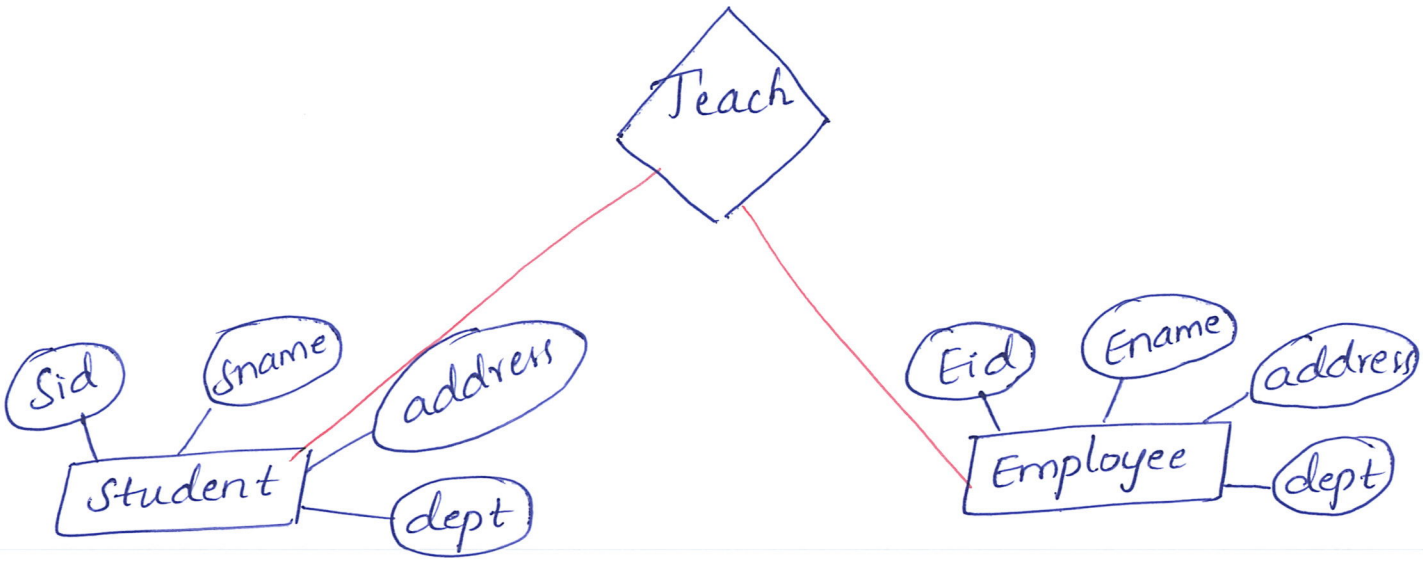


fig: before schema Refinement into DB design

Here address arise Redundancy

dept arise redundancy

So we decompose the DB design into multiple no. of tables.

Unit - IV

Transaction Management

Transaction: A sequence of sql stmts that are executed as a unit

* Execution of a program against the DB (set of read & write operations will perform)
* T is nothing but doing a simple operation
* T access data using 2 operations

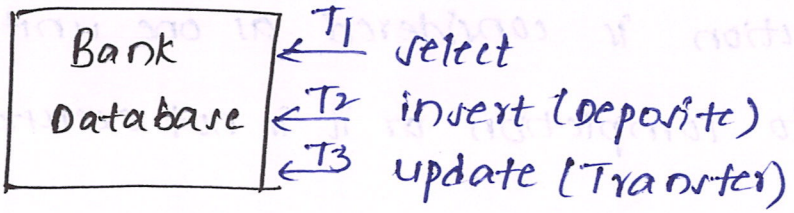
① R(x) - just Read the data item ex: R(A)

② W(x) - Read and modify the data by using same operations ex: Write(A) [A = A - 50]

Transaction Management:

* Its a program unit whose execution may change the content of database
* Collection of operations that form a single logical unit of work which accesses and possibly modifies the contents of a DB is called Transaction management

ex:



ex: T1: Read (A) A = 1000

A := A - 50 A = 1000 - 50 = 950

Write (A) A = 950

ex: T_2 : Read (A) — A = 1000

Temp := A * 0.1 — temp := 1000 × 0.1 = 100

A := A - temp — A := 1000 - 100 = 900

Write (A) — A = 900

In order to maintain consistency in a database before & after transaction, certain properties are followed. There are called ACID properties

Properties of a transaction:

* To make database transaction is in consistent state we use same properties called ACID properties
* To ensure integrity of data, the DBS should maintain following ACID properties

① Atomicity (A):

* Either all operations of the transaction are reflected properly in the database or none.
* We mean that either the entire transaction takes place at once or doesn't happen at all
* There is no midway i.e Transactions donot occur partially
* Each transaction is considered as one unit and either runs to completion or it is not executed at all
* It involves following two operations

i) Abort: If a T aborts changes mode to DB are not visible

ii) Commit: If a T commits, changes mode are visible

* Atomicity also known as "All or nothing rule":
* If transaction doesnot complete its execution, the database system restores the old values
* Atomicity is handles by the transaction management component
* To ensure atomicity DBs keeps track of the old values of any data on which a transaction performes a write

Atomicity

Ex:

	A = 1000	B = 2000
T_1: Read (A)	1000	
A := A - 50	950	
write (A)	950	
failure		
T_2: Read (B)	2000	
B := B + 50	2050	
write (B)	2050	

Ex: initially x : 500 y = 200

T_1: Read (X)	500
X : X - 100	400
write (X)	400
failure	
T_2: Read (Y)	200
Y : = Y + 100	300
write (Y)	300

Here transaction fails after completion of T_1 but before completion of T_2 (say, after write(X) but before write (Y)) then the amount has been deducted from X but not added to Y

This results an inconsistent database state

To avoid this atomicity should be ensured

T_1	T_2
① R(A) = 1000	
② A = A - 50	
	③ R(B) 2000
	④ B = B + 50
	⑤ W(B)
⑥ W(A)	
⑦ commit	

consistency will occur

* So in 'A' either all transactions are executed none will execute

After 1,2,3,4,5,6 T executed then only commit will execute

After 1,2,3,4,5 only we write (A) to ensure consistency

Consistency

A = 1000 B = 2000

T1		T2	
R(A)	1000		
A = A - 50	950		
W(A)	950		
R(B)	2000		
B = B + 50	2050		
W(B)	2050		
		R(A)	1000
		Temp = A * 0.1	100
		A = A - temp	900
		write (A)	900
		R(B)	2000
		B = B + temp	2100
		W(B)	2100

Maintain constant data in all transaction.

A+B = 950 + 2050 = 3000 A+B = 900 + 2100 = 3000

The 'C' requirement is that the sum of A & B unchanged by the execution of the transaction

Consistency: DB transaction must change affected data only in allowed ways. Any data written to the database must be valid according to all defined rules including constraints, triggers & any

combination

* It represents the consistency of the referential integrity of the database, unique primary keys in table etc..

'C'- Maintain the constant data after execution

Isolation: If several transactions are executed concurrently their operation may interleave in some undesirable way, resulting in an inconsistent state

* So by this property are transaction is executing then the other one is in waiting position

Durability: It guarantees that, once a transaction completes successfully, all the updates that it carried out on the DB persist, even if there is a system failure after the transaction completes execution

* Ensuring durability is the responsibility of a component called the recovery management

* Ensuring the isolation property is the responsibility of concurrent control component

If there is no concurrency after drawing money by one person then only the one more person will draw money in another area it waste time more so we go for concurrency

Advantages:

* waiting time decreases
* Response time decreases
* Resource utilization increases
* Efficiency increases

Why we are controlling concurrency eventhough we have there many advantages

Problems by concurrent Exe:

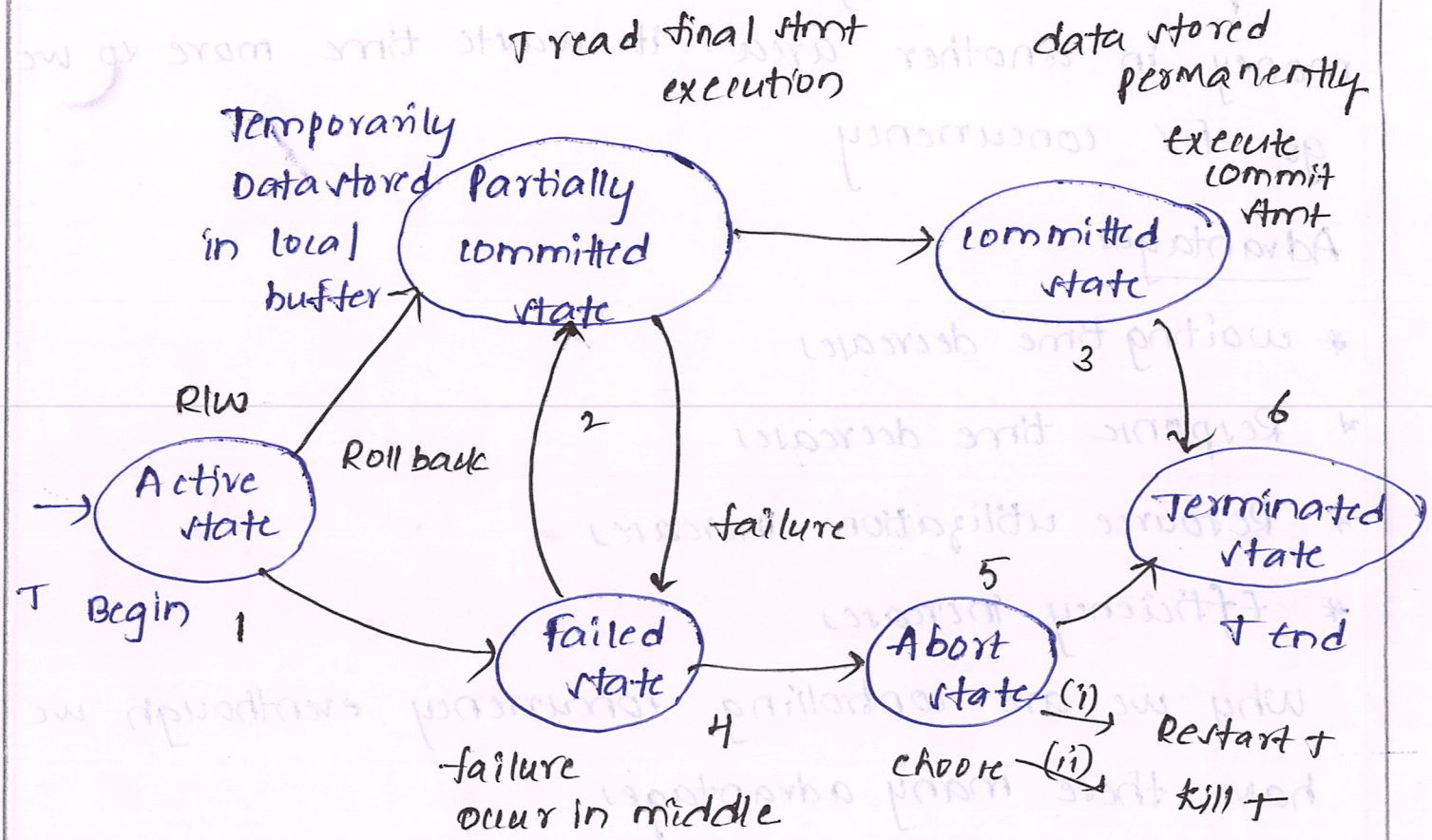
* Simultaneous execution of transaction over a stored DB can create several data integrity & consistency problems

The main problems are

* lost updates (ww conflict)
* Uncommitted data (WR conflict / dirty Read)
* Inconsistent retrievals (unrepeatable reads / RW conflict)

⇒ conflicts (WR, RW, WW) Serializability of transaction

Transaction states:



1. 1-2-3-6 || T exe successfully
2. 1-4-2-3-6 || T exe successfully
3. 1-4-2-4-5-6 || T fails due to system crash/os problem

Concurrency control:

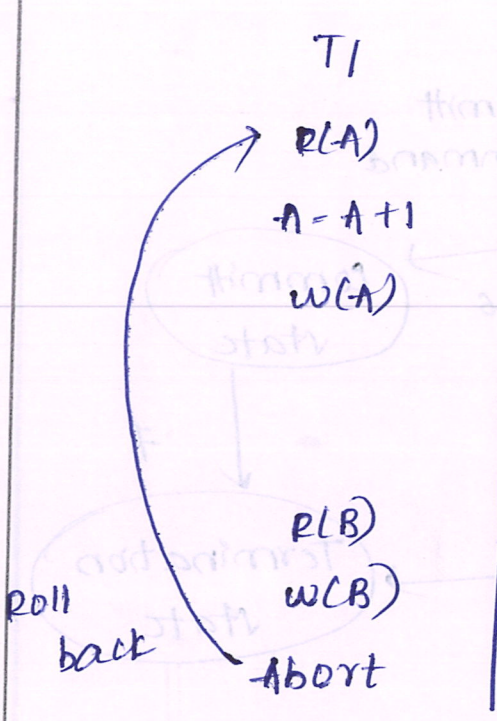
Concurrency: Executing multiple transactions at a time

Ex: ATM machines [so many members access the different atm machines at a time in different locations]

Ex: Railway ticket booking in tatkal [so many people are accessible the tatkal portal at same time].

(schedule) serializability: A schedule that is equivalent to some serial execution of a transaction

① Reading uncommitted data (WR conflict, "dirty Read"):



→ dirty read (without committing T1 we are reading A from local buffer)

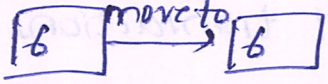
→ T2 read a value which is not committed

→ If T2 commits 6 stored permanently in DB

↓

all T done up to now will be failed again T1 starts from starting onwards A=5 in T1 & A=6 in T2 so conflicts will arise (if T1 will not Abort then A=6 in T1 also)

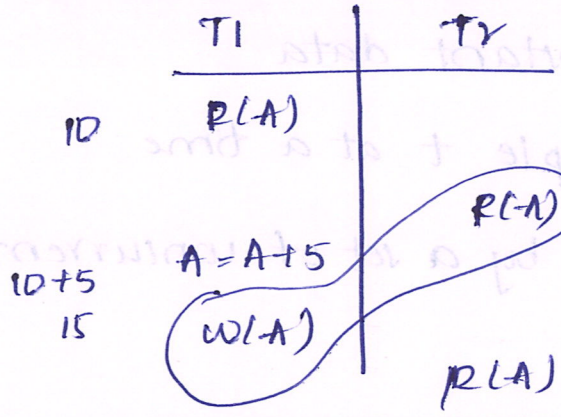
Partially committed



local buffer DB

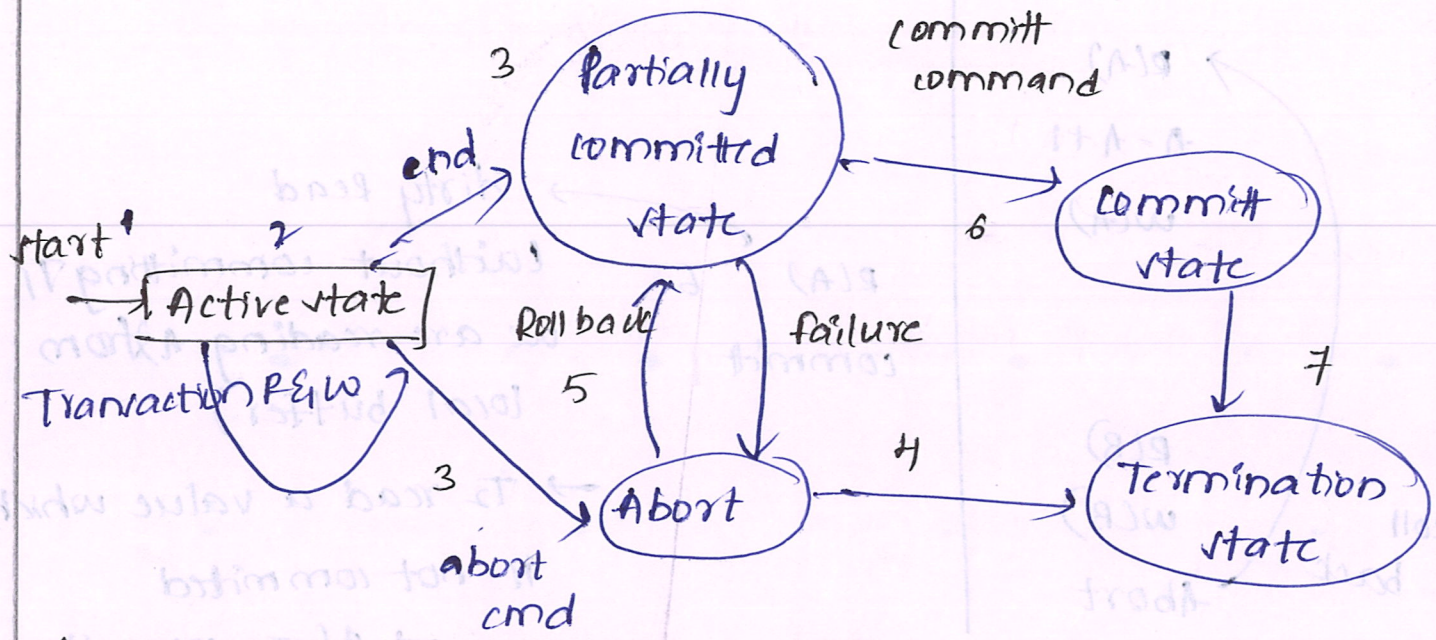
T1 T2

② Unrepeatable Read / RW conflict:



confusion will rise here what value A is 10 or 15 because of unrepeatable read operations in T2

Transaction: It is an execution of a program against the DB (set of read & write operations)



Properties of transactions: ACID

A - automicity :- all transactions are executed (or) none are executed

C - consistency :- maintain constant data after execution

I - Isolation :- one t is executing then the another one is in waiting

D - Durability :- after t is executed the modifications will updated to DB

Consistency: Maintaing constant data

Concurrency: Executing multiple t at a time

Schedule: sequence of Rlw by a set of concurrent transactions

Recoverability: converting / rollbacking failed T → partial committed T

Lock: A lock is a variable associated with a DB object (data item)

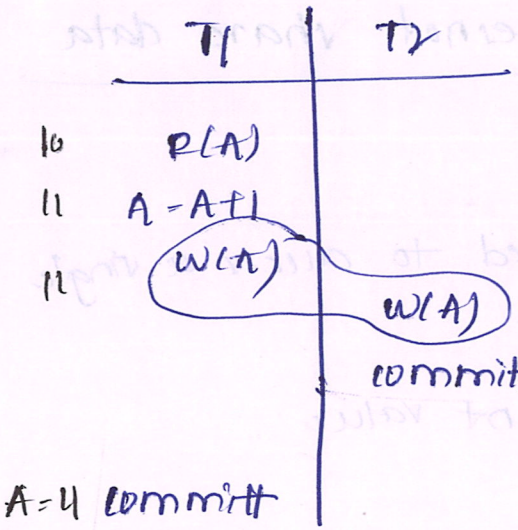
shared lock (S): only Read but not write

Exclusive lock (X): both Read & write

Dead lock: one t is waiting for another t to release a lock to prevent deadlocks perform rollbacking operations and T

Serializability: To avoid interference & guaranteed to ensure consistency

③ lost updated / ww conflict: Update of one T is overwritten by another T



→ without reading data directly if we write the data then it takes some other value

ww conflict lost updated value

Ex:

T1 (credit)	T2 (debit)
R(A) A=500	R(A) A=500
A=A+100 A=600	A=A-50 A=450
W(A) A=600	W(A) A=450

Here updated value of A in T1 is missed / lost due to ww conflict

if we run T_1, T_2 concurrently then we lost updated data

* If we run T_1, T_2 by using isolation then we got

A = 550

	T_1 (credit)	T_2 (debit)	
~~500~~	R(A)		
600	A = A+100		
600	W(A)		
		R(A)	600
		A = A-50	800-50 = 550
		W(A)	550
		commit	

* In serial execution there problems (dirty read, unrepeatable read , and lost update) would not arise since serial execution doesnot share data items
* More than 1 transaction may need to access a single data item for r/w purpose &
* database could contain inconsistent value

Schedule: The order in which the instructions are executed in the system

* A schedule is a process creating a single group of the multiple parallel transactions & executing them one by one
* It should preserve the order in which the instructions appear in each transaction
* If 2 transactions are executed at the same time, the result of one transaction may affect the o/p of the other T
* A schedule is a collection of many transactions which is implemented as a unit

Transactions will arranged in a schedule of 2 types

① Serial ② Concurrent

① Serial schedule: One after another

* The transactions are executed one after another
* Serial execution doesnot share data items among many T because every time one T is only executing

Drawbacks:

→ Inefficient
→ T suffers from longer waiting times & response time
→ low amount of resource utilization

* In serial execution dirty read, unrepeatable read, and lost update problems will not arise since it won't share data items

② Concurrent schedule: Execute multiple transactions at a time

* In concurrent schedule CPU time is shared among 2 or more T in order to run them concurrently & we must maintain proper context switching in order to maintain consistency & isolation properties of T

Recovery system → Restore the DB
→ provides high availability
→ minimize time

Failure classification: Disk crash
power outage
S/w error
a fire in m/c room

(1) Transaction failure: 2 types of errors T may no longer continue with its normal execution because

(a) logical error: some internal cdt such as bad I/p data not found, overflow or resource limit exceeded.

(b) System error: The system has entered an undesirable state (deadlock) - so reexecute.

ii) System crash: → loss the content of volatile storage & halt T processing.

→ non corrupt the nonvolatile storage content.

→ due to hardware malfunction / bug

in DB s/w or in OS

This is called Fail-stop assumption

iii) Desk Failure: Desk block loses its content

a head crash / Failure during data transfer operation

→ copies of data another disks / archival backups on tertiary media or DVD or taps } used to recover from failure

So we use recovery algorithm 2 ways:

① Action taken during normal transaction processing

② Actions taken after a failure to recover the DB

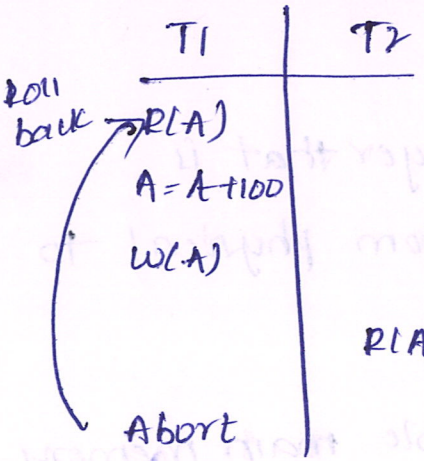
Storage structure: 3 categories

Recovery algorithm

① Transaction Roll back

② Undoing: Undo T

③ Deferred update: NO Undo / redo algorithm



If T_i fails no need to undo because the updated values just stored in local buffer only so it not update the DB. so just REDO the same operations once again to store the changes in DB

④ Immediate update: Undo / Redo Algorithm

Roll back

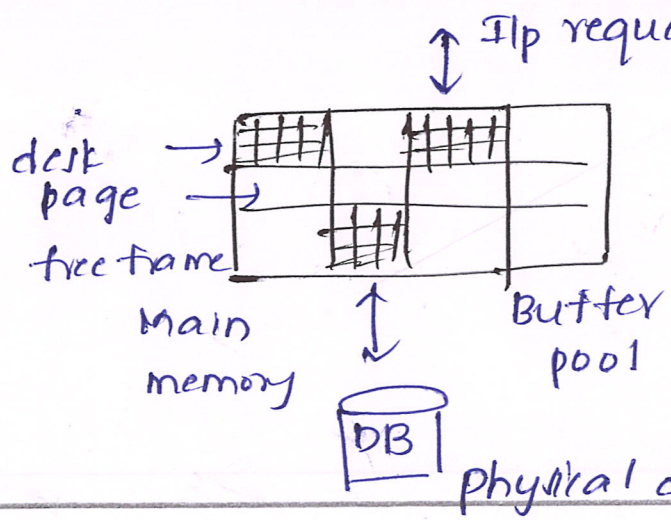
⑤ Caching / Buffering: Buffer → used to store data temporarily
cache → RAM (to store the data & speedup processing)
a region of physical memory storage

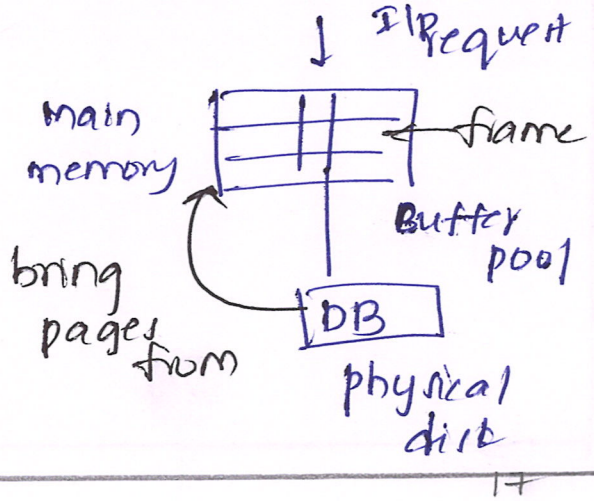
⑥ shadowpaging: It provides Atomicity & durability keeping a copy of data

Buffer Management: The Buffer manager is a s/w module of DBMS whose responsibility is to serve to all the data requests & take decision about choosing a buffer and to manage the page replacement

* The main functions of buffer manager are to speed up the processing and increase efficiency
* DB is stored in disk
* For read & write operation data base must reside in memory & modified blocks is o/p to desk once a T commit
* the Buffer manager is the s/w layer that is responsible for bringing pages from physical to main memory as needed
* The Buffer manager the available main memory by dividing the main memory into a collection of pages which are called as buffer pool

* The main memory pages in the buffer are called frames
* The goal of the BM is to ensure the data requests made by the programs are satisfied by copying data from secondary storage devices into buffer
* I/p, o/p operations occurs blw the program & the buffer area only
* If I/p operation doesnot find the requested page in the bufferpool the BM (rlw) will have to do a physical transfer the pages from the disk to a free block in bufferpool & then make the requested page placed in the buffer pool, that is available to the program requesting the original I/p operation
* A similar scenario will take place in the reverse order for an o/p operation
* The BM makes a new empty buffer available to the program for o/p of the page





* If there is no space in the buffer pool then the BM physically transfer one of the page from buffer pool to disk to provide the empty space in the B pool for the o/p

Serial Schedule

T_1		
T_1:	R(A)	1000
	A = A-100	900
	W(A)	900
	R(B)	2000
	B = B+100	2100
	W(B)	2100
T_2:	R(A)	900
	T = A*0.1	90
	R(C)	3000
	C = C+T	3090
	W(C)	3090

A+B+C = 6090

After completion of T_1 then only T_2 will be execute because in serial execution one after another the T will be executed

concurrent Schedule

T_1		T_2	
R(A)	1000		
A = A-100	900		
W(A)	900		
		R(A)	900
		T = A*0.1	90
		R(C)	3000
		C = C+T	3090
		W(C)	3090
R(B)	2000		
B = B+100	2100		
W(B)	2100		
		A+B+C = 6090	

Here some portion of T_1 will be executed, then some portion of T_2 will be executed & so on. No problem here, we have made context switching in this schedule both T_1 & T_2 execute at a time

This means we can use the results of serial execution as a measure of correctness & concurrent execution for improving resource utilization so we need serialization of concurrent transaction:

process of managing the execution of a set of T in such a way that their concurrent execution produces the same end result as if they were run serially.

Parallel Transaction: simultaneously we are executing T1 & T2

	T1	T2	
1000	R(A)	R(A)	1000
950	A = A-50	T = A * 0.1	100
950	W(A)	A = A-temp	900
2000	R(B)	W(A)	900
2050	B = B+50	R(B)	2000
2050	W(B)	B = B+temp	2100
		W(B)	2100
	A+B = 3000	A+B = 3000	

Serializability:

* The given interleaved execution of there transactions is said to be serializable if it produces the same result as some serial execution of there transactions
* The process of managing the execution of a set of T in such a way that their concurrent execution produces the same end result as if they were run serially
* If 2 transactions T1 & T2 produces the same result then T1 & T2 will be serializable schedule

Conflict Serializability:

* If a schedule S_1 can be transformed into a schedule S_2 by a series of swaps of non-conflicting instructions we say that S_1 and S_2 are conflict equivalent (conflicting inst can't be swapped)
* We say that a schedule S_1 is conflict serializable if it is conflict equivalent to a serial schedule
* If there 2 schedules are made of the same set of T_i then both S_1 and S_2 would yield the same result if the conflict resolution rules are maintained while creating the new schedule. In that case the schedule S_1 & S_2 would be called conflict equivalent

→ let I_i, I_j be the 2 instructions of transactions T_i and T_j respectively

→ Inst I_i, I_j conflict if and only if there exists some item Q accessed by both I_i, I_j & atleast one of these instructions wrote Q

* I_i = Read(Q), I_j = Read(Q) // I_i, I_j don't conflict
* I_i = Read(Q), I_j = write(Q) // I_i, I_j conflict
* I_i = write(Q), I_j = Read(Q) // I_i, I_j conflict
* I_i = write(Q), I_j = write(Q) // I_i, I_j conflict

* Conflicting parts cannot be swapped in this way because the ordering of the conflicting instructions is important and cannot be changed in any other schedule that is derived from the first

WR - dirty Read / Reading uncommitted data

RW - Unrepeatable Read

WW - lost update

Ex: Conflict serializability

→ S_1 can be transformed into S_2, a serial schedule where T_2 follows T_1 by a series of swaps of non conflicting instructions

① Swap T_1. read (B) and T_2. write (A)

② Swap T_1. read (B) and T_2. read (A)

③ Swap T_1. write (B) and T_2. write (A)

④ Swap T_1. write (B) and T_2. read (A)

→ Therefore S_1 is conflict serializable

→ these swaps donot conflict as they work with different items (A or B) in different transactions

Schedule S_1

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

Schedule S_2

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

Ex: Non conflict serializable schedule

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

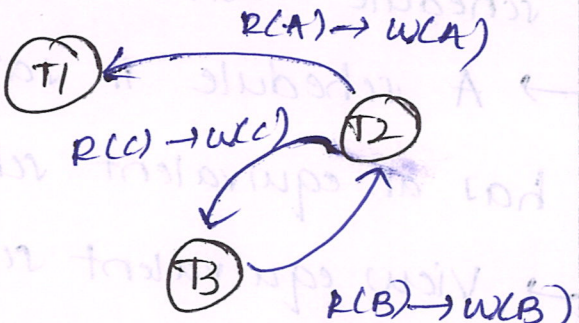
Here we are unable to swap inst in the above schedule to obtain either the serial schedule $\langle T_3, T_4 \rangle$ or $\langle T_4, T_3 \rangle$

Here RW, WW conflicts will occur because we are performing R&W on same data item A

Ex: Non conflict serializable schedule

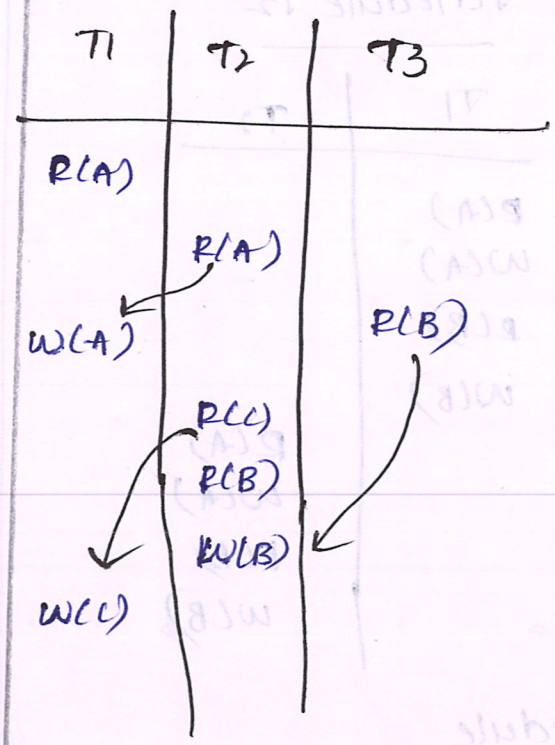
T_1	T_2	T_3
R(A)		
	R(A)	
		R(B)
W(A)		
	R(C)	
	R(B)	
		W(C)
	W(B)	

Precedence graph

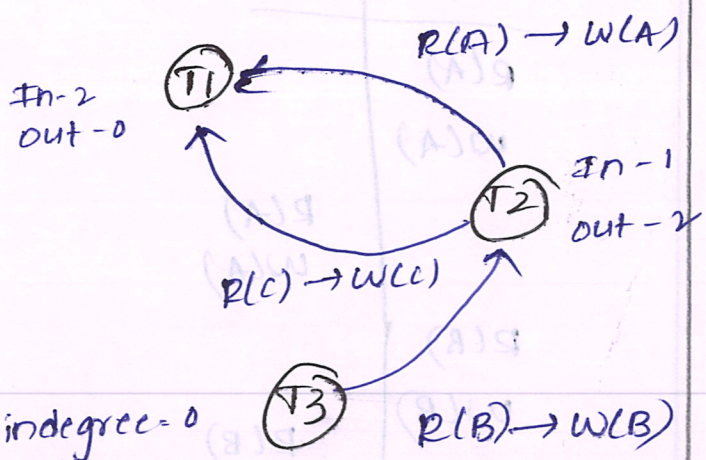


if cycle is formed then it is a non-serializable schedule

ex: Conflict serializable schedule



Precedence graph



No cycle - no loop

starting point is T3 - T2 - T1

cycle / loop - Non conflict serializable
No cycle / No loop - conflict serializable

Every conflict S.S $\xrightarrow{is}$ view SS

but not vice versa

View - Serializability:

→ Every conflict serializables is a view serializable schedule but not vice versa

→ A schedule is said to be view serializable, if it has an equivalent schedule.

→ View equivalent schedule: A schedule S_2 is said to be a view E.S to S_1 if it satisfies the

following 3 conditions

① Initial Read

② Read write sequence

③ Final update

① Initial Read: If in S_1, T_1 reads the initial value of the data item, then in S_2 also T_1 should read the initial value of the same data item

S_1

T_1	T_2
R(A)	

S_2

T_1	T_2
R(A)	

② Read-write sequence:

If in S_1, T_1 writes a value in the data item which is Read by T_2 then

In S_2 also, T_1 should write the value in the data item before T_2 reads it

S_1

T_1	T_2
R(A)	
A = A+100	
W(A)	R(A)

S_2

T_1	T_2
R(A)	
A = A+100	
W(A)	R(A)

③ Final Update: If in S_1, T_1 performs the final write operation on that data item, then in S_2 also T_1 should perform the final write operation on

that data item

S1

T1	T2
R(A)	
A=A+100	
W(A)	
	R(A)
W(A)	
committ	

S2

T1	T2
R(A)	
A=A+100	
W(A)	
	R(A)
W(A)	
committ	

Ex: View serializable schedule

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

Here in this example T2, T3 perform W(A) operations without having performed a R(A) operation this write operations are called blind writes

→ View serializable schedule with blind writes is not a conflict serable schedule.

Isolation levels of a Transaction:

→ Transaction isolation level is a state within database that specifies the amount of data can be visible to a statement in a transaction, specifically when the same data source is accessed by multiple transactions simultaneously

→ Transaction Isolation level is a part of the isolation state of a DBMS

→ Isolation is one of the ACID property

→ the T isolation level is primarily used as a means to provide accurate & reliable access to data within concurrent transaction

→ For ex, 2 different transactions might be accessing the same data simultaneously Therefore

→ if the change made on the data by one T is not passed to the other T, it can affect DB operations

→ To circumvent this, DBMS employs different transaction isolation levels that enforce read and write locks on the data 4 Isolation levels are

① Serializable

② Repeatable reads

(3) Read committed

(4) Read uncommitted

* Concurrency Controlling techinques :-

→ Concurrency is a process of managing Simultaneous execution of Transaction in a shared database.

→ Purpose of concurrency control is,

* To enforce islotion
* To preserve db consistency
* To restore conflicts (RW, WR, WW)

→ Techniques for Controlling concurrency is,

① Locking
② Time stamp based
③ Optimistic
④ Multiversion.

1) Lock-Based Protocols :-

→ A lock guarantees exclusive use of data items to a current transaction. lock based protocols used

* to access data items (lock acquire by the T)
* After completion of T it Releases lock.

→ Different types of locks are,

(i) Shared lock (S)	(ii) Exclusive lock (X)
T can Read the data item value Lock-S(B)	T can Read and Write the data item values
Eg:- Lock-S(B) // acquire shared lock on B	Eg:- Lock-X(B) // acquire Exclusive lock on B
→ It can be hold by any no. of T	→ Exclusive lock can be hold by (any) only 1 T at a time.

Ex:-

T_1	T_2
// both R & W	
Exclusive Lock: Lock-X(B)	
R(B)	
B-50	
W(B)	
unlock(B)	
Releasing lock & B can be used by another T	// Read B only
	Lock S(B)
	R(B)
	unlock(B) — Shared lock.

Note :-

① Any Number of transactions can held shared lock.

But,

② Exclusive lock can be hold only by one T at a time.

Ex :- Let A and B be two accounts that are accessed by transactions T_1 and T_2. Transaction T_1 transfers 50 es, from B to A, The T T_2 displays the total amount of money in Alc A & B.

T_1	T_2
Lock - X(B);	Lock - S(A);
R(B)	R(A)
B = B - 50;	unlock (A) -
W (B)	dock :- S(B)
unlock (B)	R(B)
Lock - X(A)	unlock (B)
R(A)	Display (A+B)
A = A + 50	
W (A)	
unlock (A)	

→ When a T: requests a lock on data item in a Particular mode and no other T has a lock on the.

Same data item in a conflicting mode, the lock can be granted.

Ex:- Consider T_2 has a shared mode lock on data item A, and another T T_1 requests a exclusive mode lock on the same data item.

2) Time Stamp Ordering Protocol :=

Time Stamping Method :- A unique identifier created by DBMS that (mod) indicates relative starting time of a Transaction.

→ it notes the starting time of a T & it always stores the highest time stamp of a T (at what time the T will complete.)

→ It is a tag that can be attached to any T or any data item which denotes a specific time.

→ This time stamp can be generated by using System clock or by logical counter

* System clock :=

Time stamp generated at the time of T started.

Logical Counter :=

Every time a new T starts. So they logical counter is incremented after a new Time stamp has been assigned to a T.

Ex := T_1 → time stamp TS(1)
9:00 AM
.
.
.
9:10 AM T_2 → time stamp TS(2)

We should be noted that,
TS(1) < TS(2)
9:00 < 9:10

* Time stamp of one T is always less than the time stamp of the next transaction.

⇒ Time Stamp Values :=

R - Read TS of a datatime (A)
W - Write TS of a datatime (A)

* R TS is a largest TS of any T that execute Read (A) Successfully that time we have to take

* W TS is a largest TS of any T that execute write (A) Successfully that time we have to take

Ex:- Read time stamp (RTS) :=

T_1 10	T_2 20	T_3 30	T_4 40
$R_1(A)$		$R_3(A)$	
$w_1(A)$	$R_2(A)$		
			$w_4(A)$

* Now find RTS (A) :-

initially RTS (A) = 0

update T_1 = 10 (0 < 10) True

update T_3 = 30 (10 < 30) True

update T_2 = 20 (30 < 20) False ×

→ No Read operations after this So, finally the highest TS of R operation is RTS(A) = 30

Ex:- Find Write Ts of A

initially WTS (A) = 0

update T_1 = 10 (0 < 10) True

update T_4 = 40 (10 < 40) True.

So, finally the highest TS of write operation is WTS(A) = 40

* Time Stamp Ordering Protocol :- $TS1 < TS2$

→ It ensures, that any conflicting Read & write operat-ions are Executed in time stamp order.

Issues that occur in Ts order are,

1) Ti issues Read (A) :-

(i) If Ts (Ti) < WTS (A) { Ti is an older T then the last T that write the value of A }. (request fails)

(ii) If Ts (Ti) ≥ WTS (A) { Ti allowed to read updated value of A }

Ex :-

T_1 10	T_2 20	T_3 30
R(A) (0<10) ✓		$R_T(A)$ (10<30) ✓
	R(A) ×(30<20)	

RTS (A) = 30

Ex :-

T_1 10	T_2 20	T_3 30
R(A) 1000	R(A) 1000	
	A = A + 100	
	W(A) 1100	R(A) 1100
	1100	

(i) Ts (Ti) < WTS (A)

Ts (T_1) < WTS (A)

R(A) in T_1 W(A) in T_2

10 < 20

⇒ This means in T_1 R(A) just Reads 1000 but and Ts is 10 in T2 Ts is 20 & it modifies the A value by w(A) operation in T2. So what ever the value updated by T2 with w(A) that cannot be Read by T_1 because Ts $(T_1) < (T_2)$ Ts. So it's fails to read.

(ii) Ts $(T_i) \geq w(A)$ Here in this Example,

Ts (T_3) R(A) $\geq$ Ts w(A)

↓ ↓

R(A) in T_3 in T_2.

$30 \geq 20$ (T)

→ So, what ever value updated in T_2 with w(A) operation that is Read by R(A) in T_3. So, it result is true & it reads final updated value.

② T_i issues WRITE (A) :=

① If Ts $(T_i) \geq w(A)$ &

Ts $(T_i) \geq R(A)$

then T_i is allowed to modify write value of A & Ts (T_i) will become current value of w(A)

(ii) If Ts $(T_i) < R(A)$

it means the younger T is already using value of A, the updation is not allowed.

(iii) If $R(A) \leq TS(T_i) \leq W(A)$.

it means that the younger T has updated a value of A, T_i is not allowed to modify A.

T_1 (10)	T_2 (20)	T_3 (30)
R(A) 9:00 AM 1000		
	R(A) 1000	
9:05	A = A+100 1100	R(A) 1100
9:10	W(A) 1100	A = A+200
9:15	9:25	W(A).
	9:30	1300
	R(A)	9:15 / 9:20
	9:35 AM 1300	

① $TS(T_i) \geq W(A)$ &
$T_3\ W(A) \geq T_2\ W(A)$

$TS(T_i) \geq R(A)$
$T_3\ W(A) \geq T_2\ R(A)$

② $TS(T_i) < R(A)$
$T_2\ W(A) < T_3\ R(A)$

③ $R(A) \leq TS(T_i) \leq W(A)$
$T_1\ R(A) \leq T_2\ W(A) \leq T_3\ W(A)$

* Thoma's Write rule :=

→ It's a modified Version of Time stamp ordering Protocol.

→ In TOP T_i issues $W(Q)$ & $TS(T_i) < WTS(Q)$ T_i is Rolled back however in Thoma's write rule if $TS(T_i) < W(Q)$ & $TS(T_i) >= R(Q)$ the write(Q) Operation can be ignored.

T_1	T_2
R(Q)	
W(Q)	W(Q)

* T_i issues Read (A) :=

T_2 R(A)

$TS(T_i) < WTS(A) \rightarrow T_1 R_t(A) < T_2\ WTS(A) \rightarrow$ fails to Read

$TS(T_i) \geq WTS(A) \rightarrow T_3\ R(A) \geq T_2\ WTS(A) \rightarrow$ it read updated value.

T_1	T_2	T_3
$R_t(A)$ 9:00		
9:05	R(A)	
9:10	A = A+100	
9:15	W(A)	9:15 / R(A) 9:20
		A = A+200 9:25
	9:35 R(A)	W(A) 9:30

* T_i issues write (A) :=

(i) $TS(T_i) \geq W(A)$ & $TS(T_i) \geq R(A)$ → Allowed to modify & it will take the W(A) value.

$T_3\ W(A) \geq T_2\ W(A)$

$T_3\ W(A) \geq T_2\ R(A)$

(ii) $TS(T_i) < R(A)$ → Already Read updated value & does not allow to update.

$T_2\ W(A) < T_3\ R(A)$

(iii) $R(A) \leq TS(T_i) \leq W(A)$

$T_1 R(A) \leq T_2\ W(A) \leq T_3\ W(A)$ → younger T has Already updated value & not allowed to modify A.

* Two-phase locking Protocol (2PL) :-

→ It is one of Concurrency control technique.

→ It requires both locks & unlocks being done in 2 phases.

[first it locks all T, after completion of the T it unlocks all T]

→ 2PL is divided into 2 phases.

1) Expanding phase / Growing phase :-

In this phase, a T may obtain locks, but may not release any lock. The advantages are,

a) holding lock un-necessarily.

b) locking due too early

c) penality to other transactions.

2) Shrinking phase :-

In this phase, a T may release locks, but may not obtain any new locks.

→ This 2 phases will have a common point that is a lock point.

↓

Initially the T first in growing phase, it will locks all the I after completion of the I it reach the Final

Lock after this growing phase cannot allows / obtain a new locks this is called Shrinking phase. & in this phase it releases the locks.

* Advantages of 2PL :-

→ This 2 phase locking protocol will Enforce intignity & Reduce Concurrency.

→ It Ensures conflict Serializability.

* Disadvantages of 2PL :-

→ It does not Ensure freedom from deadlock.

→ Cascading roll backs may occur under two-phase locking.

→ Cascading roll backs can be avoided by a modification of 2PL.

* Variations of 2phl :-

① Strict two-phase locking protocol :-

→ This protocol requires that locking should be 2phase, and all Exclusive mode locks taken by a T should be held until the T. This requirement prevents any T from reading the data written by any uncommitted T under Exclusive mode until the T committs.

→// Here Exclusive locks cannot be Released until commi
-tt,→Shared / Read L released regullarly.
→ Dead locks may occur.

② The Rigorous 2 PL protocol :=

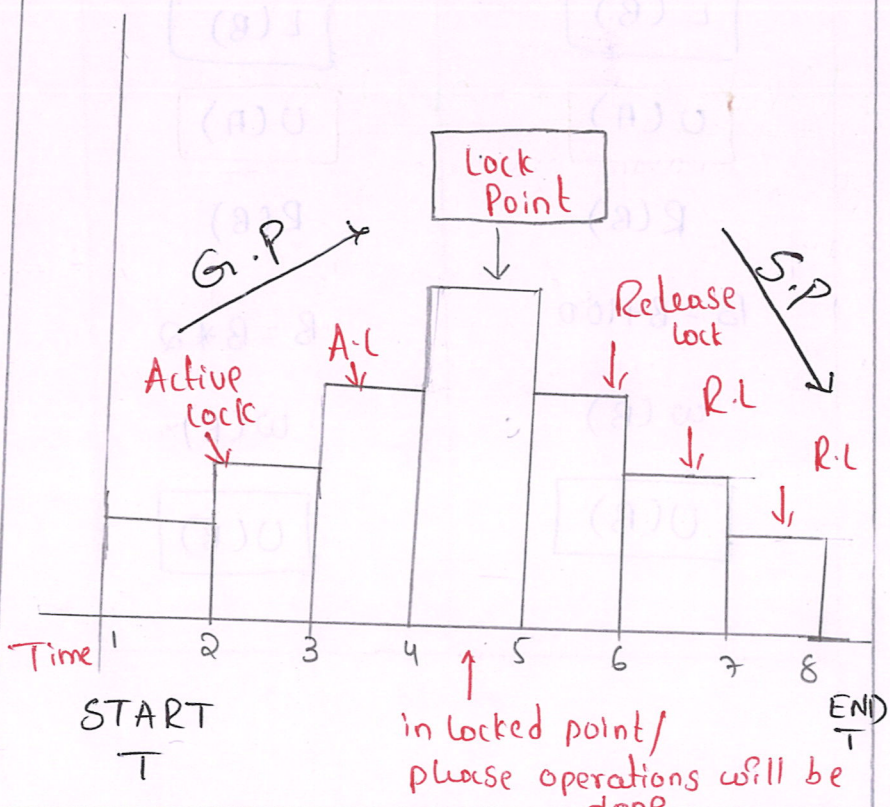
This protocol requries that all locks be held until the T commitTs.

→// Shared / exclusive locks cannot be released until committ
→ dead locks may occur
→ This also known as "Strong Strict 2PL".

*Ex for 2PL :=

T3 :=
Lock - X(B)
R(B)
B = B - 50
W(B)
Lock - X(A)
R(A)
A = A + 50
W(A)
unlock(B)
unlock(A)

Graph for 2PL :=



* After Creating a new lock then only we have to unlock the previous lock.

* All lock operations preceeds all unlock operations.

→ // Create a new lock on B before Releasing previous lock on A.

*Ex for 2PL :=

T_1	T_2
L(A)	L(A)
R(A)	R(A)
A = A+100	A = A*2
W(A)	W(A)
L(B)	L(B)
U(A)	U(A)
R(B)	R(B)
B = B+100	B = B*2
W(B)	W(B)
U(B)	U(B)

③ Conservative (static) 2PL :=

After Committ only it release locks, it avoids the cascading Roll backs & it Also the dead lock free. It holds all the locks they need before the T begin.

→ it won't wait for only lock.

*Dead lock :=

A dead lock is a situation where 2 (or) more transactions are waiting for one another to releasing locks.

* Here T_1 is waiting for T_2 and T_2 is waiting for T_1. So dead lock occur.

Lock - x → Exclusive lock

Lock - S → Shared lock

T_1	T_2
LX(x)	LX(y)
R(x)	R(y)
x = x+100	y = y*0.1
W(x)	W(y)
Here T_1 wants to perform operations on 'y', then T_1 waits until T_2 Releases lock on y	Here T_2 waits until T_1 releases lock on x

Necessary conditions on deadlocks :=

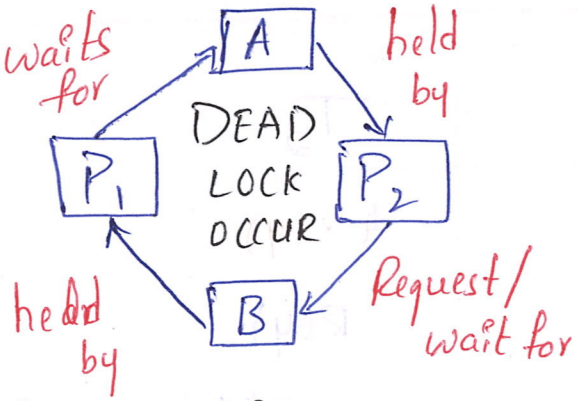
Preventions for Dead lock :=

1) Hold & wait
2) mutual Exclusion
3) No preemption
4) Circular wait

① Hold & wait :=

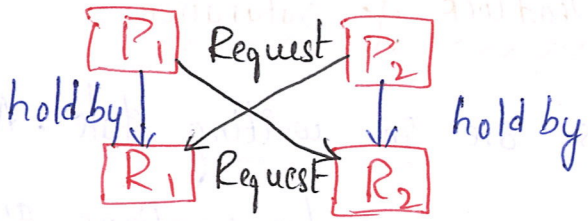
→ Process currently holding resources that were granted earlier can request new Reasources.

→ P_1 holds B & waiting for A.
P_2 holds A & waiting for B.



② Mutal Exclusion :=

It's process is requesting resources which is already locked by other process (or) A resource may be acquired Exclusive lock by only one process at a time



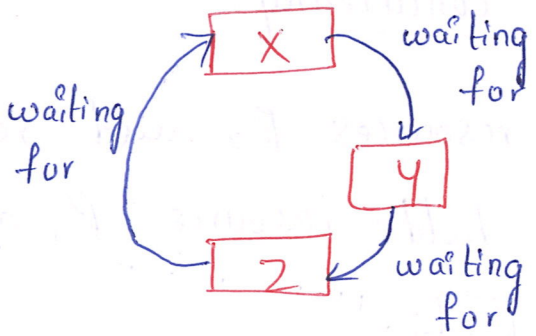
③ No preemption :-

→ Preemption :- If you work is completed than release locks.

→ Non-preemption :- Eventhough your work is completed it did not release the lock.

④ Circular wait :-

x is waiting for y and
y is waiting for z
z is waiting for x in a circular manner is known as circular wait.



* Difference between Deadlock & Saturation :=

→ Deadlock :- All processes are in waiting state. A deadlock is a situation where 2 or more transactions are waiting for one another to releasing locks.

→ T_1 is waiting for T_2, T_2 is waiting for T_1.

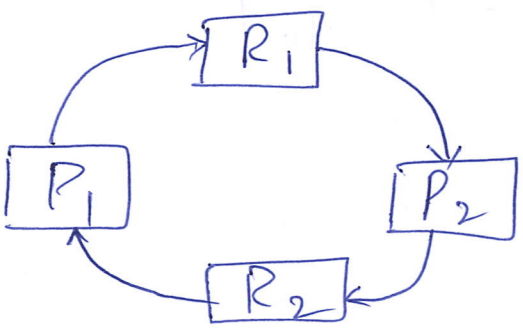
→ Deadlock is a cdt where no process proceeds for Execution and Each waits for resources, that have been acquired by the othe processes..

→ Saturation / Live lock :=

It occurs when 2 or more processes continuously repeat the same interaction in response to changes in the other processes without doing any useful work.

→ These processes are not in the waiting state & they are running concurently.

→ Process P_1 holds resources R_2 and requires R_1 while process P_2 holds resource R_1 and requires R_2



→ Deadlocks Saturation both are the conditions where the processes requesting for a resource has been delayed for a long.

In staruation, Process with high priorities continuously uses the resources preventing low priority Process to acquire the resources.

→ Saturation is the situation when a T has to wait for a indefinate period of time to acquire a lock.

* Thoma's Write rule is :-

Suppose that transaction T_i issues write (Q):

a) If $TS(T_i) < R\text{-timestamp}(Q)$, then the value of Q that T_i is producing was previously needed and it had been assumed that the value would never be produced. Hence, the system rejects the write operation and rolls T_i back.

b) $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an absolute value of Q. Hence, this write operation can be ignored.

c) Otherwise, the system executes the write operation and sets W-timestamps (Q) to $TS(T_i)$

The difference between timestamp Ordering process and Thomas write rule lies in the second rule. In timestamp ordering Protocol, if T_i issues write (Q) and TS (T_i) < W-timestamp (Q), T_i is rolled back. However, in Thoma's write rule if TS (T_i) < W-timestamp (Q) and TS (T_i) >= R-timestamp(Q), the write (Q) operation can be ignored.

Validations based Protocols :-

Validations Based protocol assumes that each transaction T_i, Executes in three phases:

(i) Read phase :- During this phase, the system Executes transaction T_i. it reads the values of the various data items and stores them in variables local to it. It Performs all write operations on temporary local variables.

(ii) Validation phase :- Transaction T_i Performs Validation test to determine that whether it can copy temporary local variables, that hold the results of write operations to the data base, without causing violation of Serializability.

(iii) Write phase :- If transaction T_i Succeeds in Validation, then System updates actual databases otherwise the System roll back T_i. To Perform the validation test, we should know when the various phases Of transactions took place. Therefore, following three timestamps are associated with transaction T_i.

* Multiple Granularity :=

Introduction :=

The concurrency control schemes described thus far, locks individual data item at a time. Executing these locks is time consuming. However, it would be advantageous to group several data items, and to treat them as a single synchroniz-ation units and lock it just once.

For example, if a transaction T_i needs to access the entire data base, and a locking protocol is used, then T_i must lock each item in the data base, which is time consuming. Instead, it would be better if T_i could issue a single lock request to lock the entire database. On the other hand, if transaction T_j needs to access only a few data items, it should not be required to lock the entire database, otherwise concurrency is lost.

Thus, there is a need of mechanism to allow the system to define levels of granularity. One way is to define hierarchy of data items and such a hierarchy can be represented graphically as a tree.

Consider the tree, which consists of four levels of nodes. The highest levels represents the entire database. Below it are the nodes of type area; the database consists of exactly these areas.

1) Start (T_i) :- The time when T_i started its execution

2) Validation (T_i), the time when T_i finished its read phase and started its validation phase.

3) Finish (T_i), the time when T_i finished its write phase.

* Advantages :-

Validation based Protocol guards automatically against cascading roll back, Since the actual writes take place only after the transaction issuing the write has committed.

* Disadvantages :-

There is possibility of starvation of long transaction, due to a sequence of conflicting short transactions that cause repeated restarts of long transaction. To avoid starvation, conflicting transactions must be temporarily blocked, to enable long transaction to finish

→ Saturation :- / Live lock :- is the situation when a T has to wait for a indefinate period of time to acquire a lock.

→ lock group of items at a time Save time.

In schedule 3, $TS(T_{14}) < TS(T_{15})$ and the schedule is Possible under time stamp protocol.

* Advantages :-

1) The timestamp ordering Protocol Ensures conflict Serializability. This is because conflicting operations are Processed in time stamp order.

2) The protocol Ensures freedom from deadlock, Since no transaction Ever waits.

* Disadvantages :-

1) There is a possibility of starvation of long transactions if a Sequence of conflicting short transactions causes repeated restarting of the long transaction. If a transaction is found to be getting restarted repeatedly, conflicting transactions need to be temporarily blocked to Enable the transaction to finish.

2) The protocol can generate schedules that are not recoverable.

Each area contains the nodes of type file as its children. Each file belong to area only. Each file contains the nodes of type record.

```
Read (A);
A := A+50;
Write (A);
Display (A+B);
```

Fig: Shows a concurrent schedule for these two transactions.

T_{14}	T_{15}
Read (B);	Read (B);
	B := B-50;
	Write (B);
Read (A);	Read (A);
Display (A+B);	A := A+50;
	Write (A);
	Display (A+B);

Fig: Schedule 3.

* Thomas's Write rule :-

Thomas's write rule is a modified version of timestamp ordering protocol.

Consider Schedule 4:

T_{16}	T_{17}
Read (Q)	
	Write (Q)
Write (Q)	

Fig: Schedule 4.

Here, T_{16} starts before T_{17}, therefore $TS(T_{16}) < TS(T_{17})$, the read (Q) operation of T_{16} succeeds, Similarly the write(Q) operation of T_{17}. When T_{16} attempts its write (Q) operation, it is rejected by the system and T_{16} is rolled back; as ($TS(T_{16}) < W$- timestamp (Q)). Since W-time stamp (Q) = $TS(T_{17})$.

In this case, T_{17} has already written Q and the value of Q that T_{16} is attempting to write is one that will never need to be read. Thus, the rollback of T_{16} is required by timestamp (Op) ordering protocol, but it is unnecessary.

→ Thomas's write rule, modifies the time stamp ordering

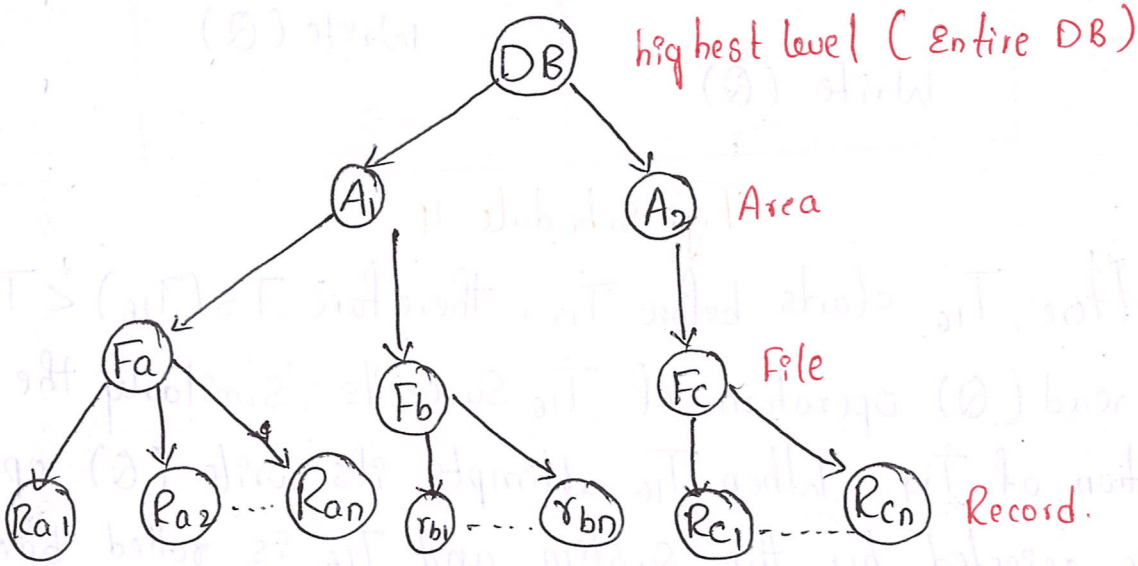
* Intension mode *

Shared mode

Explicit locking is done at lower level of Tree with only shared mode locks.

Exclusive mode.

Explicit locking is done at lower level of Tree with Exclusive & shared mode.



If transaction Ti gets an Explicit lock on file fc in Exclusive mode, then it has an implicit locks in Exclusive mode on all the records belonging to fc

A more Efficient way to gain the knowledge of weather a transaction can lock a data items or not, is to introduce a new class of lock modes, called intention lock modes. If a node is locked in an intention mode, Explicit locking can be done at a lower level of tree.

Intention locks are put on all ancestors of a node before that node is locked explicitly. Thus, a transaction does not search the entire tree to determine whether it can lock a node sucessfully.

There is an intention mode associated with shared mode and exclusively mode. If a node is locked in intenti -on shared mode, explicit locking is done at a lower levels of the tree, but with only shared mode locks. Similarly, if a node is locked in intention exclusive (IX) mode, then explicit locking is done at a lower level with exclusive mode or shared mode clocks. If a note is locked in shared and intension exclusive (SIX) mode, the sub tree rooted by that node is locked explicitly in shared mode and explicit locking is done at a lower level with exclusive mode locks. The com -patibility function for these locks modes.

-	Intension shared IS	X	S	SIX	X
IS	True	True	True	True	False
Intension exclusive IX	True	True	False	False	False
S	True	False	True	False	False
Shared intension exclusive SIX	True	False	False	False	False
X	False	False	False	False	False

*Multiple Granularity protocol :-

→ Multiple granularity protocol ensures Serializability, The protocol is given below :-

→ Each transaction T_i can lock a node Q by following these rules :-

1) It must observe the locks compatibility function.
2) It must lock the root of the tree first and can lock it in any mode.
3) It can lock a node Q in S or IS mode only if it currently has the parent of Q locked in either IS or IX mode.
4) It can lock node Q in X, SIX or IX mode only if it currently has the parent of Q locked in either IX or SIX mode.
5) It can lock a node if it has not Previously unlocked any node
6) It can unlock a node Q only if it currently has none of the children of Q locked.

→ The multiple granularity protocol locks the data items in top-down order, whereas it unlocks them in bottom-up order

* Advantages :=

→ This protocol enhances concurrency

→ It reduces lock overhead.

* Disadvantages :=

→ Deadlock is possible in this protocol.

* Applications :=

→ It is useful in application that include a mix of:

→ short transaction that access only a few data items.

→ long transactions that produce reports from an Entire file or set of files.

=> FAQ's :=

1) Explain two phase locking with algorithms.
2) What is transaction? In what ways is it different from an ordinary program.
3) Write a short notes on concept of transaction Processing.
4) Explain concurrency control scheme?
5) Explain conflict and view Serializability. Write short notes on multiple granularity locking.

6. Explain timestamp based protocol.

Consistency :=

DB transaction must change affected data only in allowed ways. Any data written to the data base must be valid according to all defined rules, including constraints, triggers and any combination. It represents the consistency of the referential intigrity of the database, unique primary keys in table, etc...

Aborted :=

If any of the checks fails and the transaction has reached a failed state, then the recovery manager rdls back all its write operations on the database to bring the database back to its original state where it was price to the execution of the transaction. T in this state are called aborted. The DB recovery module can select one of the 2 operations after a transaction aborts.

(i) Re start the transaction
(ii) kill the transaction.

Unit-V

Storage and Indexing:

Indexing: Way of optimizing performance of DB by minimizing the no.of disk access required when query is process.

Index:- Index is a data structure which is used to quickly locate and & access the data in DB table.

Ex: Suppose if we are going to search any topic in text book then no need to search each and every page, just open the text book & backside of the book index will be there, in that we search for that particular topic with page numbers.

primary key concept in index primary key (page no:

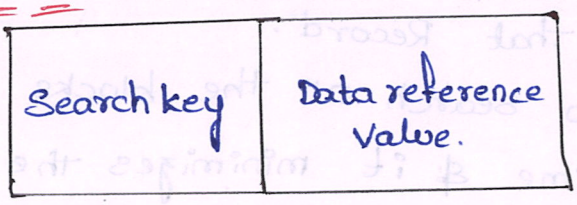
20, 37, 128)

↑ definition ↑ Example ↑ queries.

Here 20 is one index value
37 is one index value
128 is one index value.

means its a key value to search for particular concept/data

Structure of Index: It consist of search key, data reference value.

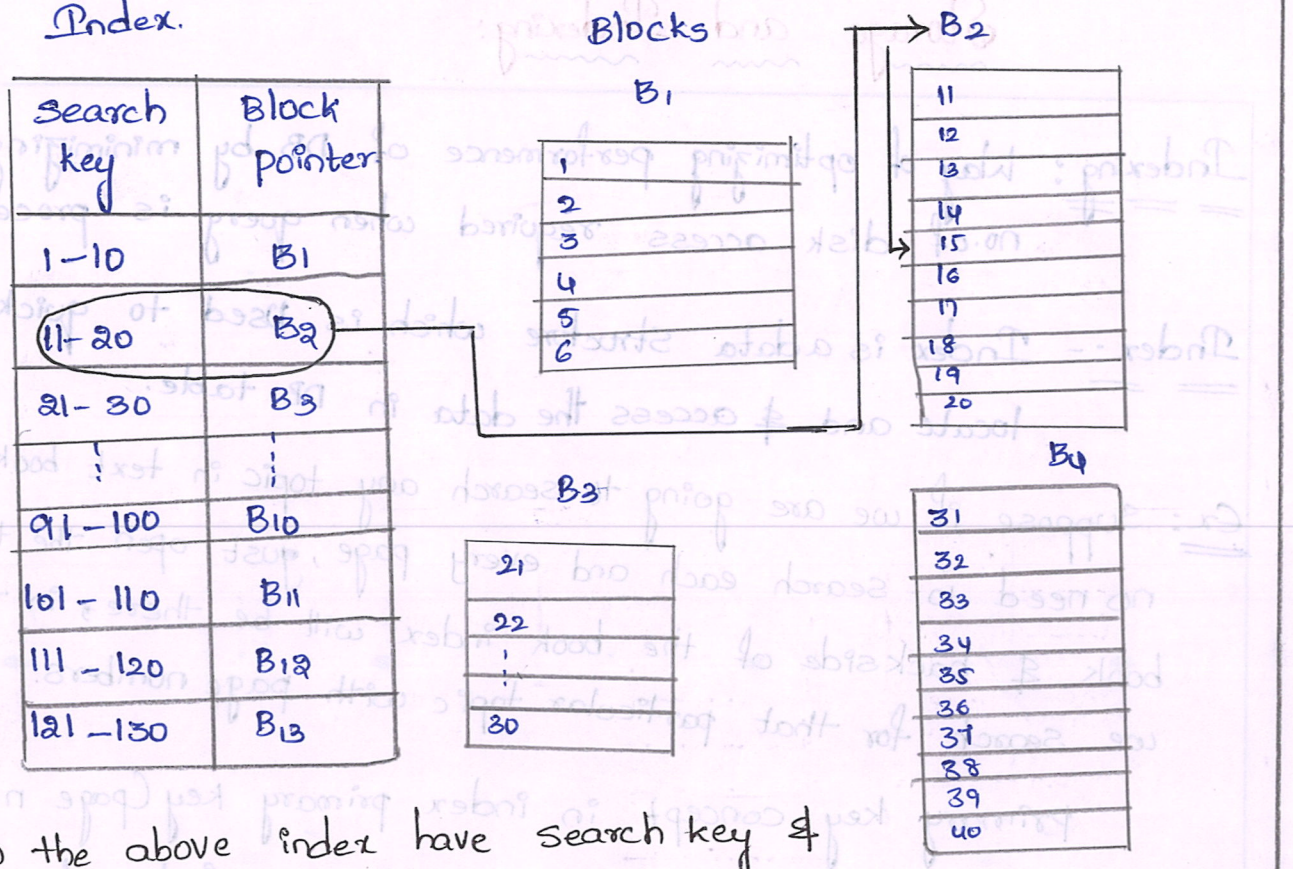


→ Search key contains the copy of primary key/candidate key of a table.

→ Data Reference Value contains set of pointers which hold the address of disk block, where perticular key value can be found.

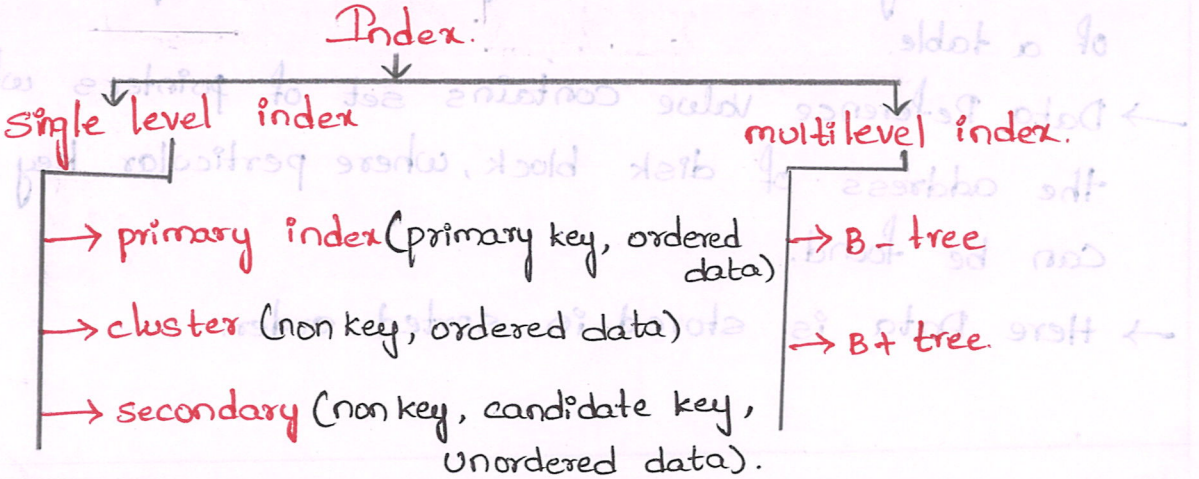
→ Here Data is stored in sorted order.

Ex: Index.



In the above index have search key & block pointer / data reference value. & we have list of block each block contains set of records from 1 to 130. suppose if we want search for a record 15 then directly we go to index & find the pointer value for that. so 15 will be available in block 2 (B2) so we directly go to that block and access that Record.

→ No need to search all the blocks
→ it save time & it minimizes the no. of disk to search
→ Access time will be decreased.



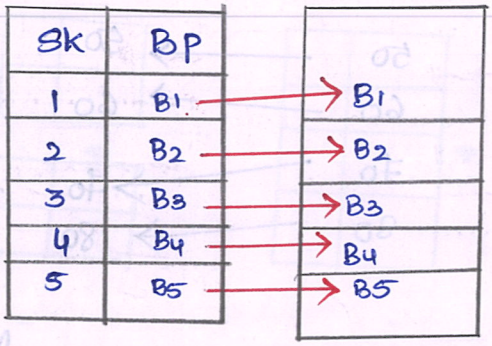
Primary index < Dense index
Sparse index

1. single index: < Primary
Secondary
Cluster

→ Index & data directly communicating.

Index data block.

Sk	Bp
1	B1
2	B2
3	B3
4	B4
5	B5



* It can't handle the large size of (data) index to place in memory.
* It leads to multiple disk access.

2. Multilevel index: < B-tree
B+ tree

Nonkey	BP
100	•
123	•
46	•

Outer index

NK	DB
100	•
15	•
67	•
123	•
234	•
45	•
46	•
68	•
1	•

inner index

100
15
67

123
234
45

46
68
1

data blocks.

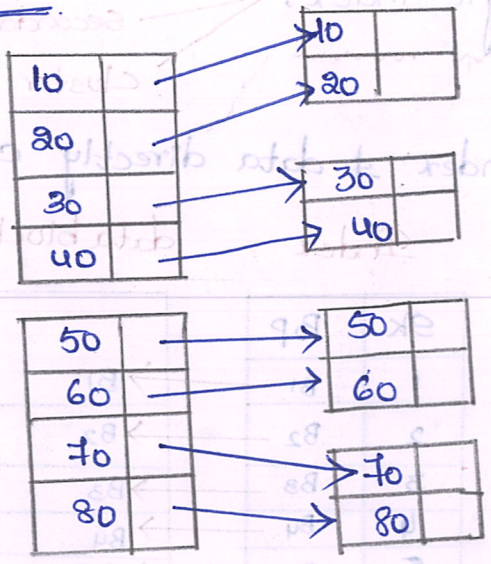
Classification of indexing: ordered index is 2 types.

1. Dense indexing: Here an index entry is created for every search key value in the DB.

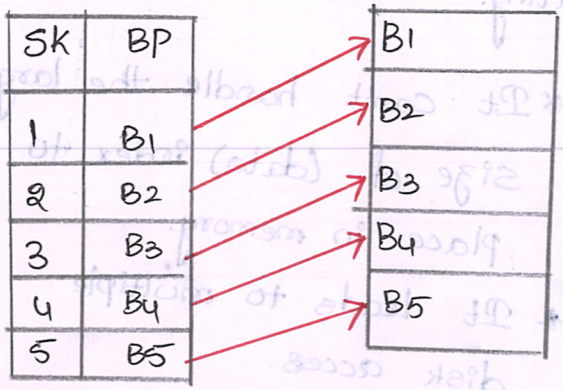
→ More search key values will be there.

→ It makes searching faster but requires more pace to store index records itself.

Ex:



Index 1-to-1 Data block



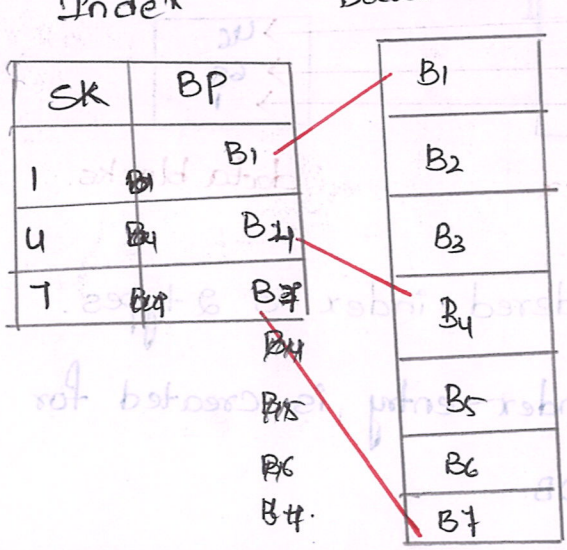
2) Sparse Indexing:

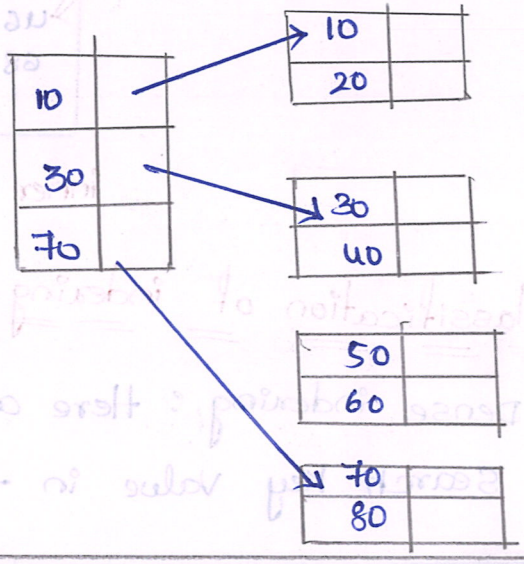
→ In sparse index index records are not created for every search key. Index entry created only for some records.

→ To search for a record, we first proceed by index record and reach at the actual location of the data.

→ If the data we are looking for it not where we directly reach by following the index, then the system starts sequential search until the desired data is found.

Index Data block.





1) Multilevel index: Index records comprise search key values and data pointers.

→ M.I is stored on the disk along with the actual DB files.

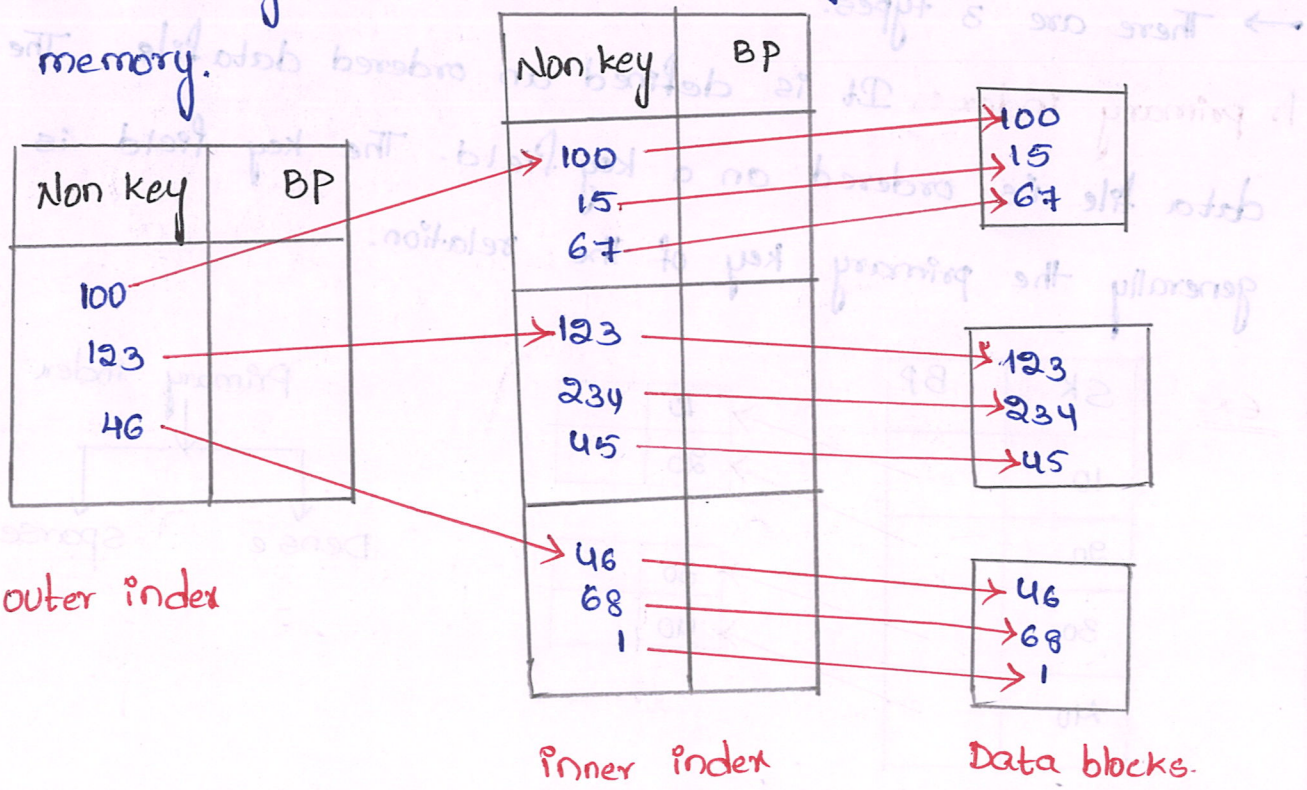
→ As the size of the DB grows, so does the size of indices.

B-trees, B+ trees.

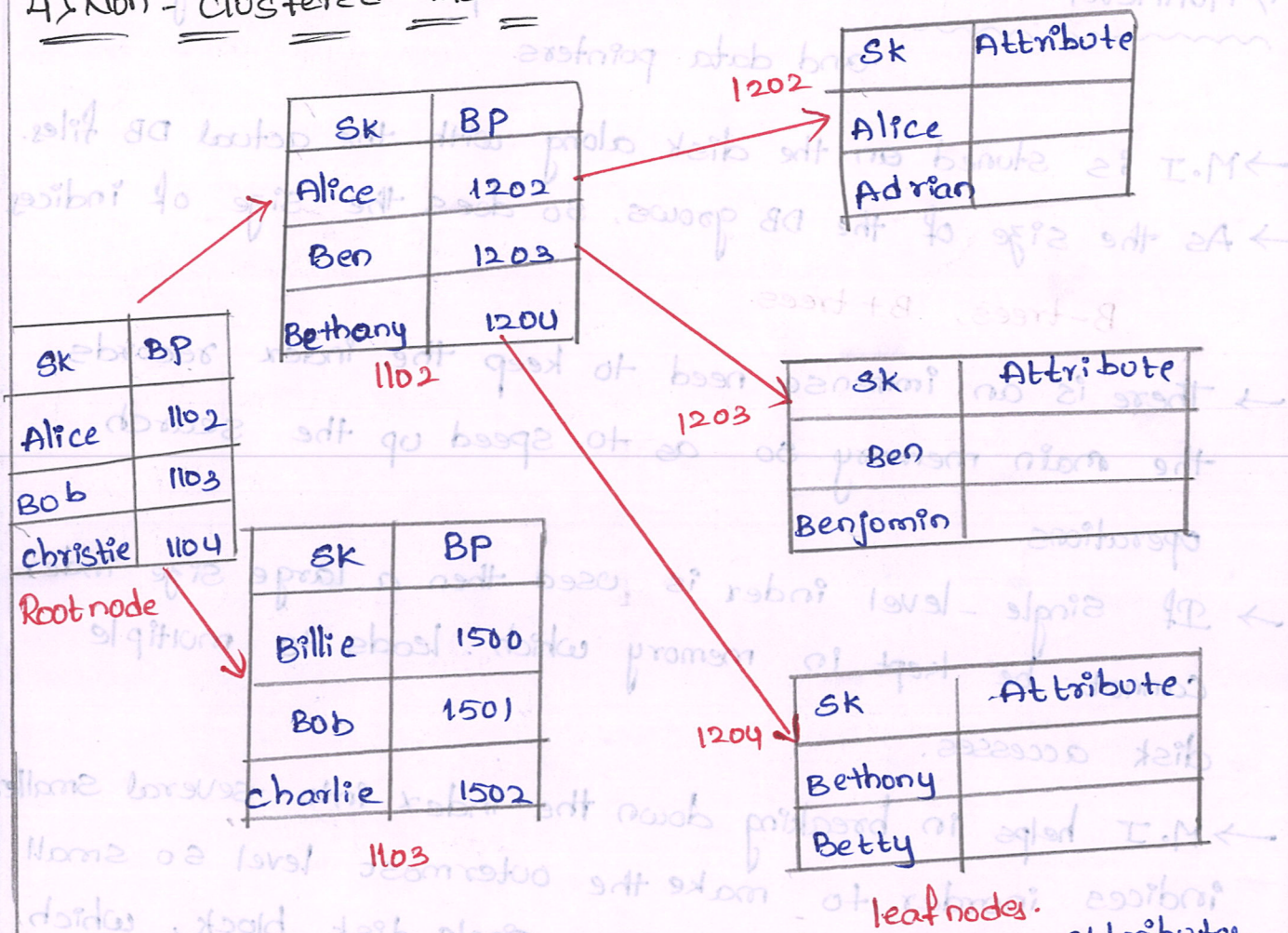
→ There is an immense need to keep the index records in the main memory so as to speed up the search operations.

→ If single-level index is used then a large size index cannot be kept in memory which leads to multiple disk accesses.

→ M.I helps in breaking down the index into several smaller indices inorder to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.



4) Non-clustered index:

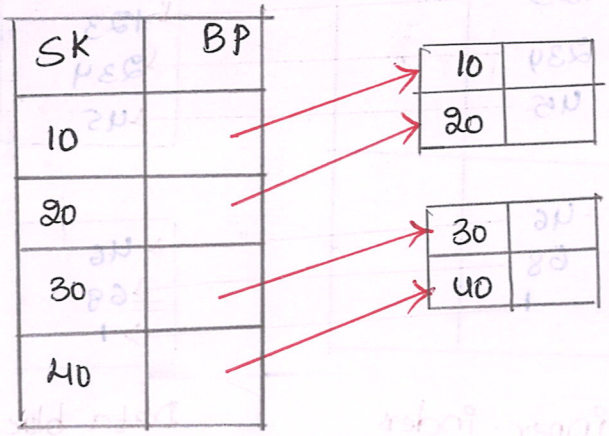


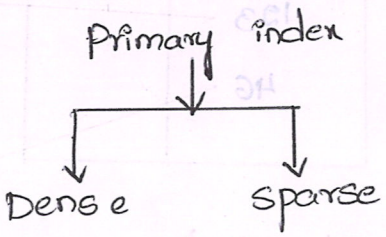
→ Indexing is defined based on its indexing attributes.

→ There are 3 types.

1. primary index: It is defined an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation.

Ex:

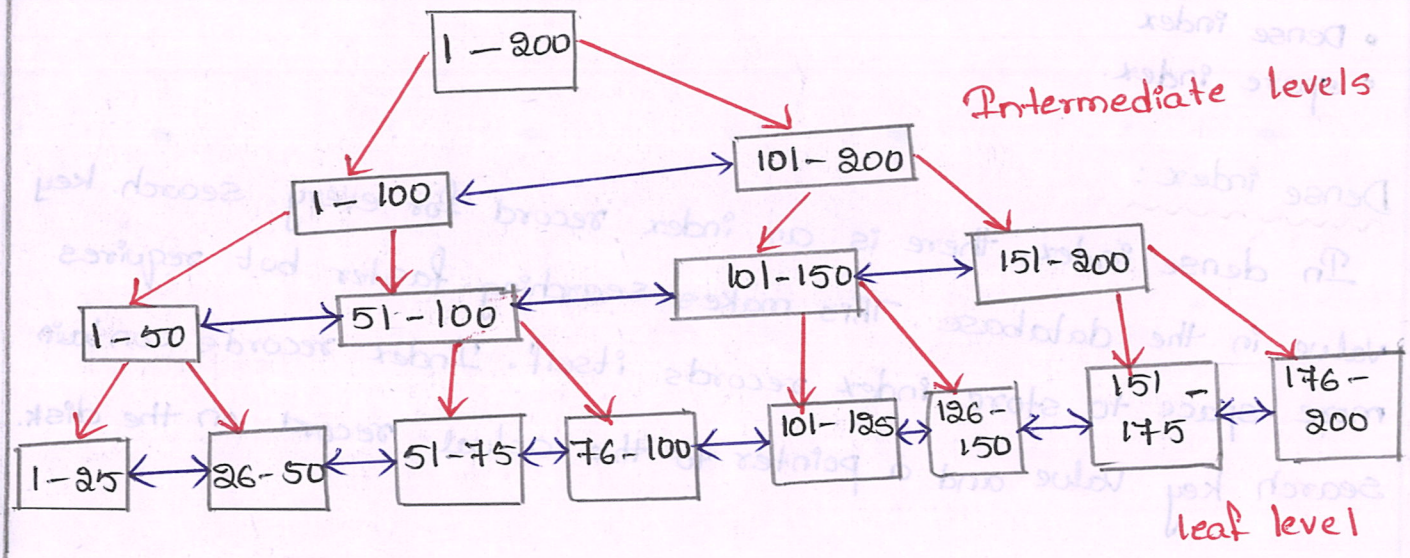




2) Secondary index :- Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non key with duplicate values

→ Also known as non-clustering index.

3) clustering index :- It is defined an ordered data file. The data file is ordered on a non key field.



Indexing :-

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books

Indexing is defined based on its indexing attributes. Indexing can be of the following types-.

Primary index : primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is genera -lly the primary key of the relation.

Secondary index: Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.

clustering index: clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

ordered indexing is of two types -

- Dense index
- spare index.

Dense index:

In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.

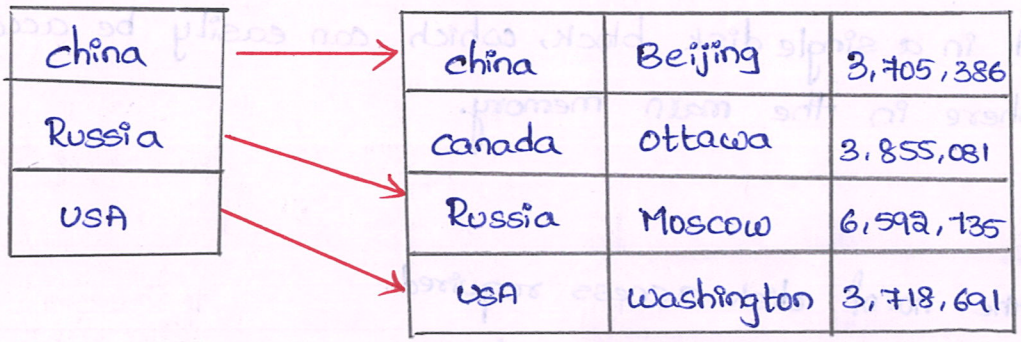


China	China	Beijing	3,705,386
Canada	Canada	Ottawa	3,855,081
Russia	Russia	Moscow	6,592,735
USA	USA	Washington	3,718,691

Sparse index:

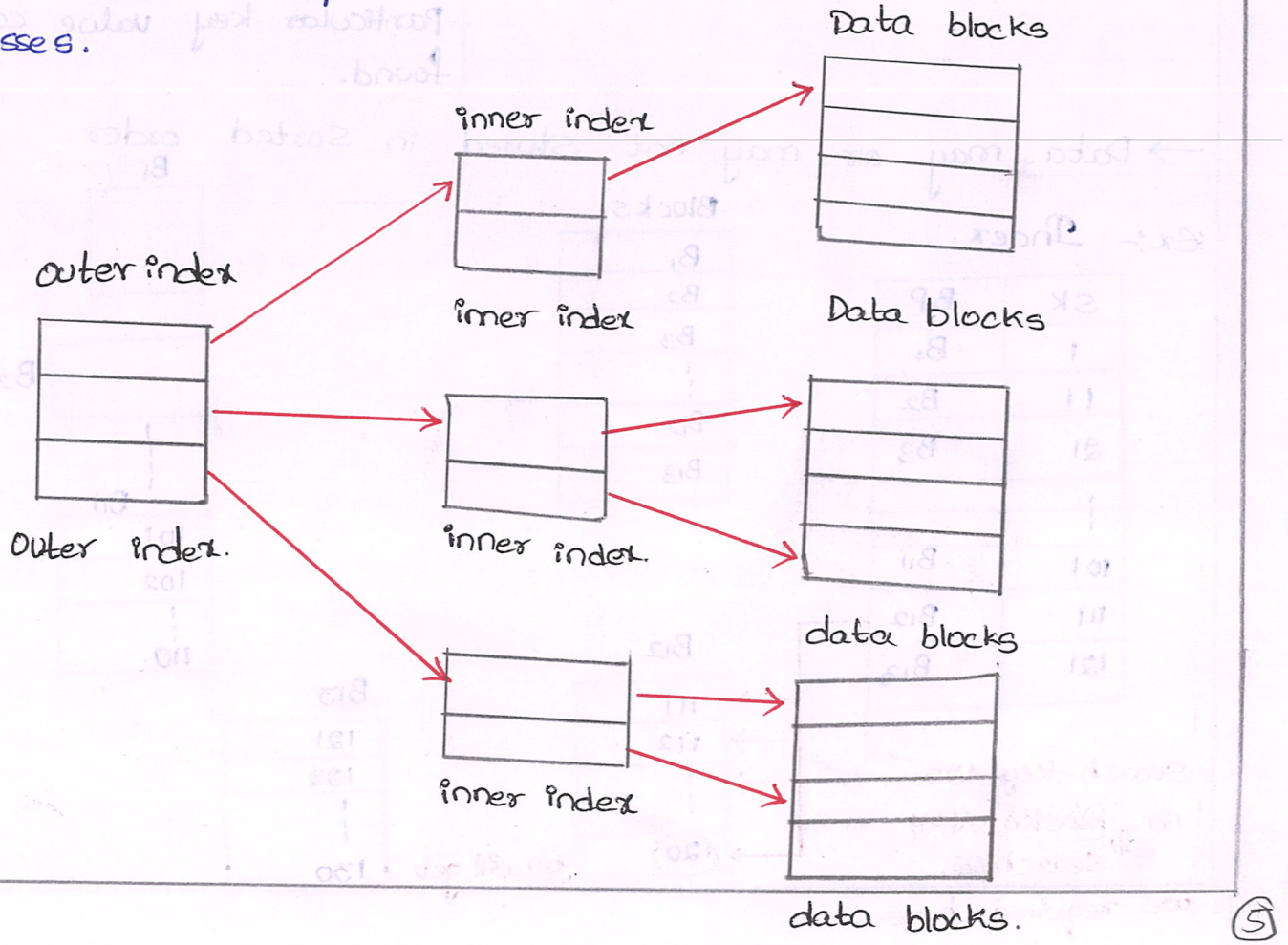
In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reachat the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts

Sequential search until the desired data is found.



Multilevel index:

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



Multilevel index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accomodated anywhere in the main memory.

Indexing :

Minimizing the no. of disk access required.

→ It's a way of optimize the performence of DB.

→ It's a Ds (data structure) which is used to quickly locate & access the data in DB table.

Structure of index :-

Search key	data reference

(or) Block pointer.

Contains a copy of P.k or C.k of a table

contains set of pointers which holds the address of disk block, where Particular key value can be found.

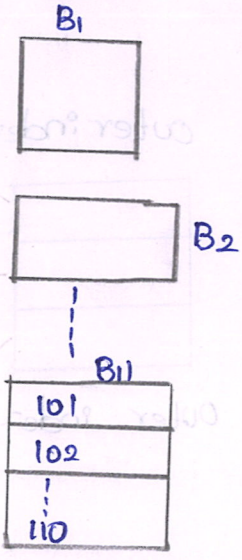
→ Data may or may not stored in sorted order.

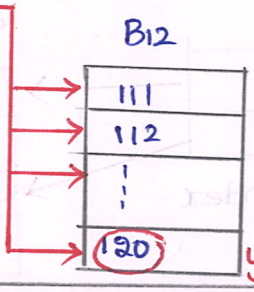
Ex:- Index.

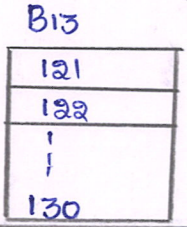
SK	BP
1	B_1
11	B_2
21	B_3
⋮	
101	B_{11}
111	B_{12}
121	B_{13}

Blocks.

B_1
B_2
B_3
⋮
B_{12}
B_{13}







Search-key 120 so 11 blocks you searches we required here.

you will get

Hashing:

For a huge database structure, it can be almost next to impossible to search all the index values through all its level and then reach the destination data block to retrieve the desired data. Hashing is an effective technique to calculate the direct location of a data record on the disk without using index structure.

Hashing uses hash functions with search keys as Parameters to generate the address of a data record.

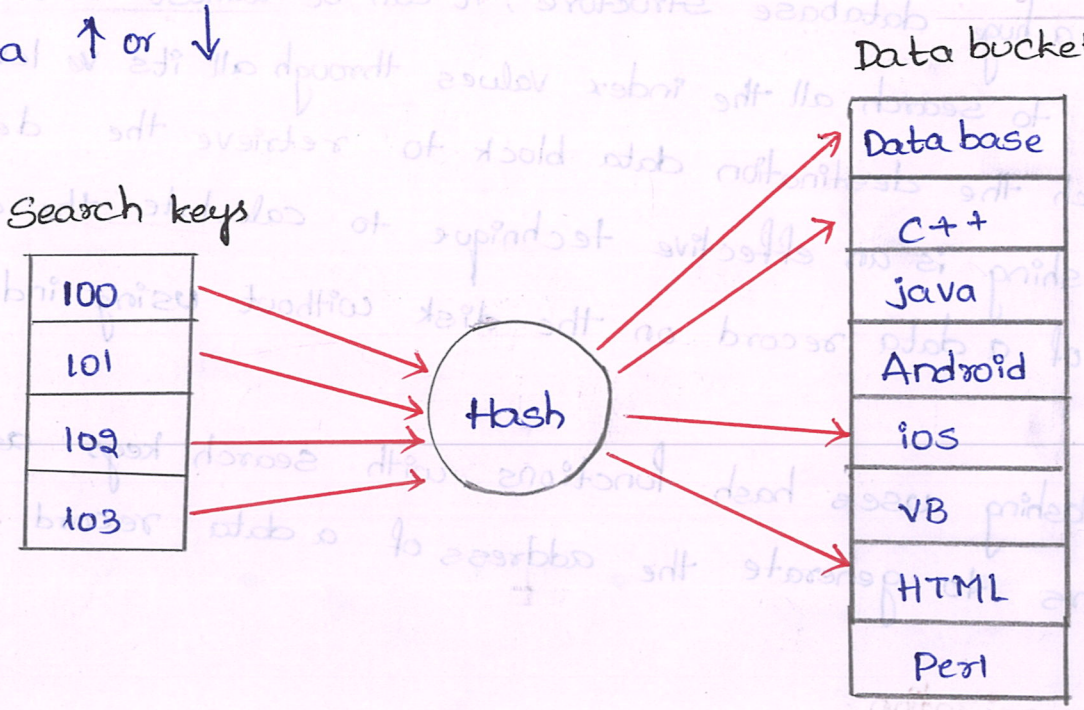
Hash organization:

- Bucket: A hash file stores data in bucket format. Bucket is considered a unit of storage. A bucket typically stores one complete disk block, which in turn can store one or more records.
- Hash function: A hash function, h, is a mapping function that maps all the set of search-keys k to the address where actual records are placed. It is a function from search keys to bucket addresses.

static hashing:

In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if mod-4 hash function is used, then it shall generate only 5 values. The output address address shall always be same for that function. The number of buckets provided remains unchanged at all times.

→ It does not expand / shrink dynamically when size of data ↑ or ↓



Operation:

In static hashing Bucket size is fixed.

→ Bucket overflow is the main problem in static hashing to overcome this we go for close hashing, (or) overflow hashing, open hashing (or) linear probing.

Insertion:

When a record is required to be entered using static hash, the hash function h computes the bucket address for search key k, where the record will be stored.

→ Bucket address = h(k)

Search:

When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.

Delete :

This is simply a search followed by a deletion operation.

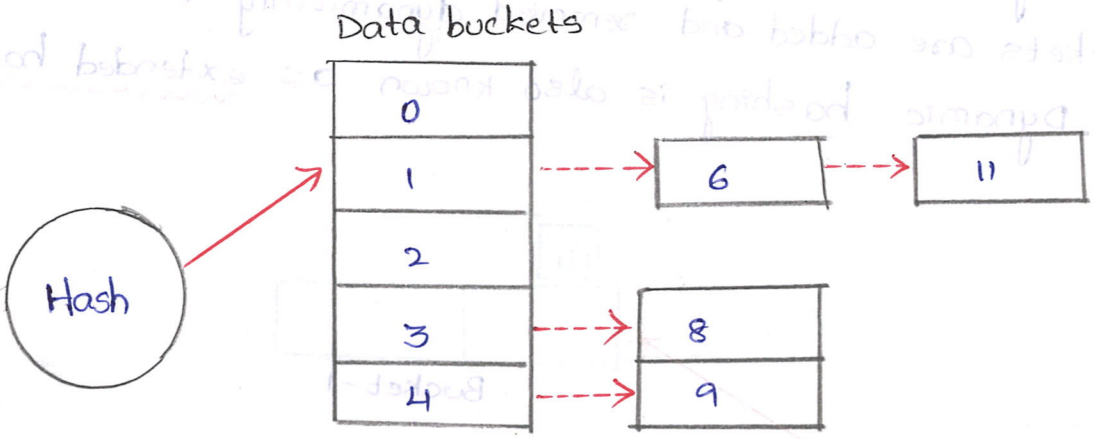
Bucket Overflow :

The condition of bucket-overflow is known as collision. This is a fatal state for any static hash function. In this case, overflow chaining can be used.

- Overflow chaining -

When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called closed hashing

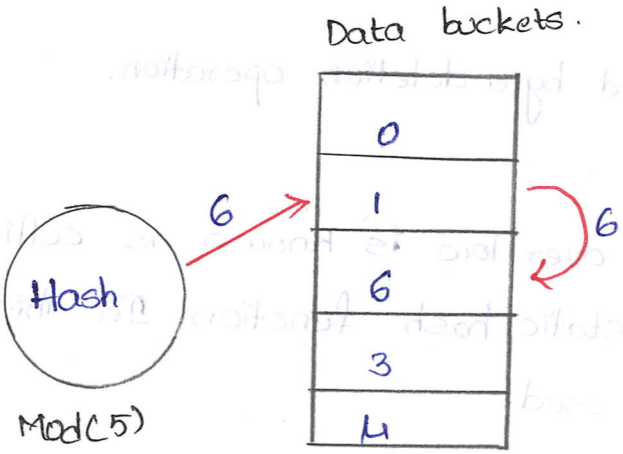
→ We add new bucket.



- Linear probing :

When a hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called open hashing.

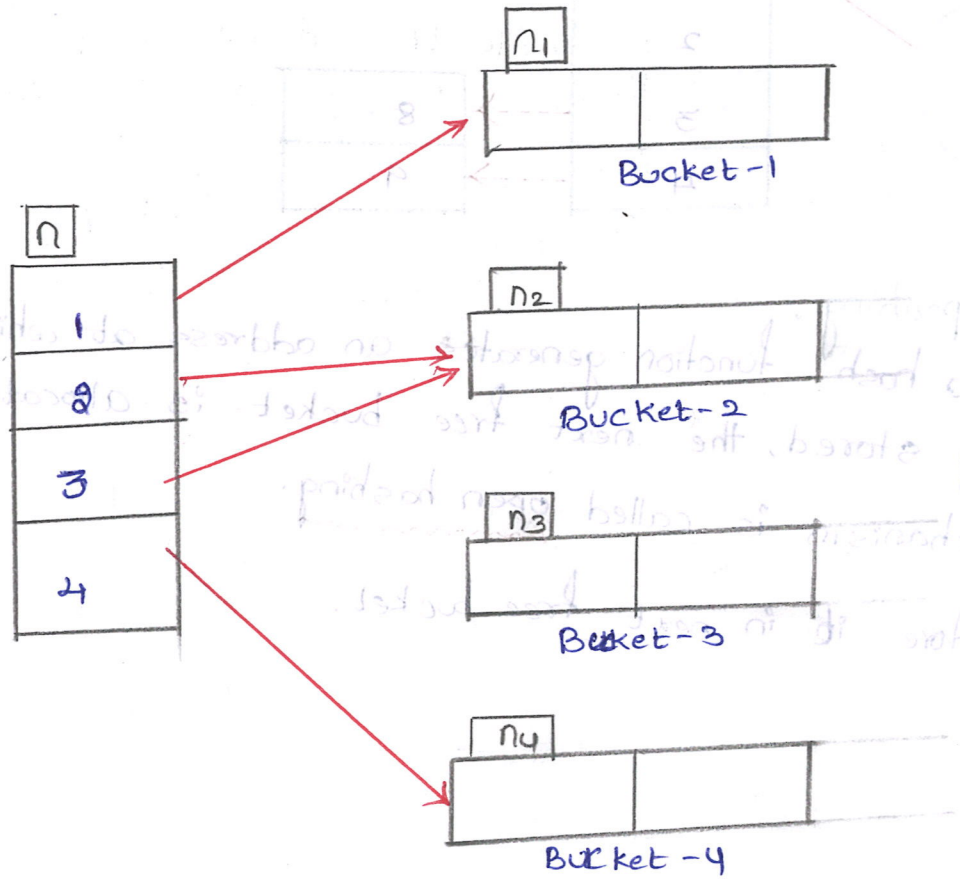
→ We store it in next free bucket.



Dynamic hashing:

→ Very less collisions occur.

→ The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing is also known as extended hashing.



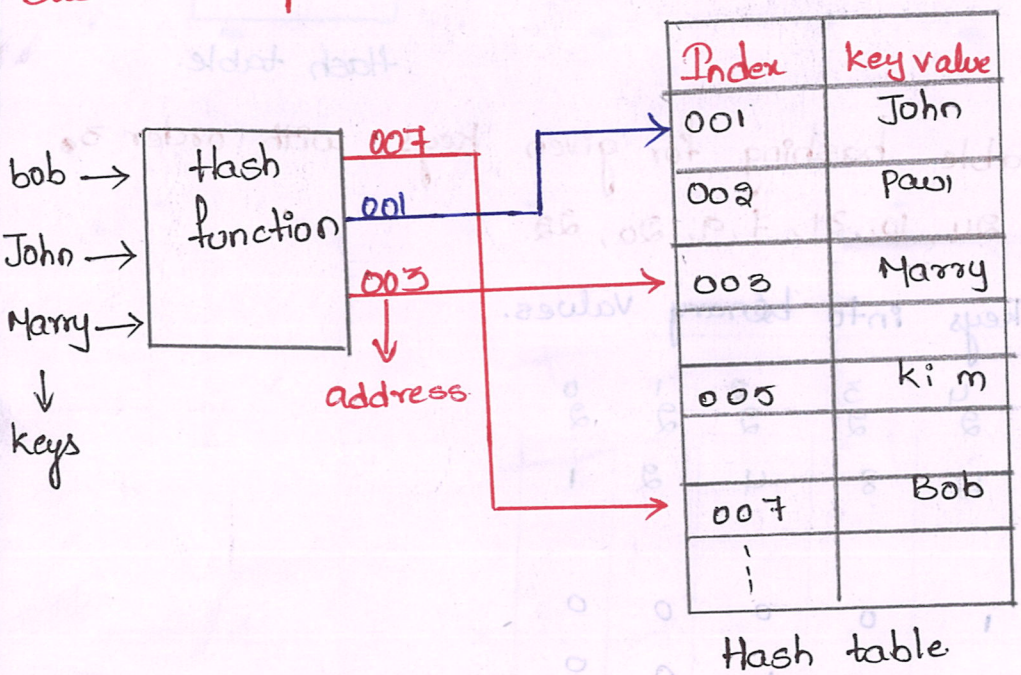
Hashing:

Hashing involves less key comparism and searching can performed in constant time.

→ The goal of a hashed search is to find the target data in only one test (directly search)

Time complexity of hashing technique is O(1)

Basic concept:



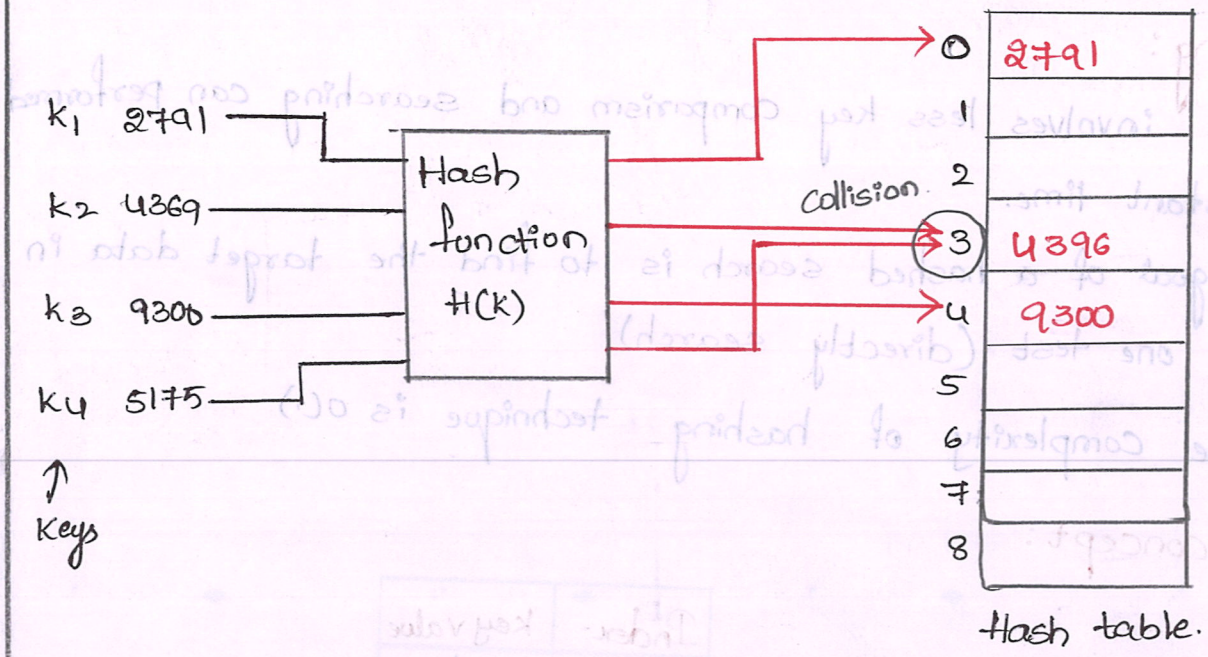
→ Each key has only 1 address.

→ Hash function generates Address for each key value.

→ collesion is the main problem of Hashing technique.

↓

It's defined as the situation when two or more distinct records are mapped to the same physical address, (address generated by the mapping, function may not be unique).



Ex:- Do extendable hashing for given keys. with order 3.

16, 4, 6, 22, 24, 10, 31, 7, 9, 20, 22.

1) convert given keys into binary values.

	2^5	2^4	2^3	2^2	2^1	2^0
	32	16	8	4	2	1
16 -		1	0	0	0	0
4 -		0	0	1	0	0
6 -		0	0	1	1	0
22 -		1	0	1	1	0
24 -		1	1	0	0	0
10 -		0	1	0	1	0
31 -		1	1	1	1	1
7 -		0	0	1	1	1
9 -		0	1	0	0	1
20 -		1	0	1	0	0
22 -		1	0	1	1	0

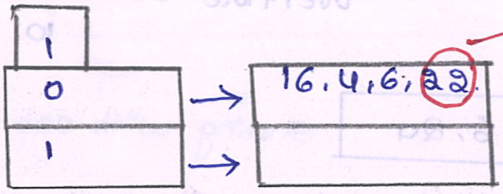
Ex: order is 3 (means we have to insert only 3 values in each bucket).

16 - 10000
4 - 00100
6 - 00110
22 - 10110
24 - 11000
10 - 01010
31 - 11111
7 - 00111
9 - 01001
20 - 10100
2 - 00010

Step 1: Take single last digit value last one

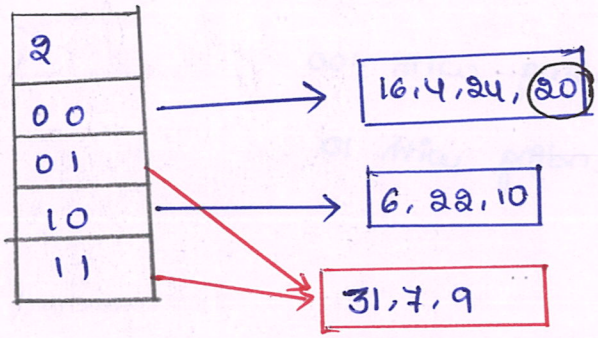
Value you will take

16 - 10000
4 - 00100
6 - 00110
22 - 10110



→ overflow occurs bcz of order 3 so goto step 2.

Step 2: take last 2 digit combinations.



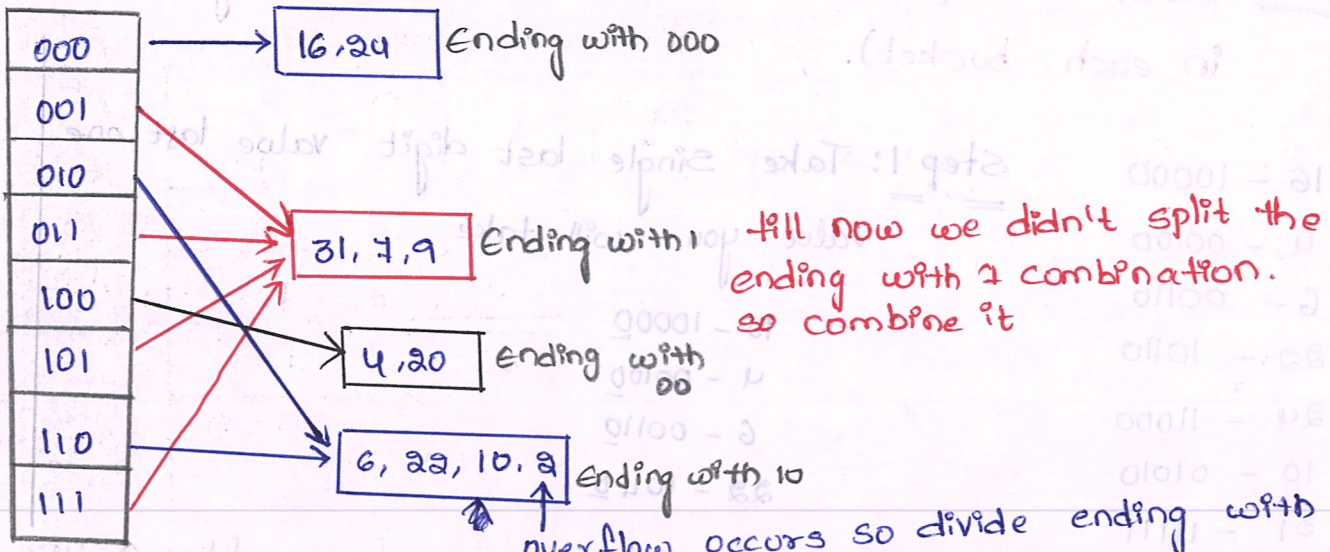
→ overflow occurs. So go with Step 3.

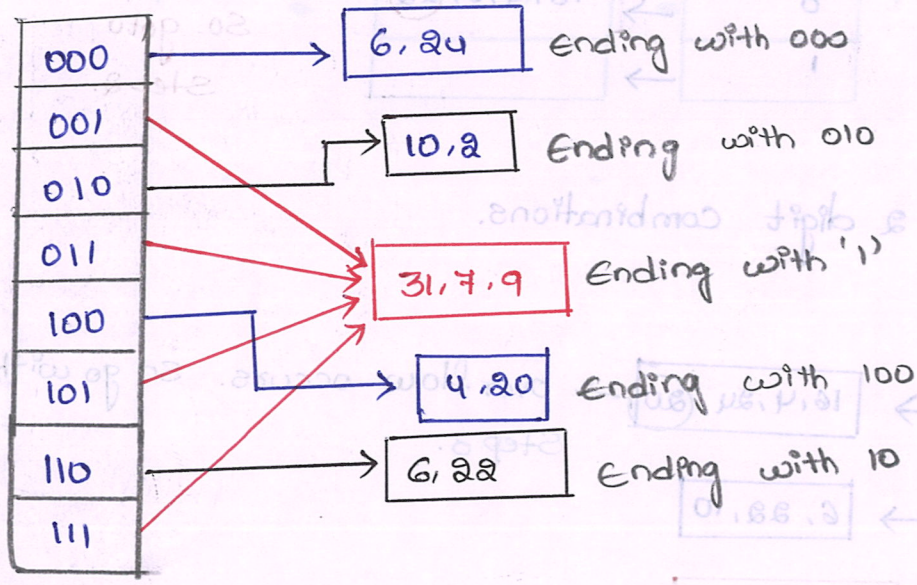
till now we didn't open 1 (ending) block

So just combine it bcz problem occured at ending zero.

So seperate 0 blocks.

Step 3: Take 3 digit combinations. Here problem occured at ending 00 combination so split it.





Note:

→ Take initally ending with 0, ending with 1 combination.

→ If overflow occurs at ending with 0 then divided / split ending with 0 into 00, 10

→ if overflow occurs at ending with 1 then split ending with 1 into 01, 11

→ if overflow occurs at ending with

(i) 00 → 100, 000

(ii) 11 → 011, 111

(iii) 10 → 010, 110

(iv) 01 → 001, 101

Here for every split we have '0' combination & '1' combination.

Hashing:

→ Hashing is a teachnique to directly search the location of designed data on the disk without using index structure.

→ Hashing method is used to index & retrieve items in a DB as it is faster to search that specific item using the shorter hashed key instead of using it's original value.

→ Data stored in the form of datablocks whose address is generated by applying a hash function in the memory location where these records are stored known as a data block or data bucket.

→ For a huge database structure, it's tough to search all the index value through all it's level & then you to reach the destination data block to get the desired need data.

→ It is also a helpful technique for implementing dictionaries.

2 types of hashing methods.

(i) Static hashing:

In static hashing the resultant data bucket address will always remain the same.

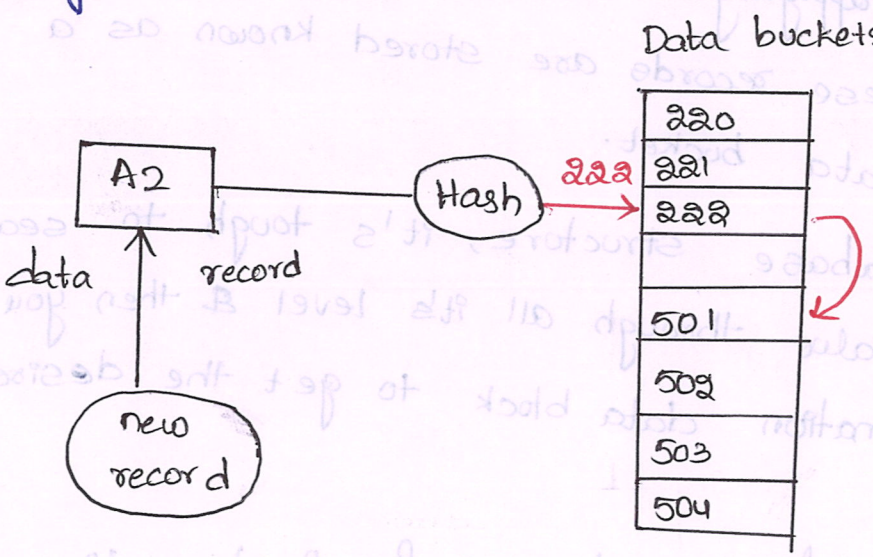
It has 2 types.

1. open hashing 2. close hashing.

→ 2) Dynamic hashing.

It offers a mechanism in which data buckets are added & removed dynamically and on demand. In this hashing, the hash function helps you to create a large no. of values.

Open hashing: In open hashing method, instead of overwriting older one the next available data block is used to enter the new record, this method is also known as linear probing.



Ex: A2, is a new record which you want to insert the hash function generates address as 222. But it's already occupied by same other value, that's why the system looks for the next data bucket 501 and assigns A2 to it.

close hashing:

In close hashing method, when buckets are full, a new bucket is allocated for the same hash and result are linked after the previous one.

Extendible hashing	Linear hashing.
1) In this a directory structure is used.	1) it allocates primary bucket pages successfully.
2) it does not require overflow pages	2) it require temporary overflow pages.
3) space utilization is move	3) less
4) cost is high	4) low
5) performance high	5) performance low.

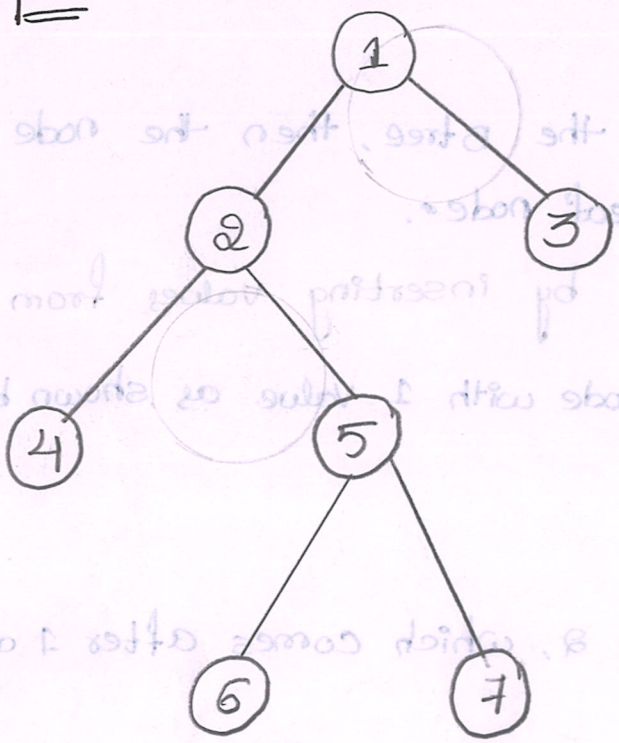
B-tree :

Btree is a self-balancing tree, and it is a m-way tree where m defines the order of the tree. Btree is a generaliza-tion of the binary search tree in which a node can have more than one key and more than two children depending upon the value of m. In the B tree, the data is specified in a sorted order having lower values on the left subtree and higher values in the right subtree.

Properties of the Btree:

- In the Btree, all the leaf nodes must be at the same level, whereas, in the case of the binary tree, the leaf nodes can be at different levels.

Example:

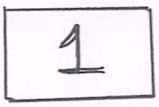


In the above tree, all the leaf nodes are not at the same level, but they have the utmost two children. Therefore, we can say that the above tree is a binary tree but not a B tree.

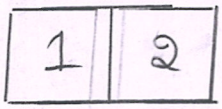
- If the Btree has an order of m, then each node can have a maximum of m in the case of minimum children, the leaf nodes have zero children, the root node has two children, and the internal nodes have a ceiling of m/2.
- Each node can have maximum (m-1) keys. For example, if the value of m is 5 then the maximum value of keys is 4.
- The root node has minimum one key, whereas all the other nodes except the root node have (ceiling of m/2 minus -1) minimum keys.
- If we perform insertion in the Btree, then the node is always inserted in the leaf node.

Create a B tree of order 3 by inserting values from 1 to 10

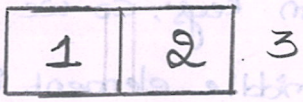
Step 1: First, we create a node with 1 value as shown below:



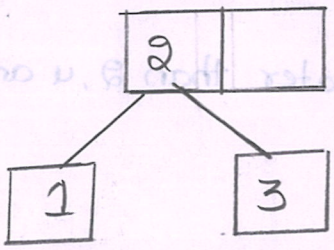
Step 2: The next element is 2, which comes after 1 as shown below:



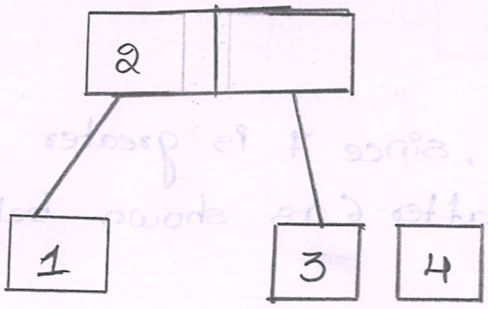
Step 3: The next element is 3, and it is inserted after 2 as shown below:



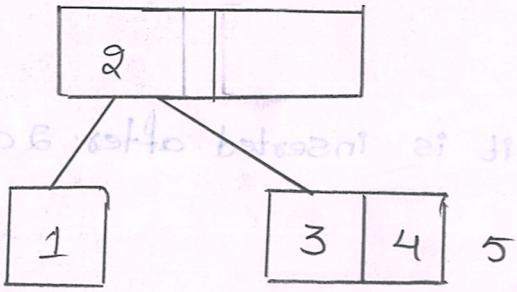
As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 2, so it moves to its parent. The node 2 does not have parent, so it will become the root node as shown below:



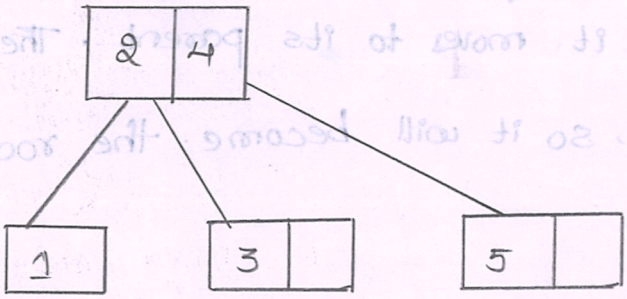
Step 4: The next element is 4. since 4 is greater than 2 and 3, so it will be added after the 3 as shown below:



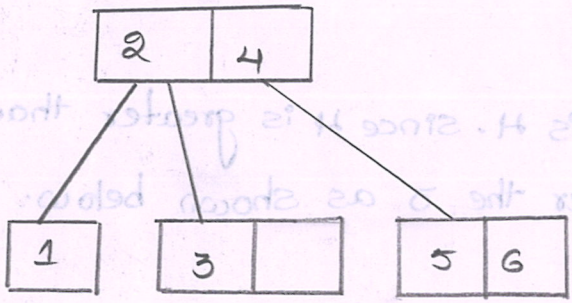
Step 5: The next element is 5. Since 5 is greater than 2,3 and 4 so it will be added after 4 as shown below.



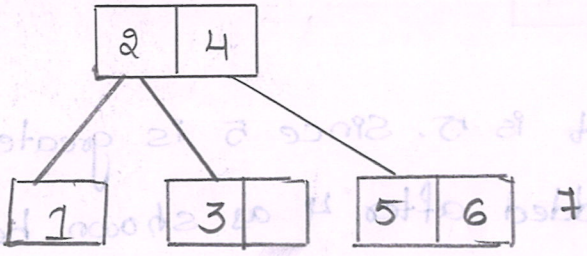
As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 4, so it moves to it parent. The parent is node 2; therefore, 4 will be added after 2 as shown below.



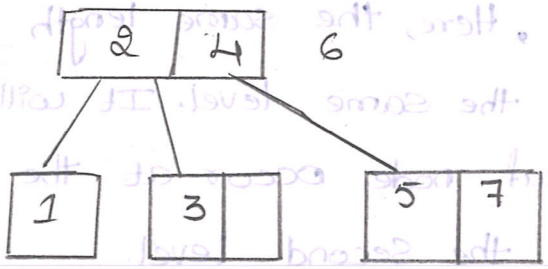
Step 6: The next element is 6. since 6 is greater than 2, 4 and 5, so 6 will come after 5 as shown below:



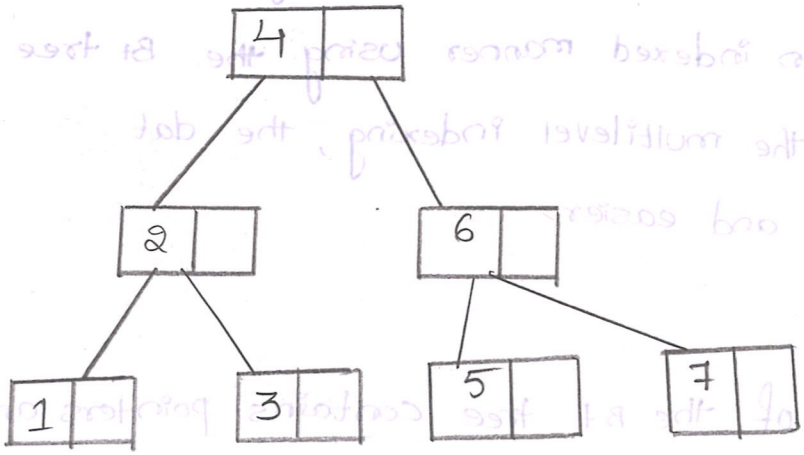
Step 7: The next element is 7, since 7 is greater than 2, 4, 5 and 6, so 7 will come after 6 as shown below:



As we know that each node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 6, so it moves to its parent as shown below:



But 6 cannot be added after 4 because the node can have 2 maximum keys, so we will split this node through the middle element. The middle element is 4, so it moves to its Parent. As node 4 does not have any parent, node 4 will become a root node as shown below.



B+ tree :

The B+ tree is also known as an advanced self balanced tree because every path from the root of the tree to the leaf of the tree has the same length. Here, the same length means that all the leaf occur at the same level. It will not happen that some of the leaf node occur at the third level and some of them at the second level.

A B+ tree index is considered a multi-level index, but the B+ tree structure is not similar to the multi-level index Sequential files.

B+ tree used?

A B+ tree is used to store the records very efficiently by storing the records in an indexed manner using the B+ tree indexed structure. Due to the multilevel indexing, the dat accessing becomes faster and easier.

B+ tree node Structure:

The node structure of the B+ tree contains pointers and key values shown in the below figure:

P_1	k_1	P_2	-------	P_{n-1}	k_{n-1}	P_n

As we can observe in the above B+ tree node structure that it contains n-1 key values (k_1 to k_{n-1}) and n pointers (P_1 to P_n).

The search key values which are placed in the node are kept in sorted order. Thus, if $i<j$ then $k_i < k_j$

Various types of nodes.

Let 'b' be the order of the B+ tree

Non-Leaf node.

Let 'm' represents the number of children of a node, then the relation between the order of the tree and the number of children can be represented as:

$$\left[\frac{b}{2}\right] \leq m \leq b$$

Let's k represent the search key values. The relation between the order of the tree and search key can be represented as:

As we know that the number of pointers is equal to the search key values plus 1, so mathematically, it can be written as:

Number of pointers (or children) = Number of search keys + 1

Therefore, the maximum number of pointers would be 'b', and the minimum number of pointers would be the ceiling function of b/2.

Leaf node :

A leaf node is a node that occurs at the last level of the B+ tree, and each leaf node uses only one pointer to connect with each other to provide the

sequential access at the leaf level.

In leaf node, the maximum number of children is:

$$\left\lceil \frac{b}{2} \right\rceil - 1 \leq k \leq b-1$$

The maximum number of search keys is:

$$\left\lceil \frac{b}{2} \right\rceil - 1 \leq m \leq b-1$$

Root node:

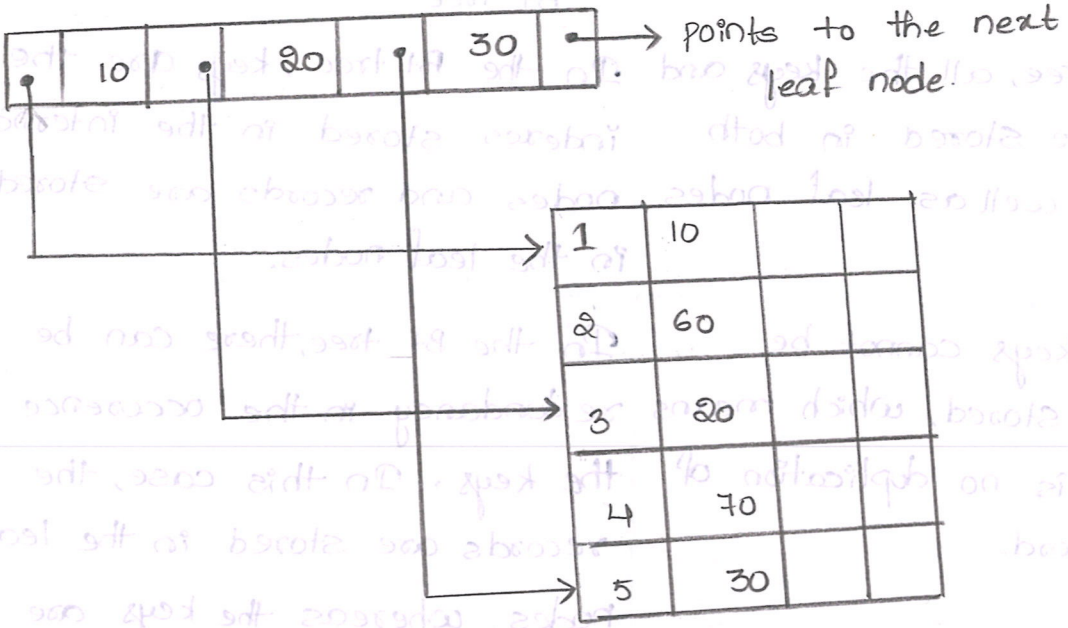
The maximum number of children in the case of the root node is: b.

The minimum number of children is: 2.

Special cases in B+ tree

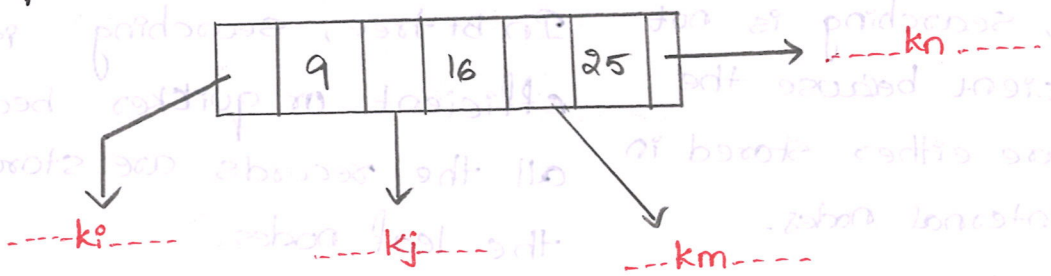
case 1: If the root node is the only node in the tree. In this case, the node becomes the leaf node. In this case, the maximum number of children is 1, i.e the root node itself, whereas, the minimum number of children is b-1, which is the same as that of a leaf node.

Representation of a leaf node in B+ tree.



In the above figure, '.' represents the pointer, whereas the 10, 20 and 30 are the key values. The pointer contains the address at which the key value is stored, as shown in the above figure.

Example of B+ tree.



In the above figure, the node contains three key values, i.e 9, 16, and 25. The pointer that appears before 9, contains the key values less than 9 represented by ki. The pointer that appers before 16, contains the key values greater than or equal to 9 but less than 16 represented by kj. The pointer that appears before 25, contains the key values greater than or equal to 16 but less than 25 represented by kn.

Difference between B tree and B+ tree.

Btree	B+ tree
In the B tree, all the keys and records are stored in both internal as well as leaf nodes.	In the B+ tree, keys are the indexes stored in the internal nodes and records are stored in the leaf nodes.
In B tree, keys cannot be repeatedly stored, which means that there is no duplication of keys or record.	In the B+ tree, there can be redundancy in the occurence of the keys. In this case, the records are stored in the leaf nodes, whereas the keys are stored in the internal nodes, so redundant keys can be present in the internal nodes.
In the Btree, leaf nodes are not linked to each other.	In B+ tree, the leaf nodes are linked to each other to provide the sequential access.
In Btree, searching is not very efficient because the records are either stored in leaf or internal nodes.	In B+ tree, searching is very efficient or quicker because all the records are stored in the leaf nodes.
Deletion of internal nodes is very slow and a time-consuming process as we need to consider the child of the deleted key also.	Deletion in B+ tree is very fast because all the records are stored in the leaf nodes so we do not have to consider the child of the node.

In Btree, sequential access is not possible.	In the Bt tree, all the leaf nodes are connected to each other through a pointer, so sequential access is possible.
In Btree, the more number of splitting operations are performed due to which heigh increases compared to width.	Bt tree has more width as compared to height.
In Btree, each node has atleast two branches and each node contains some records, so we do not need to transverse till the leaf nodes to get the data	In Bt tree, internal nodes contain only pointers and leaf nodes contain records. All the leaf nodes are at the same level, so we need to transverse till the leaf nodes to get the data.
The root node contains atleast 2 to m children where m is the other order of the tree.	The root node contain atleast 2 to m children where m is the order of the tree.