Database Management System

Data - Data is raw, unorganised facts that need to be processed.

Information - When data is process, organised, structured in a given context, to make it meaningful is called information.

Data base - collection of data is referred as data-base. it is collection of inter-related data.

Management system - set of operations which are used to store, retrive, access the data in easy and efficient manner i.e., it must provide the security of data when the system trashes or failures, and accessed by unauthorised user.

A database management system is a collection of inter-related data and a set of programs to access those data.

Applications of database system :

1. Banking
2. Air lines
3. Universities
4. Credit card transaction.
5. Telecommunication
6. Finance
7. Sales
8. Online retailers
9. Manufacturing
10. Human resources.

purpose of DBMS / Advantages of DBMS over file management system :-

1. Data Redundancy and inconsistency :

Redundancy is nothing but duplicate of same file in different applications. i.e., same information may be duplicated in several files. due to this the storage and access cost is more.

Inconsistency : when multiple copies are there of same data in different files, it is highly impossible to retrive the data currently.

example :- change of customer address may not be reflected in saving account which leads to inconsistency of retrieving the data.

2. Difficulty in accessing the data :-

To find the names of all customers to live in a particular area, the data processing department have to generate an application program to retrive the list of all customers. i.e., writing different applications to retrive data in FMS is very difficult.

3. Data isolation :-

Data are stored in various files, files can be in different formats. writing a new application to retrive each file format is difficult in FMS. (file management system).

Integrity problems : The data values stored in the database must satisfy different types of constraints (condition).

example : primary key (not null), unique key, foreign key, check constraint.

Atomicity problems : The atomicity means the entire transaction must be completed successfully or should not be done successfully.

Concurrent access anomalies : Overall performance of the system must allow multiple users to update the data in a sequential way. i.e., updation must be done correctly otherwise it creates a problem of reading a data wrongly.

A	B	C
5000	2000	1000

Transfer from A to B 2000

A	B	C
3000	4000	1000

Transfer from A to C 1000

A	B	C
4000	4000	2000

(updation is wrong).

Security problem :- Not every user of the database system should be able to access all the data.

example : Bank employee can see also the database of different customers who have open the account. he has no permission to access the details of customer account.

Database Languages :-

A Database system provides data definition language to specify overall design of the database (schema). Data manipulation language is use to generate the queries and update the values.

DDL and DML are a part of single database language such as SQL (structured query language).

Data manipulation Language (DML) :-

A DML is a language that unable users to access or manipulate data organised by appropriate data model. the type of access are
- insertion of new data.
- Deletion of information from database
- Modification of information stored in database

There are two types of DML

1. procedural DML. - requires a user to specify what data needed and how to get those data.

2. Declarative DML - it is also called as non-procedure DML where user have to specify what data are needed, without specifying how to get those data.

Query - A query is a statement requesting the retrival of information.

The portion of a DML that involves informational retrival is called a query language.

Data Definition language (DDL) :-

DDL defines the implementation of details of the database

schema's which are usually hidden from the user.

→ The data value stored in the database must satisfy different constrains

1. Domain constrains : A Domain of a possible values must be associated with every attribute.
ex: character, number, Date, Time, data types of different types.

2. Referential Integrity : This are the cases when the value which appear in one relation for a given set of attributes also appears for a set of attributes in another relation.

3. Assertion : An assertion is any condition that the database must also satisfy. i.e., domain and referential constrains.

4. Authorisation : In order to differentiate the users, the type of access they are permitted on the various datavalues in the database.

- Read Authorisation - Allowed to see the entire database and no permission is given for insertion, updation, deletion.
- Insert Authorisation - permission is given to insert new information in the database but not for deletion
- update Authorisation - permission is given to update specific data values in the database but not for deletion.
- Delete Authorisation - permission is given to delete specific datavalues in the database

History of DataBase :-

1950's to early 1960's - magnetic tapes are developed for data storage. Tapes can read the data only in a sequential way. The new data has to be copied forcebly in order into an another tape, which is very difficult when the database is very large.

Late 1960's and 1970's - Hard disk usage is used for data processing within a milliseconds.

Herarical network databases used the data structures like linked list and trees.

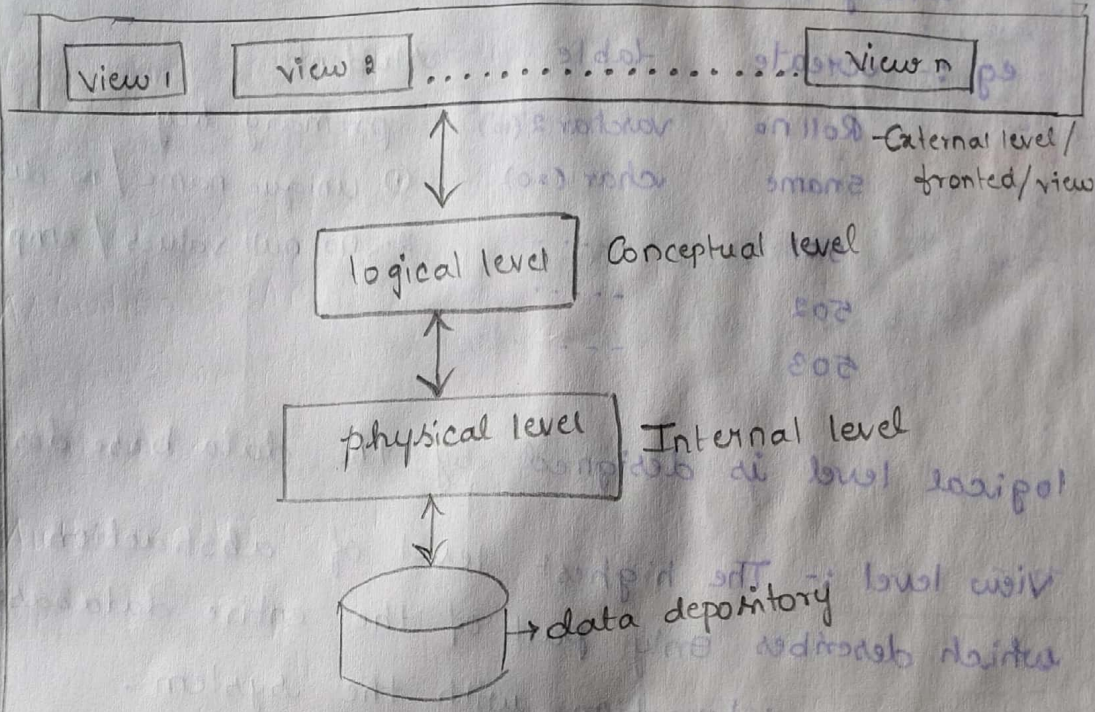
1980's - code defined the relational model which is not matching the performance of HMS and NMS. due to that IBM research center developed efficient relational DBS such as IBMDB2, Oracle, ingres, DECRDB which played a major role in processing the declaration querries i.e., relational model became the supreme to all other data models. Here only transaction processing applications are used.

Early 1990's - SQL Language was designed to support for decision support application.

Late 1990's - Due to huge growth of world wide web applications 24×7 database systems are developed.

Early 2000's - Due to emerging of XML (xtended markup language). the querry language called xquerry is used for new data base technology.

Abstraction level / Data level / views of Data



physical level :- The lowest level of abstraction describes 'how' the data is actually stored. It describes the complex low level data structures in detail.

eg :- Create table student
(Roll no varchar(10), sname char(20))

physical level it describes as

Roll no		sname	
	length 10		length 20
	offset 1		offset 1
	bytes 4		bytes 2

The physical level describes the complete storage locations in the form of bytes.

Logical level :- The next higher level of abstraction describes "what" data is stored in the database and what relationship is exist among the data.

eg :- create table student

Roll no	varchar 2 (10)	primary key
Sname	vchar (20)	① unique name / no duplicate
501	-----	2. no null values / empty.
502	-----	
503	-----	

logical level is designed by the data base designer.

View level :- The highest level of abstraction which describes only part of the entire database users make interaction with the system.

The system may provide many views for the same database. It is called as frontend applications / web applications.

Data Independence :- There are two types of data independence

1. Logical data Independence - it is a capacity to change the conceptual schema without change in the external schema or application programs.

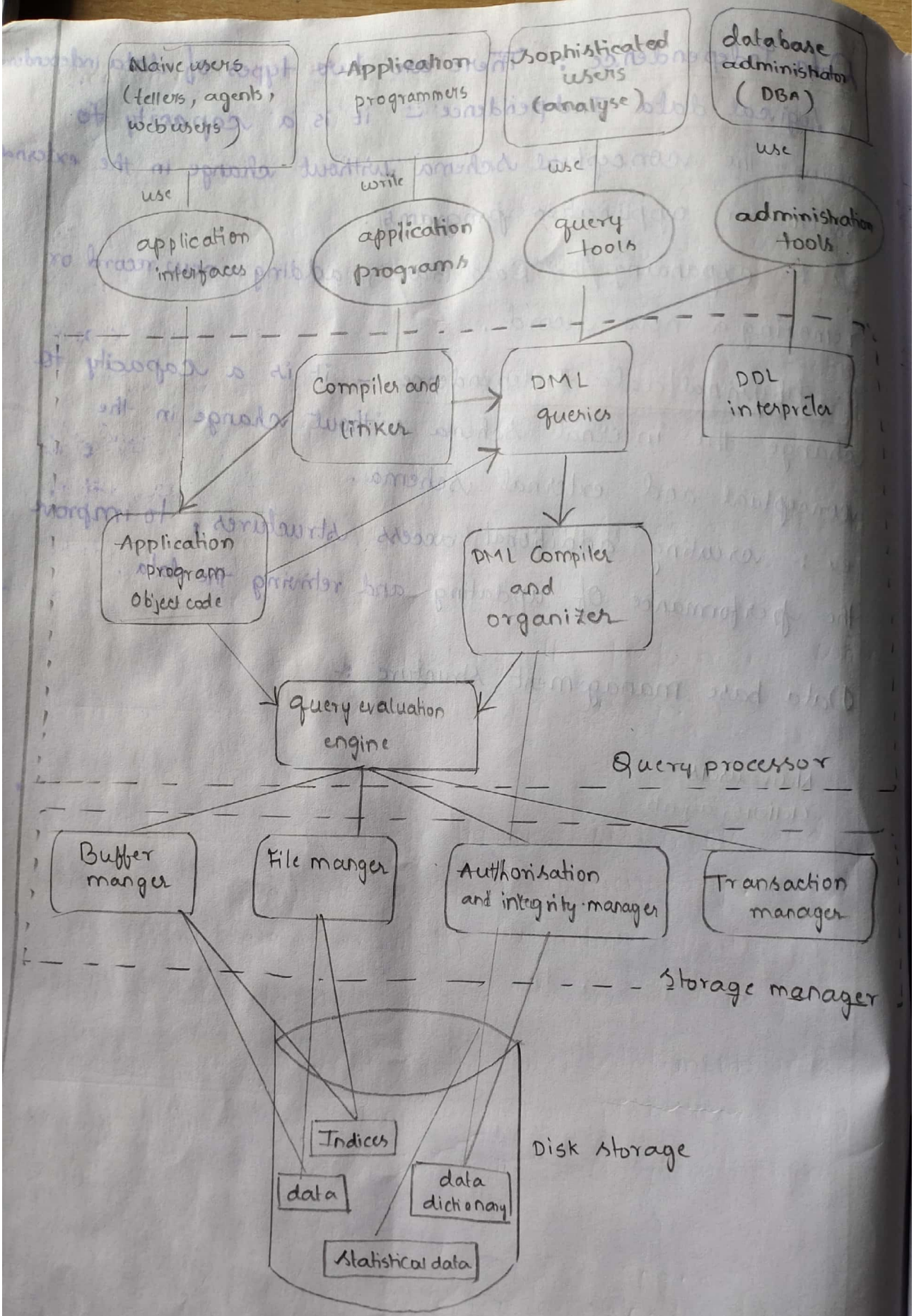
ex :- expanding the Data Base, adding a new record or removing a new record.

2. physical data Independence - it is a capacity to change the internal schema without change in the conceptual and external schema.

ex :- creating additional access structures, to improve the performance of updating and retriving the data.

Data base management structure :-

Naive users
(tellers, agents



System structure of Database.

Database users and Administrator :-

1. Naive users - The users who are going to interact with the system by invoking any one of the application program that have been written previously.
example : Transferring an amount from one account to another account.
Naive users can read the report generated from the data base.

2. Application programmers - The computer professionals who will write application programs by choosing many tools to develop user interfaces.

3. Sophisticated users - Interact with the system without writing programs. this users will break down the DML statements into instructions. that storage manager can understand.

4. Specialised users - This users will be dealing with data applications that do not fit the data processing frame work. i.e., expert systems, storing the data with complex data types and environment modeling system.

Data base administrator :- The functions of DBA for schema definition - DBA creates the original database schema by executing set of DDL statement

Storage structure and Access method definition : defines the memory allocation, permission to access the data.

Schema and physical organisation - modification : The DBA carries out the changes to the schema and storage structure in order to improve the performance;

Granting of Authorisation for data access : The DBA Grants different types of authorisation to the user to access the data. i.e., read, insert, update, delete.

The DBA is mainly responsible periodically checking the backup of databases when there is a loss of data in the case of disasters.

Making the NF free database disk space availabe to carry out normal operations.

Monitoring the jobs, checking the performance when expensive task are submitted by users.

Query processor :-

DDL interpreter - Interprets the DDL statements and records the definitions in the data dictionary (which stores meta data) about the structure of the database. (meta data means data about data)

DML compiler - Translates the DML statements in a query language into an evaluation plan consisting of low level instructions that the query evaluation engine understands.

Query evaluation engine - which execute low level instructions generated by the DML compiler.

Disk storage :-

1. Data files - which stores the database itself.
2. Data dictionary - which stores the meta data about the structure of the database.
3. Indices or Indexing - which can provide fast access to data item.

Database index provides pointers to those data items that hold a particular value.

ex : Hashing techniques.

Statistical data : it is used for analysing the database i.e., summaries of values based on various criteria.

Storage manager :-

Buffer manager - Responsible for fetching the data from disk storage into main memory, deciding what data must be kept in main memory.
It is a critical part of the database system.

File manager - Manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

Transaction manager - It ensures that database remains in a consistent state (correct state) in even though the system failures occurs.

Authorisation and Integrity manager - checks the integrity constrains on the database, also checks the authority of users to access the data.

Database design and ER diagrams :

There are 6 basic steps in designing a database

1. Requirement analysis : It understands with data to be stored, what operations and application are used and kept for database requirement and applications must be kept at the top

2. Conceptual database design :- It is used to develop a high level description of data to be stored in database, along with constraint that must told a de

3. Logical database : It is used to convert the conceptional database design into schema based upon different data models.

Schema refinement : performing the normalisation in the database schema's (reducing the redundancy in databas

physical database design : It is used to build the indexing, clustering of the table, redesigning different parts of the database in order to improve the performan of the application.

Application and security design : Identifying the entity, role of each entity, process involved in the application task. identifying the database that is accessable and not accessable by each role in the entity i.e., complete workflow of the task.

Data models :-

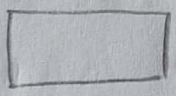
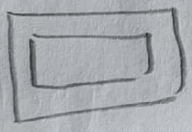
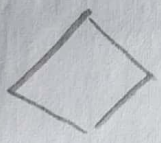
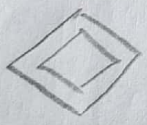
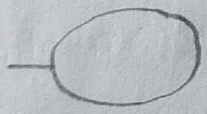
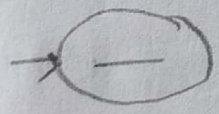
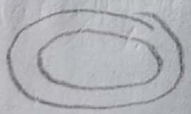
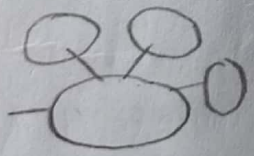
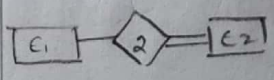
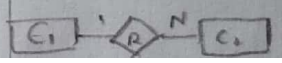
underlining the structure of database is called data model.

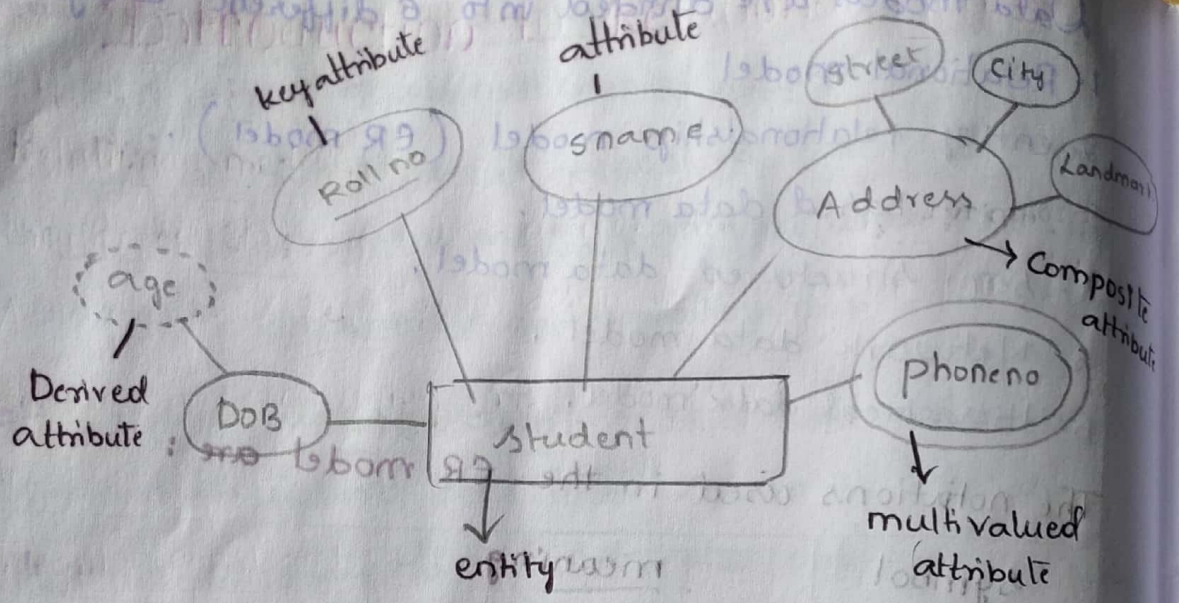
A model is said to be abstraction of real world obj or event.

Data models are divided into 6 different categories:

1. Relational model
2. entity relationalship model (ER model).
3. object based data model.
4. Semi structured data model.
5. Network data model.
6. Hierachical data model.

The notations used in the ER model are:

symbol	meaning
	Entity
	weak entity
	Relationship
	Identifying relationship
	Attribute
	Key attribute
	Multivalued attribute
	composite attribute
	Derived attribute
	Total participation of E_2 in R
	Cardinality ratio 1:N for E_1, E_2 in R



Entity :- An entity is an real time object that exists in the world and it is distinguishable from other objects.

ex :- Employee, student, car.

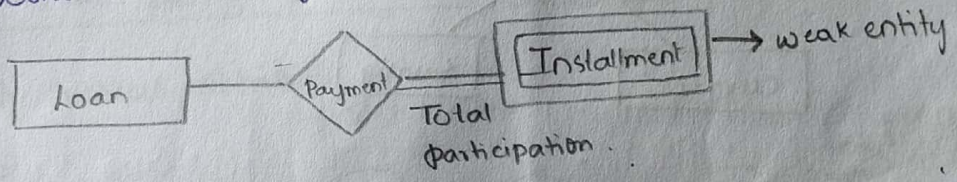
Attribute :- Each entity is describe with a set of attributes called properties of an entity.

ex :- rollno, branch, name.

Entity set :- An entity set is a set of entities of same type that shares same properties.

Relationship : It is an association among different set of entities.

Weak entity : Entity which depends on anther Entity is called weak entity. it doesn't contain any key attribute of its own. it contains total participation in entity relationship.

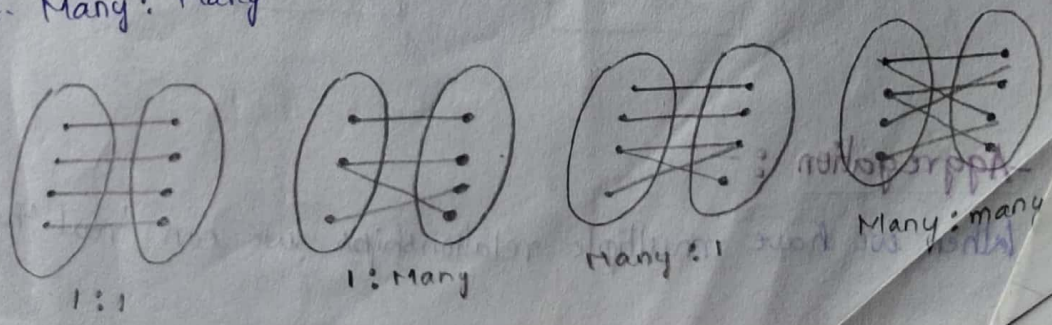


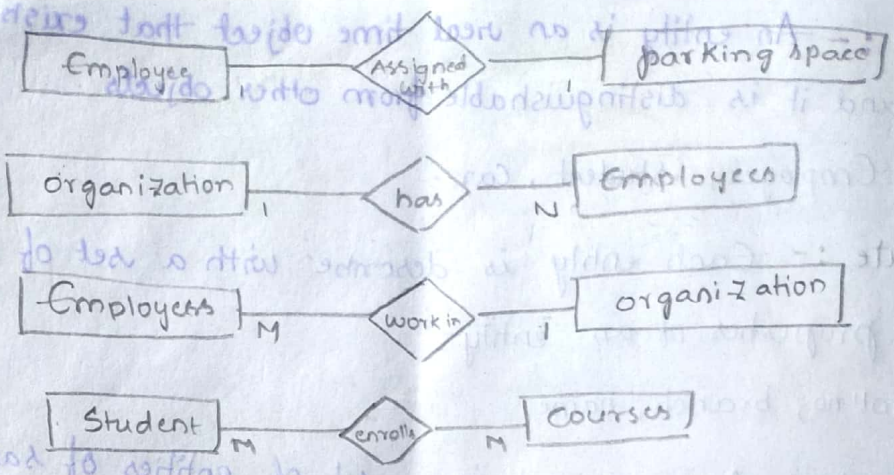
Strong entity : It contains a key attribute. it doesn't depend on any other entity. it doesn't contain any total participation in the entity relationship.

Key constrains :-

There are four types of key constrains.

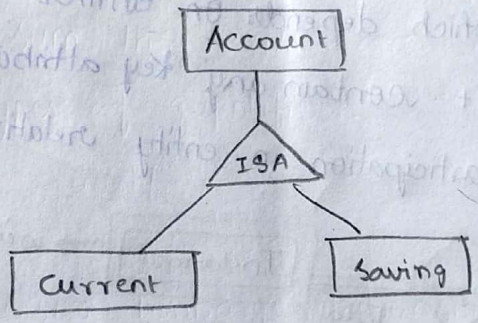
1. 1:1
2. 1: Many
3. Many: 1
4. Many: Many





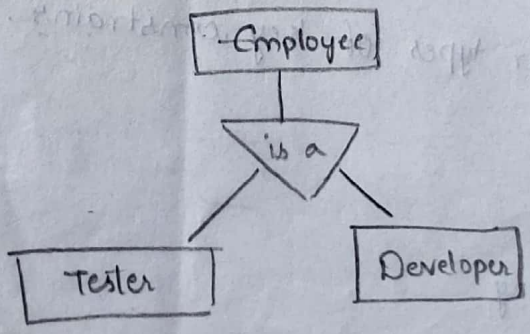
Generalisation :-

It is a bottom of approach in which two or more entities can be generalised to a higher level entity if they have same attributes in common.



Specialisation :-

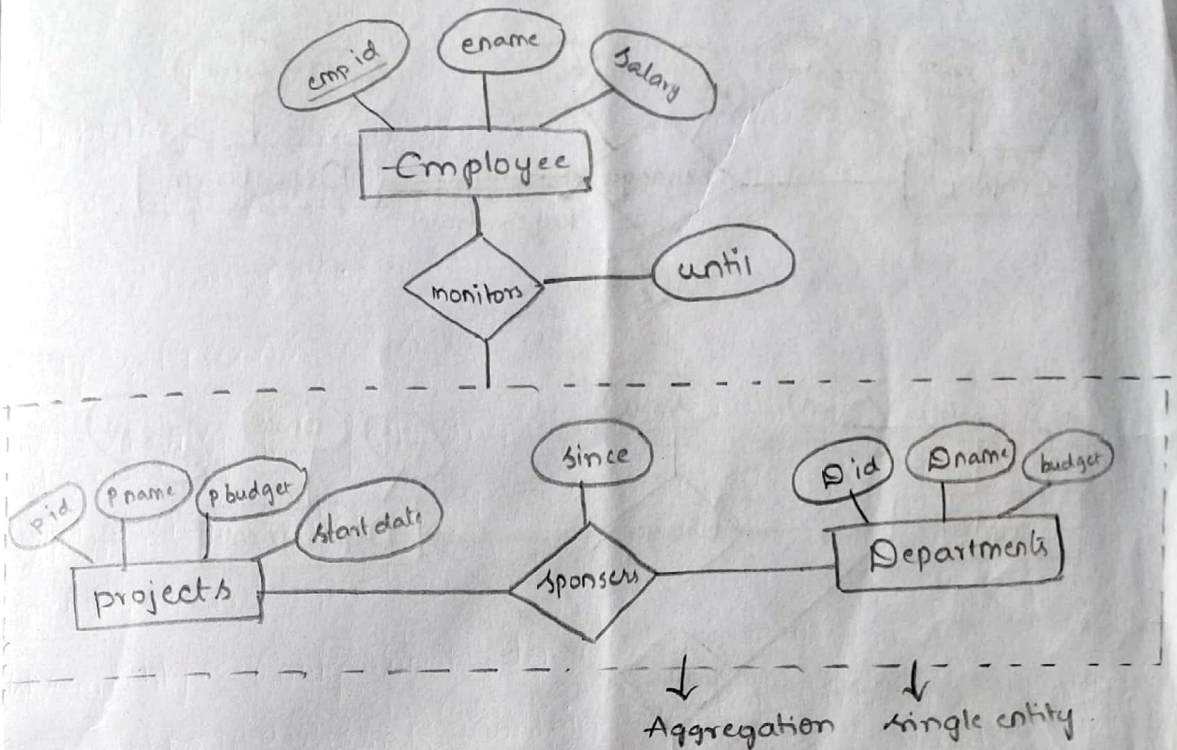
In specialisation, an entity is divided into sub-entities based upon their characteristics. it's a top-down approach. where higher level entity is specialised into two or more lower level entities.



Aggregation :-

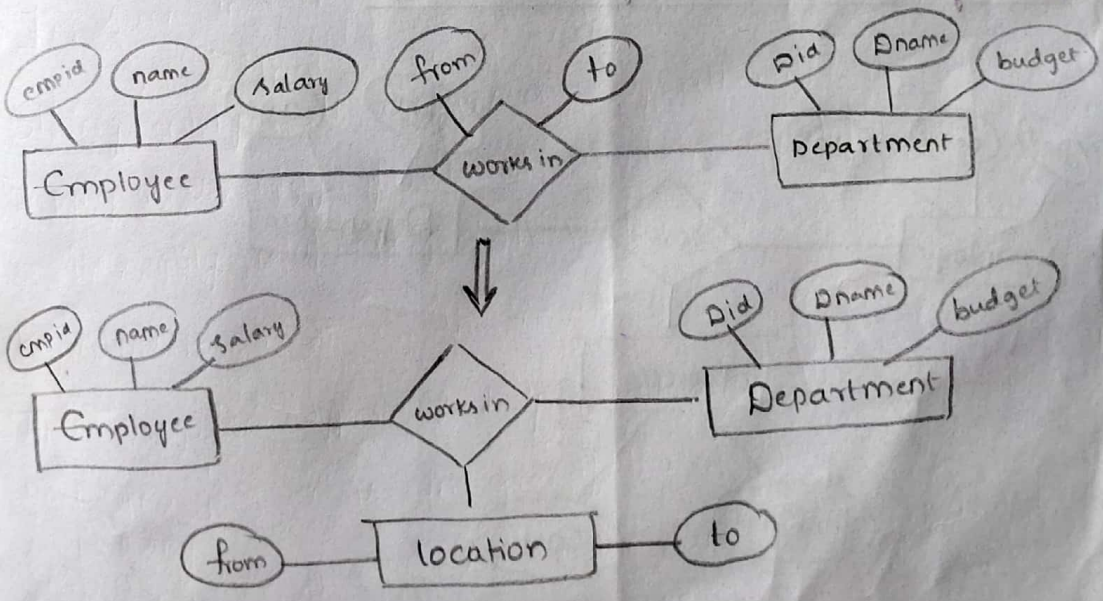
When we have multiple relationships we can treat this multiple

relationship as a single entity in order to avoid conflicts in understanding the relationships.

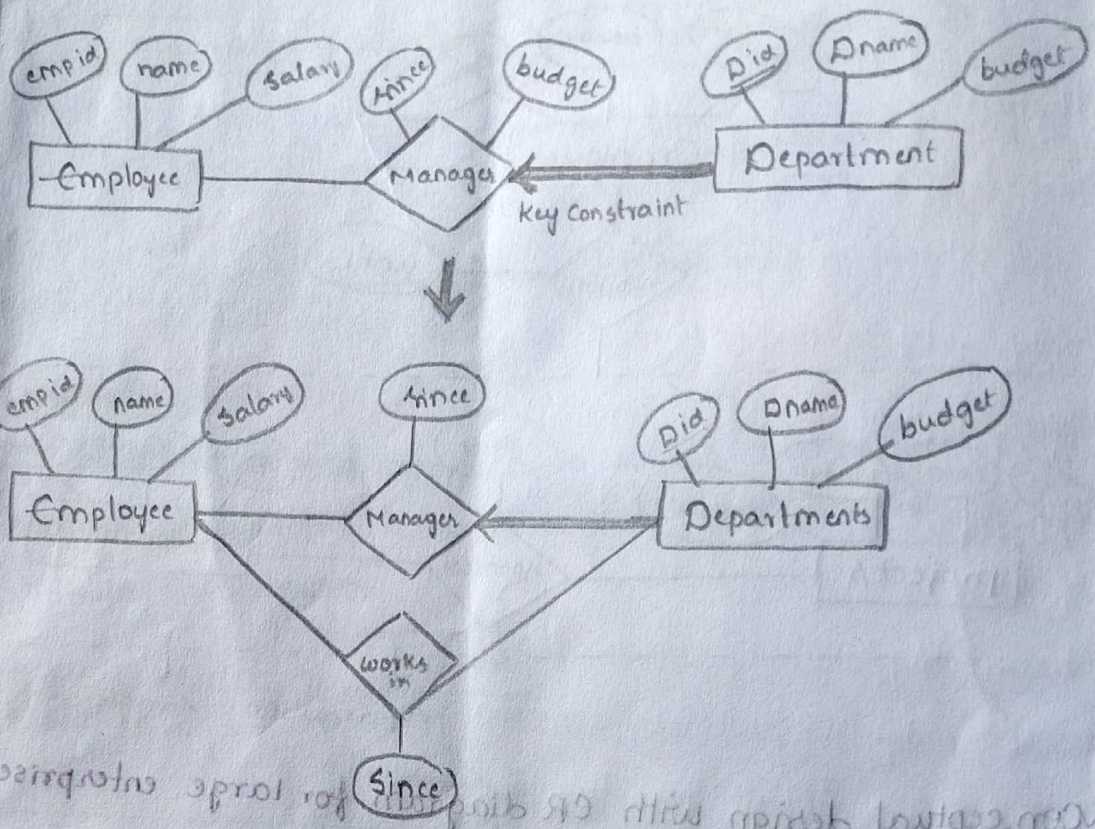


Conceptual design with ER diagram for large enterprises :-

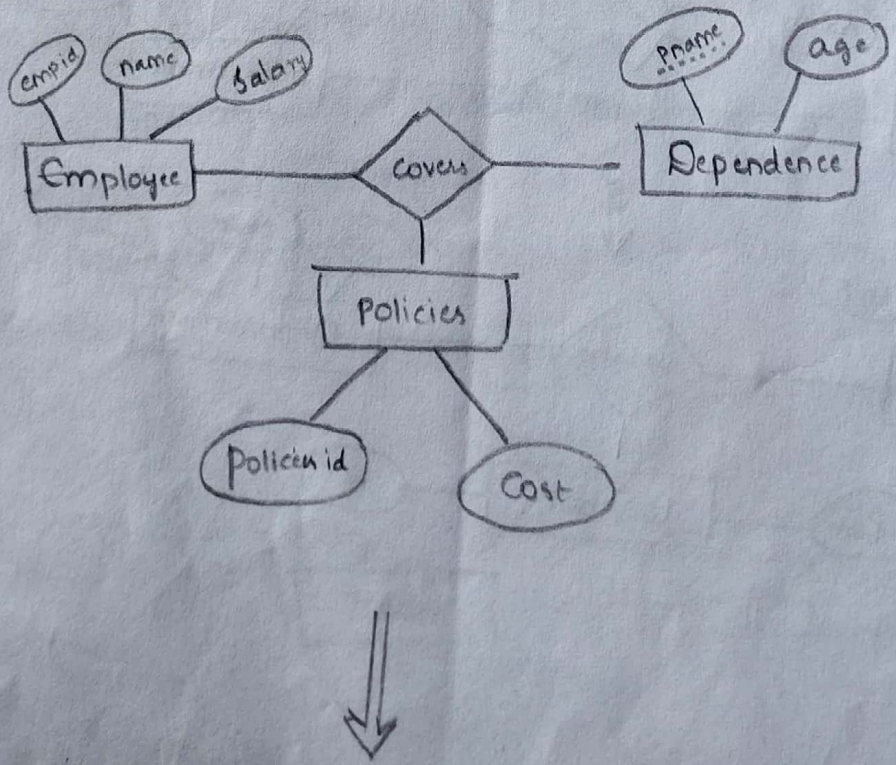
Entity versus Attribute :



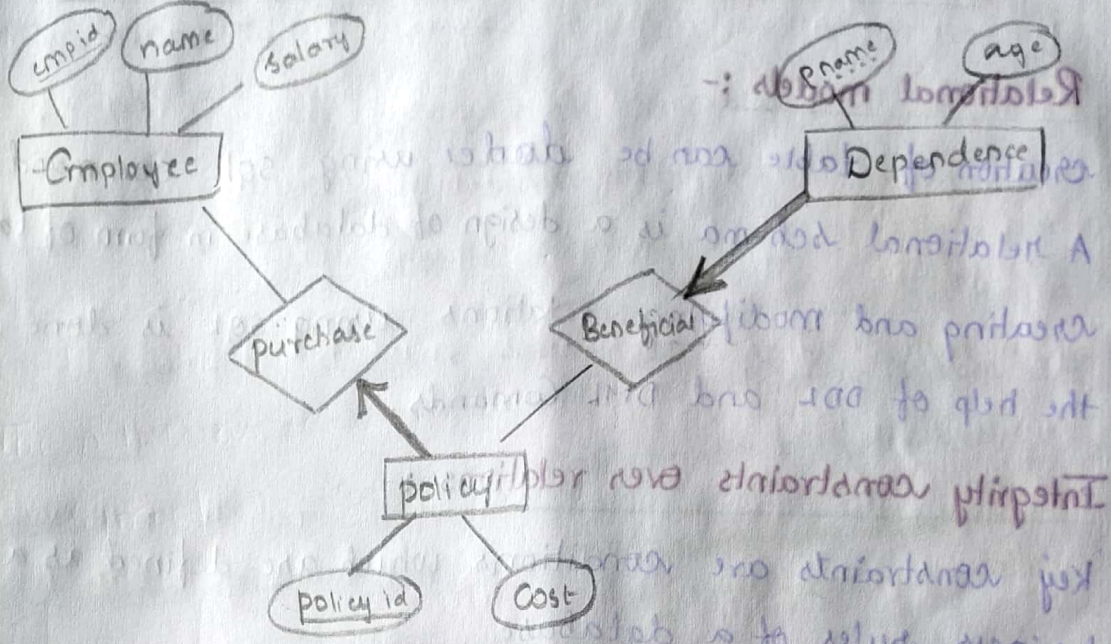
Entity versus relationships :



Binary versus Terinary relationships :-



Introduction to Relational Models



Introduction to Relational Models

Relational models :-

Creation of table can be done using SQL command.
A relational schema is a design of database in form of tables.
Creating and modifying relations using SQL is done with the help of DDL and DML Comands.

Integrity constraints over relations :-

Key constraints are conditions which are defined as a business rules of a database.

primary key :- An attribute which is used to identify records uniquely.

it should not contain any duplicate values and null values.
A null value indicates that value unknown or non-existence.
It is not equal to a space or zero in the database.
It is denoted as "null" in the database.
primary key can be created in three levels.

1. Create table tablename (Columnname1 datatype(size) primary key, columnname2 datatype (size))

2. Create table tablename (Columnname1 datatype (size), Columnname2 datatype (size), primary key (columnname1, columnname2))

3. Alter table tablename add Constraint constraint-name primary key (columnname);

Foreign key :- It is a table level constraint. to reference any primary key column from other tables. this constraints

can be used.
The table in which the foreign key is defined is called a child table.
The table that defines primary key and is referenced by foreign key is called master table (or) parent table.
The columnnames can be different but the datatype and size must be same when we use references of the table.

create table tablename (columnname1 datatype (size), columnname2 datatype (size), foreign key (columnname2) references tablename (columnname));

- tablename → present tablename
- (columnname) → The column which is defined as primary key and taken as reference to a child.
- (columnname2) → child columnname which is to be referred from parent table.

Alter table tablename add constraint constraint_name foreign key (column name) references tablename (columnname);

ex:-

department

Dep no	Dname
10	Sales
20	Finance
30	Marketing

employee

empid	ename	Depid
101	Sai	10
102	Sree	10
103	Shiva	20

create table department (deptnumber (4) primary key, dname char (10));

create table employee (empid number (4) primary key, ename char (10), deptid number (4), foreign key (deptid) references department (deptno));

unique key : This constraints can be defined at table level or column level. You can have many unique key for

a table.

It will not allow duplicate values but null values.

Create table tablename (columnname1 datatype (size), columnname2 datatype (size), unique (columnname2));

Unique key is quite opposite of primary key.

Not null :- These constraints must be defined at columnlevel only.

It will allow the duplicate values but not the null values because the name itself says not null. you can have 'n' number of columns as not null.

Create table tablename (columnname1 datatype (size) not null, columnname2 datatype (size));

candidate key :- minimum set of attributes used to define or read uniquely the records in the table..

minimum set of fields which can uniquely identify each record in the table.

There can be more than one candidate key.

-A candidate key should not contain null values and duplicate values.

empid	ename	salary	Adhano	Adhno.
101	shiva	35,000	101A1	0543
102	sri	40,000	102B2	1234
103	manu	45,000	103C6	1880
104	Ramu	60,000	104D5	1923

empid → primary key

empid, Adhar no, pan → candidate key

primary key is a part of all candidate key.

Super key : A super key is a combination of columns that uniquely identifies any row within a key.

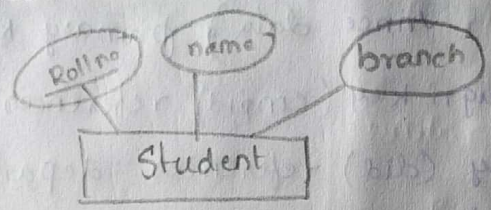
by default every relation has one superkey.

Super key = candidate key + zero or more than zero attributes.

In the above table, emp id is said to be as a superkey as well as pan no and Adhar no.

Logical database design : ER to relational

1.-Entity sets to tables

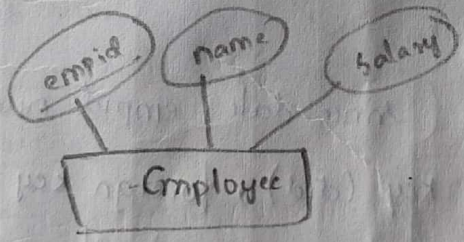


Roll no	name	branch

student → entity → tablename,

Rollno → primary key

Create table student (Rollno number(10) primary key , name char(10), branch char(5)) ;

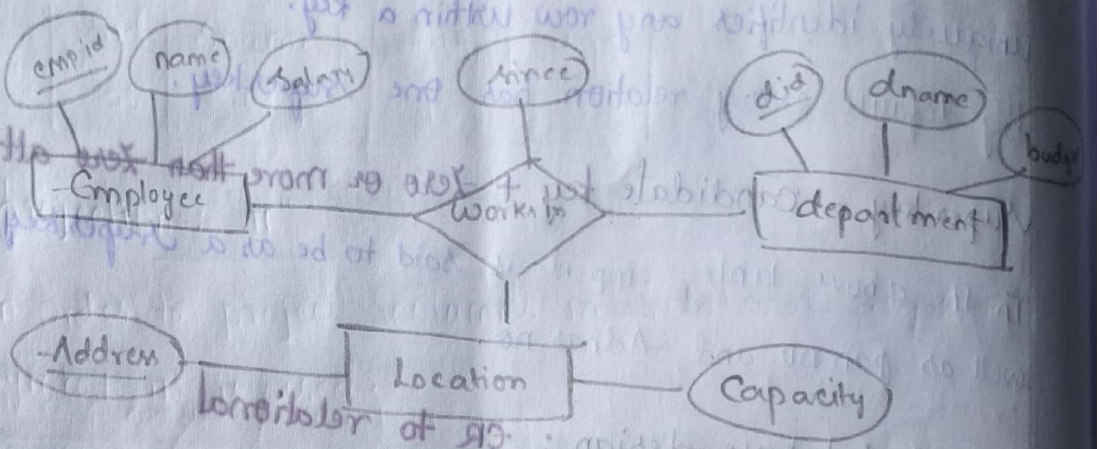


Employee → entity → tablename;

empid → primary key

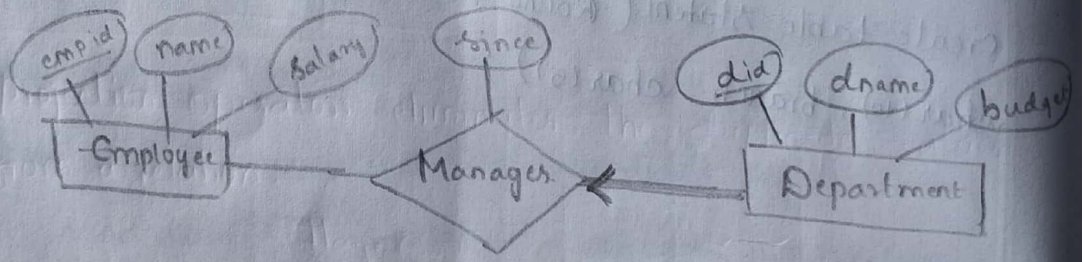
Create table Employee (empid number(5) primary key , name char(10), salary number(5)) ;

2. Relationship sets (without constraints) to tables:



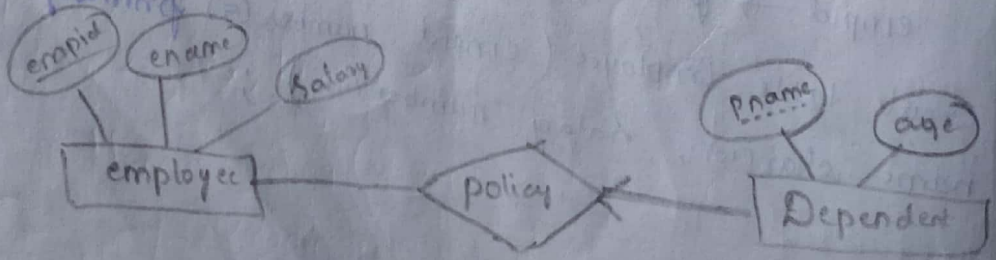
Create table works in (empid number(5), did number(10), address varchar2(10), since date, primary key (empid, did, address), foreign key (empid) references employee (empid), foreign key (did) references department (did), foreign key (address) reference location (address)) ;.

3. Translating relationship sets with key constraints :



Create table manages (since date, empid number(10), did number(10), primary key (did), foreign key (empid) references employee (empid), foreign key (did) references department (did)

4. Translating weak entity sets :-



create table dept-policy (pname char(20), age integer, empid number(10), primary key (Pname, empid), foreign key (empid) references employee(empid) on delete cascade) ;

Relational algebra :- algebra :-

Queries in relational algebra are composed using collection of operators and each query describes step by step procedure for computing a desired answer.

Relational algebra is one of the formal query language related (associated) with relational model.

The basic operators of relational models are selection, union, projection, cross product, set difference

sid	sname	rating	age
22	Dustin	4	45.0
31	Lubber	8	55.5
58	Rustg	10	35.0

S_1 of sailors

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	Rusty	10	35.0

S_2 of sailors

sid	bid	day
22	101	10/10/96
58	103	11/12/96

R_1 of reserves

Selection :- "σ"

In order to select a rows from a relation we use the operation in relational algebra is "σ". which is used to manipulate data in single relation.

eg :- $\sigma_{rating>8}(S_2)$

sid	sname	Rating	age
58	Rustg	10	35.0

projection :- "π"

In order to project columns in relational algebra the symbol "π" is used.

it allows to manipulate data in a single relation

eg :- $\pi_{sname, rating}(S_2)$

sname	rating
Yuppy	9
Lubber	8
Guppy	5
Rustg	10

Projection operators eliminates the duplicate values from a relation.

$\pi_{age}(S_2)$

Age
35.0
55.0

In selection operator we can use different comparision operators between attributes.

$<, <=, =, \neq, >=, >$

$\pi_{sname, rating}(\sigma_{rating>8}(S_2))$

sname	rating
Yuppy	9
Rustg	10

Union operator :- R ∪ S returns a relation instance containing all tuples that occur in either relation R or 'S' or both.

R and S must be union compartable. Two relations are said to be union compartable if they satisfy the following condition.

1. If they have the same no. of fields.
2. corresponding fields taken in order from left to right must have the same domains.

note : field names are not used in identifying union compatability.

A = 1, 2, 3
B = 2, 4, 5

A ∪ B
{1, 2, 3, 4, 5}

eg :- $S_1 \cup S_2$

sid	sname	rating	Age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0
28	yuppy	9	35.0
44	Guppy	5	35.0

Intersection :- R ∩ S returns a relational instance containing all the tuple that occur in both R and S. The relation R and S must be union compatable.

$S_1 \cap S_2$

sid	sname	rating	age
31	Lubber	8	55.5
58	Rusty	10	35.0

set difference :- 'R - S' returns a relation instance containing all tuples that occur in R but not in 'S'. the relation R and S must be union compatable.

$S_1 - S_2$

sid	sname	rating	age
22	Dustin	7	45.0

cross product / cartesian product :-

'R * S' returns a relation instance whose schema contains all the fields of 'R' in the same order as a appear in 'R'. followed by all the fields of 'S' in the same order as a appear of 'S'.

cross product is also called as cartesian product which makes the concatination of tuples for a given relation

$S_1 * R_1$

sid	sname	rating	age
22	Dustin	7	45.0
22	Dustin	7	45.0
31	Lubba	8	55.5
31	Lubber	8	55.5
58	Rustg	10	35.0
58	Rustg	10	35.0

sid	bid	day
22	101	10/10/96
58	103	11/10/96
22	101	10/10/96
58	103	4/10/96
22	101	10/10/96
58	103	11/12/96

The problem with cartesian product is that it repeats the unnecessary tuples and confusion regarding column data i.e., redundancy and duplication of field names.

Rename operator : It is used to overcome the conflicts which occurs in tables S_1 and r_1 of sailors and reserves. In order to overcome this problem, we use a rename operator which is denoted as 'ρ'

$$\rho(R(\bar{F}), E)$$

The above expression takes an relational algebra expression E and returns the new field name $\bar{F}$ to a given relational table called R.

Division :-

Consider two relations A and B in which A has exactly two fields (x, y) and B has exactly one field 'y' with the same domain as in 'A'. The division operation (A/B) is a set of all 'x' values such that for every 'y' values in B there is a tuple (x, y) in A

$$A/B = \Pi_x(A) - \Pi_x((\Pi_x(A) \times B) - A)$$

Ax	y
Sno	Pno
S_1	P_1
S_1	P_2
S_1	P_3
S_1	P_4
S_2	P_1
S_2	P_2
S_3	P_2
S_4	P_2
S_4	P_4

B_1 =

Pno
P_2

B_2 =

Pno
P_2
P_4

B_3 =

Pno
P_1
P_2
P_4

A/B_1 =

Sno
S_1
S_2
S_3
S_4

A/B_2 =

Sno
S_1
S_4

A/B_3 =

Sno
S_1

Joins :-

It is one of the most useful operation in relational algebra and commonly used to combine information for two or more relations.

joins can be defined as cross product followed by selection and projection.

Note :- the result of cross product will generate more values as a result whereas join operation result value will be very less and more meaningful.

join operation is performed with following symbol

Joins are divided into three types.

1. Conditional join
2. Equi join
3. Natural join

Natural join : it is a binary operator which joins two or more relations and returns the result set of all combinations of tuple where they have equal common attribute.

Example :-

Emp

Name	id	deptname
A	100	IT
B	125	HR
C	110	Sale
D	111	IT

Dept

Dept. name	manager
sale	Y
prod	Z
IT	A

-Emp ⋈ Dept

Name	id	dept. name	Manager
C	110	Sale	Y
A	120	IT	A
D	111	IT	A

Sailors

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves

sid	Bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats

Bid	Bname	color
101	Interlake	blue
102	Interlake	red
103	clipper	green
104	Marine	red

Find the names of sailors who have reserved the boat 103

$\Pi_{sname}((\sigma_{bid=103} Reserves) \bowtie sailors)$

sname
Dustin
Lubber
Horatio

Find the names of a sailors who have reserved a red boat

$\Pi_{sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie sailors)$

sname :- Dustin, lubber, Horatio

Find the colours of boats reserved by Lubber

$$\Pi_{color}\left(\left(\sigma_{sname = Lubber}\ sailors\right) \bowtie Reserves \bowtie Boats\right)$$

Color
red
green

Find the names of a sailors who have reserved atleast one boat.

$$\Pi_{sname}\left(sailors \bowtie Reserves\right)$$

sname
Dustin
Lubber
Horatio

Find the names of sailors who have reserved a red or green boat.

$$\Pi_{sname}\left(\left(\sigma_{color = red \cup green}\ Boats\right) \bowtie reserves \bowtie sailors\right)$$

Find the names of sailor who have reserved red and green boat.

$$\Pi_{sname}$$

Relational calculus :

Relational algebra is procedural whereas relational calculus is non-procedural.

we have two types of relational calculus.

1. Domain relational calculus (DRC)
2. Tuple relational calculus (TRC)

Tuple relational calculus (TRC) :- it is a variable that takes a tuples of a particular relational schemas as values.

A TRC query has the form $\{T \mid P(T)\}$ where T is a tuple variable.

$P(T)$ is a formulae that describes the tuple variable T.

Syntax for TRC :

Let Rel be a relational name, R and S be a tuple variables of attributes a and b.

Let "op" denotes an operator in the set $\{<, >, =, \leq, \geq, \neq\}$

an atomic formulae is one of the following :

1. $R \in Rel$
2. $R.a \ op \ S.b$
   (tuple, attribute, tuple, attribute)
3. $R.a \ op \ constant$
   (or)
   $constant \ op \ R.a$

A formulae is recursively defined to be one of the following where P and q are themselves formulaes and $P(R)$ denotes a formulae in which the tuple variable are appear

1. Any atomic formula

2. $\neg P$, $P \wedge q$, $P \vee q$, $P \Rightarrow q$

3. $\exists R(P(R))$ where R is a tuple variable.

4. $\forall R(P(R))$ where R is a tuple variable.

Find all sailors with the rating above 7 ?

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

Find the names and ages of sailors whose rating is above 7 ?

$\{P \mid \exists S \in \text{sailors}(S.\text{rating} > 7 \wedge P.\text{sname} = S.\text{sname} \wedge P.\text{age} = S.\text{age})\}$

Find the names of sailors who have reserved boat id = 103

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

Domain relational calculus (DRC):- A Domain Variable is a variable that ranges over the values in the domain of some attributes.

A DRC query has the form as follows:

$\{(\langle x_1, x_2, \ldots x_n \rangle) \mid P(\langle x_1, x_2, x_3 \ldots x_n \rangle)\}$ where each x_i is either a domain variable or a constant.

where 'i' is between 1 to n.

Syntax for DRC queries:

Let 'op' denotes an operator in $\{(\langle x_1, x_2 \ldots x_n \rangle) \mid P(\langle x_1, x_2 \ldots x_n \rangle)\}$

Let 'x' and 'y' denotes the domain variables. an atomic formulae in DRC will be one of the following.

1. $X \in \text{Rel}$
2. X op Y
3. X op Constant

   or

   Constant op X

When a formulae is recursively defined, we consider p & q as formulae themselves.

P(x) denotes a formulae in which the domain Variable 'x' appears.

1. Any atomic formula
2. ¬R, P∧q, P∨q, P⇒q
3. ∃x (P(x)) where x is a domain Variable.
4. ∃ x (P(x)) where x is domain Variable.

1. Find all the sailors with rating above 7 ?

$\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in \text{sailors} \wedge T > 7 \}$

Aggregate functions :-

In order to perform some computations and

There are 5 different aggregate functions or operators where SQL supports.

Aggregate operators can be applied on any column in a given relational schema.

1) count ([Distinct], A)
   - Distinct → unique
   - A → column name
2) sum ([Distinct], A)
3) AVG ([Distinct], A)
4) Min (A)
5) Max (A)

Syntax :-

Select aggregate operator (columnname) from tablename ;

Select aggregate operator (column name) from tablename where condition ;

example : Select count (rating) from sailors ;

rating
10

Select sum (rating) from sailors ;

rating
66

Select avg (rating) from sailors ;

rating
6.6

Select min (rating) from sailors ;

rating
1

Select max (rating) from sailors ;

rating
10

UNIT-2

INTRODUCTION TO SCHEMA REFINEMENT AND NORMAL FORMS

- Storing the same information in more than one place with in a database is called "Redundancy".

Problems caused by Redundancy:-

(1) Redundant storage:- same information is stored repeated

(2) Update anomalies:- If one copy of such repeated data is updated, an inconsistency is created, unless all copies are similarly updated.

(3) Insertion anomalies:-
It may not possible to store a certain information unless some other, unrelated, information is stored as well.

(4) Deletion anomalies:-
It may not be possible to delete certain information with out losing some other information, unrelated information also.

Consider a relation "Hourly_Emp." entity with attributes.

Hourly-Emps (ssn, name, lot, rating, hourly_wages, hours_worked)

→ ssn (Social Security number) is the "key attribute"

→ Suppose, that, the "hourly_wages" attribute is determined by "rating" value.

∴ This constraint is an example of functional dependency (ie) the value of "Hourly_wages" are depends upon the "rating".

Hourly_Emps:

ssn	name	lot	rating	hourly_wages	hours_worked
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

An Instance of Hourly_Emps Relation.

* If we observe the above table, the same value appears in the rating column of two tuples, the Integrity constraint tells us the same value must appear in the "Hourly_wages" column as well.

This redundancy has the same negative consequence as before:

(1) Redundant storage:-
The rating value corresponds to Hourly_wages 10, and this association is repeated 3 times.

(2) Update anomalies:-
The Hourly_wages in the first tuple could be updated without making a similar change in the second tuple.

(3) Insertion anomalies:-
We can not insert a tuple for an employee unless we know the hourly_wages for the employee rating value.

(4) Deletion anomalies:-
If we delete all tuples with a given rating value (Eg: We delete the tuples for Smethurst and Guldu) we lose the association between that rating value and hourly_wages value.

Decomposition :-

- A "decomposition of a relation schema" R consists of replacing the relation schema by two (or) more relation schemas that each contain a subset of the attributes of R and " together includes all attributes in R.

(ie) When a functional dependency occurs; the particular schema is decomposed into 2 (or) more schemas.

We can decompose Hourly_Emps into two relations.

Hourly_Emps2 (<u>ssn</u>, name, lot, rating, hours_worked)

Wages (<u>rating</u>, hourly_wages)

The Instances of above relation.

ssn	name	lot	rating	hourly_worked
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan.	35	8	40.

rating	hourly_wages
8	10
5	7

Instances of Hourly_Emps2 & Wages.

Note:- We simply/easily record the hourly_wages for any rating simply by adding a tuple to Wages, even if no employee with the rating appears in the current instance of Hourly_Emps.

changing the wage associated with rating involves updating a single wage tuple.

This is more efficient than updating several tuples, as it eliminates the potential for inconsistency.

Problems Related to Decomposition :-

Improper decomposition of relational schema can create problems.

Two important questions must be asked repeatedly.

(1) Do we need to decompose a relation?

- Several normal forms have been proposed for relations.

These normal forms of a given relational schema.

can decide whether (or) not to decompose.

(2) What problems (if any) does a given decomposition cause?

Decomposition has 2 properties.

(1) Lossless join property

(2) Dependency preservation property.

(1) The lossless-join property enables us to recover any instance of the decomposed relation from corresponding instances of the smaller relation.

(2) The Dependency-preservation property enables us to enforce any constraint on the original relation by simply enforcing some constraints on each of the smaller relations.

That is we need not to perform joins of the smaller relations to check whether a constraint on the original relation is violated.

FUNCTIONAL DEPENDENCIES:-

A Functional Dependency (FD) is a constraint between two set of attributes from the database.

An FD is denoted by $X \longrightarrow Y$.

We can read it as "Y is functionally dependent on X".

(or)

'X' is functionally determines Y.

The left hand side is called "determinents" and the right hand side is called "dependents". Here 'X' is determinent set, 'Y' is dependent set of attributes

→ Let R be a relation schema.

Let 'α' and 'β' are be non-empty sets of attributes in R.

We say that, for any legal instance $r(R)$ (or) r of R satisfies the functional dependency $\alpha \longrightarrow \beta$ on a relation for every pair of tuples t_1 & t_2 in "r".

If $t1.\alpha = t2.\alpha \Rightarrow t1.\beta = t2.\beta$, $\forall\, t_1, t_2$ in R. and $\alpha, \beta \subseteq R$.

(i.e) If their α part is equal then β part must also equa

Eg:- $r(R)$

	α	β	
t_1	x	y	
t_2	x	y.	

Let us take an instance shown below.

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

A "Legal instance" of a relation must satisfy all specified Integrity constraints, including all specified Functional Dependencies.

In the above instance,

→ The FD on A → B is not hold, because for the first two tuples 'a1' has 'b1' values, in the third tuple a1 has b2. So, the rule of FD is violated here similarly, B → C, B → D, C → D are also not hold.

→ But AB → C is holding the FD. For every AB value there is only one 'C' value

	α → β
t_1	a1 b1 → c1
t_2	a1 b1 → c1
t_3	a1 b2 → c2
t_4	a2 b1 → c3

So, we say that there is an FD between AB → C.

→ A primary key constraint is a special case of an FD. Let X → Y is a FD, & The attributes in the primary Key (or) Key play the role of X, and the set of all

attributes in the relation. plays the role of Y.

Let us take the example on Airline Reservation.

Flight(flight_no, city_arrival, city-departure, Plane_type)

Seats_free(flight_no, Jdate, seats_avl)

FD's ⟹: flight_no ⟶ city_arrival

flight_no ⟶ city_departure

flight_no ⟶ Plane_type

flight_no, Jdate ⟶ seats_avl.

"flight_no" alone can't determine the no. of seats available. Because a single flight can be traveled to and from one city to another many time. Each seat can be reserved, and there is a chance reserving same seat in different dates and different source and destinations.

→ So, flight_no and Jdate both can gives the no. of seats available on that flight on the particular date.

Closure of a set of FDS:-

The set of FDs implied by a given set F of FDs is called the "Closure of F".

It is denoted by F^+.

(or)

The set of FDs which are logically follow from F. is called "Closure of F". (F^+).

→ Note: How can we infer (or) compute the closure of given set F of FD's.

The answer is by following the 3 Armstrong Axioms rules.

ARMSTRONG AXIOMS:-

These axioms are applied repeatedly to infer all FDs implied by a set of 'F' of FDs.

Let 'X, Y, Z' are sets of attributes over a relation schema 'R'.

(1) Reflexivity:-

If $X \supseteq Y$ then $X \longrightarrow Y$.

(2) Augmentation:-

If $X \longrightarrow Y$ holds, then $XZ \longrightarrow YZ$ also holds for any "Z".

(3) Transitivity:-

If $X \longrightarrow Y$ holds,

$Y \longrightarrow Z$ holds,

then $\boxed{X \longrightarrow Z}$ also holds.

Theorem-1:- Armstrong Axioms are SOUND and COMPLETE

(1) Armstrong axioms are sound, in that they generate only FDs in F^+ when applied to a set F of FDs. (ie) any new FD derived by the axioms is indeed a member of the closure.

(2) They are also complete, in that repeated application of these rules will generate all FDs in the closure of F^+.

Additional rules of closure :-

4. **UNION :-**

If $X \rightarrow Y$ holds, and $X \rightarrow Z$ holds, then

"$X \rightarrow YZ$" also holds.

5. **Decomposition :-**

If $X \rightarrow YZ$ holds then

$X \rightarrow Y$ holds

$X \rightarrow Z$ also holds.

6. **Pseudo Transitivity Rule**

If $\alpha \rightarrow \beta$ holds &

$\gamma\beta \rightarrow \delta$ holds then

$\alpha\gamma \rightarrow \delta$ holds.

Different Types of Keys :-

Super Key :-

It is a set of all attributes (or) some attributes that are used to identify (or) determine a tuple in schema.

(or)

A super key can be defined as a set of attributes of a relation schema upon which all attributes of the schema are "**Functionally Dependent**".

→ The set of all attributes is a "**trivial superkey**".

Let us take the schema:

Emp (ssn, name, salary, dot)

The List of superkeys are:

→ ssn

→ ssn, name

→ ssn, name, salary

→ ssn, name, salary, dot

No two tuples can determine the same super key.

A super key is a set of one (or) more attributes that taken collectively, allows us to identify uniquely in a the relation.

A superkey may contain extraneous attributes.

Let us take student schema:

Student (ID, name, dep_name, tot_credits).

In the above schema ID can alone identify all other attributes. So, ID is a superkey.

'Name' did not identifies all tuples in a relation, so it is not a super key.

Pair <ID, Name> both can identify the tuples in a student relation. So, <ID, name> can be a superkey.

(From H.F. KORTH Reference Book).

Candidate Key:-

Minimum set of attributes that are require to identified a tuples in a relation is known as "candidate key".

A candidate key is a subset of super key.

It is also called Minimal super key.

(i.e) If we remove any attribute from candidate key it is not possible to identify the tuples in a relation

→ Candidate keys are often referred to as "primary keys".

In students relation 'ID' is the candidate key.

3.6

In the sailors relation, Boats relation, of the candidate Keys are: 1) sid is a candidate key for sailors

2) bid is a candidate key for Boats.

In Reserves the combination of {sid, bid} are the candidate key.

Either sid (or) bid can alone is not enough to identify the "day". we need both of them to identify the entire tuple.

So, <sid, bid> is the minimal superkey (or) candidate key (or) primary key.

Prime Attribute :-

An attribute which is a part of any candidate key is called prime attribute.

In the above example: of Reserves schema: <sid, bid are prime attributes.

Non-prime attribute :-

An an attribute (or) set of attributes which is not a part of any candidate key called "Non-prime attributes".

Sailors → prime : sid

Non-prime: sname, rating, age

Reserves → prime: sid, bid

Non-prime: Day.

Let us consider a schema $R = \{A, B, C, G, H, I\}$ and the set F of FDs, and find closure of F (F^+).

$\{A \to B,\ A \to C,\ CG \to H,\ CG \to I,\ B \to H\}$

From Transitivity :-

$A \to B$

$B \to H$ hold then "$A \to H$" holds.

From Union rule :-

Since $CG \to H$ & $CG \to I$ holds, then "$CG \to HI$" holds

From pseudo transitivity :-

$A \to C$

$CG \to H$ then $AG \to I$ holds.

Ex.2 :- Let us take $R = \{A, B, C, D, E, F\}$

set of F of FDS.

$A \to B$

$A \to C$

$CD \to E$

$CD \to F$

$B \to E$

⇒ From $A \to B$, $B \to E$, then we get $A \to E$

⇒ From $A \to C$, $CD \to E$ then $AD \to E$ holds.

⇒ From $A \to C$, $CD \to F$ then $AD \to F$ holds.

⇒ From $CD \to E$, $CD \to F$ then $CD \to EF$ and $AD \to EF$ also holds.

In this way we use armstrong axioms to compute F^+. In this procedure, when a FD is added to F^+, it maybe

already present, and in that case there is no change to F^+.

Closure of Attribute set:- $(\alpha)^+_F$

Let α be set of attributes, the set of all attributes functionally determined by α under a set F of functional dependencies is called as the "closure of α under F". We denote it by α^+. (or) $(\alpha)^+_F$

Let us take a set of FD's. (F).

$A \rightarrow BC$

$AC \rightarrow D$

$D \rightarrow B$

$AB \rightarrow D$

If we apply Armstrong Axioms on F.

$A \rightarrow B$
$A \rightarrow C$
] → From Decomposition of $A \rightarrow BC$

$D \rightarrow B$

$AC \rightarrow D$

$AB \rightarrow D$

$AB \rightarrow D$, from $A \rightarrow B$, If we replace B in place of A,

then $AA \rightarrow D$ ⇒

$\Rightarrow A \rightarrow D$ is derived.

$AC \rightarrow D$, from $A \rightarrow C$, we replace C with A

$AA \rightarrow D$

"$A \rightarrow D$" is derived Again.

(i.e) an attribute A can determine B, C, D. So, the attribute closure of A ⇒ $A^+ = \{A, B, C, D\}$

B is not deriving any attribute, from reflexivity property, B can derive itself.

So, Attribute closure of B

(i.e) $B^+ = \{B\}$.

Like wise :- $C^+ = \{C\}$

$D = \{B, D\}$

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

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

Algorithm to compute F^+ :-

$F^+ = F$.

repeat

for each FD f in F^+

apply reflexivity and augmentation rules on f,

add the resulting FD to F^+.

for each pair of FDs f1 and f2 in F^+

if f1 and f2 can be combined using transitivity

add the resulting FD to F^+.

Untill F^+ does not change any further.

Algorithm to compute α^+ (Attribute closure of α).

Let α be set of attributes.

result := α

repeat

for each FD $\beta \rightarrow \gamma$ in F do.

begin.

if $\beta \subseteq$ result then result := result $\cup$ γ;

end

until (result does not change).

→ (i.e) suppose, $A \rightarrow B$

$A^+ = \{A, B\}$

$A \rightarrow B$, $A \rightarrow C$

$A^+ = \{A, B, C\} \Rightarrow A^+ = \{A, B\} \cup \{C\}$.

Cover of a set of FDs:-

Let F and G be two sets of FDs on Relational schema R. then

F is a cover of G if "$F^+ = G^+$."

(i.e) the closure F is Equivalent to closure of G.

∴ The set of FDs are logically implied by F are the same as set of FD that are logically implied by G.

Minimal Cover / Canonical cover (F_c):-

A cover is said to be minimal cover/ canonical cover, if it has no redundant terms. (or) extraneous attributes.

Extraneous attribute:-

an attribute of a FD is said to be extraneous. If we can remove it without changing the closure of the set of FDs.

- Consider a set F of FDs and the FD $\alpha \rightarrow \beta$ in F. (∵ α, β are set of attributes).

(1) An attribute A is extraneous in α if $A \in \alpha$, and

F is logically implies $(F - \{\alpha \to \beta\}) \cup \{(\alpha - A) \to \beta\}$.

(2) An attribute A is extraneous in β if $A \in \beta$, and the set of FDs $(F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$ logically implies F.

Ex:-1:

Suppose, we have $AB \to C$, and $A \to C$ in F.

Then B is extraneous in "$AB \to C$",

Because A alone can determine 'C'. So, B is extraneous. (or) redundant.

Ex-2:

consider a relational schema $R = \{A, B, C, D\}$, and the set of FDs.

$A \to BC$

$AC \to D$

$D \to B$.

$AB \to D$.

From, $A \to BC$

$A \to B$.

$A \to C$ are derived.

If $A \to C$ is a FD, then $AC \to D$ will be $AA \to D$

(i.e) $A \to D$. is derived.

From,

From $A \to BC$, $A \to B$ is derived, If we replace it in $AB \to D$, then $AA \to D$ (i.e) $A \to D$.

So, from $\left.\begin{array}{l} A \to BC \\ AC \to D \\ AB \to D \end{array}\right\}$ we can derive $A \to B$, $A \to C$ and $A \to D$.

So, now the FDs are

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

$D \to B$,

$AC \to D$.

$AB \to D$.

$\therefore$ $A \to D$, and $D \to B$, by transitivity "$A \to B$", which is a redundant term, so, $A \to B$ will be eliminated.

$\therefore$ $AC \to D$, $AB \to D$ derives $A \to D$, so, both the FDs are also redundant. we eliminate them also.

Hence the Final FDs are.

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

(i.e) $\boxed{\begin{array}{l} A \to CD \\ D \to B. \end{array}}$ is the minimal (or) canonical cover of F.

(i.e) $(F_c) = \begin{array}{l} A \to CD \\ D \to B. \end{array}$

Use of Minimal (or) Canonical Cover:-

Suppose that we have a set of FDs F on a relation schema. Whenever a user performs an update on a relation, the database must ensure that the update does not violate any FDs, that is all the FDs in F are satisfied in the new database state.

(*) The System must roll back the update if it violates any FDs in the set F.

(*) We can reduce the effort spent in checking for.

violations by testing a simplified set of FDs that has the same closure as the given set.

Any database that satisfies the simplified set of FDs also satisfies the original set, and vice versa, since the two sets have the same closure.

∴ However the simplified set is easier to test.

So, Inorder to avoid the checking each and every FD, if we find canonical cover / minimal cover of a FD, it is easy and less time taking process in testing of FD.

Trivial Functional Dependency:-

Any FD $\alpha \rightarrow \beta$ is trivial, if β is a subset of α.

(∴ α, β are set of attributes).

(i.e) $\beta \subseteq \alpha$.

(or)

Any FD $\alpha \rightarrow \beta$ holds, where α is a superset of β.

(i.e) $\alpha \supseteq \beta$.

Algorithm for Finding Minimal / canonical cover:-

$F_C = F$

Repeat

→ Use the union rule to replace any dependencies in F_C of the form $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1\beta_2$

→ Find a FD $\alpha \rightarrow \beta$ in F_C with an extraneous attribute either in α (or) in β.

/Note: The test for extraneous attributes is done using F_C not 'F'/

If an extraneous attribute is found, delete it from

$\alpha \rightarrow \beta$ in F_c.

until (F_c does not change).

∴ (i.e). First, we are checking for any redundancy, then if any redundant FD are there just remove it. until there is no redundant FD is found.

*Loss-less Decomposition:-

Let $r(R)$ be an instance of Relation schema R.

Let F be a set of FDs on $r(R)$.

Let R_1, R_2 form a decomposition of R.

→ We say that the decomposition is a lossless decomposition if there is no loss of information by replacing $r(R)$ with two relation schemas $r_1(R_1)$ and $r_2(R_2)$.

→ More precisely, we say the decomposition is lossless if, for all legal database instances. (i.e), the database instances that satisfied the specified FDs, on relation r, contains the same set of tuples as the result of the following Query:

```
Select *
from ( Select R1 from r)
        NATURAL JOIN
        (Select R2 from r).
```

(i.e) $\boxed{\Pi_{R1}(r) \bowtie \Pi_{R_2}(r) = r}$

If we project r onto R_1 & R_2, and compute the natural join of the projection results, we get back exactly r.

(or)

Let $R = \{R_1, R_2, \ldots R_n\}$ be a decomposition of Relation 'D'.

Then "D" is said to be lossless decomposition,

If for each $r(R)$, we have.

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie \cdots \bowtie \Pi_{R_n}(r).$$

Ex:- Let us take the relation instance 'r', with attributes S, P, D.

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
S2	P2	D2
S3	P1	D3
S1	P1	D3
S3	P1	D1

$\Pi_{SP}(r) \bowtie \Pi_{PD}(r)$

The relation instance r has 3 tuples.

with the decomposed relations $\Pi_{SP}(r)$ and $\Pi_{PD}(r)$.

After joining $\Pi_{SP}(r) \bowtie \Pi_{PD}(r)$, we lose some information. (i.e) these are some extra tuples are there after joining.

The result of $\Pi_{SP}(r) \bowtie \Pi_{PD}(r)$ is not the original relation instance "r". Because, it has more tuples than the original relation 'r'. So, it is a lossy decomposition

(*) All the decompositions, which are used to eliminate redundancy must be Lossless.

Let R be a relation and F be a set of FDs that hold over R.

The decomposition of R into relations with attribute sets R_1 and R_2 is lossless, iff F^+ contains the FD.

either • $R_1 \cap R_2 \rightarrow R_1$

or • $R_1 \cap R_2 \rightarrow R_2$.

In otherwords, if $R_1 \cap R_2$ form a superkey of either R_1 or R_2, the decomposition of R is a lossless decomposition.

The Example for Lossless Decomposition is:

Hourly_Emp is decomposed into Hourly_Emps2, and "rating and wages" table.

ssn	name	lot	rating	hours_worked
123	Attoshoo	48	8	40
231	Smiley	22	8	30
131	Smethurst	35	5	30
434	Guldu	35	5	32
612	Madayan	35	8	40

rating	wages
8	10
5	7

Instance of Hourly_Emps2 and Rating_wages.

After joining these two tables, we got the original Hourly_Emp table. Hence it is the Lossless join decomposition.

Dependency Preservation:-

- Let R be a relation.

  $R = \{R_1, R_2, \ldots R_n\}$ be a set of decompositions of R.

- Let F be the set of FDs that holds on R.

  Let F_i be the set of FDs in F^+ that holds on R_i,

  then the decomposition is said to be "Dependency preserving if.

$$F^+ = F_1 \cup F_2 \cup F_3 \ldots \ldots \cup F_n$$

(i.e) If we take the dependencies $F_1, F_2 \ldots F_n$ and compute the closure of their 'union', we get back all dependencies in the closure of F.

∴ We need enforce only the dependencies in $F_1, F_2 \ldots F_n$, all the FDs in F^+ are sure to be satisfied.

∴ To enforce F_1, we need to examine only R_1,

To enforce F_2, we need to examine only R_2.

-- etc.

Example:-

It is much needed to compute the closure F^+ while computing the projection of F.

Let 'R' be a relation with attributes A B C.

which is decomposed into relations with attributes

AB and BC.

The set of FDs 'F' over R is $A \to B$, $B \to C$, $C \to A$.

- $A \rightarrow B$ is in F_{AB},

  $B \rightarrow C$ is in F_{BC},

But this decomposition is not in dependency preserving

Here $C \rightarrow A$ is ignored.

∴ The closure of F contains all dependencies in F, plus

$A \rightarrow C$, $B \rightarrow A$ and $C \rightarrow B$.

Consequently, F_{AB} contains $A \rightarrow B$ and $B \rightarrow A$.

F_{BC} contains $B \rightarrow C$ and $C \rightarrow B$.

∴ Thus $F_{AB} \cup F_{BC}$, contains.

$A \rightarrow B$, $B \rightarrow C$, $B \rightarrow A$ and $C \rightarrow B$.

∴ From $C \rightarrow B$, $B \rightarrow A$, by transitivity $C \rightarrow A$ is derived

∴ Hence, the closure of dependencies $F_{AB} \cup F_{BC}$ is now includes $C \rightarrow A$ also.

∴ Thus, the decomposition preserves the dependency

$C \rightarrow A$

SOME IMPORTANT TERMS:-

→

SPURIOUS TUPLES:-

The effect of Lossy decomposition is that when R1 & R2 are joined, some extra tuples are generated, these extra tuples are called as "spurious tuples".

Ex:- $\Pi_{SP}(R) \bowtie \Pi_{PD}(R)$ relation, <S1 P1 d3> and <S3 P1 d1> are not the tuples that are in original relation. These two tuples are called "Spurious tuples".

Full functional Dependency:—

In a FD 'F', y is fully FD on 'X',

(i.e) $X \rightarrow Y$ in R gives full FD, If removal of any attribute from X makes the dependency does not hold anymore on 'R'.

(∴ where X, Y are set of attributes, $\{X, Y\} \in R$).

(or)

If $\alpha \rightarrow \beta$ is Fully Functionally Dependent, then subset of attribute 'α' can not imply 'β', then it is said to be fully Functionally Dependent.

Ex:- $\{sid, bid\} \longrightarrow day$

∴ 'sid' alone does not imply "day".

bid alone does not imply "day".

Hence "day" is fully functionally dependent on {sid, bid}

Partial Functional Dependency:—

If $\alpha \rightarrow \beta$ is a FD in R, α, β are set of attributes.

If we remove any attribute from 'α' (or) the subset of α can determine β; (i.e) the functional Dependency is still holds good, then the FD is said to be

"partial Function dependency".

$\{sid, sname\} \longrightarrow \{rating, age\}$ but

$\{sid\} \longrightarrow \{rating, age\}$, sid, sname can determines. rating, age, but if we remove sname, sid still determine "rating and age".

NORMAL FORMS.

- "Codd" proposed the relational Database Model. He also introduced the concept of Normal forms.
- Normal forms tries to prevent the redundancy in relation schema.
- Normal forms provides guidelines to how to decompose the current schema (or) relation into smaller relation
- As we mentioned earlier, redundancy can leads to different types of anamolies such insertion, deletion, updation anomolies, which leads to inconsistency.
- In order to avoid inconsistency and redundancy problems, the normal forms provide several guidelines

Types of Normal Forms:-

First Normal Form (1NF)

Second Normal Form (2NF)

Third Normal Form (3NF)

Boyce-Codd Normal Form (BCNF),

- All these normal forms are graded (i.e) If a relation is in BCNF, is also in 3NF, every relation in 3NF is also in 2NF and every relation in 2NF is also in 1NF.

→ There other types of Normal forms also there, 4NF and 5NF.

These Normal forms uses the concept of Functional Dependencies.

First Normal Form (1NF):-

A relation is in First Normal form, if every field contains atomic values (ie) No nested structures no lists, no sets.

(or).

The relation schema is in 1NF, if every attribute of 'R' takes only single (or) atomic value.

"Atomic value" means, the values which ~~can~~ be not further ~~divided~~.

Examples:-

Ex-1:-

SId	name	age	dependent
22	Dustin	45	Adam, smith.
29	Brutus	33	Jhon, Joseph
31	Lubber	55	Lindsey

} Multiple values.

1NF:-

SId	name	age	dependent
22	Dustin	45	Adam
22	Dustin	45	smith
29	Brutus	33	Jhon
29	Brutus	33	Joseph.
31	Lubber	55	Lindsey

} single values | atomic values.

Ex-2:-

Name		Address.		
Title	Fullname	Street	city	state.

} Nested structures.

1NF :-

Title	Fullname	Street	City	State

No nested structure

Second NORMAL FORM — (2NF):

1) A relation R is in 2NF if every non-prime attribute of R is fully functionally dependent on each "relation key" (i.e. candidate key).

2) And the relation schema must be in 1NF.

(i.e) No partial FDs are holding on R.

Ex:- Let us take Reserves schema:

Reserves (sid, bid, day).

∴ sid, bid ⟶ day

On a particular day a sailor can reserve a boat.

Ex:-2 :- Loan_schema (c_name, Loan_no, amount)

cid, Loan_no ⟶ amount

∴ The customer can take several loans, but for a particular loan_no & particular customer_id (cid) the amount must be fixed.

candidate key :- {cid, loan_no}

Here cid and Loan_no combinely forms the candidate key, neither of c_name (or) loan_no is not candidate key.

Non_prime attribute — {amount}

Here amount is Fully Functionally dependent on.

{card, loan_no}.

Prime Attribute:- An attribute, which is a part of a candidate Key.

Non_Prime attribute:- An attribute which is not a part of candidate key

Ex3:-

courses(course_no, title, loc, time)

FDs:- course_no → title

course_no, time → Loc

When course_no and time is same, the 'loc' is also same

Candidate Key:- {course_no, time}

. Closure of course_no, time gives all the attributes.

Non_prime attribute:- {Loc, title}

∴ The property of 2NF is every non_prime attribute is Fully FD on relational Key (i.e candidate key).

"Loc" is FFD on {course_no, time}.

But "title" only depends on "course_no"

"course_no" is a subset of candidate key.

∴ 'course_no' is not FFD on candidate key.

So, this is not in 2NF,

∴ Hence we have to Break the above schema into two relations.

∴ Because, it shows the redundancy in the form of FD, course_no → title.

Hence, the decomposed relations are:

→ course1 (course_no, title)

→ course2 (course_no, time, loc)

Now the above relation is in 2NF.

Ex-4:-

Banker (Br_code, c_name, agent).

FD's :- 1) agent → Br_code

2) c_name, br_code → agent.

→ "An agent can only work in one Bank." can be a statement of 1st FD agent → Br_Code.

→ "c_name, br_code → agent", In a particular bank and a particular customer can have only one agent.

candidate keys → {c_name, br_code}
{c_name, agent}.

Non_prime attributes :- No Non_prime attributes

(Note: If there are no_non prime attributes then by default it is in 2NF).

∴ So, The above Relation is in "2NF" But there is a redundancy still appears.

(i.e) Whenever a customer approach an agent he has to mention br_code. For several customers having

having same agent, we have to write the same branch_code for several times.

Third Normal Form (3NF) :-

Def 1:- A relation is in 3NF, it must be in 2NF & No non_prime attribute is functionally dependent on other non_prime attributes.

(i.e) The relation R does not include any transitive dependeneres.

Ex:1 :- Banker (Br_code, c_name, agent).

As we see earlier, it has No non_prime attributes. So, it is in 2NF,

∴ Hence there are no non prime attributes, 3NF is also satisfied. and there are no transitive dependeneres as well.

But there is still exists a redundancy problem, which is identified and solved by BCNF.

~~11/4/24~~ (OR)

ALTERNATE DEFINITION OF 3NF :-

Def-2:- Let R be a relation schema,
F be the set of FDs given to hold over R,
X be a subset of attributes of R, and
A be an attribute of R,
" R is in 3NF" if, for every FD $X \rightarrow A$, one of the following statement is true:

- $A \in X$; (i.e) it is a trivial Functional Dependency (or
- X is a super key (or)
- A is a part of some key for R.

3-NF

Ex-2 :- Let us take the example of Hourly_Emp Relation

SSN	name	lot	rating	Hourly_wages	hourly_worked
123	Attichoo	48	8	10	40
231	smiley	22	8	10	30
131	Smethurst	35	5	7	30
434	Guldu	35	5	7	32
612	Madayan	35	8	10	40

Hourly_Emp Relation Instance

In the above example the only key is 'ssn', which alone determines all other attributes.

But there is a FD (ie) "Rating → Hourly_wages." which is a transitive dependency.

(ie) ssn → ratings → Hourly_wages.

"The consequence is that we can not record the fact that employee "ssn" has "rating" without knowing Hourly_wages for that rating.

"This condition leads to insertion, deletion and update anomalies. So, Inorder avoid these anomalies, we have to split the table in to 2 relations, to get the relation into 3NF.

Hourly_Emp2 (ssn, name, lot, rating, hours_worked)

Rating_wages (rating, hourly_wages).

∴ Now, in the above 2 relations, don't have any transitive dependency, and

there are no non_prime attributes, which are depends on other non_prime attributes, it satisfies

the property of 2NF, each attribute value is taking single values, so, it is in 1NF.

∴ Hence the above 2 relations Hourly_Emp2, and Rating_wages are in 3NF.

Boyce-Codd Normal Form (BCNF):- (3.5 NF)

Let R be a relation schema,

F' be the set of FDs given to hold over R,

X be the subset of Attributes of R, &

A be an attribute of R.

'R' is in BCNF if, for every FD $X \to A$,

one of the statements is true:

- $A \in X$; that is it is a trivial FD.

  (ie) $X \supseteq A$.

- X is a superkey (ie) $X \to R$.

  ($X \to R$ means, X may give R).

A relation which is in BCNF is definitely in 3NF

Ex: 1:- Let us consider supplied_by schema:

: Supplied_by (sup_id, item, price, gift_item)

FDs: sup_id, item ⟶ price

price ⟶ gift_item.

The FD: "sup_id, item ⟶ price" describes particular sup_id and item has a fixed price.

The FD: "price ⟶ gift_item" describes a particular worth has some gift_items, such as for purchase of

40,000 Rs/- , a 400 rupees Doll is free gift.

candidate Keys: { sup_id , Item }

Non-prime attributes : { price, gift_item }

(ie) { sup_id, item } is a superkey.

∴ The above relation is not satisfying any of 2NF, 3NF, BCNF.

2NF :- The Non-prime attribute gift_Item is depends on other non-prime attribute "price".

3NF :- It violates the property of transitive dependency

price ⟶ gift_item

{sup_id, item} ⟶ price

BCNF :- "price" is not a superkey, so, it is not in BCNF. also.

∴ So, now we have split supplied_by schema into;

supplied_by1 (sup_id, item, price)

supplied_by2 (price, gift_item)

Now superkeys : 1) { sup_id, item }

2) { price }

The above 2 relations satisfies all the properties of 2NF, 3NF, and BCNF.

Now price is a superkey, which is in BCNF.

∴ There are no transitive dependencies, so the 2 relations are in 3NF.

Hence, the above 2 relations satisfies,

(i) BCNF, 3NF, 2NF and 1NF.

(ii) also, they are loss less decomposition

(iii) also, they are dependency preserving.

Comparison (or) Relation between 3NF and BCNF:-

Let us look at the similarity (or) relation between 3NF and BCNF from the definition.

→ Let "R" be a relation schema,

"F" be a set of FDs to hold over R,

"X" be a subset of attributes of R.

"A" be an attribute of R.

If for every FD "$X \longrightarrow A$" in F,

(1) If R is in 3NF, one of the following to be true

- $A \in X$: (i.e) $X \supseteq A$, it is a trivial FD. (or)
- X is a superkey (or)
- A is a part of some key for R.

(2) If R is in BCNF,

- $A \in X$: (i.e) $X \supseteq A$, it is a trivial FD (or)
- X is a superkey.

The definition of 3NF is similar to BCNF, with the only difference is there is a third condition in 3NF, which is not present in BCNF.

(*) $X \supseteq A$ means, A is fully functionally dependent on X.

X is a superkey, that gives this relation is in 2NF.

If the first two conditions are not satisfied, we look for another condition, (i.e).

"A is a part of some key for R".

that means, attribute A must be a part of candidate key.

(i.e). $X \rightarrow A$, A can't be a non-prime attribute.

→ The first two conditions (i.e) trivial FD and X is a superkey is a direct way of preventing redundancy. Hence, "BCNF" ensures that there is no redundancy in Relation schema.

→ But the 3rd condition leads some redundancy. Hence a relation is in 3NF, there is a chance of redundancy, But the relation "R" is in BCNF, there is no chance of redundancy.

Note:-

A good Database Decomposition must have the given properties.

(1) No Redundancy

(2) Lossless join Decomposition

(3) Dependency preserving Decomposition.

But, it is not always guaranteed that a relation can have all the 3 properties.

• Lossless join property is a necessary condition, we must have to ensure that, a relation must be decomposed as lossless join only.

But, the property of redundancy and Dependency preservation is not always hold.

- We have trade of between "property of Redundancy and "Dependency preservation"

(i.e), We can get either

(1) 3NF, Lossless join, Dependency preservation

(i.e) we may have some redundancy

(or) (2) BCNF, Lossless join.

(i.e) BCNF is a direct way to remove redundancy.

(*) BCNF, Lossless join, and Dependency preservation can not be always guaranteed.

→ Example :-

Let us take the relation schemas, course, teachers, student.

courses (course_no, title, description, loc, time)

teachers (emp_id, ename, salary, course_no, time, loc)

student (Rollno, sname, course_no, grade)

FDs :-

(1) course_no ⟶ title, description

(2) course_no, time ⟶ loc.

(3) emp_id ⟶ ename, salary.

(4) emp_id, time ⟶ course_no.

(5) Rollno ⟶ sname

(6) Rollno, course_no ⟶ grade.

candidate keys :— { course_no, ttime }
(or)
(prime keys) { emp_id, Ttime }
{ Rollno, course_no }

From FDs, ①, ③, ⑤ are not in BCNF,

(i-e) $X \rightarrow A$, either X is a super key, (or)
$A \in X$; trivial FD.

from, course_no ⟶ title, description
(X) ⟶ (A).

Here "course_no" is not the super key, &
{ course_no, ttime } is the super key.

Ily, ③ & ⑤ also failed to satisfies the condition.

→ ②, ④, ⑥ FD are satisfies the BCNF condition

So, inorder to make the relation schemas into BCNF, we have to split the schemas as:

From ① & ②

course 1 (course_no, title, description)

course 2 (course_no, ttime, loc)

From ③ & ④

Teacher 1 (emp_id, ename, salary)

Teacher 2 (emp_id, ttime, course_no)

From ⑤ & ⑥

Student 1 (Rollno, sname)

Student 2 (Rollno, course_no, grade)

Now all the new relations are in BCNF and

if we perform. Natural join between course1 and course2, that will gives the orginal courses relati

(i-e) course1 ⋈ course2 = courses.

Hence, Lossless join property is ensured.

→ All the above new decomposed schemas contain all the FDs that are imposed on orginal schema, Hence dependency preserving decomposition is also ensured/satisfied.

∴ The above relation schemas have the properties of (1) BCNF

(2) Lossless join decomposition

(3) Dependency preserving.

Other kinds of Dependencies:-

Functional Dependencies are probably most common and important kind of constraint from the point of view of Database Design.

However, there are other kinds of Dependencies

(1) Multi Valued Dependencies

(2) Join Dependencies.

By using these two dependencies, we can identify potential redundancy problems that can not be detected using FDs alone.

Multi-Valued Dependencies (MVD)

Let R be a relation schema,

Let X and Y be subsets of attributes of R,

The multivalued Dependency (MVD) $X \rightarrow\rightarrow Y$ is said to hold over R if,

in every legal instance r of R, each 'X' value is associated with a set of Y values and this set is independent of the values in other attributes.

→ Formally, if the MVD $X \rightarrow\rightarrow Y$ holds over R and $Z = R - XY$, the following must be true for every legal instance $r(R)$:

If $t_1, t_2 \in r$, and

$t_1.X = t_2.X$, there must be some $t_3 \in r$, such that

$t_1.XY = t_3.XY$ and

$t_2.Z = t_3.Z$

$r(R)$

	X	Y	Z
t_1	a	b1	c1
t_2	a	b2	c2
t_3	a	b1	c2
t_4	a	b2	c1

From the above instance $r(R)$, if we are given the first two tuples and told that MVD $X \rightarrow\rightarrow Y$ holds over this relation, we can infer that the relation instance must also contains the third tuple.

→ The above table suggests another way to think about

MVDs: If $X \rightarrow\rightarrow Y$ holds over R,

then $\Pi_{YZ}(\sigma_{X=a}(R)) = \Pi_{Y}(\sigma_{X=a}(R)) \times \Pi_{Z}(\sigma_{X=a}(R))$

in every legal instance R, for any value a, that appears in the X-column of R.

→ In Otherwords,

consider group of tuples in R, with same X value.

- In each group consider, "the projections onto the attributes YZ"
- This projection must be equal to the cross-product of the projections onto Y and Z.

(i.e) for a given X-value, the Y-values and Z-values are independent.

From this definition, It is easy to see that $X \rightarrow\rightarrow Y$ must hold whenever $X \rightarrow Y$ holds.

If the $X \rightarrow Y$ holds there is exactly one Y-value for a given X-value.

And this condition in the MVD definition hold trivially. But "the converse doesnot hold."

Example on MVD:—

Let us take a relation "CTB" with attributes "course, teacher, Book".

CTB (course, Teacher, Book)

	course	Teacher	Book.
tuple t_1	physics 101	Green	Mechanics
t_2	physics 101	Green	optics.
t_3	physics 101	Brown	Mechanics
t_4	physics 101	Brown	optics.
t_5	Math 301	Green	Mechanics.
t_6	Math 301	Green	vectors.
t_7	Math 301	Green	Geometry.

- A Teacher T, can teach a course C, and recommend text book B for the course.

(*) There are no FDs.

*. The Key is : {course, teacher, Book}.

→ However, the recommended text for a course are independent of the teacher.

Note → 1) The relation schema CTB is in BCNF; we would not consider decomposing it further, if we looked only at the FDs. that hold over "CTB" relation instance.

2) There is "redundancy",

(i.e) the fact that Green can teach Physics 101 is recorded once per recommended text for the course. by observing first two tuples in CTB relation.

Similarly, the fact that optics is a text for physics 101 is recorded once per potential teacher.

(3) The redundancy can be eliminated by decomposing CTB into CT and CB.

→ The redundancy in this example is due to the constraint:

" The text for a course are independent of the teacher ",

which cannot be expressed interms of FDs.

This constraint is an example of a MVD.

→ The decomposed relations are: CT and CB.

CT

Course	Teacher
physics 101	Green
physics 101	Brown
Math301	Green

Instance of CT

CB

Course	Book
physics 101	Mechanics
physics 101	optics
Math301	Mechanics
Math301	Vector
Math301	geometry

Instance of CB

→ The constraints course textBooks are independent of Teachers can be expressed as C →→ T. In terms of MVDs, this constraint can be read as follows:

→ If (there is a tuple showing that) C is taught by a teacher "T", and.

(there is a tuple showing that) C has a Book B as text, then (there is a tuple showing that) C is taught by T and has text book 'B'.

Fourth Normal Form :- (4NF)

→ Fourth Normal Form is a direct generalization of BCNF.

→ Let R be a relation schema,

X and Y be non-empty subsets of the attributes of R

F be the set of FDs that includes both FD and MVDs

"R is said to be in 4NF, if,

for every MVD $X \rightarrow\rightarrow Y$ that hold over R, one of the following statements is true:

- $Y \subseteq X$ (or) $XY = R$ (or)
- X is a super key.

⇒ $X \rightarrow\rightarrow Y$ is a trivial MVD if $Y \subseteq X \subseteq R$ (or) $XY = R$;

such MVDs are always hold.

⇒ Example :-

The relation "CTB" is not in 4NF, because $C \rightarrow\rightarrow T$.

is a non-trivial MVD and C is not a key.

We can eliminate the resulting redundancy by decompo

-sing CTB into CT and CB:

CT(Course, Teacher)

CB(course, Book)

Now each of these relations are in 4NF.

→ There is a condition, that are proposed by "Date and Fagin", under these condition we can safely ignore the MVD information.

The condition is:

"If a relation schema is in BCNF, and atleast one of its keys consist of a single attribute, it is also in 4NF."

Join Dependencies:-

→ A join dependency is a further generalization of MVDs.

A join dependency (JD) $\bowtie \{R_1, R_2, \ldots R_n\}$ is said to hold over a relation R,

If $R_1, R_2, \ldots, R_n$ is a lossless-join decomposition of R.

→ An MVD $X \twoheadrightarrow Y$ over a relation R can be expressed as the join dependency $\bowtie \{XY, X(R-Y)\}$.

As an example in CTB relation, the MVD $C \twoheadrightarrow T$ can be expressed as the join dependency $\bowtie \{CT, CB\}$.

→ Unlike FDs and MVDs, there no set of sound and complete inference rules for JDs.

Fifth Normal Form :- (5NF)

A relation schema R is said to be in 5NF if, for every JD $\bowtie \{R_1, R_2, \ldots R_n\}$ that hold over R, one of the following statement is true:

- $R_i = R$ for some i, (or)
- The JD is implied by the set of those FDs over R in which the left side is a key for R.

- We have not presented inference rules for FDs & JDs taken together, there is need for explanation of second condition.
- We must be able to show that the decomposition of R into $\{R_1, R_2, \ldots R_n\}$ is lossless join whenever the "Key dependencies" (FDs in which the left side is a key for R) hold.
- JD $\bowtie \{R_1, R_2, \ldots R_n\}$ is a trivial Join Dependency if $R_i = R$ for some i; such a JD always hold.

→ Date and Fagin, identifies some condition again, under which we can safely ignore JD information, and we will detects using only FD information.

"If a relation schema is in 3NF and each of its keys consists of a single attribute, it is also in 5NF".

The condition identified in this result are sufficient for a relation to be in 5NF but not necessary.

Inclusion Dependency:-

An inclusion dependency is a statement of the form that some columns of a relation are contained in other column.

(*) A Foreign key constraint is an example of inclusion dependency.

Inclusion dependencies are very intuitive and quite common. However they have less influence on database design.

→ An example of inclusion dependency is:
an ER diagram that involves ISA hierarchies also leads to Key-based inclusion dependencies.

→ Most inclusion dependencies are in practice are "Key-Based", that is, involves only keys.

→ "Foreign key" constraint is a good example of "Key-Based Inclusion Dependencies".

→ If all inclusion dependencies are key-based, we rarely have to worry about splitting attribute groups that participate in inclusion dependencies, since decompositions usually do not split the primary key.

Note: However, that going from 3NF to BCNF always involves splitting of some key (Ideally, not the primary key), since the dependency guiding the split is of the form $X \longrightarrow A$, where A is a part of a key.

Example:- Reserves (sid, bid, day)
Sailors (sid, sname, rating, age)
Boats (bid, bname, color).

"sid" is a foreign key in reserves and primary key in sailors. So, there is an inclusion dependency exist between the relations "sailors and Reserves".

18/11/21

UNIT-4

*NORMALIZATION

→ Database design based on the ER model may have some amount of inconsistency, uncertainity, redundancy. To eliminate these drawbacks some refinement has to be done on the database

→ The refinement process is called Normalization

→ It is the technique of organizing the data into a multiple related tables, inorder to minize data redundancy (data redundancy is duplicate copies)

→ The different stages of normalization is called normal forms. They are 5 types:

1) First Normal Form (1NF)
2) Second Normal Form (2NF)
3) Third Normal Form (3NF)
4) Boyce-code Normal Form (BCNF)
5) Fourth Normal Form (4NF)
6) Fifth Normal Form (5NF)

→ the reason to go for this normal forms is due to 3 anolomies in dbms

1) Insertion
2) Updation
3) Deletion.

Emp_id	Empname	Emp address	Emp_dept
101	Rick	Delhi	D001
102	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

Insertion Anolomie:
Suppose a new employee joins a company who is under training and currently not assigned to any department then we will not able to insert that data.

Deletion Anolomie:
Suppose if the company closes the department D890 from the above table then employee information also gets deleted

Updata Anamlie:
In above table if we want to update the address of employee Rick then it should be done in two departments.
Suppose it gets updated in only one department then there is a problem of retrieving the data.

19/11/21

Functional Depedency:
In a given relation R X,Y are the attributes, attribute Y is functionally dependent on attribute X if each value of X determines exactly one value of Y, which is represented as $X \rightarrow Y$

→ FD helps us to maintain the quality of the data in the database
→ It helps to find the difference between good and bad database design.

Rules:
There are 3 important rules to be followed for functional dependencies in DB.

→ Reflexive rule:
If X is set of attributes and Y is subset of X, then X holds the value of Y.

→ Augmentation rule:
When X FD on Y i.e $X \rightarrow Y$ holds (satisfies) and C is an attribute set then $AC \rightarrow BC$ also holds ie adding attributes to C attribute set do not change the basic dependencies is called augmentation rule.

→ Transitivity rule:

This rule is very much similar to the transitive rule in algebra. If $X \to Y$ holds $Y \to Z$ then $X \to Z$ holds. X determines Y is called functionally determining the Y value.

Advantages:

1) It avoids data redundancy
2) Helps to maintain the quality of data in database
3) Helps to define meanings and constrains of databases
4) Helps to identify bad designs.

Types:

1) Partial Dependencies
2) Multi-valued Dependencies
3) Transitive Dependencies
4) Full functional Dependencies.

Partial Dependency:

It occurs when a non-prime attribute is functionally dependent on part of a candidate key.

Ex:

Sid	Pid	Sname	Pname

Sid, Pid are candidate key
Sid is primary key. (prime attribute)
Sname, Pname are non-prime attribute

→ Sname is functionally dependent upon Sid
Pname functionally dependent upon Pid

Sid → Sname, Pid → pname.

Ex: course_no coursename Facultyname.

Course_no - Primary key.

Coursename, Faculty name are non-prime attributes

course_no → coursename, Faculty name

First Normal Form:

A relation is in First Normal Form if every attribute in every row can contain only one single value i.e atomic in nature.

Student

First name	Last name	knowledge
Thomas	Mueller	Java, C++, PHP
Ursula	Meier	PHP, Java
Igor	Mueller	C++, Java

⇓

First Name	Last Name	knowledge
Thomas	Mueller	Jave
Thomas	Mueller	C++
Thomas	Mueller	PHP
Ursula	Meier	PHP
Ursula	Meier	Java
Igor	Mueller	C++
Igor	Mueller	Java

Second Normal Form:

The data must be in a First Normal Form. All non-prime attributes should be fully functional dependent on the candidate key.

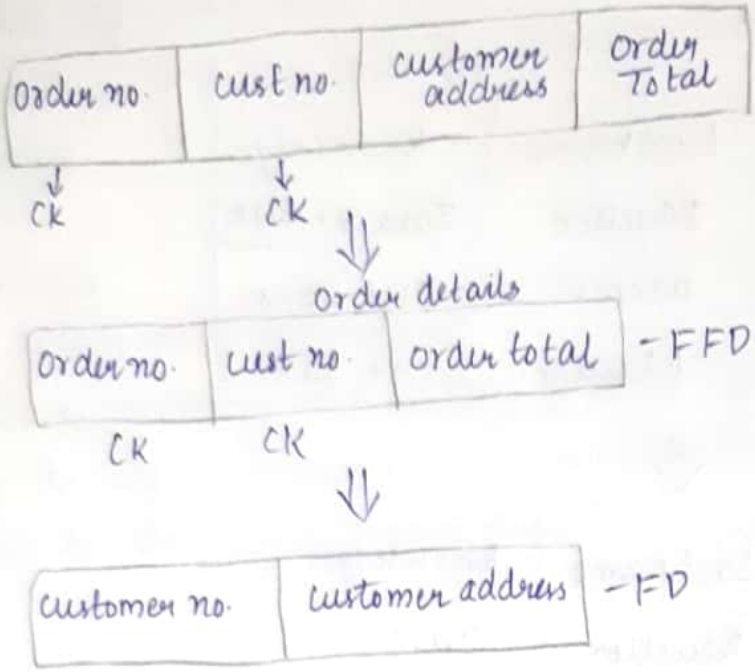
→ Each column i.e not a part of any candidate key depends on the entire candidate key.

→ We must check whether the database is falling insertion, updation, deletion anomalies or not.

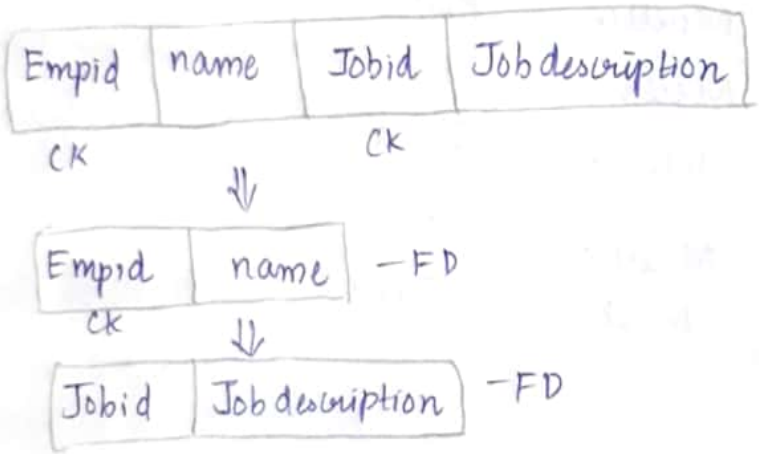
→ If it falls the above problem that means there is a partial dependencies between the columns.

→ If we remove all the partial dependencies then the table becomes into a 2NF

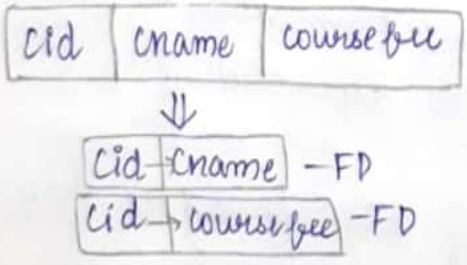
Ex: 1)



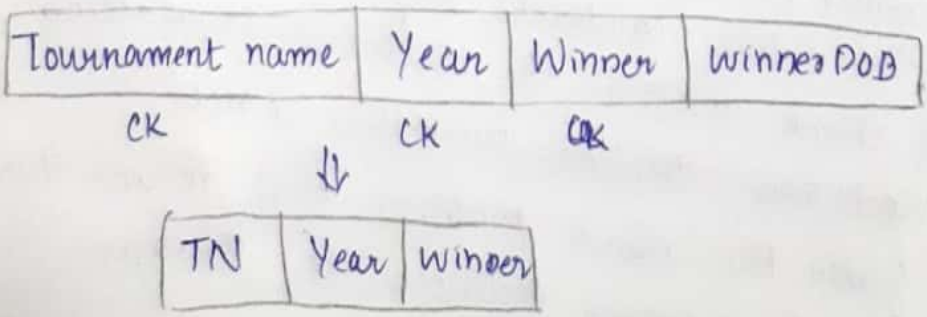
2)



3)



4)



b)

ccode	date	cname	seat	Room

CK

⇓

ccode	cname

PF

⇓

ccode	date	Room	seat

Fk

NOTE:

→ To Test 2NF

i) is the table satisfied 1NF or not

ii) Identifying candidate keys correctly

iii) For each candidate key which columns or what a columns are not part of a candidate key

iv) Every non-prime attributes really depend upon the entire candidate key

Full Functional Dependency (FFD):

It is the dependency where non-key attributes are functionally dependent on complete candidate key.

Ex

Eid	Ename	Pid	Ptitle	Hrly worked

⇓

Eid → Ename → FD

Pid → Ptitle → FD

Eid, Pid → Hrly Worked → FFD

Decomposition in DBMS:

The process of breaking up or dividing a single relation into 2 or more sub relationals is called Decomposition of a relation.

→ Types of Decomposition:

1) Loseless Decomposition

2) lossy Decomposition

Loseless Join decomposition

→ Consider relation R which is decomposed into subrelationals $R_1, R_2 \dots R_n$.

→ This decomposition is called Lossless Join decomposition when we join all the sub relations, generates the same relation as R that must decompose.

$$R_1 \bowtie R_2 \bowtie R_3 \dots \bowtie R_n = R$$

$\bowtie$ → natural join

A	B	C
1	2	1
2	5	3
3	3	3

R

A	B
1	2
2	5
3	3

R_1

B	C
2	1
5	3
3	3

R_2

A	B	C
1	2	1
2	5	3
3	3	3

$R_1 \bowtie R_2 = R$

NOTE:
Lossless Join decomposition is also called as non-additive Join decomposition because the resultant relation is same before and after decomposition. i.e no extra tuples obtained

Lossy Join decomposition:

→ Consider relation R which is decomposed into subrelationals $R_1, R_2 \dots R_n$.

→ The decomposition is said to be Lossy Join decomposition when we join the subrelationals which doesnot ~~join~~ produce original relation that first decomposed.

→ It contains extra tuples.

$$R_1 \bowtie R_2 \bowtie R_3 \dots \bowtie R_n \supset R$$

A	B	C
1	2	1
2	5	3
3	3	3

A	C
1	1
2	3
3	3

R_1

B	C
2	1
5	3
3	3

R_2

A	B	C	
1	2	1	
2	5	3	
2	3	3	Extra tuple
3	5	3	
3	3	3	

$R_1 \bowtie R_2$

Method 2:

Consider Relation R which is decomposed into subrelationals R_1 and R_2 to say that the decomposition is lossy or loseless the following conditions gets satisfied:

1) $R_1 \cup R_2 = R$
2) $R_1 \cap R_2 \neq \phi$
3) $R_1 \cap R_2$ = superkey of R_1 or R_2

If all the 3 conditions gets satisfied then decomposition is said to be losseless.

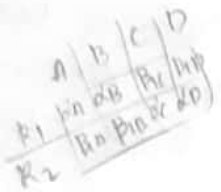
Problem 1:

$R(A, B, C, D)$

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

R: $R_1(A, B)$

$R_2(C, D)$



Find whether decomposition of R is loseless or lossy.

Sol: 1) $R_1(A, B) \cup R_2(C, D) = (A, B, C, D) R$

union of both subrelations must contain all the attributes of Relation R.

Condition 1 is satisfied.

2) Intersection of both subrelationals must not be equal to null.

$R_1(A, B) \cap R_2(C, D) = \phi$

Condition 2 fails

So the above decomposition is said to be lossy decomposition

Problem 2:

$R(A, B, C, D)$

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

$R_1(A, B)$, $R_2(B, C)$, $R_3(B, D)$

Sol: 1) $R_1(A, B) \cup R_2(B, C) \cup R_3(B, D) = (A, B, C, D) R$

Condition 1 satisfied

2) $R_1(A, B) \cap R_3(B, C) \cap R_3(B, D) = B$

Condition 2 satisfied.

3) $R_1 \cap R_2 = B$, $R_2 \cap R_3 = B$, $R_1 \cap R_3 = B$

B is said to be as super key which derives all attributes of relation R

Condition 3 satisfied

So above decomposition is losseless decomposition.

Problem-3:

$R(A,B,C)$, $R_1(A,B)$, $R_2(B,C)$

1) $R_1(A,B) \cup R_2(B,C) = (A,B,C)R$

2) $R_1(A,B) \cap R_2(B,C) = B$

3) $R_1 \cap R_2 = B$

condition 1,2,3 satisfied.

So the above decomposition is losseless decomposition.

Method-3:

A table TABLE-LOSSY $(1:n, 1:k)$ is used to test for the type of decomposition Row i is for relation schema R_i of the decomposed relation and column j is for attribute A_j in the original relation.

→ for each decomposed relation R_i do

if an attribute A_j is included in R_i

then TABLE-LOSSY $(i,j) := \alpha_A$ // Place a symbol α_A in row i and column j //

else TABLE-LOSSY $(i,j) := \beta_{iA}$ // place a symbol β_{iA} in row i and column j //

→ change := true

while (change) do

for each FD $X \to Y$ in F do

if rows i & j exist such that same symbol appears in each column corresponding to the attribute of x then if one of the symbols in the y column is α_s then make the other α_s else the symbols are β_{pm} & β_{qm} then make both of them, say β_{pm} else

change = false

i := 1

lossy := true

while (lossy and $i \le n$) do

for each row i of TABLE-LOSSY

if all symbols are α_s

then lossy := False
else i := i+1;

Problem:

R(A, B, C, D, E)

FD: AB → CD, C → D / $R_1(A,B,C)$, $R_2(B,C,D)$, $R_3(C,D,E)$

A → E

	A	B	C	D	E
R_1	α_A	α_B	α_C	β_{1D} α_D	β_{1E}
R_2	β_{2A}	α_B	α_C	α_D	β_{2E}
R_3	β_{3A}	β_{3B}	α_C	α_D	α_E

NOTE:

If any one of the rows contains all 'a' values then decomposition is lossless otherwise lossy.

→ For above relation there are no all 'a' values in R_1, R_2, R_3 so the decomposition is lossy.

Problem:

R(A, B, C, D)

FD: A → B, A → C, C → D

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

	A	B	C	D
R_1	α_A	α_B	α_C	β_{1D} α_D
R_2	β_{2A}	β_{2B}	α_C	α_D

Since all 'a' values are present in R_1 it is lossless decomposition.

Problem:

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

$R_1(B,E)$, $R_2(A,C,D,E,F)$ / FD: A → B, C → DE, AC → F

	A	B	C	D	E	F
R_1	β_{1A}	α_B	β_{1C}	β_{1D}	α_E	β_{1F}
R_2	α_A	β_{2B}	α_C	α_D	α_E	α_F

lossy decomposition.

Problem:

$R(A, B, C, D, E)$

FD: $A \to BC$, $CD \to E$, $B \to D$, $E \to A$

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

	A	B	C	D	E
R_1	α_A	α_B	α_C	β_{1D} / α_D	β_{1E}
R_2	α_A	β_{2B} / α_B	α_C	α_D	α_E

R_2 rows contains α values

$\therefore$ Lossless decomposition

Closure Set of Attributes:

The set of all those attributes which can be functionally determined from an attribute set is called closure of that attribute set.

→ closure of a attribute set is denoted as $\{A\} = \{A\}^+$

Ex: $R = \{A\ B\ C\ D\ E\ F\ G\}$

Satisfies F.D's: $\{A \to B, BC \to DE, AEF \to G\}$

Find the closure of $\{AC\}^+$

Sol: $\{A\}^+ = \{A\}$

$(AC)^+ = \{A\ C\ B\ D\ E\}$

We cannot determine $AEF \to G$ because in AC closure F is not present

Trivial Functional Dependency (TFD)

If $A \to B$ and $B \subseteq A$, then this dependency is called trivial functionally dependency

Ex: $A \to A$, $B \to B$ } TFD

$AB \to B$, $A \to B$, $B \to B$ } TFD

Rollno → Rollno

Rollno, Sname → Sname

Eid, Ename → Eid

Eid, Ename → Ename

Non-trivial Functional Dependency (NTFD)

If $A \to B$ and $B \not\subseteq A$ if a FD: $A \to B$ is true where B is not a subset of A, then this dependency is called non-trivial functional dependency.

Ex: $AB \to B$, $A \to Bx$ } NTFD

Eid, Ename → Email Address.

Dependency Preserving Decomposition

A decomposition of relation R into $R_1, R_2 -- R_n$ is dependency preserving decomposition with respect to set of functionally dependencies F that holds on R only if the following condition is true $(F_1 \cup F_2 \cup F_3 -- \cup F_n)^+ = F^+$

where $F_1, F_2 -- F_n$ are the set of functional dependencies of relation $R_1, R_2, -- R_n$

- $(F_1 \cup F_2 \cup --- \cup F_n)^+$ is a closure of union of all sets of functionally dependencies.
- F^+ is a closure set of functionally dependencies F of R

Problem:

Given $R = (A\ B\ C)$

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

$R_1 = \{A\ B\}$ $R_2 = \{B\ C\}$

$R_1 \to F_1$, $F_1 = (A)^+, (B)^+, (AB)^+$

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

$= \{B\ C\}$ ($\because A$ is a TFD, $A \to A$)

$= \{B\}$ ($\because C$ is not present in R_1)

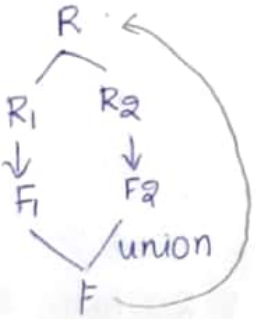
$A \to B$ — (1)

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

$= \{C\ A\}$ ($\because B \to B$ is a TFD)

$= \{A\}$ ($\because C$ is not present in R_1)

$B \to A$ — (2)



$(AB)^+ = ABC$

$= \{C\}$ ($\because AB \to AB$ is TFD)

$(AB)^+ = \phi$ ($\because C$ is not in R_1)

$FD_1 = \{A \to B, B \to A\}$

$R_2 \to F_2$

$F_2 = (B)^+, (C)^+, (BC)^+$

$(B^+)^+ = \{BCA\}$

$= \{CA\}$ ($\because B \to B$ is TFD)

$= \{C\}$ ($\because A$ is not in R_2)

$B \to C$ — (1)

$(C)^+ = \{CAB\}$

$= \{AB\}$ ($\because C \to C$ is TFD)

$= \{B\}$ ($\because A$ is not in R_2)

$C \to B$ — (2)

$(BC)^+ = \{BCA\}$

$= \{A\}$ ($\because BC \to BC$ is TFD)

$= \phi$ ($\because A$ is not in R_2)

$FD_2 = \{B \to C, C \to B\}$

$FD_1 \cup FD_2 = \{A \to B, B \to A, B \to C, C \to B\}$

$\therefore FD_1 \cup FD_2 \neq F$

i.e We got only $A \to B, B \to C$ but not $C \to A$.

The given 'R' Relation is not a dependency preserving decomposition

Problem:

$R(A, B, C, D)$

$FD: \{AB \to C, C \to D, D \to A\}$

$R_1 = (ABC)$ $R_2 = (CD)$

Sol: $R_1 \to F_1$

$F_1 = (A)^+, (B)^+, (C)^+, (AB)^+, (BC)^+, (CA)^+, (ABC)^+$

$(A)^+ = \{A\}$

$= \phi$ ($\because A \to A$ is TFD)

$(B)^+ = \{B\}$

$= \phi$

$(C)^+ = \{CDA\}$

$= \{DA\}$

$= \{A\}$

$C \to A$.

$(AB)^+ = \{ABCDA\}$

$= \{CDA\}$

$= \{AC\}$

$AB \to AC$.

$(BC)^+ = \{BCDA\}$

$= \{DA\}$

$= \{A\}$

$BC \to A$.

$(CA)^+ = \{CADA\}$

$= \{D\}$

$= \phi$

$(ABC)^+ = \{ABCD\}$

$= \{D\}$

$= \phi$

$FD_1 = \{C \to A, AB \to C, BC \to A\}$

~~$(D)^+ = \{DA\}$~~

~~$= \{A\}$~~

~~$D \to A$~~

$F_2 = (C)^+, (D)^+, (CD)^+$

$(C)^+ = \{CDA\}$

$= \{DA\}$

$= \{D\}$

$C \to D$

$(D)^+ = \{DA\}$

$= \{A\}$

$= \phi$

$(CD)^+ = \{CDA\}$

$= \{A\}$

$= \phi$

$FD_2 = \{C \to D\}$.

$FD_1 \cup FD_2 \neq F$

We got $AB \to C, C \to D$ but not $D \to A$

∴ Not a depedency preserving decomposition

$R(A, B, C, D, E, F)$

$A \to BC, C \to A, D \to E, F \to A, E \to D$

$R_1(A, C, D)$ $R_2(B, C, D)$ $R_3(E, F, D)$

	A	B	C	D	E	F
R_1	α_A	β_{1B}	α_C	α_D	β_{1E}	β_{1F}
R_2	β_{2A}	α_B	α_C	α_D	β_{2E}	β_{2F}
R_3	β_{3A}	β_{3B}	β_{3C}	α_D	α_E	α_F
					β_{3E}	

Q) R(A B C D E)

FD: $\{A \to BC, CD \to E, B \to D, E \to A\}$

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

Check whether loseless or lossy.

Sol:

	A	B	C	D	E
R_1	α_A	α_B	α_C	β_{1D}	β_{1E}
R_2	α_A	β_{2B} (α_B)	β_{2C} (α_C)	α_D	α_E

Since row R_2 has all α values.

∴ Loseless.

Q) R(P, Q, R, S)

$R_1(P, Q, R)$ $R_2(R, S)$

FD: $\{PQ \to R, R \to S, S \to P\}$

Check dependence preserving.

Sol: $F_1 = (P)^+, (Q)^+, (R)^+, (PQ)^+, (QR)^+, (PR)^+, (PQR)^+$

$(P)^+ = \{P\}$
$= \phi$

$(Q)^+ = \{Q\}$
$= \phi$

$(R)^+ = \{RSP\}$
$= \{SP\}$
$= \{P\}$

$R \to P$ -①

$(PQ)^+ = \{PQ, RS\}$
$= \{RS\}$
$= \{R\}$

$PQ \to R$ -②

$(QR)^+ = \{QRSP\}$
$= \{SP\}$
$= \{P\}$

$QR \to P$ -③

$(PR)^+ = \{PRS\}$
$= \{S\}$
$= \phi$

$(PQR)^+ = \{PQRS\}$
$= \{S\}$
$= \phi$

$F_1 = \{R \to P, PQ \to R, QR \to P\}$

$F_2 = (R^+)^+, (S)^+, (RS)^+$

$(R)^+ = \{RSP\}$
$= \{SP\}$
$= \{S\}$

$R \to S$ -①

$(S)^+ = \{SP\}$
$= \{P\}$
$= \phi$

$(RS)^+ = \{RSP\}$
$= \{P\}$
$= \phi$

$F_2 = \{R \to S\}$

$F_1 \cup F_2 = \{R \to P, R \to S, PQ \to R, QR \to P\}$

We got $PQ \to R$, $R \to S$ but not $S \to P$

∴ Not dependence preserving.

a) $R(ABCD)$

FD: $\{A \to B, A \to C, C \to D\}$

$R_1(ABC)$ $R_2(CD)$

check dependency preserving

Sol $F_1 = (A)^+, (B)^+, (C)^+, (AB)^+, (BC)^+, (CA)^+, (ABC)^+$

$(A)^+ = \{ABCD\}$
$= \{BCD\}$
$= \{BC\}$

$A \to BC$ — (1)

$(AB)^+ = \{ABCD\}$
$= \{CD\}$
$= \{C\}$

$AB \to C$ — (2)

$(ABC)^+ = \{ABCD\}$
$= \{D\}$
$= \phi$

$(B)^+ = \{B\}$
$= \phi$

$(BC)^+ = \{BCD\}$
$= \{D\}$
$= \phi$

$(C)^+ = \{CD\}$
$= \{D\}$
$= \phi$

$(CA)^+ = \{CABD\}$
$= \{BD\}$
$= \phi$

$F_1 = \{A \to BC, AB \to C\}$

$F_2 = (C)^+, (D)^+, (CD)^+$

$(C)^+ = \{CD\}$
$= \{D\}$

$C \to D$ — (1)

$(D)^+ = \{D\}$
$= \phi$

$(CD)^+ = \{CD\}$
$= \phi$

$F_2 = \{C \to D\}$

$F_1 \cup F_2 = \{A \to BC, AB \to C, C \to D\}$
$= \{A \to B, A \to C, C \to D, AB \to C\}$

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

$\therefore$ Dependency preserving.

6/12

Third Normal Form:

→ It is used to reduce the duplication of the data and ensure differential integrity.

→ It must satisfy the following conditions:

1) Table must be in 2NF.
2) No transitive dependency must be there i.e all non-prime attributes must depend on prime attributes only.

Eg * | Rollno | Sname | Subjectid | Subjectname | Address |

⇓

① | Rollno | Sname | Subjectid | Address |

② | Subjectid | Subjectname |

(∵ Roll → Subject id → Subjectname)

*

Rollno	Subjectid	marks	Exam type	Total marks
101	G2	20	mid	25
102	G3	65	external	75

① | Rollno | Subjectid | marks |

② | Subjectid | Examtype | Total marks |

→ A relation schema 'R' is in 3NF if it satisfies 2NF and non-prime attribute of R is transively dependent on key (PK) of R

(OR)

→ A relation is in 3NF if functional dependency X → Y satisfies any one of the following conditions.

if FD: X → Y

i) X → Y is a trivial FD i.e Y ≤ X

Eg: AB → A

ii) if X → Y, then X is a superkey

iii) if X → Y, then (Y → X) is a prime attribute (part in CK)

Eg: R(ABC)

FD: $A \to B$, $B \to C$

R is in which Normal Form

Sol $(A)^+ = \{ABC\}$ $(B)^+ = \{BC\}$

$= \{BC\}$ $= \{C\}$

$A \to B$, $A \to C$ $B \to C$ (∵ transitive)

It is not in 3NF.

BCNF:

→ It is an extension of 3NF, it is also called as 3.5 NF

→ To convert 3NF into BCNF, it must satisfy the following conditions:

1) Table must be in 3 NF.
2) For every functional dependency, $A \to B$, A has to be super key of that particular table.
3) A super key is a group of single or multiple keys which identifies rows in a table.

$X \to Y$ is a trivial FD

$X \to Y$, then X is a superkey.

Eg:

(P) Stdid	(NP) subject	(NP) professor
101	physics	kumar
102	Maths	shiva
103	chemistry	Aruna

⇒

(SK) stdid	Professor id
101	F1
102	F2
103	F3

(SK) Professorid	Professor	subject
F1	kumar	physics
F2	shiva	Maths
F3	Aruna	chemistry

→ After dividing, every table should contain superkey (SK)

Eg:

student	Course	Teacher

⇓

① | student | Cid |

② | Cid | teacher |

Fourth Normal Form:

For a table to satisfy 4NF the following conditions must be considered

1) Table should be in BCNF
2) A table should not have any multivalued dependency.

3NF

Ex:

Rollno	Name	Pincode	city

PK → Rollno
Rollno → name, pincode, city
Rollno ← Pincode → city (Transitive dependency)

Rollno	Name	Pincode

Pincode	city

Prime → non prime attribute ✓
Non-prime attribute → non-prime attribute ✗

Sid	Sname	marks	grade

Sid → Sname, marks, grade
marks → grade
(Non-prime) (Non-prime)

Sid	Sname	marks

marks	grade

BCNF

1) Candidate key: It is a column in a table which has the ability to become primary key.
2) Determinant: It is any attribute on which some other attribute is fully functionally dependent.
3) BCNF: A relation R is said to be BCNF if and only if "every determinant is a candidate key"

ID	Ename	Qualification	Grade	Did	Dname
1	Sai	BE	C	20	IT
2	Shiva	ME	B	30	CSE
3	Ramesh	PhD	A	10	ME

ID → Ename, Did

ID, Qualification → grade

Did → Dname

ID Qualification Grade

↳ CK + PK + determinant → grade

↳ qualification → grade

CK + determinant

Relation 1

ID	Qualification	Grade
1	BE	C
2	ME	B
3	PhD	A

ID → Ename

↳ CK + PK + determinat

ID → Did

↳ CK + PK + determinant

Relation 2

ID	Ename	Did
1	Sai	20
2	Shiva	30
3	Ramesh	10

Did → Dname

Relation 3

Did	Dname
20	IT
30	CSE
10	ME

Multivalued Dependency:

A multivalued dependency requires min. of 3 columns in which there should be atleast 2 attributes that depend on 3rd one.

→ These 2 attributes are dependent on each other.

Conditions:

1) There should be atleast 3 columns in a table
2) For every dependency $A \rightarrow B$, for every value of A contains multiple values of B then the dependency is referred as multivalue dependency.
3) In the relation of 3 columns $R(XYZ)$, if there exists multivalued dependencies b/w X & Y then $Y \perp Z$ should be independent to each other.

NOTE:

→ A table with multivalued dependency violates the normalisation standard of 4NF. because it creates unnecessary redundancies and can contribute to inconsistent data.

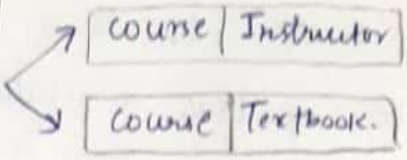
→ To overcome this problem, for 4NF it is necessary to break this information into 2 tables.

Sid	Subject_enrolled	Activity_enrolled
45	Economics	Painting
45	History	Hockey
33	Physics	Drawing
59	Chemistry	Singing
40	Python	Journalism

Sid	Subject_enrolled
45	Economics
45	History
33	Physics
59	Chemistry
40	Python

Sid	Activity_enrolled
45	Painting
45	Hockey
33	Drawing
59	Singing
40	Journalism

Course	Instructor	Textbook
Management	X	Sai
Management	Y	Peter
Management	Z	Peter
Finance	A	Wilson
Finance	A	Grin



Advantages:

1) It helps in removing the redundancy and anomolies in the database
2) Data Integrity and consistency can be maintained through normalisation and restricted constraints.

Fifth Normal Form:

It is also known as project Joint normal form (PJ/NF)

→ It is designed to minimize redundancy in relational databases by seperating symetrically connected relationships in multple formats to store multivalued data

→ A relation R is in 5NF if and only if every non-trivial join dependancy in R is simplified by the candidate key of R

→ A relation break up into 3 relations must contain loseless join property, that has no invalid or extra tuples of attributes which are recreated when relations are again joined together through natural join

Properties:

A relation R with attributes, its values and tuples is in 5NF if and only if the following condition must be satisfied:

1) The relation R should already in 4NF
2) The relation R should not contain additional non-loss decomposed data (after join dependency)

Join Dependency: A join dependancy is a further generalisation of MVD (multivalue dependancy).

JD $\bowtie \{R_1, R_2, -- R_n\}$ is said to hold over a relation R if $R_1, R_2, -- R_n$ is a lossless join decomposition of R

i.e It says that a relation once decomposed into 2 or more subrelations then it must be capable of joining back to the original relation.

→ This can be done when we maintain proper combinations of subrelations which when joined forms the original relation in a loseless manner.

Ex: Teacher

Subject	Tname	class
Physics	X	6
Maths	Y	7
Social	Z	8

R_1 → subject, Tname

R_2 → Tname, class

R_3 → Subject, class

rializability:-

Recoverability:-

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	

A recoverable schedule is one where, for each pair of transaction T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the commit operation of T_j.

In the above example. T_2 performs only read(A) operation.

→ If the system allows T_2 to commit immediately after executing the read(A) instruction.

→ T_2 commits before T_1 does.

→ now suppose T_1 fails before it commits.
→ since T_2 has read the value of data item A written by T_1.
→ now we must abort the T_2 to ensure atomicity.
→ since T_2 already committed & we cannot abort it.
→ This situation is highly impossible to recover correctly from the failure of T_1.

Cascadeless schedules

T_1	T_2	T_3
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

Even if a schedule is recoverable to recover correctly from the failure of a transaction T_i, we may have to rollback several transactions.

In the above schedule. T_1 writes a value of A that is read by Transaction T_2.

T_2 writes a value of A that is read by T_3

Suppose that at this point T_1 fails, T_1 must be rolled back.

→ Since T_2 is dependent on T_1. T_2 must be rolled back.

→ T_3 is dependent on T_2. T_3 must be rolled back.

→ This phenomenon in which a single transaction failure leads to a series of transactions rollback is called "Cascading rollback"

Cascadeless schedules :- For each pair of transactions T_i and T_j, such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.

→ It is easy to verify that every cascadeless schedules is also recoverable.

It is desirable to restrict the schedules to those where "Cascading rollbacks cannot occur."

Such schedules are called cascadeless schedules

Implementation of Isolation:-

There are various concurrency control schemes that can be used to ensure that even when multiple transactions are executed concurrently only acceptable schedules are generated.

→ A trivial example of concurrency control scheme is

→ A transaction acquires a lock on the entire database before it starts and releases the lock after it has committed.

→ while a transaction holds a lock, no other transaction is allowed to acquire the lock, and all must therefore wait for the lock to be released.

→ As a result of locking policy, only one transaction can executed at a time.

Therefore only serial schedules are generated.

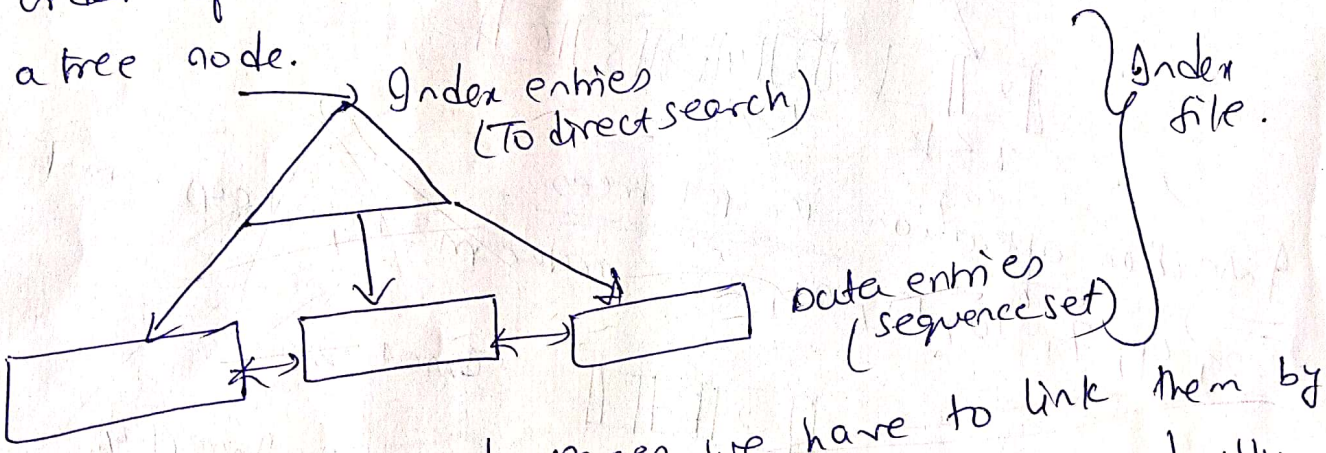
→ These trivially serializable is very eas to verify and they are cascadeless as well

rees :- It is a balanced tree in which the internal nodes direct the search and the leaf nodes contains the data entries. (1)

→ Since the tree structure grows and shrinks dynamically it is not feasible to allocate the leaf pages sequentially as in ISAM.

→ Every node contains m entries, where $d \leq m \leq 2d$. The value of d is a parameter of the B+ tree, called the order of the tree, and is a measure of the capacity of a tree node.

→

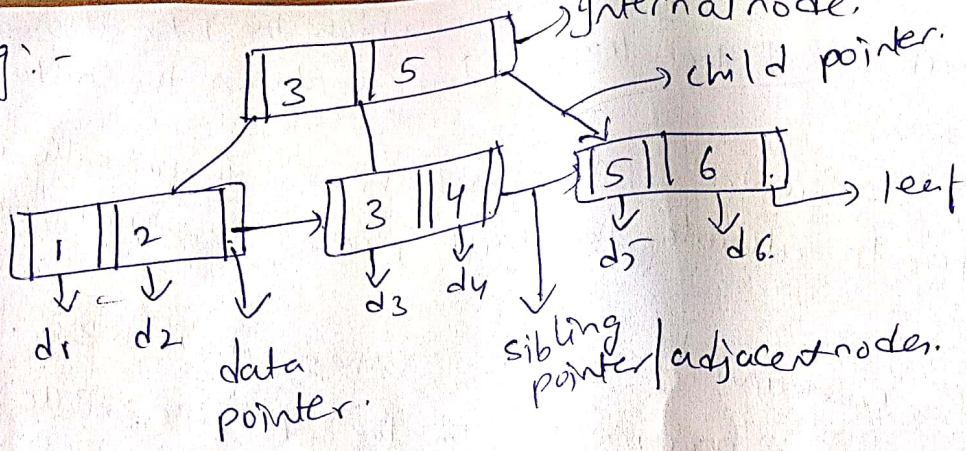


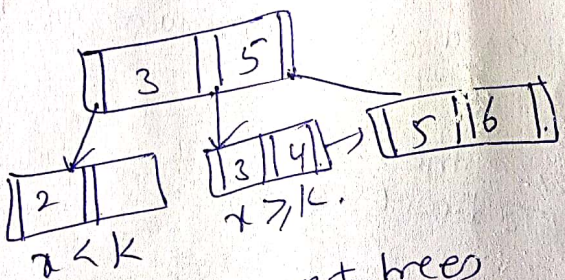
→ To retrieve all leaf pages, we have to link them by using page pointers, by organizing them into a doubly linked list.

→ Characteristics of B+ trees

(1) Data records are only stored in leaves.
(2) Internal nodes stores just keys.
(3) All leaf nodes are interconnected with each other for faster access.
(4) Keys are used for directing a search to the proper leaf.
(5) B+ trees combines features of ISAM, B trees.

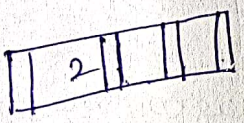
Eg:-



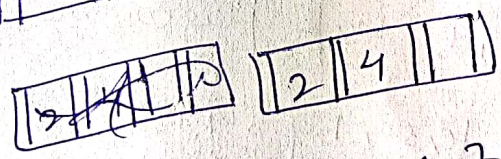


Insertion operation in B+ trees

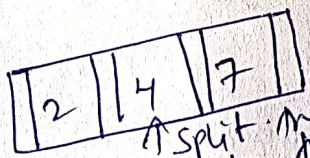
2, 4, 7, 10, 17, 21, 28 order $m = 4$, keys $= m-1 = 3$



Insert 2



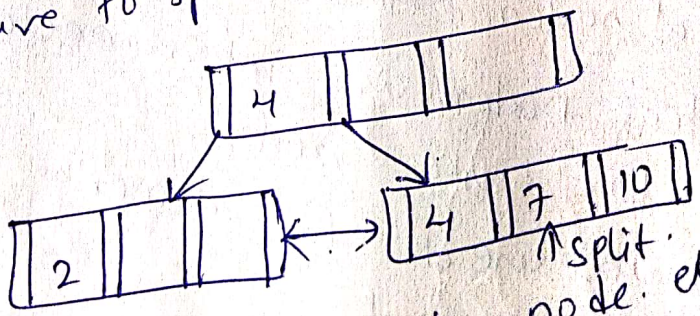
Insert 4.



Insert 7.

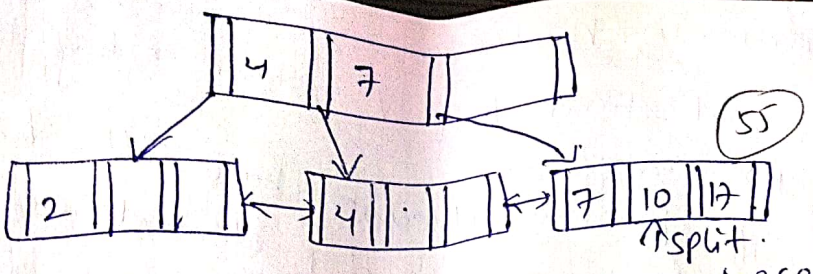
↑ split. The middle element always.

To insert 10 into the node since the keys limit is 3 so we have to split the middle element as follows

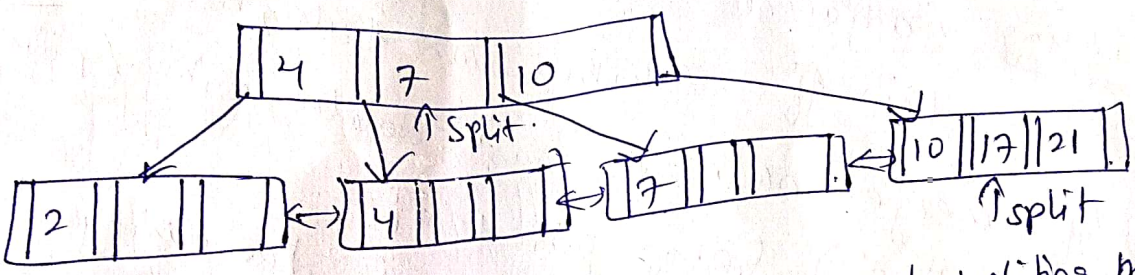


Insertion of 10

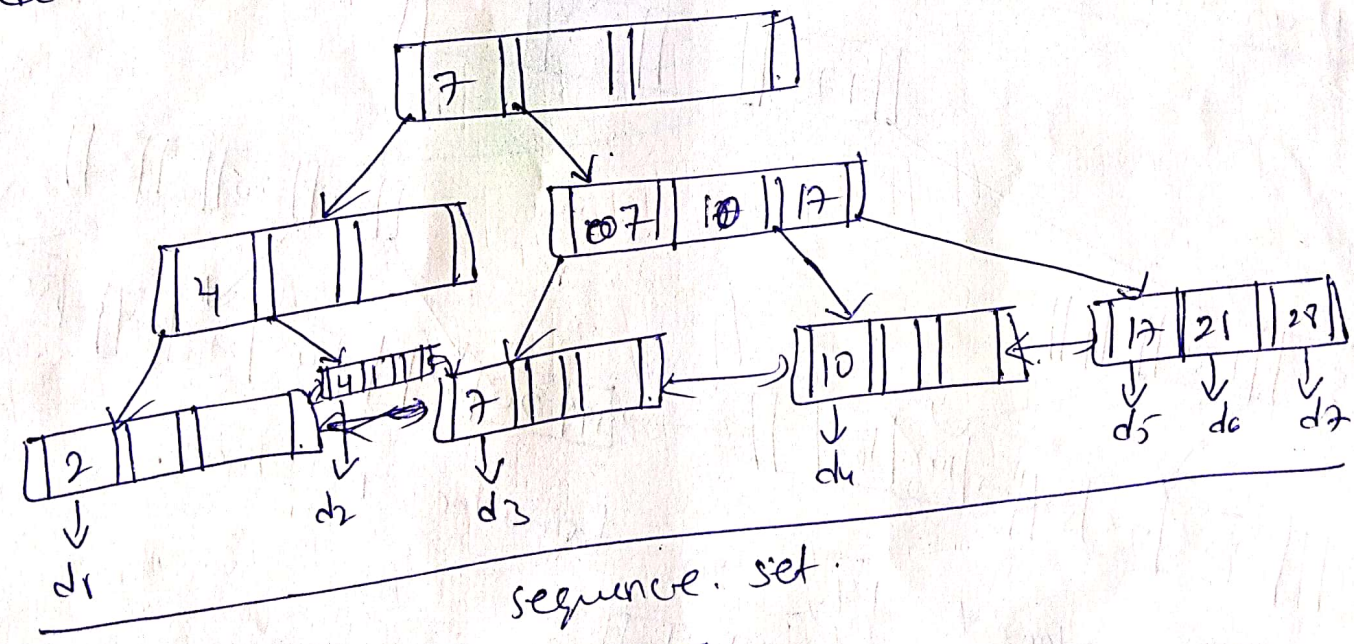
To insert 17 again split the node. element 7 as follows

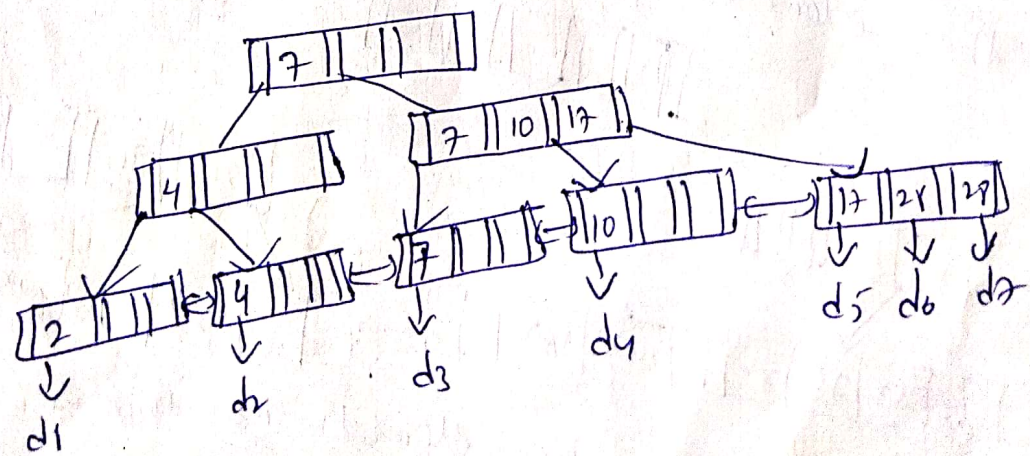


To insert 21 and repeat the same procedure of split the node 7, 10, 17 as follows.



To insert 28, and repeat same procedure of spliting the node as follows.





Deletion operation in B+Trees

→ Start at root, find leaf L where entry belongs.

→ Remove the entry.

→ If L is atleast half-full, done!

→ If L has only d-1 elements or entries,

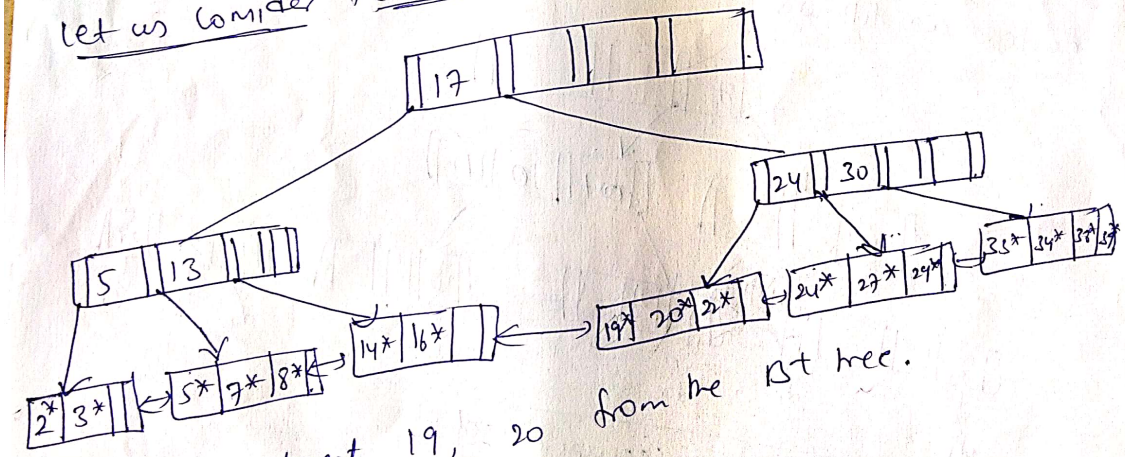
(i) Try to redistribute, borrowing from sibling (adjacent node with same parent as L).

(ii) If - redistribution fails, merge L and sibling.

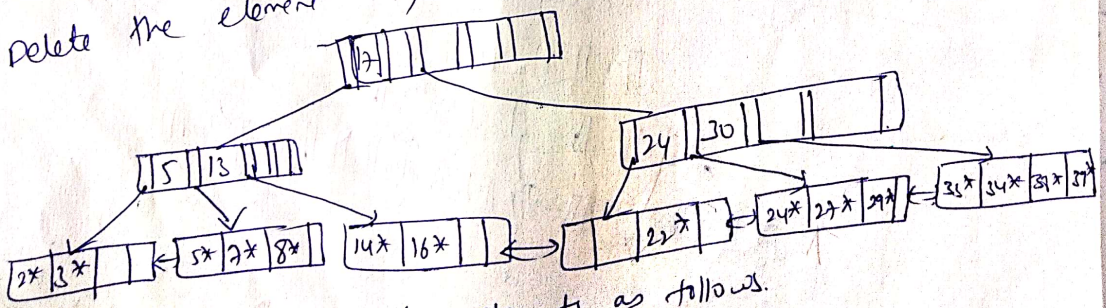
→ If merge occured, must delete entry (pointing to L or sibling) from parent of L.

→ Merge could propagate to root, decreasing height.

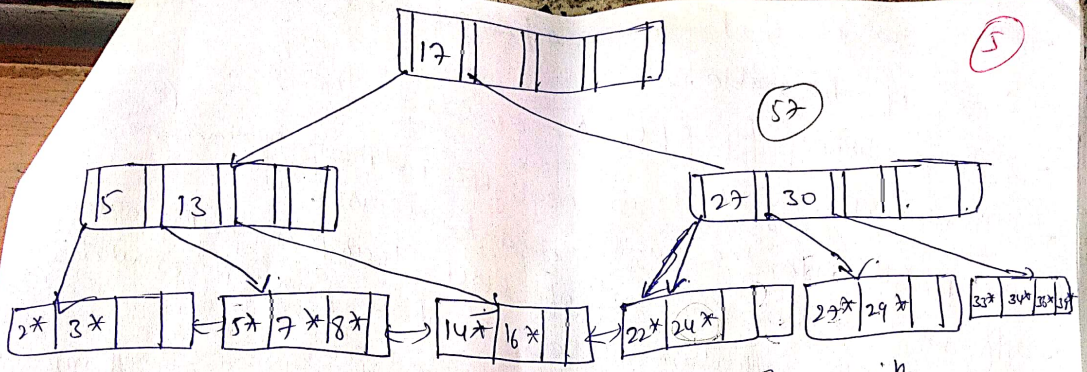
Let us consider the following B+ tree



Delete the element 19, 20 from the B+ tree.

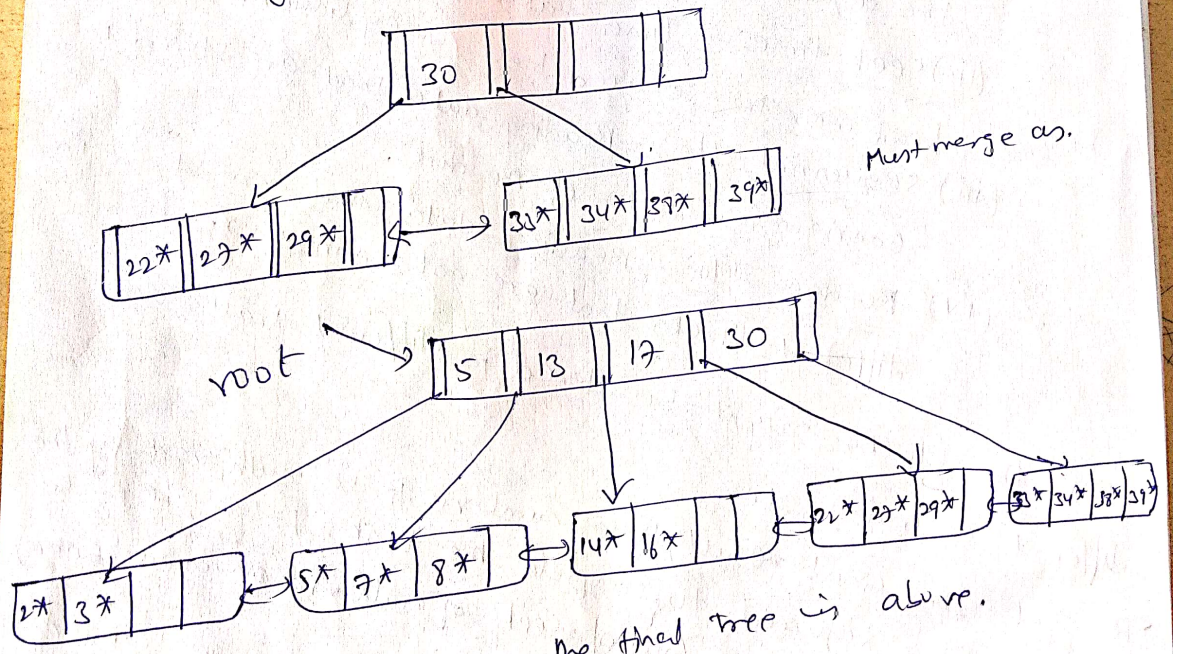


Now rearrange the elements as follows.



Deleting 19 is easy, but deleting 20 is done with redistribution. Now the middle key is copied up.

Now deleting 24*



After deleting 19, 20, 24, the final tree is above.

Measures of Query cost:- In DBMS, the cost involved in executing a query can be measured by considering the no. of different resources that are listed below:-

...mplete at the time of crash.

...mote Backup systems) – (11)

→ Traditional transaction processing systems are centralized or Client-server systems.

→ such systems are vulnerable to environmental disasters such as fire, flooding or earthquakes.

→ There is need for transaction processing systems that can function in spite of system failures or environment disasters.

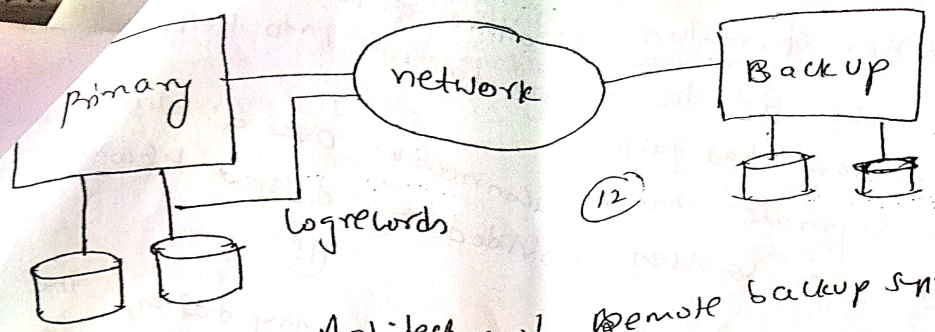
→ such systems must provide high availability, the time for which the system is unusable must be extremely small.

→ we can achieve high availability by performing transactions processing at one site called the primary site and having a remote backup site.

→ where all the data from the primary site are replicated.

→ The remote site must be kept synchronized with the primary site, as updates are performed at the primary.

→ Synchronization is done by sending all log record from primary site to the remote backup site.



Architecture of Remote backup system.

→ When primary site fails, the remote backup site takes over processing.

→ It performs recovery using its copy of the data from the primary and the log records received from the primary.

→ Remote backup site is performing recovery actions that would have been performed at the primary site when the latter recovered.

→ Standard recovery algorithms with minor modifications can be used for recovery.

→ Once recovery has been performed, the remote backup site starts processing transactions.

→ Performance of a remote backup system is better than performance of a distributed system with two phase commit

Several issues must be addressed in designing remote backup system.

(1) Detection of failure:- Failure of communication can make doul the remote backup into believe that the primary has failed.
Eg:- separate modem connection over a telephone line, with services provided by different telecommunication companies.

(2) Transfer of Control:- When the primary fails, the backup site takes over processing and becomes the new primary.
→ When original primary site recovers, it can either play the role of remote backup, or take over the role of primary site again before of log of updates must be done again.

(3) Time to recover:- If the log at the remote backup grows large, recovery will take long time.
→ The remote backup site can periodically process the redo log records that it has received, and can perform checkpoints, so that earlier parts of the log can be deleted.
→ A "hot-spare" configuration makeover by the backup site almost instantaneous.

→ Time to commit:- To ensure that the updates of a committed transaction are durable, a transaction must not be declared committed until its log records have reached the backup site.

ree of durability can be classified as.

e-Safe:- A transaction commits as soon as its commit log record is written to stable storage at the primary site.

(14)

(ii) Two-very safe:- A transaction commits as soon as it commit log record is written to stable storage at the primary and the back up site.

(iii) Two-safe:- Same as two-very safe if both primary and backup sites are active, If only the primary is active, the transaction is allowed to commit as soon as its commit log record is written to stable storage at the primary site.

Log:- :- Structure used for recording database modifications is called "log".

Log record:- The log is a sequence of log records, recording all the updates activities in the database.

An update log record :- Contains.

(1) Transaction identifier which is a unique identifier of the transaction that is performed the write operation.

(2) Data item identifier unique identifier of the data item written. i.e location on disk.

(3) Old value:- value of the data prior to write

(4) New value:- value of the data item will have after write.

LSN:- In database page, to identify which operations have been applied to a database page.

Unit-5

Transaction Concept:- Transaction is a program unit whose execution may change the contents of a database.

Database before transaction
Database after transaction

After transaction operation data must be in "Consistent state"

Transaction is used to represent a logical unit of database processing that must be completed in a its entirety to ensure correctness.

→ "Transaction must be completed, ~~done~~ and data must be in consistent state".

ACID properties of a transaction:

Atomicity:- [ALL or None the transaction must de ~~done~~]

→ Ensures that a transaction will run to completion as an indivisible unit, at the end of which either no changes have occurred to the database, or database has been changed in a consistent manner

Eg: A = 2000, B = 3000/- [local buffer DB]

Ti
R(A, a)
a - 500
write(A, a)
Read(B, b)
b = b + 500
write(B, b)

Transfer 500 from A to B.

After successful completion
A = 2000 - 500 = 1500/-
B = 3000 + 500 = 3500/- } 5000

So the data is in consistent state after all Tramactions done without any interrupt.

If suppose if power failure occurs at write(A, a)

then A = 1500
B = 3000 } → 4500 ≠ 5000.

so the data is in inconsistent data.

Consistency:- Correctness or ensures that if the database was in a consistent state before the start of a tramaction, then or termination, the database will also be in a consistent data state.

Sum (A,B) = Sum (A,B)

before transaction

10,000
-500

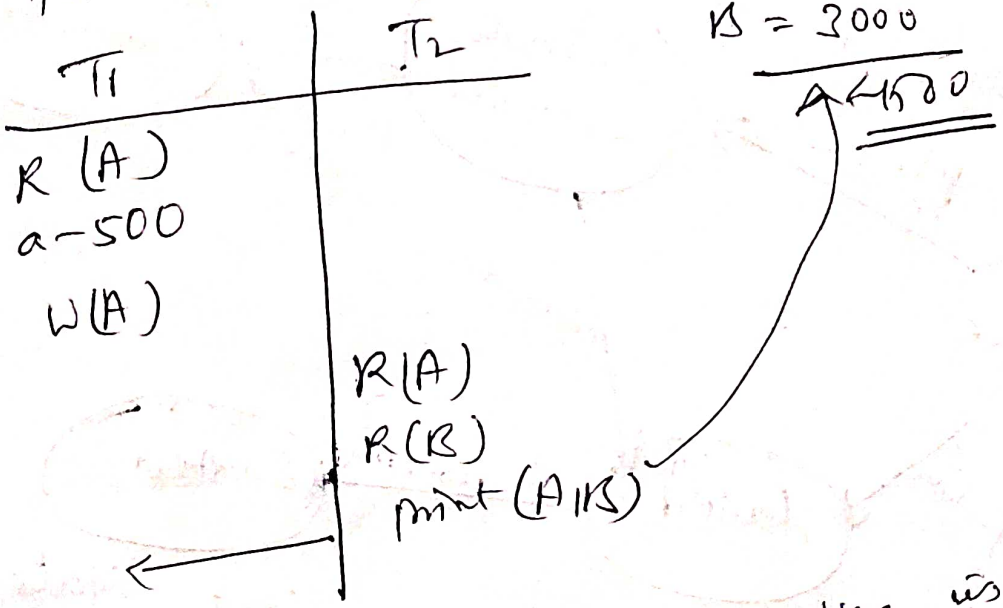
after transaction

10,500

Isolation:- Indicates that the actions performed by a transaction will be isolated or hidden from outside the transaction until it terminates.

A = 2000, B = 3000

A = 1500
B = 3000
4500



Actually the sum must be 5000, transaction is in not inconsistent state. because once T1 is completed his transaction no other transaction T2 must be allowed to perform any other action. otherwise data will be in inconsistent state: as stated above the example.

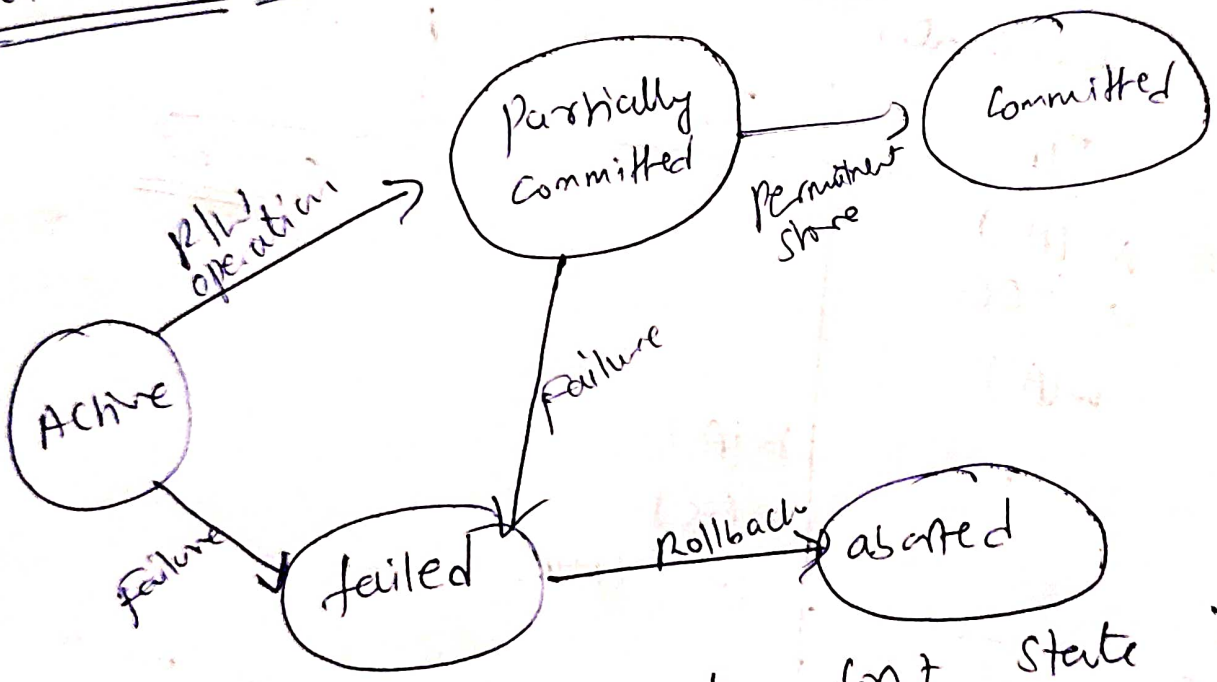
Durability:- All updates done by a transaction must become permanent.

commit

(or)

Ensures that the commit action of a transaction on its termination will be reflected in the database.

Transaction State:-



① **Active state :-** This is the first state in the life cycle of a transaction.

→ A transaction is called in an active state as long as its instructions are getting executed.

→ All the changes made by the transaction now are stored in the buffer in main memory.

(2) Partially committed state

→ After the last instruction of transaction has executed, it enters into a partially committed state.

→ After entering this state, the transaction is considered to be partially committed.

→ It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.

(3) Committed state:-

→ After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state.

→ Now the transaction is considered to be fully committed.

Note:- After a transaction has entered the committed state, it is not possible to rollback the transaction.

→ It is not possible to undo the changes that has been made by the transaction.

→ The only way to undo the changes is by carrying out another transaction called as compensating transaction that performs the reverse operation.

(4) **Failed state:-**

→ When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a failed state.

(5) **Aborted state:-**

→ After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.

→ To undo the changes made by the transaction, it becomes necessary to rollback the transaction.

→ After the transaction has rolled back completely, it enters into an aborted state.

Terminated state:- This is the last state in the life cycle of a transaction.

→ After entering the committed state or aborted state the transaction finally enters into a terminated state where its life cycle finally comes to an end.

Concurrent Execution:- It implies interleaving execution of operations of a tramaction

Benefits:-

(1) It helps in reducing waiting time

→ There may be a mix of tramactions running on a system, some short and long.

→ If tramactions run serially, a short tramaction has to wait for a preceding until long tramaction to complete, which can lead to unpredictable delays in running a tramaction.

→ If the tramactions are operating on different parts of the database, it is better to let them run concurrently, sharing CPU cycles and disk acces among them.

→ Concurrent execution reduces the unpredictable delays in running tramactions.

(2) Improved Throughput and resource utilization:-

→ A tramaction consist of many steps.

→ some involve I/O activity, others involve CPU activity.

→ CPU and the disk in a computer system can operate in parallel.

→ The parallelism of the CPU and I/O system can therefore be exploited to run multiple transactions in parallel.

→ The no. of transactions executed in a given amount of time, the processor and disk utilization also increase, i

→ the processor and disk spend less time idle, or not performing any useful work.

Schedule:- It represents the order in which instructions of a transaction are executed.

→ A schedule 's' of n transactions $T_1, T_2 \dots T_n$ is an ordering of operations of the transactions in chronological order.

or

When several transactions are executing concurrently then the order of execution of various instructions is known as schedule.

T_1	T_2
R(X)	
	W(X)

$T_1 \rightarrow T_2$

Types of schedule:-

Serial schedule:-

↳ Does not interleave the actions of any operations of different transactions.

↳ Always ensure a consistent state.

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

$T_1 \rightarrow T_2$ or $T_2 \rightarrow T_1$.

T_1 : read (A);
A := A - 50;
write (A);
read (B);
B := B + 50;
write (B)

T_2 : read (A);
temp := A * 0.1;
A := A - temp;
write (A);
read (B);
B := B + temp;
write (B);

A = 1000 B = 2000

T_1	T_2
read (A)	
A := A - 50	
write (A)	
read (B)	
B := B + 50	
write B	
	read (A)
	temp := A * 0.1
	A := A - temp
	write (A)
	read (B)
	B := B + temp
	write (B)

$T_1 = 855$

$T_2 = 2145$

3000

T_1 followed by T_2

T_1	T_2
	read(A) temp := A * 0.1 A := A - temp write(A) read(B) B := B + temp write(B)
read(A) A := A - 50 write(A) read(B) B := B + 50 write(B)	

850
2150
1 1
3000

T_2 followed by T_1

$T_1 = 850$, $T_2 = 2150$

T_1	T_2
read(A) A := A - 50 write(A)	
	read(A) temp := A * 0.1 A := A - temp write(A)
read(B) B := B + 50 write(B)	
	read(B) B := B + temp write(B).

Concurrently Executing of T_1 followed by T_2 & T_2 followed by T_1, T_2 by The B data will be in inconsistent state

To overcome this problem we have to go for complete schedule.

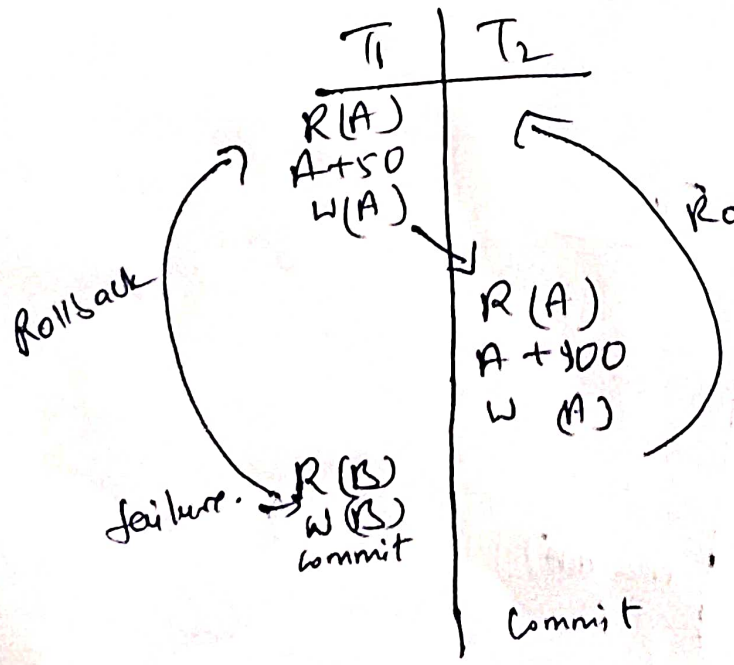
The no. of possible schedules for a set of n transactions is much larger than n!

② Complete schedule :- If the last operation of each transaction is either abort or commit.

T_1	T_2
R(A)	R(A)
W(A)	
commit	W(A)
	Abort

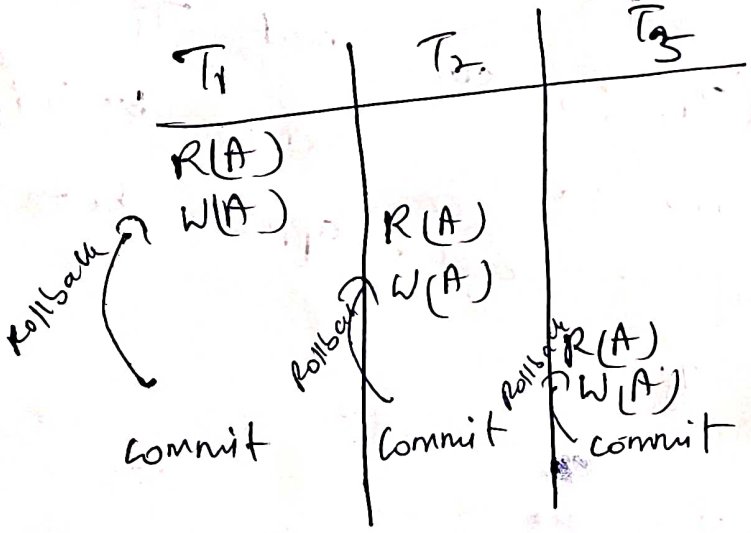
} Complete schedule.

③ Recoverable schedule :- It is one where for each pair of transactions (T_i, T_j) such that T_j reads a data item that was previously written by T_i, the commit operation of T_i should appear before commit operation of T_j.

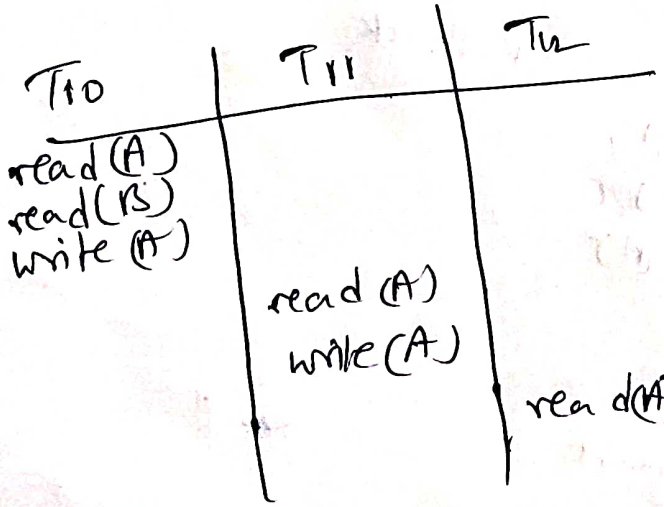


A recoverable schedule is one where, for each pair of tramachions T_i and T_j such mat T_j reads a data item previous written by T_i, the commit operation of T_i appears before the commit operation of T_j.

(4) cascadeless schedule:- Is one where for each pair of tramachions (T_i, T_j) such mat T_j reads a data item written by T_i, then the commit operation of T_i should appear me read operation of T_j.



or



→ Tramachion T_{10} writes a value of A that is read by tramachion T_{11}.

→ Tramachion T_{11} writes a value of A that is read by tramachion T_{12}.

→ suppose that, at this point T_{10} fails.

→ T_0 must be rolled back.

→ since T_{11} is dependent on T_{10}, T_{11} must be rolled back.

→ since T_{12} is dependent on T_{11}, T_{12} must be rolled back.

→ This phenomenon, in which a single tramachion failure leads to a series of tramachion rollbacks, is called cascading rollback

→ cascading rollback is undesirable, since it leads to the undoing of a significant amount of work.

→ It is desirable to restrict the schedules to those where cascading rollbacks cannot occur.

→ such schedules are called "cascadeless schedules."

→ A cascadeless schedule is one where for each pair of tramachions T_i and T_j

such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.

→ It is easy to verify that every cascadeless schedule is also recoverable.

Serializability:-

→ The database system must control concurrent execution of transactions, to ensure that the database state remains consistent.

→ A serializable schedule always leaves the database in consistent state.

→ A serial schedule is always a serializable schedule. In serial schedule, a transaction only starts when the other transaction finished execution.

→ Non-serial schedule needs to be checked for serializability.

→ A non-serial schedule of 'n' number of transactions is said to be serializable schedule, if it is equivalent to the serial schedule of those 'n' transactions.

→ There are 2 types of serializability.

(1) Conflict serializability
(2) view serializability

→ Conflict serializability

→ A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its non-conflicting operations.

Conflicting operations

Two operations are said to be in conflict, if they satisfy all the following 3 conditions.

(1) Both the operations should belong to different tranactions.
(2) Both the operations are working on same data item.
(3) Atleast one of the operation is a write operation.

ie Let S be a schedule in which there are two consecutive instructions I_i and I_j, of tranaction T_i and T_j, respectively where $(i \neq j)$.

→ If I_i and I_j refer to different data items, then we can swap I_i and I_j without

affecting the results of any instruction in the schedule.

→ If I_i and I_j refer to the same data item Q, then the order of two steps may matter.

→ Here we will use only read and write instructions

There are 4 cases that we need to consider

① I_i = read (Q), I_j = read (Q).

The order of I_i and I_j does not matter, since the same value of Q is read by T_i and T_j, regardless of the order.

② I_i = read (Q), I_j = write (Q). If I_i comes before I_j then T_i does not read the value of Q that is written by T_j in the instruction I_j.

→ If I_j comes before I_i, then T_i reads the value of Q that is written by T_j.

→ The order of I_i and I_j matters.

(3) I_i = write(Q), I_j = read(Q). The order of I_i and I_j matters for reasons similar to those of the previous case.

(4) I_i = write (Q), I_j = write (Q). Since both instructions are write operations, the order of these instructions does not affect either T_i or T_j.

→ We say I_i and I_j conflict if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation.

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Schedule - 1

Concurrency Control

Lock-based protocols:-

→ To ensure serializability is to require that data items be accened in a mutually exclusive manner, i.e while one transaction is accessing a data item, no other transaction can modify that data item.

→ The most common method used to implement this requirement is to allow a transaction to accen data item only if it is currently holding a lock on that item.

Locks:-

→ There are various modes in which data item can be locked.

(1) Shared lock:- If a transaction T_i has obtained a shared-mode lock (denoted by S), on data item Q, then T_i can read, but cannot write Q.

(2) Exclusive lock:- If a transaction T_i has obtained an exclusive-mode lock (denoted by X) on data item Q, then T_i can perform both read and write Q.

→ A tramachion requests a shared lock on data item Q by executing the lock-S(Q).

→ A tramachion request a exclusive lock through the lock-X(Q).

→ A tramachion can unlock a data item Q by unlock (Q). instruction.

T1 :- lock-X(B);
read(B);
B := B-50;
write(B);
unlock(B);
lock-X(A);
read(A);
A := A+50;
write(A);
unlock(A).

Tramachion T_1

T2 :- lock-S(A);
read(A);
unlock(A);
lock-S(B);
read(B);
unlock(B);
display(A+B).

tramachion T_2.

→ To access a data item, tramachion T_i must first lock that item.

→ If the data item is already locked by another tramachion in a incompatible mode, the concurrency control manager will not grant a lock until all incompatible locks held by another tramachion have been released.

T_1	T_2	Concurrency control Manager
lock-X(B)		grant-X(B, T_1)
read(B) B := B-50 write(B) unlock(B)		
	lock-S(A)	grant-S(A, T_2)
	read(A) unlock(A) lock-S(B)	
		grant-S(B, T_2)
	read(B) unlock(B) display(A+B)	
lock-X(A)		grant-X(A, T_2)
read(A) A := A+50 write(A) unlock(A)		

The above table explains how the concurrency control manager is assigned to locks and released the locks b/w two transactions T_1 & T_2.

Granting of locks:-

→ suppose a transaction T_2 has a shared-mode lock on data item,

→ Another transaction T_1 requests an exclusive mode lock on the data item.

→ T_1 has to wait for T_2 to release the shared-mode lock.

→ Meanwhile, a transaction T_3 may request a shared-mode lock on the same data item.

→ The lock request is compatible with the lock granted to T_2, so T_3 may be granted shared-mode lock.

→ At this point T_2 may release the lock, but still T_1 has to wait for T_3 to finish.

→ Suppose T_4 new transaction requests a shared mode lock on the same data item, and it is granted the lock before T_3 is released.

→ In this process T_i never gets the chance of exclusive-mode lock on the data item.

→ This situation is called starvation.

→ To avoid starvation of transactions by granting locks in the following manner.

⇒ When a Transaction T_i requests a lock on data item Q in a particular mode M, the concurrency control Manager grants lock provided that

(1) There is no other transaction holding a lock on Q, in a mode that conflicts with M.

(2) There is no other transaction that is waiting for a lock Q, and that made its lock request before T_i.

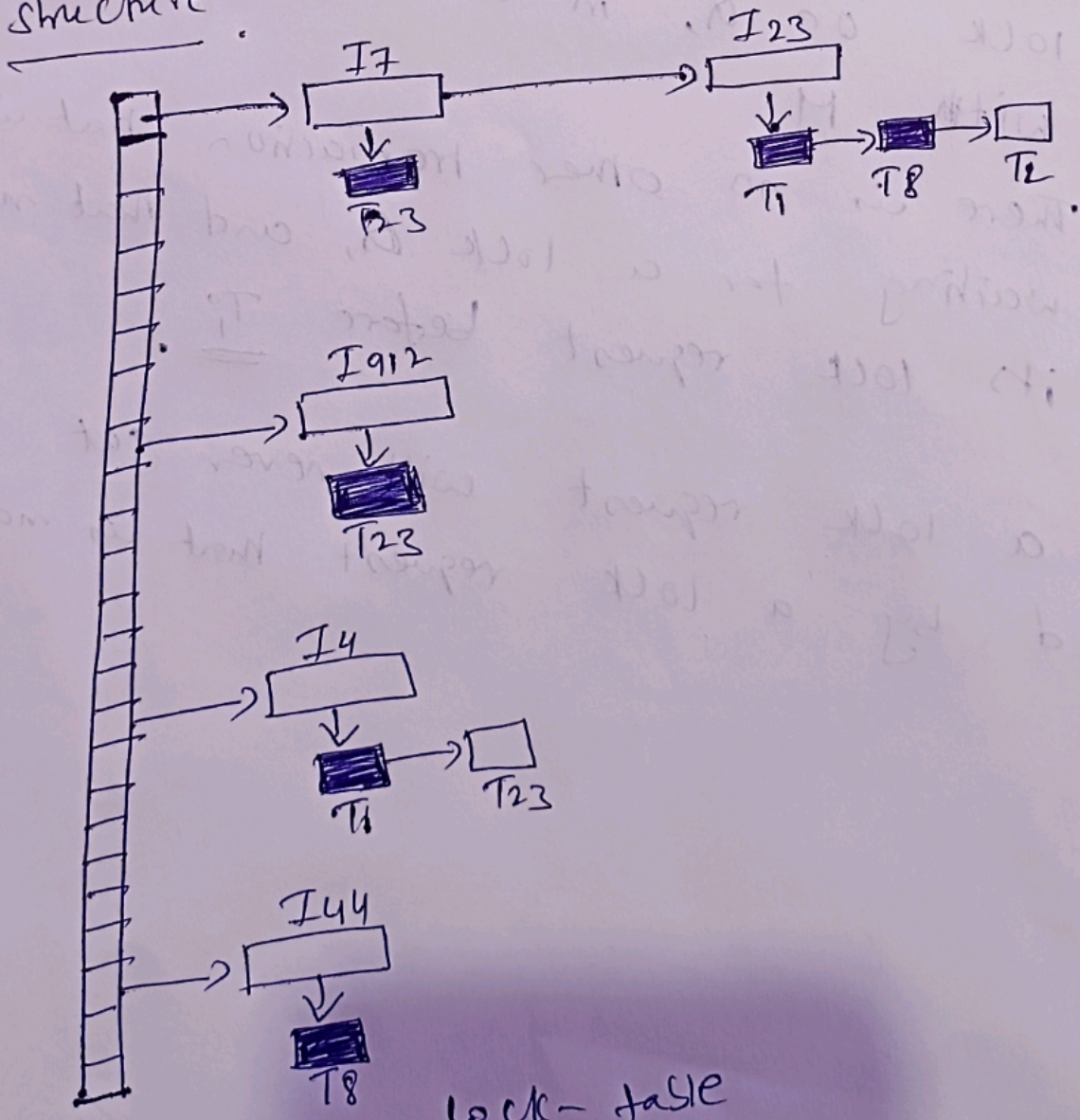
Thus, a lock request will never get blocked by a lock request that is made later.

Implementation of locking

→ A lock manager can be implemented as a process that receives messages from transactions and sends messages in reply.

→ The lock-manager process replies to lock-request messages with lock-grant messages, or with messages requesting rollback of the transaction (in case of deadlock).

→ The lock manager uses the following data structure.



lock-table

→ For each data item that it currently locked, it maintains a linked list of records.
→ one for each request, in the order in which the requests arrived.
→ It uses a hash table indexed on the name of data item.
→ To find the linked list if any for a data item.
→ This table is called the lock-table.
→ Each record of the linked list for a data item notes which transaction made the request, and what lock mode is requested.
→ The record also notes if the request has currently been granted.

In the above lock-table.
→ Table contains locks for 5 different data items.
→ I4, I7, I23, I44 and I912.
→ lock-table uses overflow chaining.
→ There is a linked-list of data items for each entry in the lock-table.
→ There is also a list of transactions that have been granted locks, waiting for locks.

→ Granted locks are filled in black rectangles

→ waiting request are the empty rectangles. □.

→ T_{23} has been granted locks on I_{9} & I_2 and I_7. and it is waiting for I_4.

Lock-Manager processes requests this way

→ When a lock request menage arives, it adds a record to the end of the linked list for the data item, if the linked list is present.

→ Otherwise it creates a new linked list containing only the record for the request.

→ It always grants the first lock request on a data item.

→ If the transaction requests a lock on a item on which a lock has already been granted, the lock manager grants the request only if it is compatible with all earlier requests.

→ If the earlier requests have been granted already.

→ otherwise the request has to wait until lock is released.

Validation based protocols:-

→ Each transaction T_i executes in two or three different phases in its lifetime, depending on whether it is read-only or an update transaction.

→ The phases are

① **Read phase:-** During this phase, the system executes transaction T_i.

→ It reads the value of the various data items and stores them in variables local to T_i.

→ It performs all write operation on temporary local variables, without updates of the actual database.

② **validation phase:-** Transaction T_i performs a validation test to determine whether it can copy to the database the temporary local variables that hold the results of write operation. without causing violation of serializability.

(3) Write phase:- If transaction T_i succeeds in validation step 2 done,

→ The system applies the actual updates to the database.

→ The system rolls back T_i.

→ Each transaction must go through three phases.

To perform the validation test, transaction T_i has 3 different timestamps.

(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.

The validation test for T_j requires that, for all transactions T_i with $TS(T_i) < TS(T_j)$ one of the 2 conditions must hold

① Finish(T_i) < Start(T_j).

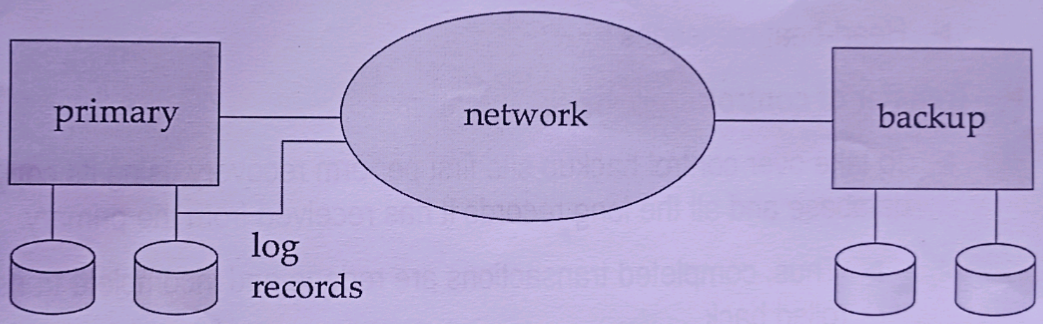
→ Since T_i completes its execution before T_j started, the serializability order is indeed maintained.

② The set of data items written by T_j does not intersect with the set of data items read by T_j, and T_i completes its write phase before T_j starts its validation phase.

Start (T_j) < Finish (T_i) < validation (T_j)

Remote Backup Systems

- Remote backup systems provide high availability by allowing transaction processing to continue even if the primary site is destroyed.



Remote Backup Systems (Cont.)

- **Detection of failure**: Backup site must detect when primary site has failed
  - to distinguish primary site failure from link failure maintain several communication links between the primary and the remote backup.
  - Heart-beat messages
- **Transfer of control**:
  - To take over control backup site first perform recovery using its copy of the database and all the long records it has received from the primary.
    - Thus, completed transactions are redone and incomplete transactions are rolled back.
  - When the backup site takes over processing it becomes the new primary
  - To transfer control back to old primary when it recovers, old primary must receive redo logs from the old backup and apply all updates locally.

Remote Backup Systems (Cont.)

- **Time to recover**: To reduce delay in takeover, backup site periodically proceses the redo log records (in effect, performing recovery from previous database state), performs a checkpoint, and can then delete earlier parts of the log.
- **Hot-Spare** configuration permits very fast takeover:
  - Backup continually processes redo log record as they arrive, applying the updates locally.
  - When failure of the primary is detected the backup rolls back incomplete transactions, and is ready to process new transactions.
- Alternative to remote backup: distributed database with replicated data
  - Remote backup is faster and cheaper, but less tolerant to failure
    - more on this in Chapter 19

Remote Backup Systems (Cont.)

- Ensure durability of updates by delaying transaction commit until update is logged at backup; avoid this delay by permitting lower degrees of durability.
- **One-safe:** commit as soon as transaction's commit log record is written at primary
  - Problem: updates may not arrive at backup before it takes over.
- **Two-very-safe:** commit when transaction's commit log record is written at primary and backup
  - Reduces availability since transactions cannot commit if either site fails.
- **Two-safe:** proceed as in two-very-safe if both primary and backup are active. If only the primary is active, the transaction commits as soon as is commit log record is written at the primary.
  - Better availability than two-very-safe; avoids problem of lost transactions in one-safe.