

DBMS Lab

MySQL

* MySQL is a database system used on the web and run on a server.

* MySQL uses standard SQL (standard structured query language)

* MySQL is very first, reliable and easy to under

- use

SQL

* SQL lets you access and manipulate databases

* It allows all operations. It being used in MySQL, Oracle, SQL server, PostgreSQL

constraints

PRIMARY key $\rightarrow$ id INT PRIMARY key (Auto-Increment)

FOREIGN key $\rightarrow$ FOREIGN KEY (dept-id) REFERENCES
department (id)

CHECK $\rightarrow$ age INT, CHECK (age $\geq$ 18)

ENUM $\rightarrow$ gender ENUM ('male', 'female');

create and insert $\rightarrow$ demo lab/lecture 2

create database sm;

create table department (
id INT primary key Auto-Increment,

name varchar(50) Not Null Unique,

location varchar(100) Not Null

time
create-date date Not Null

);

Insert into department (id, name, location, create-date)
values (NULL, 'ESE', '2nd floor', '2010-04-10');

Alter table

te. Lecture-3

varchar(100)

students(id, name, email, address,
phone)

Add multiple column

(Alter table students Add email varchar(255) not-null after name;)

ALTER Alter table students ADD (phone varchar(50),
email varchar(255) not-null after name;)

multiple column @ after
use kora jay na

After roll;

Modify existing column

ALTER table email students modify email varchar(150);
email int(11);

Rename column and change data types:-

Alter table students change phone contact varchar(20);

Drop column :-

Alter table table.name
Drop column column.name;

ADD and DROP primary key

Alter table students

ADD primary key (id);

Alter table students

DROP primary key;

Auto-increment

Auto-increment

disable

Alter table students

→ ADD

→ DROP column

→ modify

→ change @

→ ADD primary key (id)

→ DROP primary key

→ Rename to new-name

* ADD Foreign key

→ Alter table students

ADD constraint FK-student

Foreign key (dept-id) references department (id)

⇒ Delete (Pk) ⇒

alter table student

drop constraint foreign key FK_students;

✱ Add unique constraint

⇒ alter table table-name

ADD unique (column-name);

— Drop unique

alter table table_name

drop index column name;

✱ Add default

Alter table (students)

alter email set Default 'abc@example.com';

Remove

alter table students

alter email drop default

table rename

→ Alter table table-name
Rename to new-table-name

update data

update table name

set column1 = value1, column2 = value2

where id = 1; → condition

Ex:- v.vit

delete data

provide 10% salary increment

update teachers

set salary = salary + (salary * 0.1)

where designation = 'Assistant professor'

Delete data

Syntax: Delete From table-name where condition;

Delete From students where id=1;

It means to delete the record from the table where the condition is satisfied.

Example: Delete From students where id=1;

From table-name

[where condition]

[Example: Delete From students where id=1;

[where condition]

[Example: Delete From students where id=1;

[where condition]

Example: Delete From students where id=1;

SELECT STATEMENT (lecture-4)

* Select is a SQL command used to retrieve or read specific data from one or more tables in a database

* It allows users to specify columns, apply condition and filter the result according to their needs.

Syntax:-

SELECT column1, column2

FROM table_name

[WHERE condition]

[GROUP BY column1, column2]

[HAVING condition]

[ORDER BY column1, column2]

[LIMIT row-count];

* SELECT student.name, teacher.name FROM students, teacher;

Select Example Lecture-4

Students (id, name, phone, roll, email, address, dept-id)

Teachers (id, name, designation, city, salary)

* Retrieve specific column

select name, phone from students;

* Retrieve all column

select * from students

* Distinct :- it is used to remove duplicate values from the result set

select distinct column1, column2 from table name;

* Aggregate Functions

gives maximum value

* count, Avg, MAX, MIN, SUM

select count (distinct address) from students;

00.14 9269
00.015 9269
00.000 9269

Where clause

* The where clause is used to filter rows/records based on condition.

* Where is used with update, delete, select

Syntax:-
select column, column, from table name
where condition;

— x —
Exercises

① select name, designation from teachers
where salary > 50,000;

② select count(id) of st from students;

③ select count(id) from students where
address = 'dhaka';

* select * from students where not cpga = 4.00;

cpga 4.00
min 3.00
show 20%

order by by default ascending order
ascending (Asc)
descending (Desc)

↓ The order by keyword use to sort the result-set in ascending or descending order

* By default, it sorts the records in ascending order

* order by descending order

* select * from students order by desc regpa desc;
(also (order by name, address);

Exercise

① select * from teachers order by salary desc;

② select * from student order by name, city;
name asc, city asc;

Exercises

- ① Select * from teacher where salary ≥ 40000 AND salary ≤ 60000 AND Address = 'Dhaka'.
- ② select * from teacher where salary ≥ 70000 AND (designation = 'Assistant professor' OR designation = 'Lecturers')

Simple joint

Select students.name, students.roll, students.departments.name
From students, departments where
where students.dept_id = department.id

for not equal ($<>$)

between and

In

select * from students where address In ('Dhaka', 'sylhet')

question problem solve 2022

2

① select book-title, Author-name, name from books,
~~borrowers~~, Authors, publisher

Where books.author-id = Authors.author-id AND
books.publisher-id = Publisher.publisher-id;

(not equal →
equal =)

② select * from ~~bro~~ borrowers where borrower-id Not
In (select borrower-id from borrow-book);

* select author-name from authors where author-name
LIKE '_e%';

⑥ select book-title, Author-name from books, Authors
where books.~~book~~ Author-id = Authors.Author-id;

⑦ insert into writers (writer-name, writer-address)
select ~~author~~ writer-name, ~~author~~ address from authors;

5) ~~select books.book-title, count(borrow-book.borrower-id)
from books, borrow-book where
borrow-book.borrower-id = books.book-id~~

5) select books.~~book-title~~^{id}, count(borrow-book.~~book-id~~^{id})
from books, borrow-book where books.book-id = borrow-book.
group by borrow-book.book-id
order by count(borrow-book.id) desc

6) update borrowers set phone = Replace(phone, '337', '999')
where phone LIKE '%337%';

7) select borrowers.*, count(borrow-book.id) from
borrow-book, borrowers, where
borrow-book.borrower-id = borrowers.borrower-id
group by borrower-id
having count(count(borrow-book.id)) > 5;

8) ~~update borrowers set phone = 1337
where phone = 1337;~~



values

~~ADD current time~~

alter table department add column ^{date times} timestamp TIME
default CURRENT_time;

alter table department add column ^{join date} date time Not NULL
Default current timestamp;

Mid 2nd

① Insert into employee (name, designation, email, salary) values

① create table employee (
(niloy, 'Developer', 'n@gmail', 3000.00),
(Bishal, 'Developer', 'B@gmail', 6000.00);

② alter table add phone varchar(50) Not NULL unique
after email;

③ update employee set salary = salary + (salary * 0.1);

④ Delete from employee where designation = 'Developer';
↳ case case sensitive

Final Session - 2021 - 22

① Select learners. * From learners, courses, enrollments where enrollments.students_id = students.id
AND enrollments.courses_id = courses.id

AND courses.title = 'database';

② Select count (distinct students_id) From enrollments; (At least 1 course enrollment)

③ Select count (student_id) From enrollments where course_id = 2;

④ Select * From instructors where designation = 'senior' AND salary > 50,000;

⑤ Select * From learners order by city;

⑥ update instructors set salary = salary + (salary * 0.05) where salary < 50000;

(vii) ~~select total-marks from assessments where~~
~~id = 3;~~

(viii) select SUM(obtained-marks) from results
where id = '3';

(ix) select * from assessments;

(x) select courses.* from courses, ~~teacher~~ where
instructor-id = 1 AND course-level = 'basic';

(xi) select * from employees where
salary > (select avg(salary) from employees);

————— X —————

DBMS class-1

Database:- Database is a collection of ^{related} data and data is a collection of facts and figures that can be process to produce information.

DBMS:- A DBMS is a software that allows creation, definition and manipulation of database, Dbms is actually a tool used to perform any kind of operation on data. in database, also provides protection and security to database.

why use DBMS

- * To develop software application less time
- * Data independent and ~~effe~~ efficient use of data
- * For more security of data
- * To use user friendly declarative query language,
- * For uniform data administration,

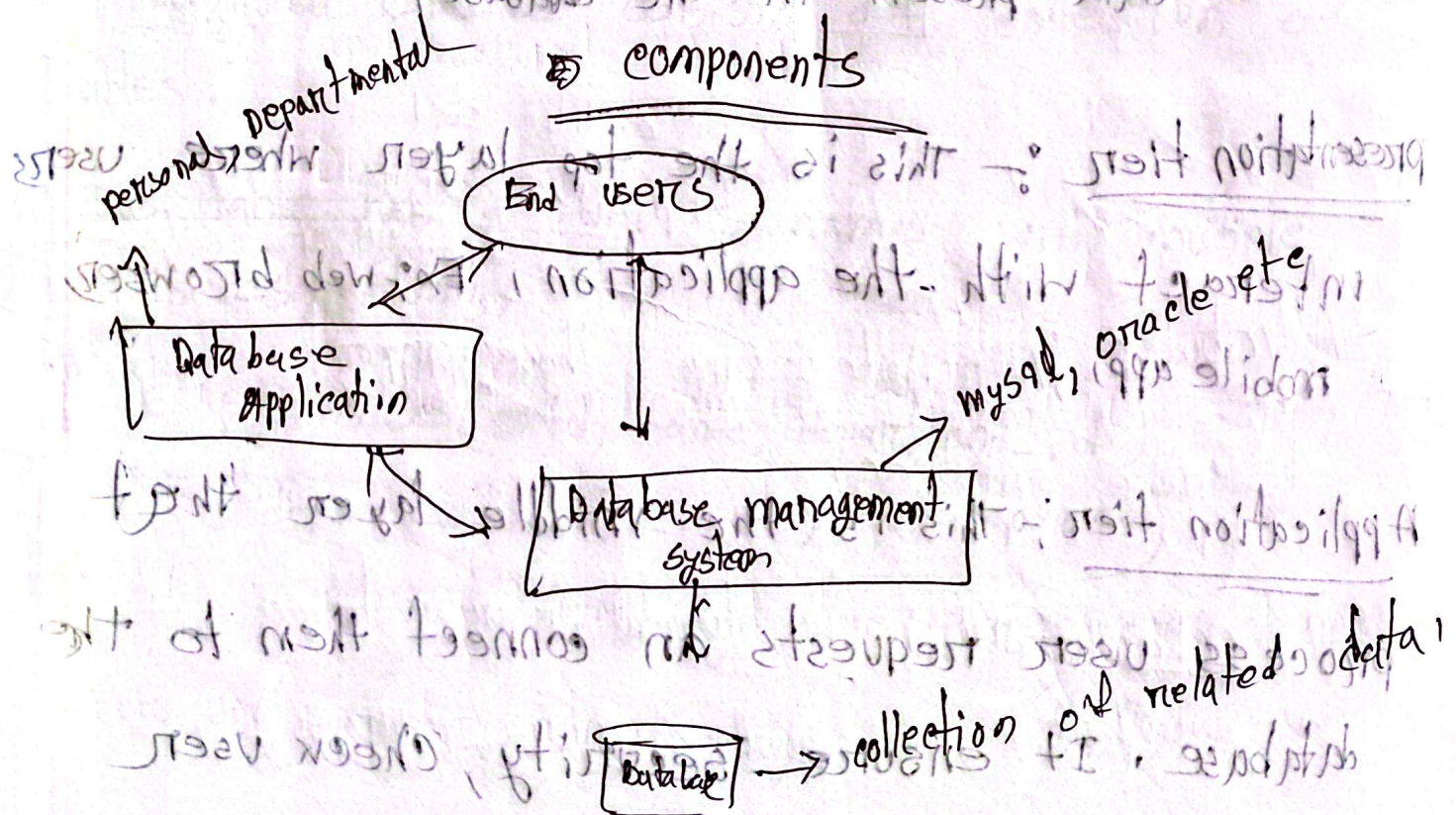
Characteristics of DBMS (advantage)

- * Real world Entity :- A modern DBMS is more realistic and uses real-world entities to design its architecture.
- * Relation based table :- DBMS allows entities and relations among them to form table.
- * ACID properties :- DBMS follows the concepts of Atomicity, consistency, isolation, and durability (ACID).
- * Less controlling Redundancy
- * providing storage structure for efficient query process
- * providing backup and Recovery of data
- * providing multiple user interfaces
- * restricting unauthorized Access
- * providing consistency consistency is a

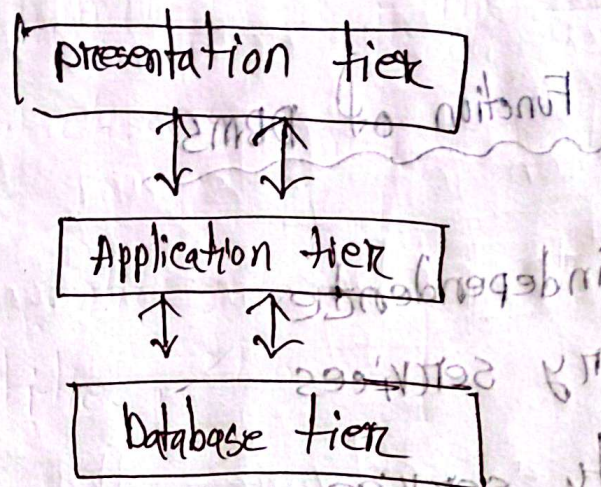
process state where every relation in a database remains consistent.

Function of DBMS

- * provides data independence
- * provides Recovery services
- * provides utility service
- * control Redundance
- * provides a clear and logical view of the process that manipulates data



3-tier Architecture of DBMC



A 3-tier Architecture separates its tiers each other on the complexity of the users and how they use the data present in the database.

presentation tier :- This is the top layer where users interact with the application, Ex: web browser, mobile app.

Application tier :- This is the middle layer that process user requests and connect them to the database. It ensure security, check user

input : Ex: web server, APIs

Databases :- This is the bottom where all data is stored and managed. It handles data retrieval, Insert, updates and security. Ex: mysql, ~~orag~~ oracle.

Lecture - 2

Relational DB

* A Relational Database consists of collection of tables.

* Each table has an unique name with multiple attributes/columns,

Ex :- COURSES (id, name, department_id)

= department (id, d.name, location, phone)

(It is a logical representation of a table)

Entity :- An Entity can be real world object, either animate or inanimate. Ex: in school database students, teachers, classes

Entity set :- An entity set is a collection of similar type of entities.

Entity type :- collection of entities which have common attributes.

(Student) → entity type

Id	name	Age
1	Prity	18
2	Sohan	21

→ entity

entity set

(1, 2) → Id

Attributes :- Attributes are those properties which describe entities.

schema :- It is a logical representation of a database. Ex: Student (id, name, address)

Instance :- change able data table. Instance change when Addition, deletion and updates.

Keys :- A set of attributes that collectively identify an entity in an entity set.

Super

Super key :- A set of attributes that collectively identify an entity in an entity set.

Ex:- teacher (id, name, phone, address)

SK = {id}, {phone}, {id name}, {phone, address}

candidate key :- A minimal super key is called a candidate key.

candidate key: (id, phone)

CK = {id}, {phone}

primary key :- one of the candidate key that is selected by designer.

selected by designer

PK = {id}

Foreign keys:-

Foreign keys are the columns of a table that points to the primary key of another table. Ex:- department_id

$SK \rightarrow CK \rightarrow PK \rightarrow FK$

~~X~~

Example: class (name, address)

Relationships = correlations among entity

Example: employee work-for company

Weak entity

The entity always dependent on another entity

Dependent depends on Employee

Attributes details

• Simple :- cannot be divided further

Ex:- id, phone, email

Single valued :- It hold only one value

Ex:- id, dob

multi-valued :- It hold more than one values

Ex:- phone, email

composite :- composite attributes are made of more than one simple attribute, Ex:- name → first name, last name

complex :- combination of composite and multi-valued attribute

Ex :- Address → present (post code, district), permanent (same pattern)

derived :- which can be derived from another attribute or set of attributes

Ex :- age $\xrightarrow{\text{derived}}$ dob

Ex:- Employees (id, name, email, phone, address, dob; gender, designation, salary)

complex = address

composite = name

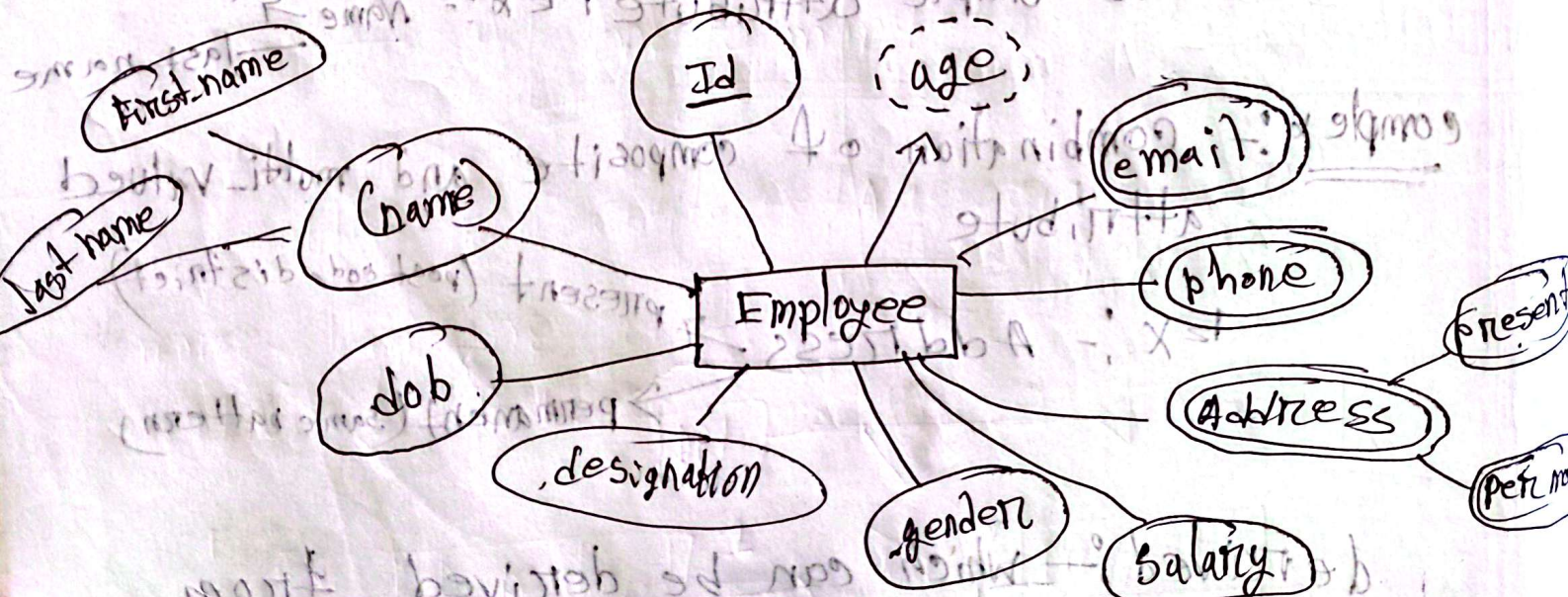
multi valued = email, phone

stored = dob

derived = age, number of employee, cpa

single valued = designation, salary, id, gender

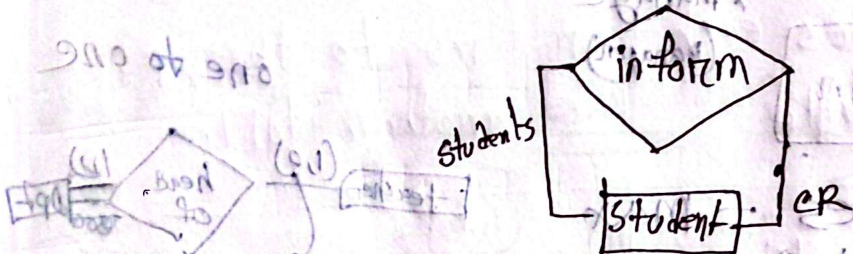
simple single valued = Id



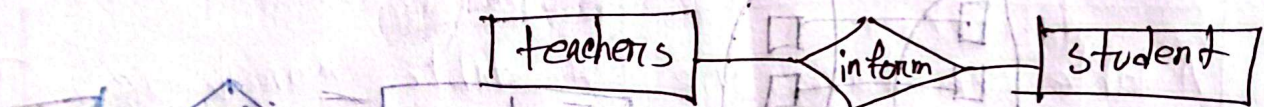
Relationship

Degree :-

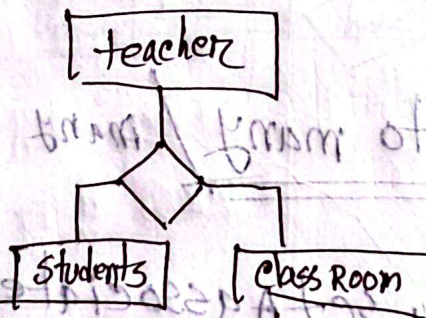
- unary (1 degree) : ER Inform students



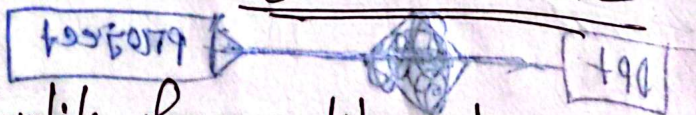
binary (2 Degree) : teachers inform students



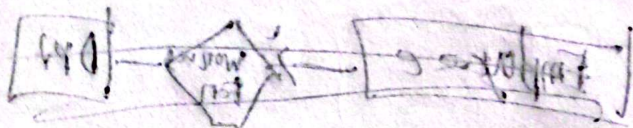
ternary (3)



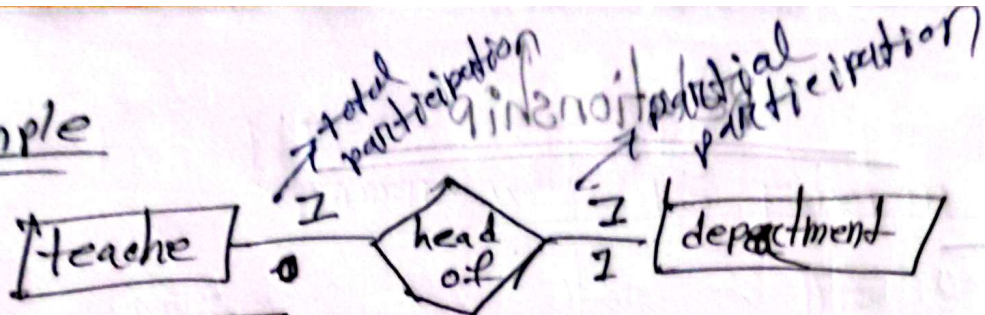
one to one



one entity from entity set A can be correlated with at most one entity of entity set B



Example



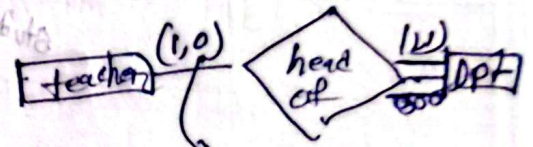
mapping cardinalities

participation = min

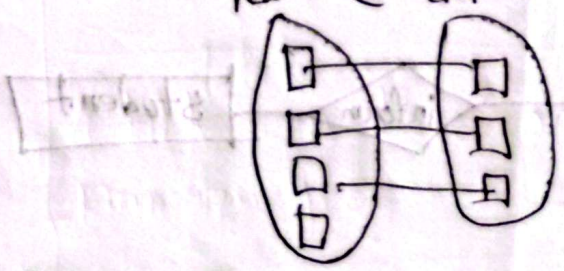
mapping cardinalities = max
(describing binary relationship sets)

Teacher(1,0) department(1,1)

one to one



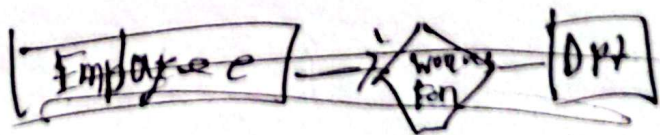
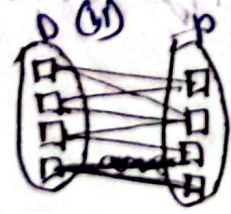
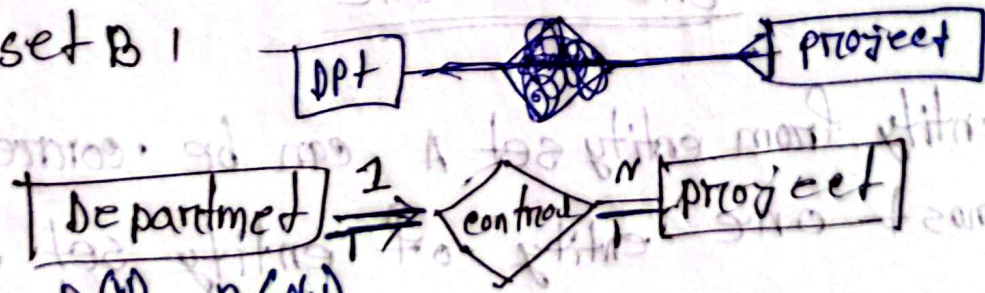
300 is teacher head of dept 222



one to many / many to one

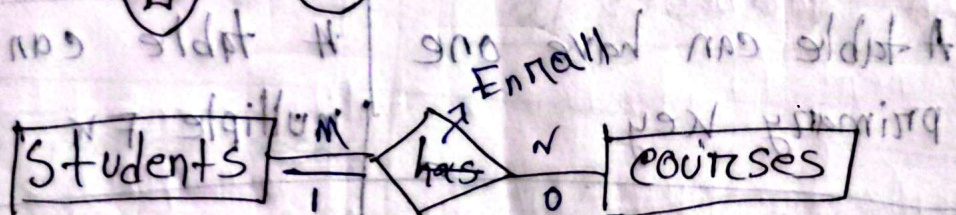
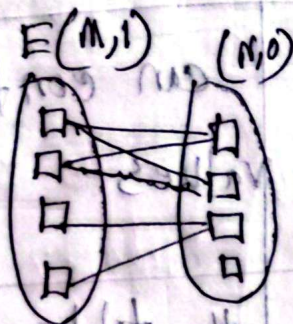
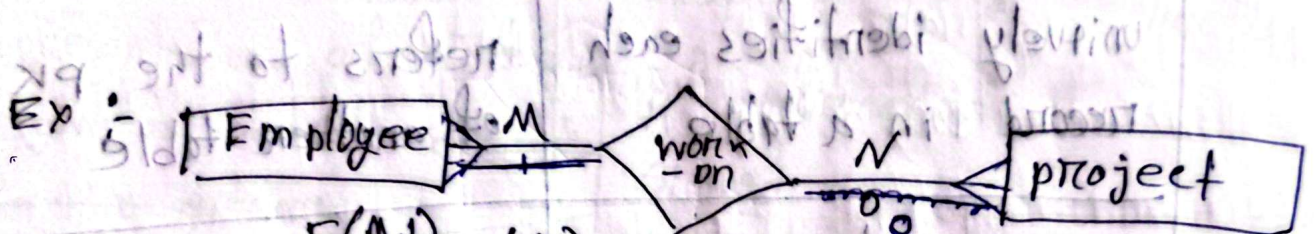
one entity from entity set A associated with more than one entity from entity set B

Ex:-



Many to many

one entity from A can be associated with more than one entity from B,



one student can have multiple courses

Diagram of primary key VS Foreign key

primary key VS Foreign key

Difference between primary key and foreign key

<u>primary key</u>	<u>Foreign key</u>
uniquely identifies each record in a table.	refers to the PK of another table
can not contain NULL values	can contain NULL values
A table can have one primary key	A table can have multiple FK
student-id in the students table	student-id in the class & Enrollments table

Schema vs instance

schema

A schema is a visual representation of a database.
It give us overall description of the database

Ex: Users (id, name, email)

instance

Instance is a collection of information that is stored in a database at a particular moment in time.

Ex:-

<u>Users</u>		
id	name	email
1	x	x@gmail.com
2	y	y@gmail.com

It give us overall description of the database.

and represent the real-world data in the database

managed by Database Administrator

Managed by users and application

DBMS vs DB

DBMS

It is a software that allows creation, definition and manipulation of database.

Use to manage and manipulates data in databases.

components of DBMS are:
DDL, query process,
storage manager etc.

Ex: - A university database
students and courses table

DB

It is a collection of Data.

use to Stores and organizes data

The components of DB are :-
table, row, col.
columns, relationships

MySQL, Oracle,
MongoDB.

Entities of cricket tournament management system

Emp. player table

player (player-id, name, dob, gender, nationality, role,

team (team-id, team-name, coach-id, captain-id)

match (match-id, team-id, team2-id, venue-id, match-date,
match-time, status, result)

tournaments (t-id, name, start-date, end-date, location, sponsor)

Venue (venue-id, city, capacity)

umpire (umpire-id, name, dob, country)

coach (coach-id, name, dob, nationality, team-id)

sponsor (sponsor-id, name, industry, contribution-amount)

player-stats (player-id, matches-played, runs-scored,
wickets-taken, batting-average, bowling-average)

match-score (score-id, match-id, player-id, wickets, balls-
overs-bowled)

Match-schedule (schedule-id, tournament-id, match-id, s-date, s-time)

Ticket (ticket-id, match-id, price, seat-number, buyer-name, buyer-email)



Player-injury (in-id, player-id, injury-type, injury-date, recovery-date)

Employee (Employee-id, name, phone-number, dob, email)

Fan (fan-id, name, email, phone-number, favorite-team-id)

ER model (diagram) lecture-3

We do not show foreign key in ER diagram

step-1 :- Identify Entities Required

step-2 :- Identify the attributes and primary key for each entity

step-3 :- Identify the relationship needed

step-4 :- Identify cardinality ratio and participation

step-5 :- Draw the diagram

01/07/2025 12:11

Weak entity



- * every weak entity have to have total participation
- * weak entity does not contain primary key

Lecture-3

ER model (diagram)

* We do not show Foreign key in ER Diagram

* step-1 :- Identify ^{the} Entities Required

* step-2 :- Identify the Attributes and primary key for each Entity

* step-3 :- Identify the Relationship needed

step-4 :- Identify cardinality ratio and participation

step-5 :- Draw the Diagram

* Facebook-DB format using ER Model :-

Step-1 : Identify entities :- user-profile, post, post-comment, post-live

Step-2 : Attributes

user-profile (u-id, name, email, password, DOB, age, phn)

post (p-id, text-content, image-url, videos, created-time, modification-time)

post-comment (pc-id, text-content, timestamp, etime)

post-live (pl-id, p-time)

Step-3 : Relationship

user-profile $\times$ user-profile $\xrightarrow{N}$ friendship $\xrightarrow{N}$ user-profile
user-profile $\times$ posts $\xrightarrow{N}$ user-post

④ user-profile ① can ② post-like

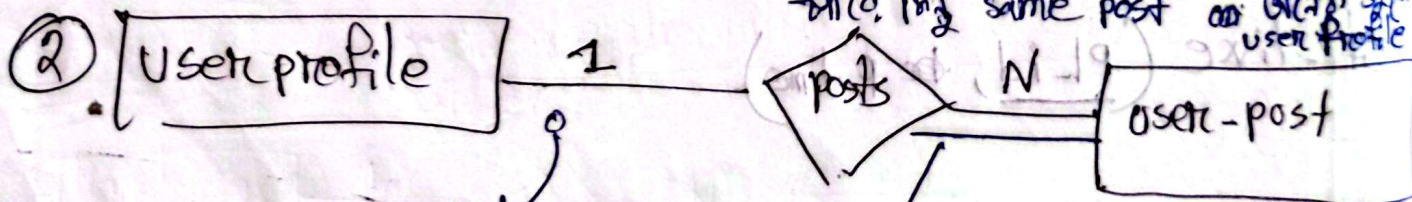
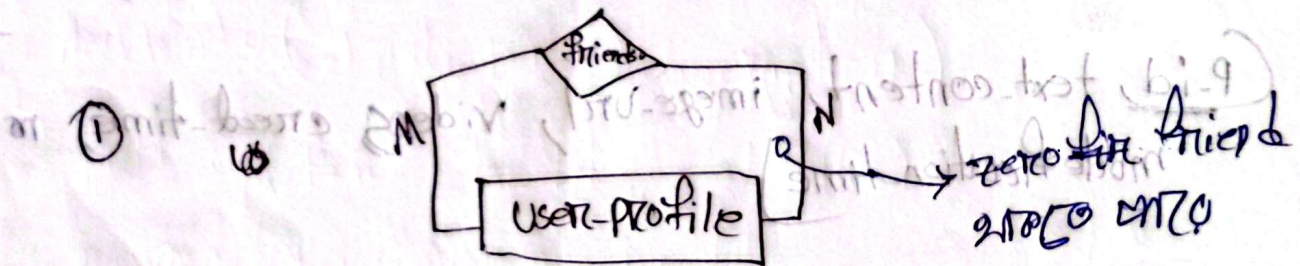
:-

* user-profile ① comments ② post-comment

* user-post ① has ② post-comment

* user-post ① has ② post-like

Step-4 :- cardinality ratio and participation



एक ही user-profile

same post कर सकता है

आप, यह कर सकते हैं

एक ही post

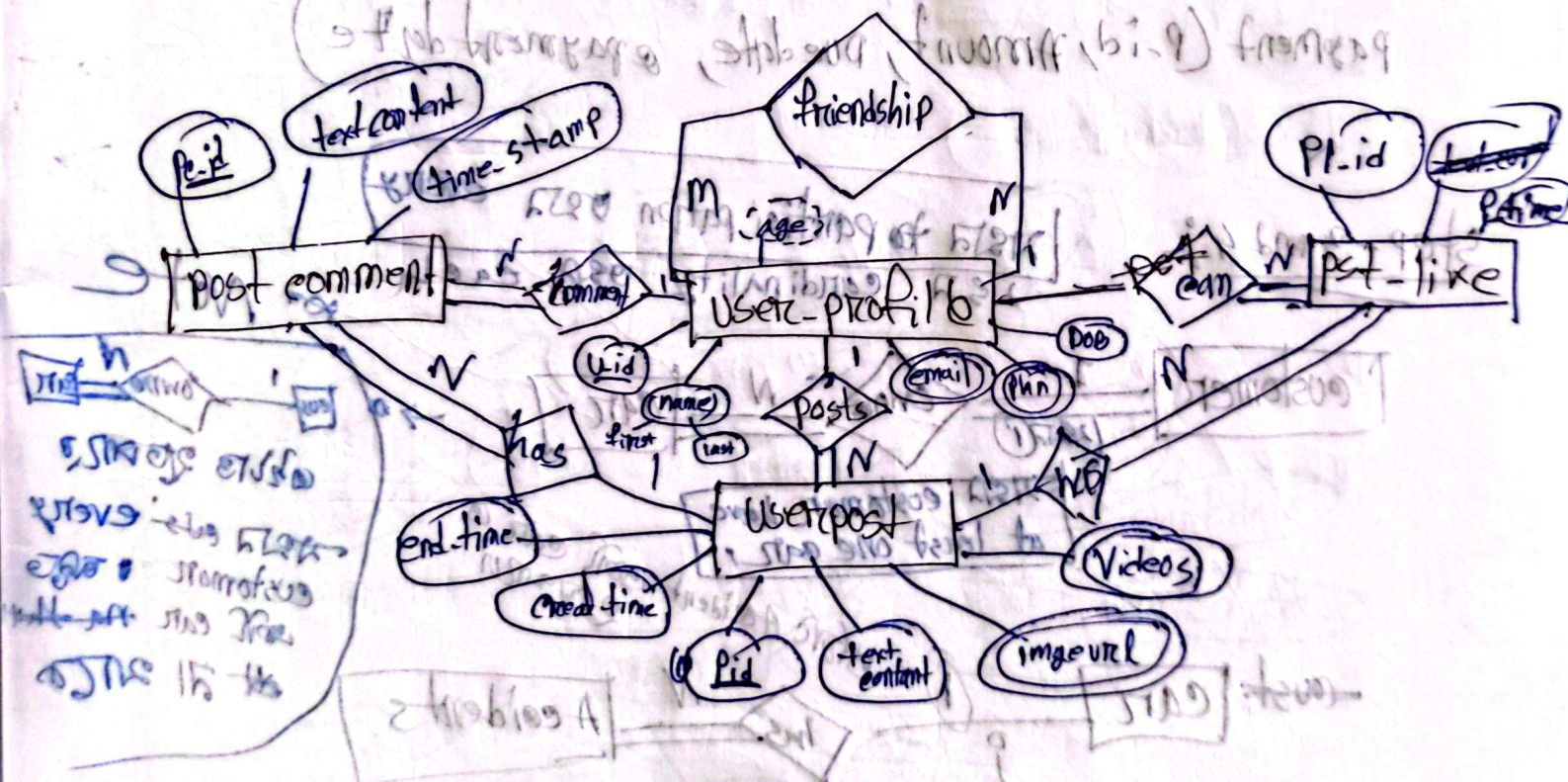
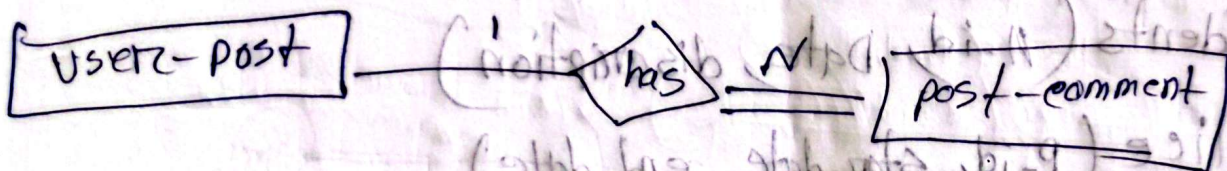
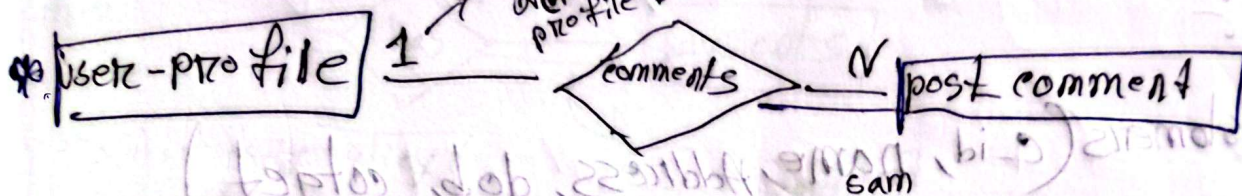
user-profile कर सकता है

एक user-profile

25 2)



एक निम्नलिखित
 Same like अनो अर्थि user
 ए पोस्ट लाइन न
 post-line user profile
 को टैग निम्नलिखित न



Lecture slide - 3.1

Assignment - 3

Mid - 1
2021-2022

Step-1:- customers, cars, Accidents, police, payment

Step-2:-

customers (c-id, name, Address, dob, contact)

cars (car-id, model-name, year)

Accidents (A-id, Date, description)

police (P-id, start-date, end-date)

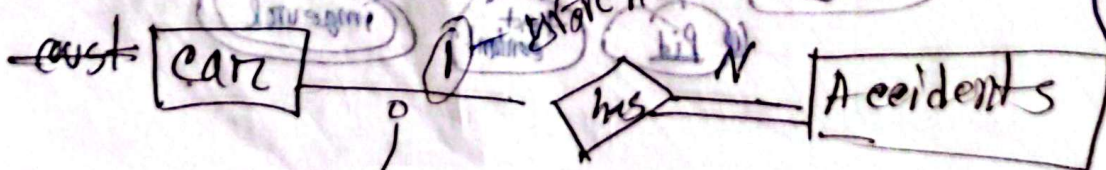
payment (P-id, Amount, Due date, & payment date)

Step-3 and 4:-

maintain participation every
cardinality every each

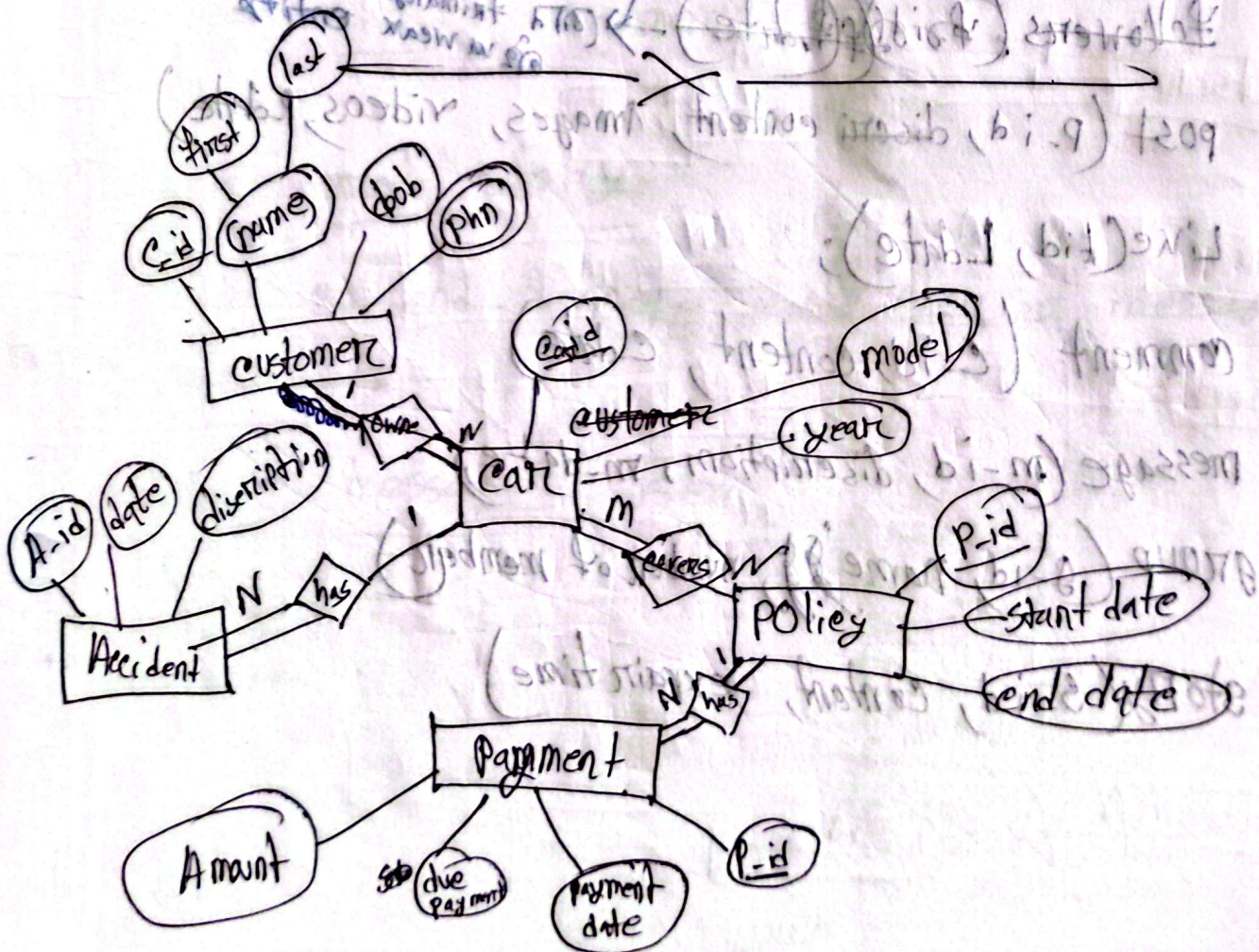
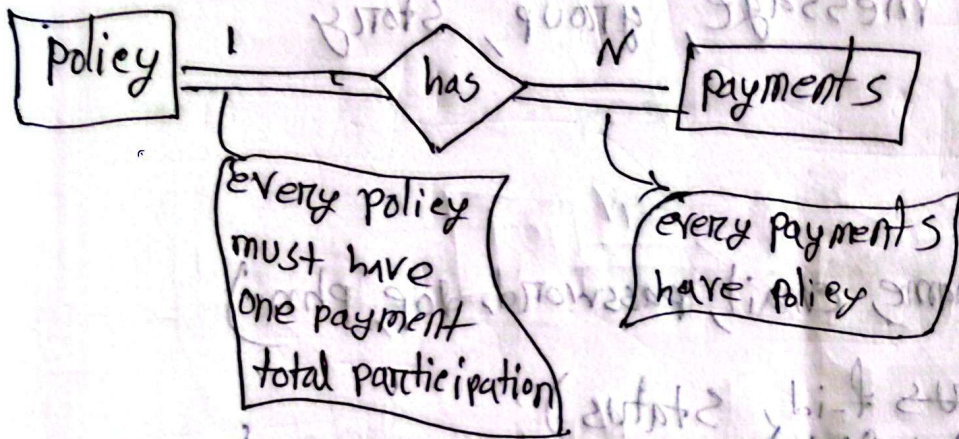
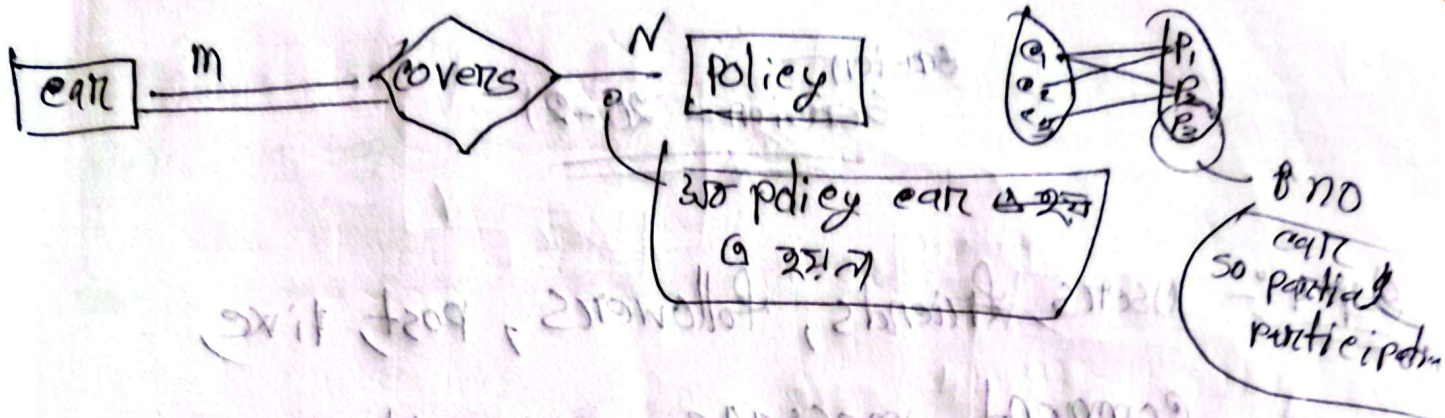


every customer have
at least one car



every car Accident 1 to N

Note
every customer
have at least one car



Step-1:- Users, friends, followers, post, like, comment, message, group, story

Step-2:-

Users (u-id, name, email, password, dop, phn)

Friends (status f-id, status)

Followers (f-id, f-date)

~~Followers (f-id, f-date)~~ → (f-id, f-date) primary key not
or weak entity

post (p-id, discription content, images, videos, p-date)

Live (l-id, l-date);

comment (c-id, content, c-date)

message (m-id, discription, m-date)

group (g-id, name, number of member)

story (s-id, content, expire time)

step-3:-

* User can send friend friend requests one each other

* User can follow each other User

* User create/share post

* User comment on post comments

* User likes in post like

* User share post share

post has like

post has comments

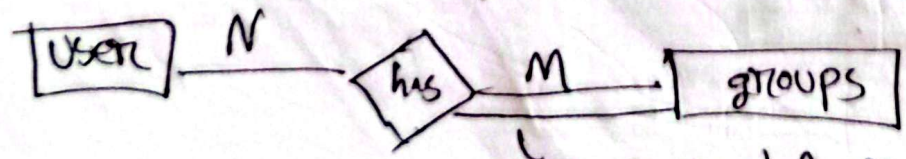
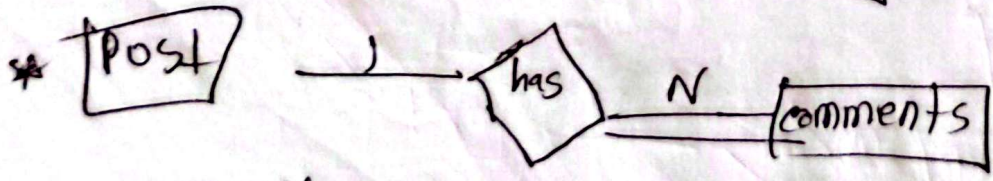
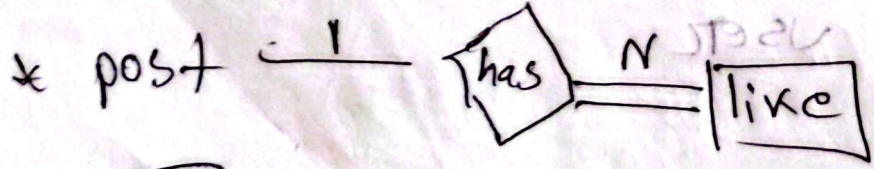
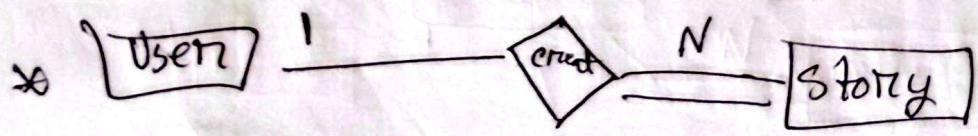
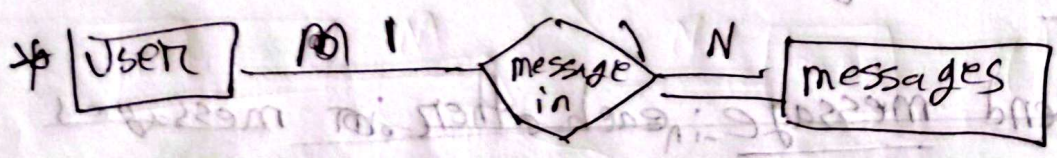
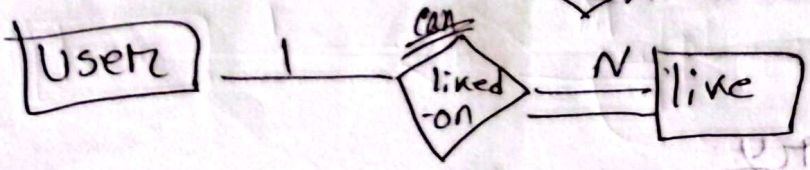
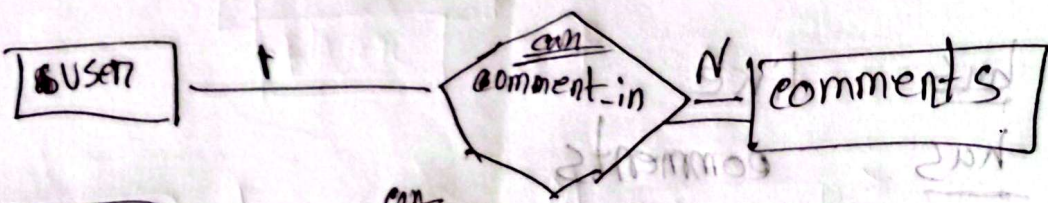
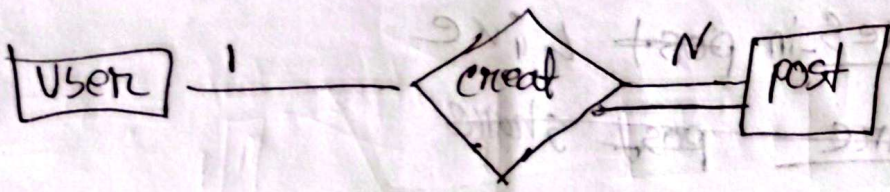
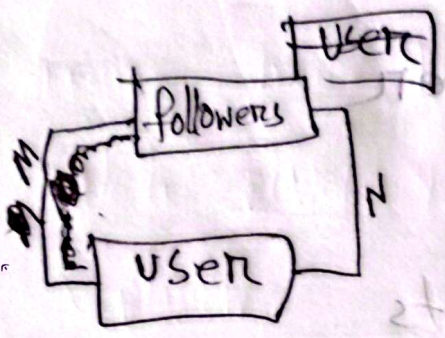
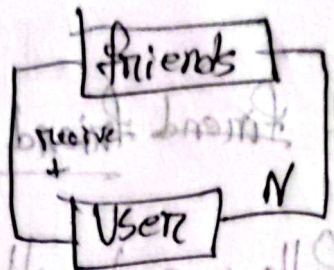
post

* User create story

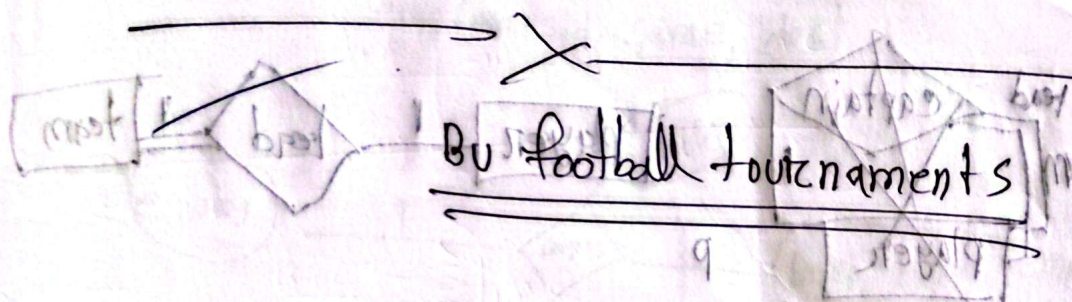
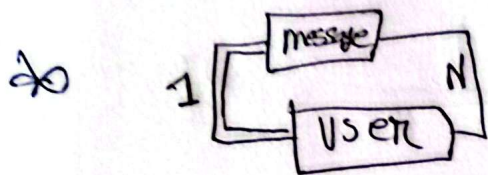
* User can send message one each other in messages

* User message User

Step-49 friendship



groups must have at least one user



step-1 :- teams, player, injury, matches, result

step-2 :- Identify each Attributes

teams (t-id, name, manager, coach) ~~FK = player-id~~

player (P-id, name, dob, skill level, position) ~~injury Records~~
FK = team-id

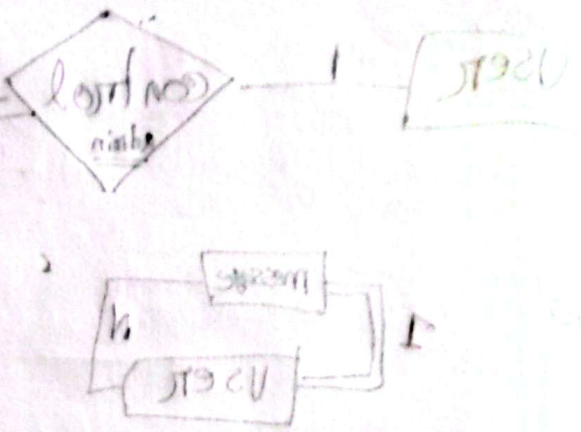
~~game~~ (g)

match (m-id, goal-team1, goal-team2, Date) ~~FK = team1/team2~~

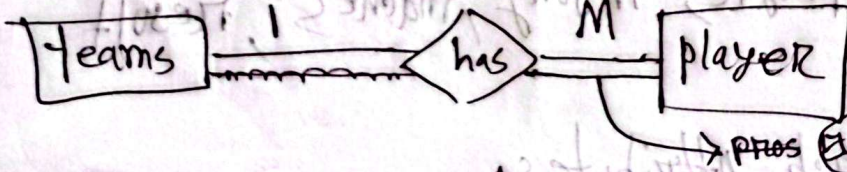
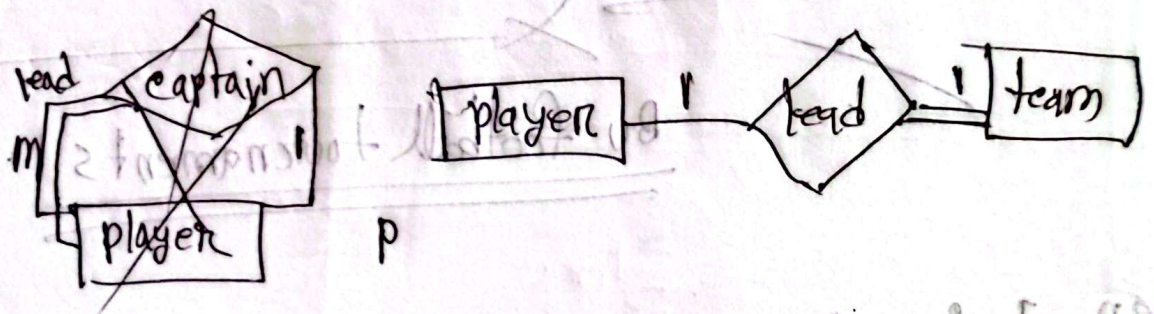
injury (i-id, injury-type, date) FK = player-id

Step-3 :

- * player lead team
- * teams has player
- * matches has played team
- * player has injury



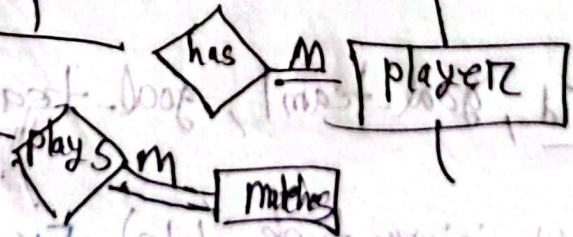
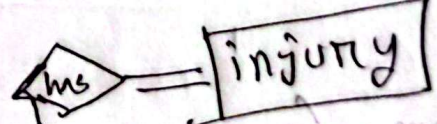
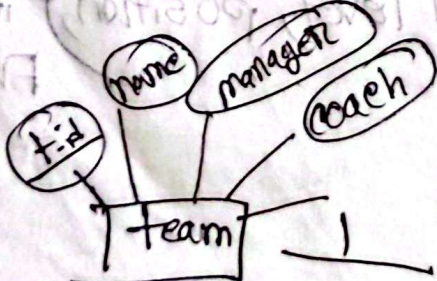
Step-4 :



(each team must have one player)

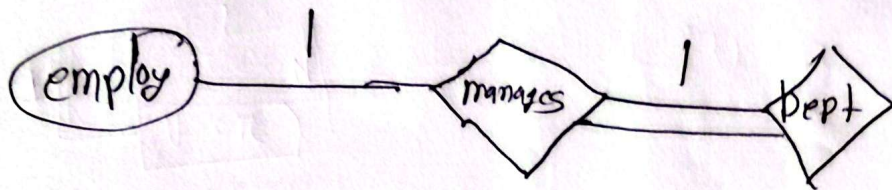


(each team must have one match)

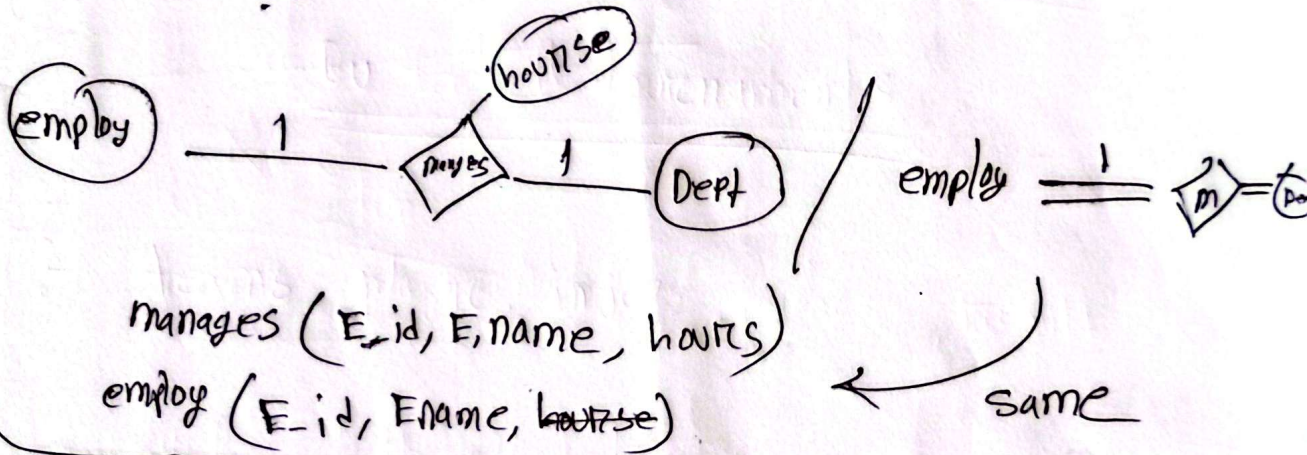


ERD to schema diagram

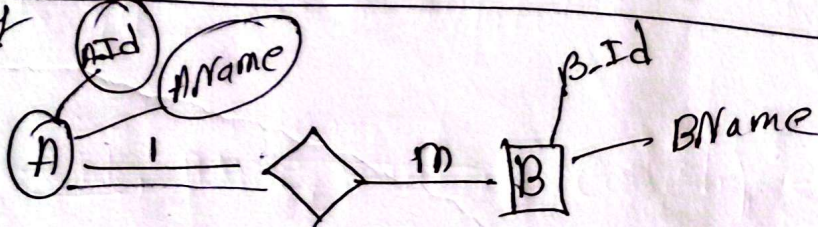
one to one



Dept (D-id, name, emp-id)
 emp (emp-id, name, dob)



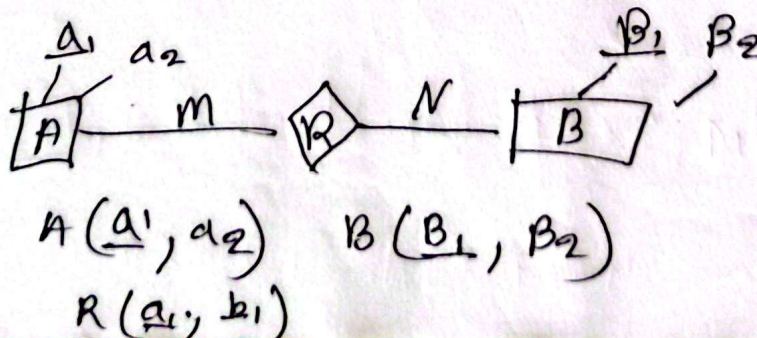
one to Many



many to many

B (B-id, B-name, A-id)
 A (A-id, A-name)

many to many



61

Lecture - 5

Limit

* Limit clause is used to specify the number of records to return from a query

* It is used with select statement

* The limit clause also used with optional offset to skip a certain number of rows before return the results

✖ select column from table name
where condition

Limit number offset number ;

→ (offset use 40, 67
2000 (2000
offset 1 → skip 1st
rows)

LIVE

* LINE operator used is used in where clause to search for a specific pattern in a column.

* there are two wild cards often used in conjunction with line operator.

(%) → represent (zero, one ^{or} multiple characters)

(_) → represent one, single character

syntax: select column-name from table-name

where column-name like pattern;

select * from students where address like 'e%';

(second character output into e)

String Function

select concat

* select concat (name, address) as information from students

* **Length()**: select name, Length(address) from students;
(return length of string)

* **Lower()**: select name, lower(address) from students;
(convert Lower case)

* **Upper()**: same as Lower();

* **Left()**: select Left(address, 2) from student;
Right() → (return 2 character return)

LPAD()
RPAD(): pads the right side of a string with another string

select LPAD(roll, 6, '0') from student;

LPAD() → Left [101 → 000101]

RPAD() → Right

* **Replace()**:

select name, Replace(address, 'Dhaka', 'Dhaka city') from students;

[Dhaka → Dhaka city]

Lecture - 6

JOIN

A join clause is used to combine rows from two or more tables, based on a related column between them.

Inner join :- Returns records that have matching values in both tables.

Ex:- select s.name AS student-name, d.name AS Dept-name
from students s inner join departments d
ON s.dept-id = depts.d.id;

Left join :- return all records from the left table and the matched records from the right table.

Syntax:- select column_name from table1 left join table2
ON table1.column_name = table2.column_name

VIEW

VIEW

❖ create VIEW VIEW-name As
select column1, column2, from table table-name
where condition;

Date & Function

❖ curdate() = current-date();

current-time();

current-timestamp();

❖ select datediff (join-date, current-date());

→ date ko unit ko (VVI) days output (day)

select dayname (curdate());

→ year, day
❖ select month (current-date());

select hour (current-time());

→ minute,

39) select e.title from courses c join enrollments e
on c.course_id = e.course_id
group by c.course_id e.id, e.title

Having count(*) > (
select Avg(enroll-count)
from (
select count(*) as enroll-count
from enrollment
group by course_id
) as subquery
);

43) update instructors set phone
set phone = concat('1880', Right(phone, length(phone) - 10))
where length(phone) > 14;

V.V.F

set phone = concat('1880', Right(phone, 10));

Replace 2017/01/17/01 → 01/7/17/01

~~30~~ update students set phone = Replace(phone, '1880', '01')
where phone like '1880 %';

30 Find the courses that have more enrollments
than the average number of enrollments
per course

⇒ select c.title from courses c
join enrollment e on c.id = e.courseid
group by c.id, c.title
having count(*) > L

select avg(enroll-count)

from (

select count(*) as enroll-count

from enrollments

group by course-id

AS (subquery)

3(a) iii

Session 10-20

select w.wname from worker w, manages m

where ~~w.wname = m.wname~~ .. m.wname = w.wname

w.street = (select street from worker
where wname = m.managername)

and

w.city = (select city from worker
where wname = m.managername)

—————>

(iv)

update works

set salary = salary * 1.075

where wname in (select manager_name from
manager)

Viii

~~select wname from~~

select wname from works

where orgname = 'Facebook'

and datediff(date, (current-date), date) > 365 * 5 ;

Lecture - 4 → Theorems

Functional Dependency (FD)

* Functional Dependency is a relationship that exists between the primary key and non-key attribute with in a table!

(determination)

$X \rightarrow Y$

(dependency)

* the left side of FD is known as determinant, the right side of the production is known as a dependent

* Data is dependent on FD, FD is not dependent on Data.

* Let R be a relation schema

$\alpha \in R$ and $\beta \in R$

The functional dependency

$\alpha \rightarrow \beta$

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

* input different → output same

different output same
 $4 \rightarrow 1$
 $5 \rightarrow 1$

~~$x \rightarrow y$~~

* $x \rightarrow y$

if x same = y also same

α	β
1	6
1	6

* if y different = x can be same or different

$1 \rightarrow 6$
 $2 \rightarrow 6$

* if x same then y must same
 if x different then y can be (same or different)

FD: ~~$x \rightarrow y$~~
 if $t_1[x] = t_2[x]$
 then $t_1[y] = t_2[y]$

different output same
 $4 \rightarrow 1$
 $5 \rightarrow 1$

not

* $x \rightarrow y$

if x same = y also same

α	β
1	6
1	6

* if y different = x can be same or different

$1 \rightarrow 6$
 $2 \rightarrow 6$

* if x same then y must same
 if x different then y can be (same or different)

FD: $x \rightarrow y$
 if $t_1[x] = t_2[x]$
 then $t_1[y] = t_2[y]$

Where closure set $F = F_2$ that why
is equivalent of FD



Canonical cover (chapter-7)

A canonical cover is a simplified and reduced version of the given set of functional dependencies.

$R(ABCD)$

FD:

$B \rightarrow A$

$AD \rightarrow B$

$C \rightarrow ABD$

canonical cover (FC)

$B \rightarrow A$

$AD \rightarrow C$

$C \rightarrow ABD$

Ans:

At first break every $\neq b$

$B \rightarrow A$

$AD \rightarrow B$

$AD \rightarrow C$

$C \rightarrow A$

$C \rightarrow B$

$C \rightarrow D$

overall = $\boxed{\begin{matrix} B \rightarrow A \\ AD \rightarrow C \\ C \rightarrow BD \end{matrix}}$

Ans exist

$AD^+ = ADCB$

Without $AD \rightarrow C$ FD

$AD^+ = ADBA$

if C is missing
so reduced
200 (10 min)

Now

$B^+ = BA$

Now with $B \rightarrow A$ combination

$B^+ = B$

if A is missing
so remove 200 (10 min)

again

$AD^+ = ADCB$

without first $AD \rightarrow B$

$AD = ADCB$

if A is missing
so remove 200 (10 min)

Trivial : $A \rightarrow B$ has trivial functional dependency if B is a subset of A

Ex : $A \rightarrow A$, $B \rightarrow B$, $AB \rightarrow A$ $\rightarrow$ A determinant of A is only

Non-Trivial : $A \rightarrow B$ has non trivial functional dependency if B is not a subset of A

Ex : $A \rightarrow B$, $AB \rightarrow ABC$

$\rightarrow$ is determination of B by A is non-trivial

~~Simple~~ $\rightarrow$ FD
closure set

It is the complete set of all possible attributes that can be functionally derived from given functional dependency using inference rules

Ex: $R(ABCDE)$ → attributes name
→ table name

FD:

$A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

$B^+ = \{B, AB, D\}$

$B^+ = \{BD\}$ and $(AB)^+ = \{BCDE\}$

Equivalence of FD

Question

$R(ABCDEH)$

Ans:

$F_1: A \rightarrow C$

$AC \rightarrow D$

$E \rightarrow AD$

$E \rightarrow H$

now closure set of F_1

$A^+ = \{A, C, D\}$

$AC^+ = \{AC, D\}$

$E^+ = \{E, A, D, C, H\}$

$E^+ = \{E, H, A, D, C\}$

$F_2: A \rightarrow CD$

$E \rightarrow AH$

now closure set of F_2

$A^+ = \{A, C, D\}$

$AC^+ = \{AC, D\}$

$E^+ = \{E, A, H, C, D\}$

$E^+ = \{E, A, H, C, D\}$

Candidate key

~~1~~ minimal number of candidate keys are called candidate key

$AB \rightarrow CD$ [here A is the subset of super key AB]

ଯଦି A already define ହୁଏ ତାହା candidate
 key ନୁହେଁ ତଥାପି A not candidate but BC can be
 candidate key ଯଦି BC ଏ A ନୁହେଁ ।

All candidate key = prime member

Attributes those are not member of candidate key = non-prim

$$R(AB \odot D)$$
$$A \rightarrow B, B \rightarrow C \vdash A \rightarrow C$$

এখানে (২) এ কোনটি বৈধ (কি) ওয়াস নিষ ২৬৮

ମନେ ମାତ୍ର D ଟୁଟି ଟୁଟି ଧନ) ହୁ ନିମ୍ନ ଟୁଟି 260
 ଯାହା 3ଟି attribute
 ଉପରେ କିଛି ନାହିଁ

$$(DA)^+ \Rightarrow DABC \text{ (all attribute include)}$$

now $(DB)^+ = DBAC$

$$D^+ = DCAB$$

କାରଣ candidate key = ~~DA, DB, DC~~ D, A, B, C

⊗ R(A B C D E F)

$AB \rightarrow C, C \rightarrow D, B \rightarrow A, E$

step-1: $ABF^+ = BFAECD$

$$BFC^+ = ABFCEDE$$

ଅନ୍ୟ attribute ଓ
 ଯାହା ଟୁଟି 30
 (super key)
 candidate key

[non-prime because ABF is a
 sub-set of $ABFC$
 that's why non-prime
 those attributes]

prime = ~~A, B, F~~

non-prime = ~~C, D, E, A~~

steps even for you (select prime) select since

* $R(ABCDE)$

$AB \rightarrow CD, D \rightarrow A, BC \rightarrow DE$

Ans — $BD^+ = BDACDE$

$AB^+ = ABCDE$

$BC^+ = BCDEA$

all are candidate key
Because there is no
any subset of super
key

prime = A, B, C, D

non-prime = ~~none~~ E

* Important to have FD in each table

why needed?

* It helps to normalization

* It helps to remove redundancy and avoid anomalies

* It help find candidate key, ~~primary~~ super key and other constraints

* Ensures data integrity

why not needed

* Some table (likes history tables) may not have clear FD

* temporary or intermediate table may not need FD

* In denormalized table, FDs may be intentionally relaxed for performance.

Normalization

* It is a technique to remove or reduce Redundancy from a table.

Anomalies in DBMS

* There are three type of anomalies that occur when database is not normalized :

Insertion

If suppose a new employee joins the company who currently not assigned to any department then we would not be able to insert the data into table

empid	dept
10	1
10	1
10	1

empid	name	address	dept
1	N	X	1001
1	N	X	1002
2	M	Y	1003
.	.	.	.

update

In the above table we have two rows for employee N as he belong to two dept. If we want to update address we need have to update the same in two rows it take time, somehow, the correct address update

delete :- If we want to delete employee
 in m that also delete information of
 department 1003 no removal of supervisor as it is a technique to remove a staff

First normal form (1NF)

The rules of first normal form, an attribute
 of a table can not hold multiple values,
 It should hold atomic values

1NF ✗		convert		1NF ✓	
Id	phn			Id	phn
1	017, 018			1	017
2	010			2	018
				3	010

(2NF)

* A table said to be 2NF if

① Table in 1NF

② no non-prime attribute is dependent on
 the proper subset of any candidate key
 of table

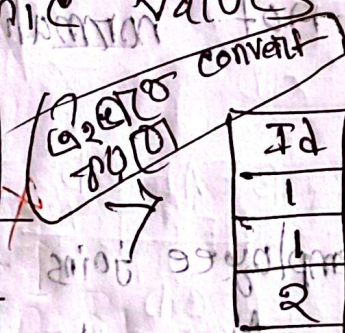
delete :- If we want to delete employee in m that also delete information of department 1003 we have to remove it. This is a technique to remove it.

First normal form (1NF)

The rules of first normal form, an attribute of a table can not hold multiple values. It should hold atomic values.

1NF ✗

Id	phn
1	017, 018
2	019



1NF ✓

Id	phn
1	017
1	018
2	019

(2NF)

* A table said to be 2NF if

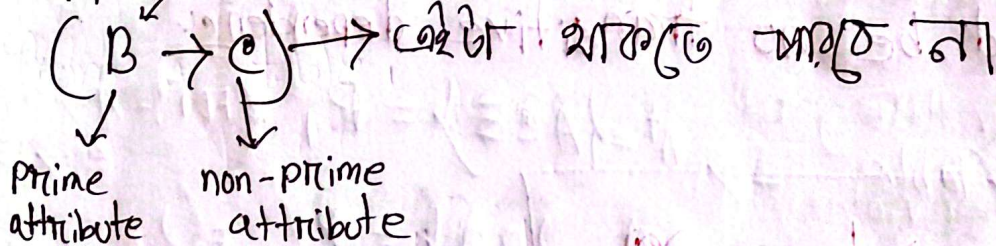
@ table in 1NF

① no non-prime attribute is dependent on the proper subset of any candidate key of table.

on

⑥ All the non-prime should be fully & functional dependent on candidate key

proper subset of candidate key



Ex: - $R(A, B, C, D, E, F) \rightarrow A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$

A candidate $AF^+ = ABCDEF$

candidate key = AF

prime = A, F (proper subset $\begin{matrix} A \\ F \end{matrix}$) } $AF \neq$ subset

non-prime = BCDE

here

$A \rightarrow B$

[A = proper subset of candidate key and B = non-prime]

So the this relation not in 2nd normal form

$R(A, B, C, D) \rightarrow \{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$

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

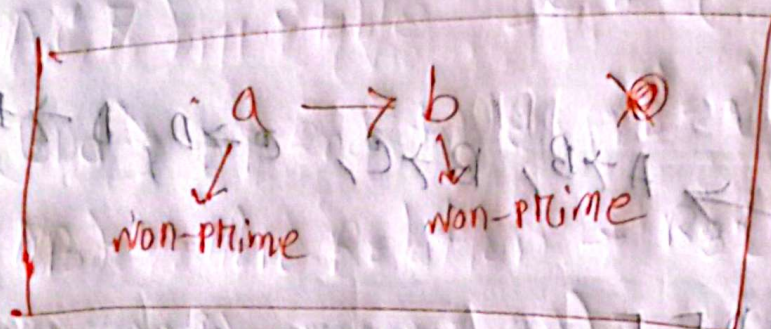
candidate key (prime) = A

non-prime = B, C, D

prime there is no proper subset of candidate key

So this $R(A, B, C, D)$ Relation exist 2NF

- Functions to 3NF of should be
- * table must be 2NF
 - * there should be no transitive dependency in table



Ex:- $R(ABCD)$

FD: $AB \rightarrow C, C \rightarrow D$

closure set $AB^+ = ABCD$

candidate key = AB

prime attribute = A, B

non-prime = C, D

* For each FD L.H.S must be on SK or R.H.S is a prime attribute
 here $C \rightarrow D$, here D is non-prime
 depended on C it also non-prime so
 occurs transitive dependencies in table

so this relation not in 3rd normal form

* Relation $R(A, B, C, D, E)$ = session = (2021-22)

Book_seller (ISBN-NO, salesman, sold-date, commission, discount-amount) also

Write as $R(A, B, C, D, E)$ FD: $A \rightarrow B, B \rightarrow C, C \rightarrow D, A \rightarrow C, A \rightarrow E, C \rightarrow E, B \rightarrow D$

* It exists this table satisfies 1NF because each attributes has single values

* check for 2NF:-

cluster set $AB^+ = \{A, B, C, E, A, B, D\}$

candidate key = AB

prime attribute = A, B

non-prime attribute = C, D, E

* none-prime attribute should not depend on prime attribute

Let's check dependences

$A \rightarrow C$ (partial dependency violates 2NF)
 $A \rightarrow E$ [violates 2NF]

$B \rightarrow d$ violate in 1st normal form

for normalize to 2NF we break the relation to remove partial dependencies

$R_1(A, C, E) \rightarrow$ table $\rightarrow A^+ = \{A, C, E\} = R_1$ $F_1: A \rightarrow C, C \rightarrow E$
 $R_2(B, d) \rightarrow$ 2nd table $\rightarrow B^+ = \{B, d\} = R_2$ $F_2: B \rightarrow d$
 $R_3(A, B) \rightarrow$ for Relation $\rightarrow R_3\{A, B\} = R_3$ $F_3: A \rightarrow B$
 $\rightarrow$ (candidate key 2/3 are)

check for 3NF

~~NO~~ there should not be no transitive dependence in table

check each FD in R_1

$C \rightarrow E$ transitive dependence [violate 3NF]
 $A = \text{student}$

for normalize to 3NF we split relation

R_1 into R_A, R_B

now all the final decomposition

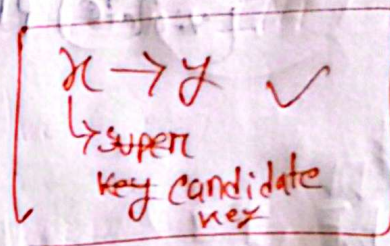
$R_A(A, C)$ and $R_3(A, B)$ (main relation to connect salesman)
 $R_B(A, E)$
 $R_2(B, d)$

Boyce codd normal form (BCNF)

It is advance version of 3NF, BCNF is stricter than 3NF.

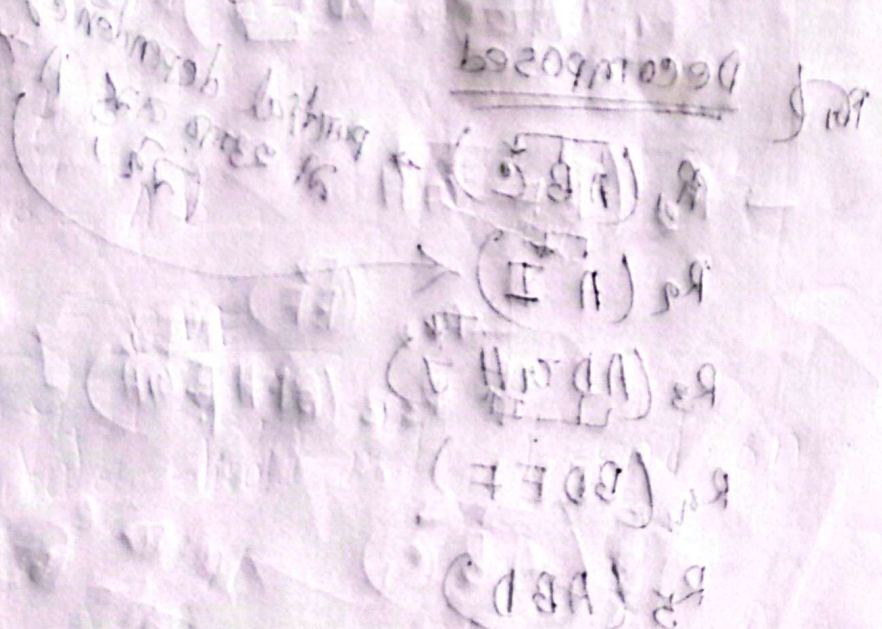
* For every functional dependency $X \rightarrow Y$, X should be ^{candidate key or} ~~candidate~~ super key of the table.

* It is in 3NF



→ X

→ Y



2NF decomposed

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

$AB \rightarrow C$ (violate) PD

$AD \rightarrow GH$ (violate) PD

$BD \rightarrow EF$ (violate) PD

$A \rightarrow I$ (violate) PD

$H \rightarrow J$

Now

$ABD^+ = ABDEGH EF$

candidate key = ABD^+

Decomposed

$R_1(A B C)$

$R_2(A I)$

$R_3(A D G H J)$

$R_4(B D E F)$

$R_5(A B D)$

partial dependency
A → I
(R₂)

3 NF decomposition

Q1) $R(A B C D E)$

$A \rightarrow B$

$B \rightarrow E$

$C \rightarrow D$

$(AC)^+ = ABCDE$

candidate key = AC

Now 2-3 NF covered

$R_1(\overline{A} \overline{B} \overline{E})$ - 2NF

$R_{11}(\overline{A} \overline{B})$

$R_{12}(\overline{B} \overline{E})$ } 3NF

$R_2(\overline{C} \overline{D})$ - 2NF

$R_3(\overline{A} \overline{C})$ - 2NF

not decomposed

* $R(AB\bar{C}D\bar{E}\bar{F}\bar{G}\bar{H}\bar{I}\bar{J})$

$AB \rightarrow C$ $A \rightarrow D\bar{E}$ (violate), $B \rightarrow F$ (violate)

$F \rightarrow \bar{G}\bar{H}$ $D \rightarrow \bar{I}\bar{J}$

now

candidate key $(AB)^+ = ABCDEFGHIJ$

now

2NF

$R_1(A \overline{DE} \overline{IJ})$

$R_{11}(A \overline{DE})$

$R_{12}(\overline{DE} \overline{IJ})$

$R_2(B \overline{F} \overline{GH})$

$R_{22}(\overline{BF})$

$R_{23}(\overline{BFH} \overline{G\bar{H}})$

$R_3(\overline{ABC})$

violates 3NF
 $A \rightarrow D$
 $A \rightarrow E$
 $A \rightarrow F$

3NF

3NF

trans e
not violate

BCNF

Ans. no (2)

$R(P, R, S, T)$

$P \rightarrow R, S$
 $SR \rightarrow T$
 $S \rightarrow R$
 $S \rightarrow P$

$P^+ = P, R, S, T$
 $SR^+ = SR, S, T, P$
 $S^+ = S, P, R, T$
 $S^+ = S, P, R, T$

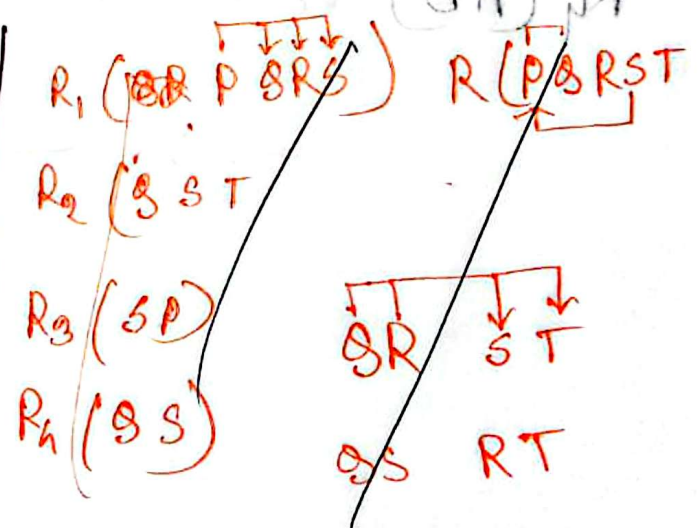
So, candidate key = $\{P, S\}$

Prime = P, S

non-prime = R, T

now 3NF

$P \rightarrow R, S$
 $SR \rightarrow T$
 $S \rightarrow R$
 $S \rightarrow P$



R_1

Session 20-21 6(a)

(a) on 2nd

book (access

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

$AB^+ = AB C D E$

$B \rightarrow C$ (invalid for 2NF)

$B \rightarrow E$ (11)

$B \rightarrow D$ (11)

candidate key = AB

$2, 1, 0 \leftarrow 9$
 $1, 0 \leftarrow 9, 0$
 $2 \leftarrow 0$
 $9 \leftarrow 2, 0$

Now de

$R_1(B, C)$

$R_2(B, E)$

$R_3(B, D)$

$R_4(AB)$

All are full fill

Not prime

candidate key

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

$2, 1, 0 \leftarrow 9$
 $1, 0 \leftarrow 9, 0$
 $2 \leftarrow 0$
 $9 \leftarrow 2, 0$

2NF

Lecture 7-6

Transaction

set of operation used to perform a logical unit of work

A transaction in database is a single unit of work that consists of one or more SQL operations (like update, insert, delete) executed together.

→ The main reason we use transaction is to keep the data correct and safe, especially when many people are using the database at the same time.

concurrent environment

→ transaction which successfully completes its execution is said to have been committed.

update users
set balance = 1000;
where id = 2;

update users
set balance = 1000;
where id = 1;
commit;

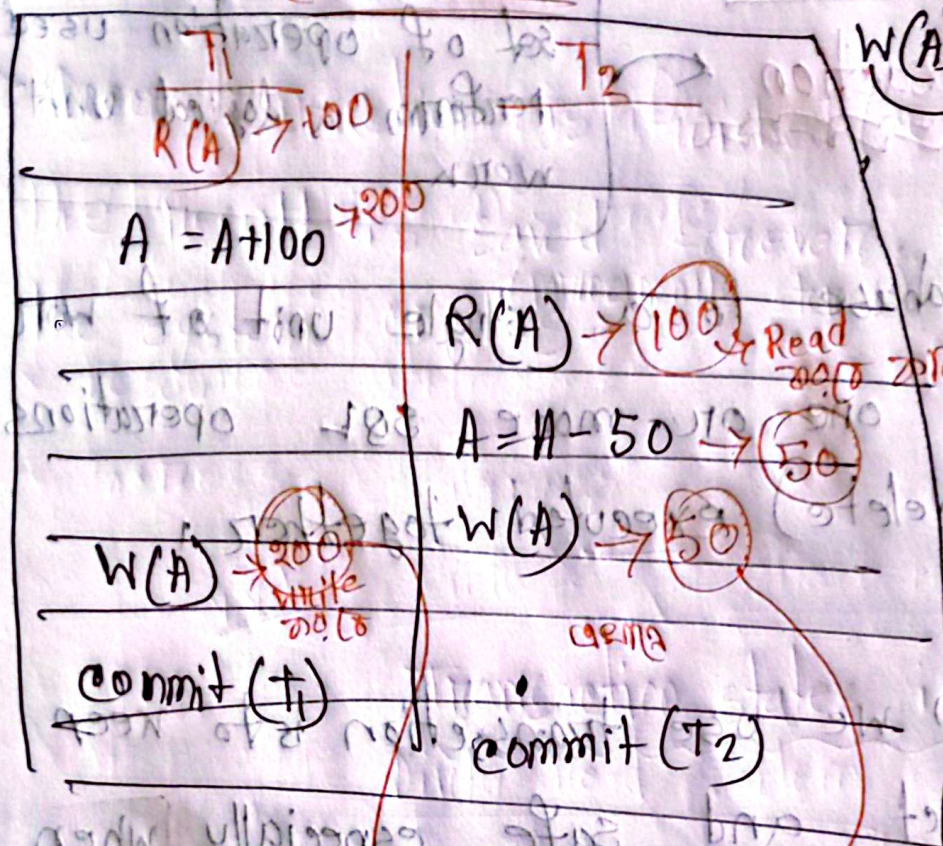
if all operation are successfully, commit command makes the changes permanent, or otherwise rolled back.

$A = 100$

Example

$R(A)$ → Read (select) attributes

$W(A)$ → write (update)



at end T_1 $A = 200$, T_2 $A = 50$

Different so rolled back 260

$\text{commit}(T_2)$ 260

* Real Life example ?

→ Bank system → transfer money and receive money

→ online banking system

→ E-commerce systems

without transaction

- ⇒ It may create an inconsistent results
- ⇒ It may create problems in concurrent ~~environment~~ execution

ACID properties



Atomicity : All or Nothing

- ⇒ Atomicity, ensure that all the operation in the transaction ^{are} executed completely or none at all,

consistency : Valid state before and after

- ⇒ Before transaction start and after transaction completed, the sum of money should be same.

Isolation : No interference between transactions (no mixing up)

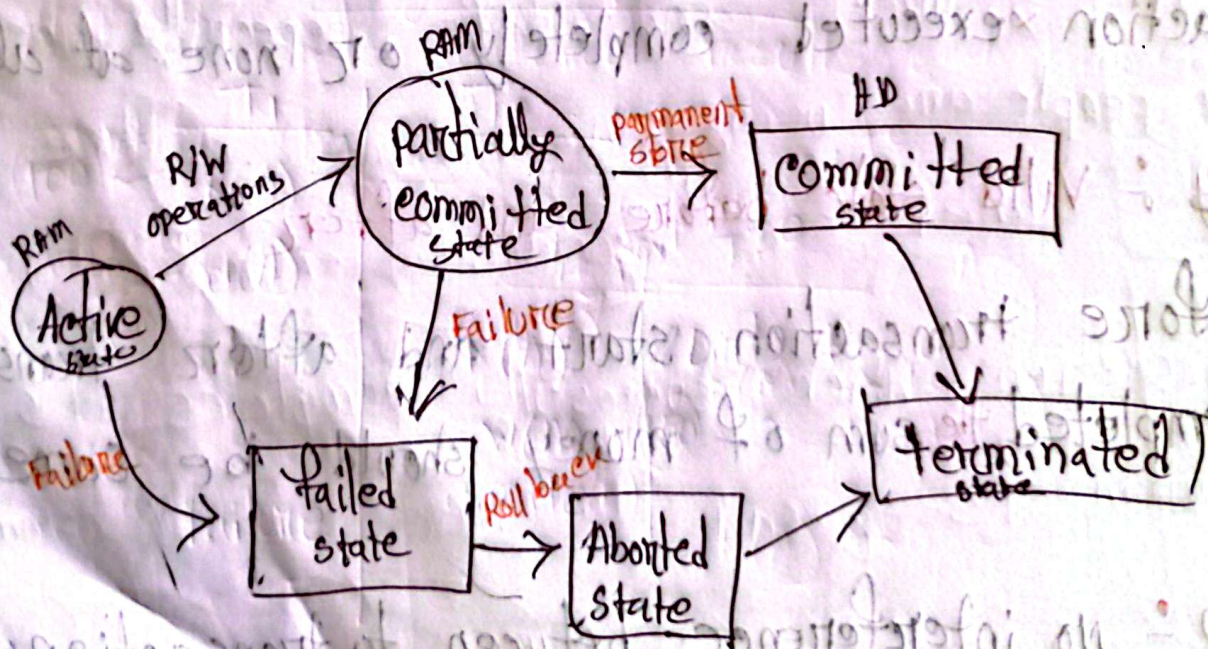
- ⇒ if many transaction run at the same time, they should not effect each other,

Ex: If two people Booking the last train seat at the same time, only one should get it, not both,

Durability :- permanent result

⇒ Once transaction is finished (committed) the result is saved forever, even if the system crashes, (permanently stored)

Transaction states in DBMS



Active state :-

- This is a starting state.
- The transaction is being executed, it is reading or writing data.

partially committed ?

- * All operations are done, but the transaction is not yet save permanently.
- * It's waiting for the final step (commit).

committed :-

- * The transaction is successfully completed.
- * changes are saved permanently in the database.

Failed :-

- * something went wrong (like system crash or error)
- * the transaction can not continue.

Aborted :-

- * The transaction is rolled back (canceled)
- * It canceled all the changes in database,

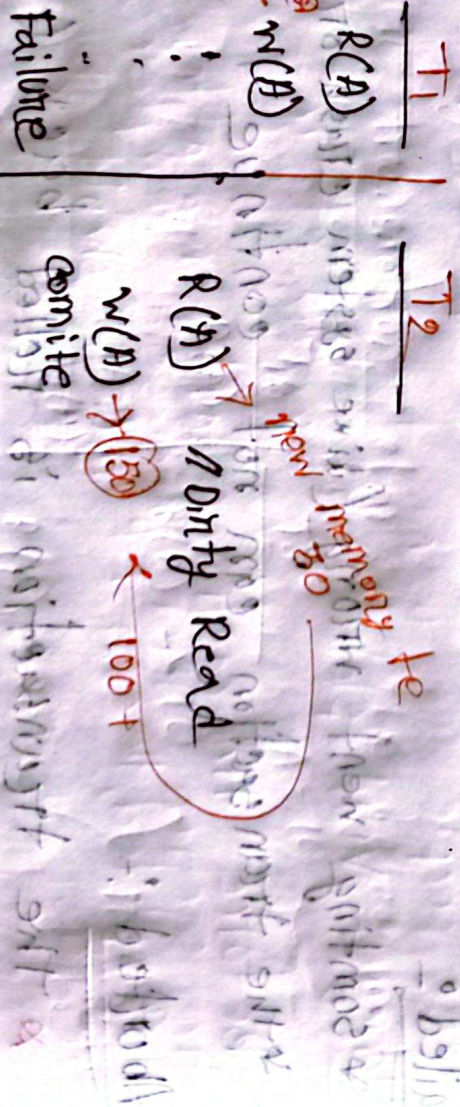
terminated state :-

- * The ~~to~~ transaction is fully completed, either committed or aborted.
- * All ~~resour~~ resource are released, and the transaction ends.

Concurrency: Advantage

- Decreases waiting time
- Decrease response time / increase performance
- Resource utilization
- Increase efficiency

Dirty Read Problem



- Transaction T_1 updates the balance of an account but has not committed yet.
- Transaction T_2 reads this uncommitted balance.
- If transaction T_1 fails and rolls back, then

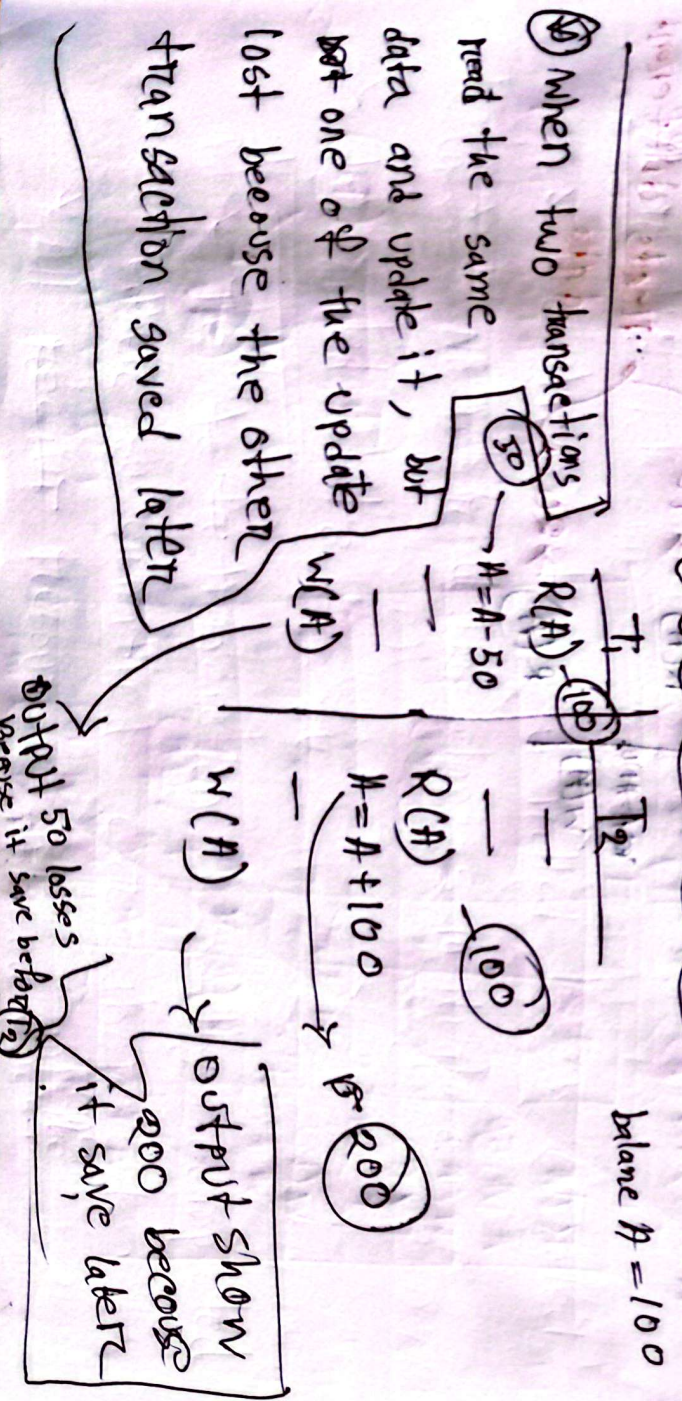
~~For~~ transaction ② read wrong data, which is called ~~lost update~~ Read

Solution:-

- Don't read any value from local buffer ~~from~~!
- Read value directly from memory if needed
- If read from local buffer ^{before} committing wait for previous transaction,

• Read value after committing, ensure no rollback can happen.

Lost Update Problem



Solution:

Use locks - Lock the data while one transaction

is using it, so other must wait.

unrepeatable read problem

When a transaction reads the same data more than once and gets different results each time, because another transaction change the data in between

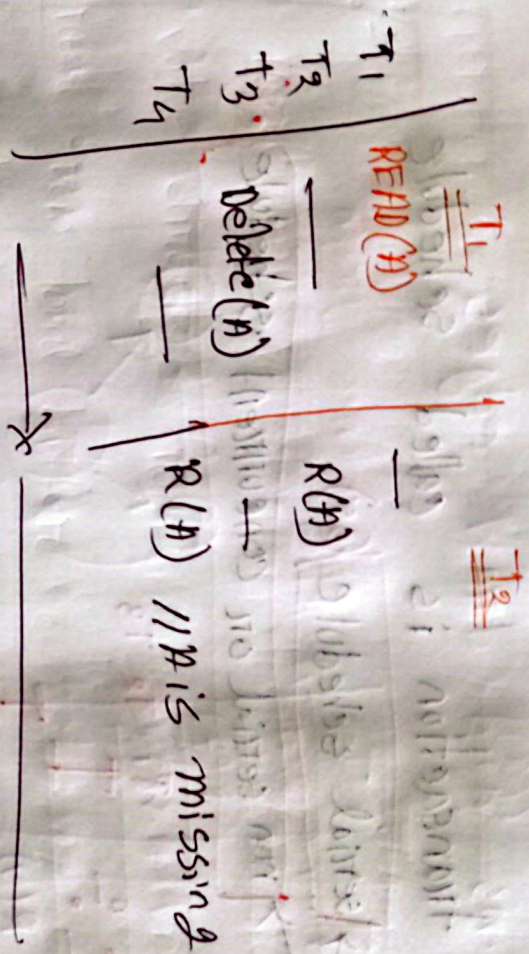
T_1	T_2
$R(A)$	$R(A)$
$W(A)$	$R(A) \rightarrow 10$
	$R(A) \rightarrow 20$

$A = 10$

$\therefore$ threads different



Phantom - Read



initials = form of history

Subsequence of history

1

(h)

(h)

(h)

(h)

(h)

(h)

(h)

(h)

(h)

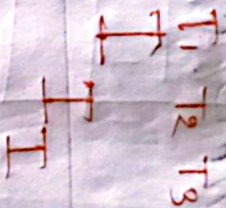
Schedule

A order of transaction is called schedule

two type $\rightarrow$ serial schedule (S)

$\rightarrow$ non serial or concurrent schedule (CS)

Serial schedule :-



one after another,
transactions run one by one, no overlapping
(no problem with isolation)

CS :-

we need to manage isolation

Serial sh schedule

T_1

$R(A)$	—
$A = A - 50$	—
$W(A)$	—
$R(B)$	—
$B = B + 50$	—
$W(B)$	—
commit	—

T_2

$R(A)$	—
$A = A + 0.1$	—
$A = A - T$	—
$W(A)$	—
$R(B)$	—
$B = B + T$	—
$W(B)$	—
commit	—

non-serial

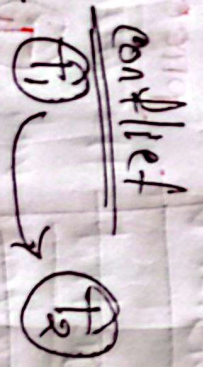
T1	T2
$r(A)$	—
$w(A)$	—
—	$r(A)$
$r(B)$	—
$w(B)$	$r(B)$
<u>commit</u>	<u>commit</u>

conflict serializable

~~$R \rightarrow W$~~
 ~~$W \rightarrow W$~~
 ~~$R \rightarrow R$~~

conflict

Precedence graph



ଏକାମ ଥିଲେ ମଧ୍ୟ
 ଠିକ୍‌ ଶେରୀଲାଇସ୍‌ କରା ଯାଇଁ

ଯଦି ଏହାକୁ
 ଠିକ୍‌ ଶେରୀଲାଇସ୍‌ କରା ଯାଇଁ

non-conflict



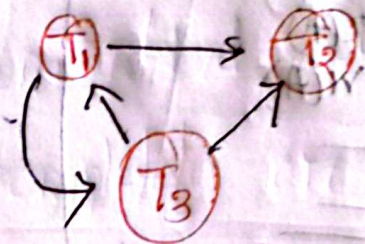
ଏକାମ ଥିଲେ
 ଠିକ୍‌ ଶେରୀଲାଇସ୍‌

ଯଦି ଏହାକୁ
 ଠିକ୍‌ ଶେରୀଲାଇସ୍‌ କରା ଯାଇଁ

* check conflict point in the other transactions and draw edges

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

precedence graph

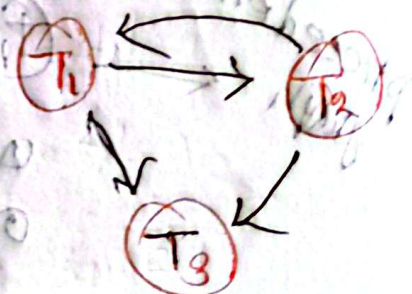


this graph is cycle so
non-conflict serializable

~~$R_1(B), R_3(C), R_1(A), w_2(A), w_1(A), w_2(B),$
 $w_3(A), w_1(B), w_3(B), w_3(C)$~~

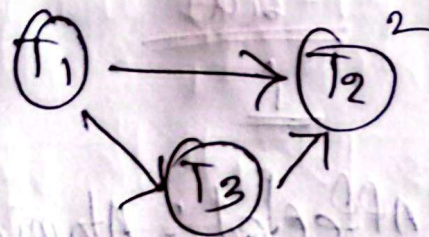
T_1	T_2	T_3
$R(B)$		
$R(A)$		$R(C)$
	$w(A)$	
$w(A)$	$w(B)$	
		$w(A)$
$w(B)$		$w(B)$
		$w(C)$

precedence graph



non-conflict serializable

T ₁	T ₂	T ₃
R(x)		
	R(y)	
		R(y)
	w(y)	
w(x)		
		w(x)
	R(x)	
	w(x)	



there is
no cycle
so this is

conflict serializable
schedule

depends on incoming edge

T₁ → T₃ → T₂

Time
Incoming
edge = 0

T ₁	T ₂	T ₃
R(x)		
w(x)		
	R(y)	R(y)
	w(y)	w(x)
	R(x)	
	w(x)	

Season 21-22

6(a)

i

properties Affected :- Atomicity

Explanation :- Atomicity ensure that all operation in the transaction executed completely or none at all.
here deposit is executed but with drawal is not executed, completely, which violating atomicity.

To maintain atomicity, The entire transaction should be rolled back

ii

properties Affected :- consistency

Explanation :- consistency ensure that the database move one valid state to another valid state.
Before transaction start and after transaction finished the sum of money should be same.
otherwise violating consistency.

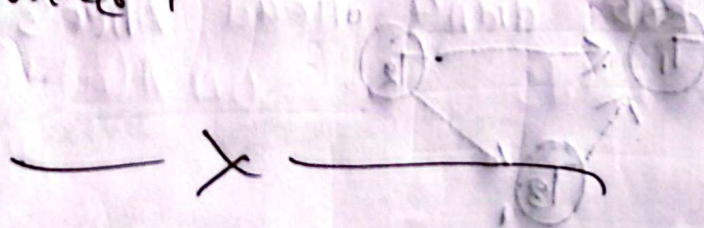
violate $\div$ property ~~III~~ The system should check the rules before doing the transaction.

Violate properties affect: isolation

* use concurrency control to prevent this.

properties affect: durability

* if the deposit is save, even after system restart, it's means durability is maintained, otherwise violated.



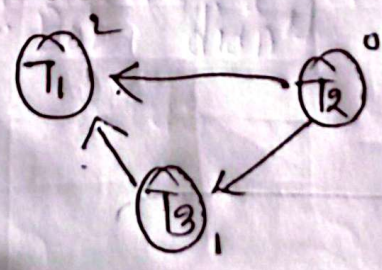
$$T \leftarrow S \leftarrow T$$

5(c)

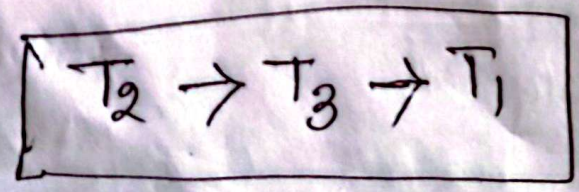
Find conflicts and Build precedence graph

T_1	T_2	T_3
	$R(A)$	
	$w(A)$	
		$R(C)$
	$w(B)$	
		$w(A)$
		$w(C)$
$R(A)$		
$R(B)$		
$w(A)$		
$w(B)$		

precedence graph :



there is no-cycle so this is conflict
serialized schedule



overview

7(b)

T_1	T_2
$R(c)$	—
$c = c - 30$	—
—	$R(c)$
—	$c = c - 20$
—	$w(c)$
—	committed
$w(c)$	
commit	

$c = 50$ tickets;

* it represents lost update problem, when two transactions read the same data, but one of the updates is lost because the other transaction saves later.

* hence T_1 (Alice) read available tickets = 50 and prepares to reserve 30, but it does not commit due to interruption

* T_2 (Bob) starts later and also reads 50 tickets and successfully reserves 20 tickets, so 30 are available

tickets = 30

⑤ ~~T₁~~ here T_1 commit later
than T_2 , that's why T_2 update
lost and the Final ticket count
→ count become $(50-30) = 20$ tickets,

—x—

✗ Why not Dirty Read?

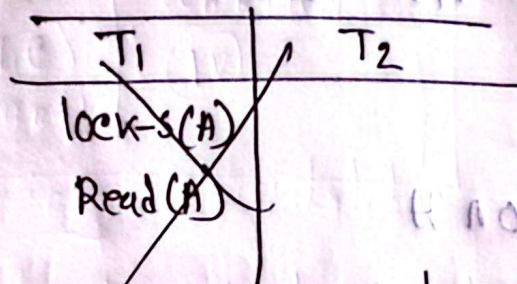
✗ A dirty Read happens when one transaction
Read uncommitted data from another transaction,
here Alice Bob didn't read Alice's uncommitted
data.

01/07/2025 11

19-20

7(b)

Q2 Add lock use when lock the data while one transaction use it, so other transaction must wait.



ADD lock and un lock instructions

T1 :-

lock-S(A)

lock(B)

$r(A)$

$r(B)$

$B = A + B$

$w(B)$

UNLOCK(A)

UNLOCK(B)

T2 :-

lock(A)

lock(B)

$w(A)$

lock(B)

$r(B)$

unlock(A)

unlock(B)

Two phase locking (2PL) protocol :-

* All transaction must acquire all locks before release any.

Has two phase

① Growing phase (acquire locks)

② Shrinking phase (release locks)

Deadlock Analysis (free)

This implementation is not deadlock-free

Because —

① T_1 acquires s-lock on A

② T_2 acquires x-lock on B

③ Now :-

④ T_1 needs x-lock on B (but T_2 hold it)

⑤ T_2 needs s-lock on A (but T_1 holds it in shared mode)

→ This create circular wait condition,
where each transaction is waiting for
each other.

Indexing

Search-key | pointer

* Indexing mechanisms to speed up access desired data.

* It helps the database find data faster without scanning every row.

types of Indexing

Primary Indexing

~~* It is an index~~

* Primary Index is an index created on the primary key of a table.

* Main file sorted

* primary key is used for sorting and filing.

* Example of Sparse Indexing

* no. of entries = no. of blocks acquired in index file by main file

index file

main file

Search key	Point
1	
4	
6	

id	name
1	a
2	b
3	b
4	d
5	e
6	f

Clustered indexing

* It is a type of indexing where the data is physically stored in the same order as the index.

* Main file is sorted

(Ordered)

* Main file is sorted

Index file

* There will be one entry for each unique value of the non-key attributes.

* data is sorted in order of the id index column (main file or column reference)

* only one clustered index per table

* very fast for range queries

SK	point
1	
2	
3	
4	

N.K
1
2
3
4
5
6

Block-1

Block-2

Block-3

Secondary

* Data is not sorted based on secondary index column. (main file unsorted)

* It is created on non-primary key columns.

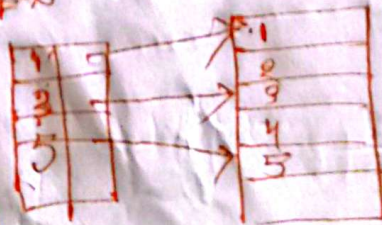
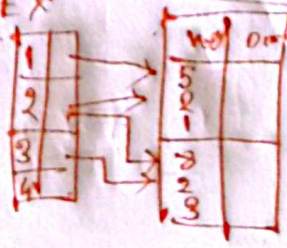
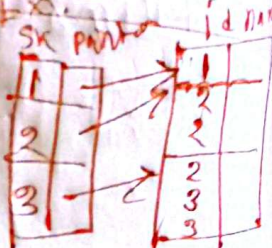
- Duplicates are allowed

dense indexing

Stk	Pointers
1	
2	
3	
9	
56	
32	
10	

34	
1	
27	
3	
92	
5	
2	
8	
100	
4	

↳ example of dense indexing

Feature	Primary	Secondary	Cluster
Based on	Primary key	non-primary key column	Any column
Data Sorting	Data is physically sorted by primary key	Data is not sorted by secondary index	Data is physically sorted by the indexed column
Uniqueness	Always Unique	can have duplicate	can be unique or not
Number per table	one per table	many allowed	only one per table
Storage use case	Fast access using primary key	Fast search on non-key column	Fast access in range queries
Example	Ex: 	Ex: 	Ex: 
Storage	Index and data are stored together	Index and data is separate from data	Index and data are stored together

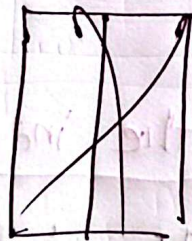
Dense Index

A Dense Index is an index that has one entry for every record in the main file,

~~or~~ Sparse Indexing : that has one entry for each block/page.

~~It~~ Takes more space than sparse

~~It~~ is fast searching, since no rows are skipped
Size larger than sparse Index



index file

Biology	
math	
ese	

one entry (for every record)

main file

7601	Biology
7201	Biology
7802	math
1201	ese

~~When~~ when is it preferable to use a Dense index instead of sparse Index

⇒ use a Dense index when

~~you need~~

you need fast search

~~* Dense index work well for columns that have duplicate values (like name, etc)~~

~~* Sparse index work only when data is sorted and unique~~

* Dense index one has one entry for every record

So, searching is very fast, because you can directly find data

* Data is not sorted on the index column

* If the data in the table is not sorted by the index field, you can't use sparse index. (column)

* Dense index works well even if the table is unsorted

③ The table is small :

- * dense index ~~to~~ uses more space (becos it ^{more entries})
- * It's work well when the table isn't too big

④ When data changes often

- * ~~If~~ you add or delete rows often, dense index is better
- * It is easier to update

Ex:-

⑤ Dens

~~For~~ Dense index is good for finding values in Range,

01/07/2025