05/08/2020

@ Data

Data is defined as facts or figures, or information that is stored in or used by a computer. An example of data is an email.

@ Database

A database is a collection of information that is organised so that it can be easily accessed, managed and updated.

@ DBMS

DBMS is a software for storing and retrieving user's data while considering appropriate security measures. It consists of a group of programs which manipulate the database.

@ Characterstics of DBMS

- → provide security & removes redundancy.
- → support of multiple view of the data.
- → support multi-user environment.
- → allows entities & relations among them to form tables.

@ Advantages

- → Data independence
- → Efficient data access
- → Data integrity & security
- → Data administration
- → Concurrent access & crash recovery
- → Reduced application development time
- → Redundancy
- → Backup & Recovery
- → Concurrent access
- → Security

@ Disadvantage
- → Danger of a Overkill
- → Complexity
- → Qualified personnel
- → Costs
- → Lower efficiency

@ Application
- → Banking : For customer information, account activities, payments, deposits etc.
- → Sales : Use for storing customers, product and sales information.
- → Universities : For student information, course reg., colleges and grades.
- → Telecommunication : It helps to keep call records, monthly bills, maintaining balances etc.

17/08/2020

@ Schema
- → Design of database
- → Blueprint of database

eg. SMS
- → student (rollno, name, age, branch)
- → Faculty (fid, fname, age, dept)
- → Staff (id, sname, sage)

@ NOTE -
Levels of data abstraction - OR 3-tier architecture of DBMS

end users ← View level OR External level

Schema design ← Logical Level OR Conceptual Level

Storage ← Physical Level OR Internal level

20/08/2020

@ Database Languages
- → DDL (Data definition language) } used for specifying the database schema.
  create, alter, drop
- → DML (Data manipulation language) } used for accessing & manipulating data in a database.
  insert, update, delete

@ NOTE

→ CREATE - to create the database instance
→ ALTER - to alter the structure of database
→ DROP - to drop database instance
→ TRUNCATE - to delete tables in a database instance
→ RENAME - to rename database instance

} DDL

→ SELECT - to read records from tables(s)
→ INSERT - to insert records(s) into tables(s)
→ DELETE - delete all records from table
→ UPDATE - update the data in table(s)

} DML

24/08/2020

@ Data Models

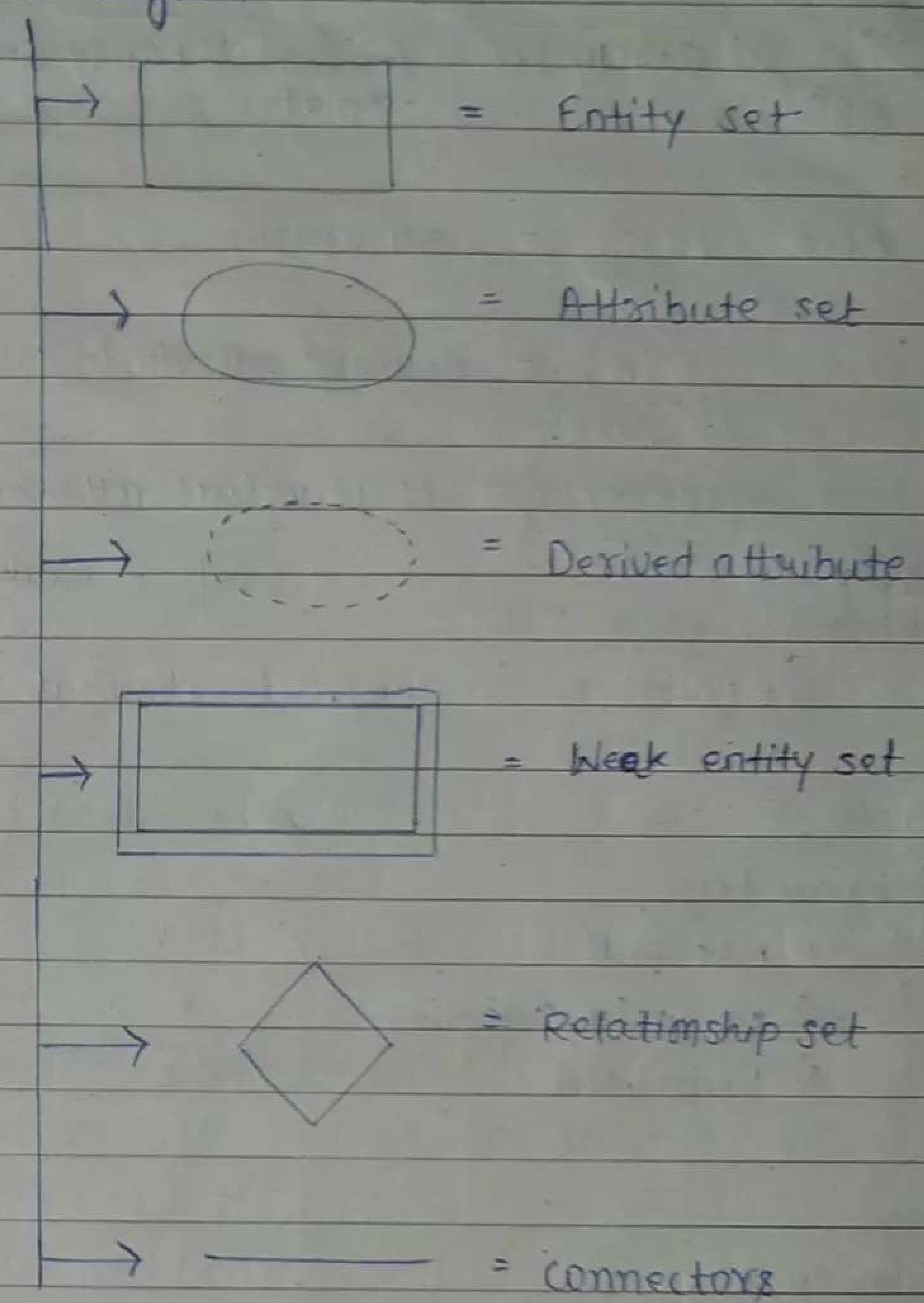
(Model → Prototype)

→ ER Model
(Entity Relationship Model)
↓
Any real
world object
that can be distinguished
from others.

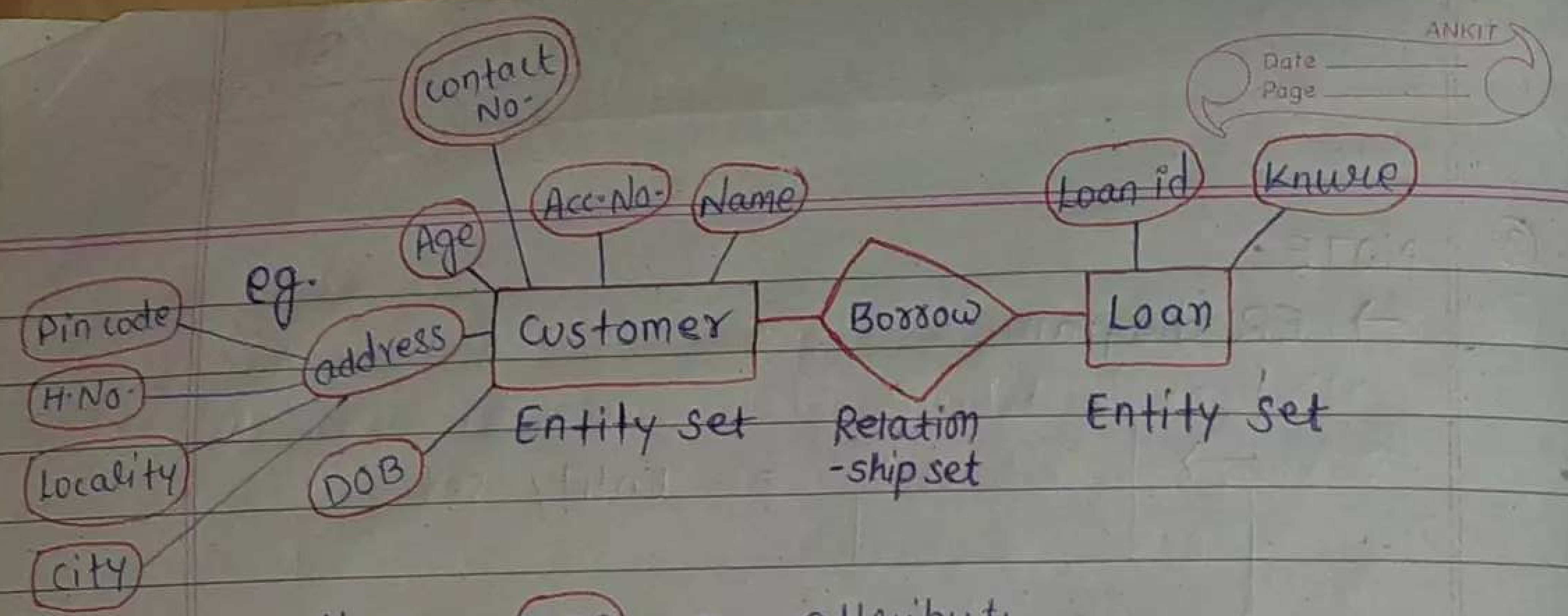
eg. AKTU
↓
Each student = Entity
↓
Set of students = Entity set

@ NOTE

→ ER diagram



eg.



Here, age = attribute

DOB = derived attribute

contact No. = Multivalues attributes

Customer = strong entity set

Loan = Weak entity set

address = composite attribute

28/08/2020

@ NOTE

→ Primary key

1) Not null
2) Uniquely Identify
3) Underlined

→ Key

① Primary key
② Super key
③ Candidate key
④ Foreign key

31/08/2020

@ Types of attribute

- Simple attribute
- Composite attribute
- Derived attribute
- Single-value attribute
- Multi-value attribute

@ Keys

- Super key
- Candidate key
- Primary key

@ Degree of Relationship

- Binary = degree 2
- Ternary = degree 3
- n-ary = degree

@ Mapping Cardinalities

- One-to-one
- Many-to-one
- One-to-many
- Many-to-many

@ Participation Constraints

- Total participation
- Partial participation

@ Relational Model

↗ STUDENT → Relation

Data

E.No.	Roll no.	Name	F.Name	Age	Branch	Add.
CS1	1	X	YY	21	CS	LKO
EC1	2	Y	YX	22	EC	LKO
CS2	3	X	XZ	21	CS	GKP
ME1	4	P	PQ	20	ME	GKP

Here,
student = Relation
Column name = attribute

S_1 = {E.No, Name, F.Name} = ✓
S_2 = {Name, F.Name} = ✗
S_3 = {Roll No., Name, Branch} = ✓
S_4 = {E.No., Roll No., Name, F.Name, Age, Branch, Add.} = ✓

Here, S_1, S_3 & S_4 = key

@ NOTE

1) All possible keys = Super key
2) Minimal Super key = Candidate Key

In above example,

{Roll no.}
{Enroll no.} → candidate key

3) One of the candidate keys chosen by the database designer to uniquely identify the entity set = Primary Key

02/09/2020

eg. Book

~~Reg~~ Ref. No.	Reg. No.	Title	Publisher
26	16	A	XY
27	17	B	YZ
28	18	C	AB
29	19	D	CD
30	20	E	EF
31	21	F	GH

{Ref. No., Reg. No., title, publisher}

{Ref. No.}
{Reg. No.} → candidate key

All possible keys = Super key

STUDENT

Primary key ← Roll No.; Foreign key ↑ d.No.

Roll No.	Name	Age	City	d.No.
1	X	21	LKO	d_2
2	Y	20	LKO	d_1
3	X	20	GKP	d_3
4	Z	21	VNS	d_2
5	P	20	LKO	d_4

DEPT.

Primary key ↖ d.No.

d.No.	d.Name	d.loc
d_1	CS	Yamuna
d_2	EC	Ganga
d_3	ME	Yamuna

Reg. Form

Roll No.	5
Name	P
Age	20
City	LKO
d.No.	d_4
SUBMIT	

Here, d_4 doesn't exist.

Foreign key

Foreign key are the columns of a table that points to the primary key of another table.

Foreign keys ensure data integrity., eg. can help to avoid orphan records.

ASSIGNMENT - 1

03/09/2020

1) Explain the difference b/w DBMS & file system.

DBMS - ① It is a software system used for creating & managing the databases. DBMS provides a systematic way to access, update & delete data.
② DBMS supports multi-user access.
③ Data consistency is more due to the use of normalization.
④ DBMS is highly secured.
⑤ Data redundancy is low.

Vs

File system - ① It is a software that manages & controls the data files in a computer system.
② Does not support multi-user access.
③ Data consistency is less.
④ File system is not secured.
⑤ Data redundancy is high.

2) Explain database languages with example.

Read, update, manipulate & store data in a database using Database Languages. The following are the database languages
→ Data definition language
→ Data manipulation language

\# DDL - The language is used to create database, tables, alter them etc. With this, you can also rename the database, drop them. It specifies the database schema.

CREATE - Create new database, table, etc.
ALTER - Alter existing database, table, etc.
DROP - Drop the database
RENAME - Set a new name for the table.

\# DML - The language used to manipulate the database like inserting data, updating table, retrieving record from a table, etc. is k/a DML.

SELECT - Retrieve data from database
INSERT - Insert data
UPDATE - Update data
DELETE - Delete all records.

3) Define superkey, candidate key, primary key & foreign key.

Super key - Super key is the superset of primary key. The super key contains a set of attributes, including the primary key, which can uniquely identify any data row in a table.

Candidate key - The candidate keys in a table are defined as the set of keys that is minimal & can uniquely identify any data row in the table.

Primary key - One of the candidate keys choosen by the database designer to uniquely identify the entity set.

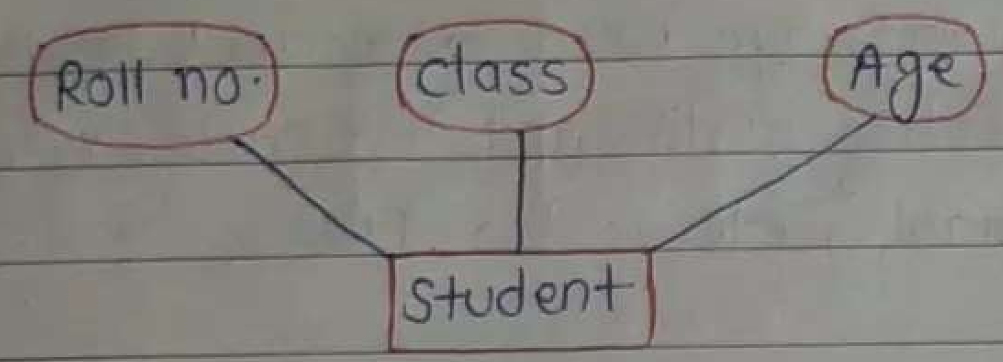
Foreign key - Foreign key are the columns of a table that points to the primary key of another table. Foreign key ensure data integrity.

4) Explain various types of attributes with examples.

Types of attributes -

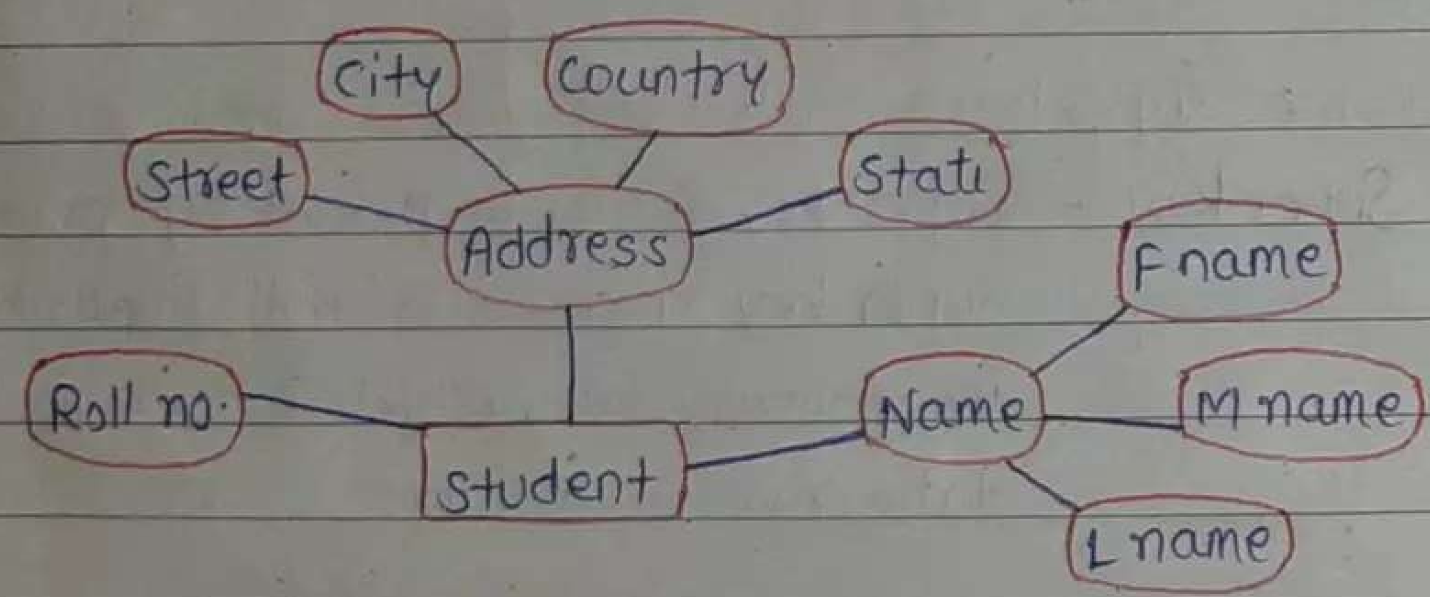
Simple attributes

→ Those which can not be divided further.



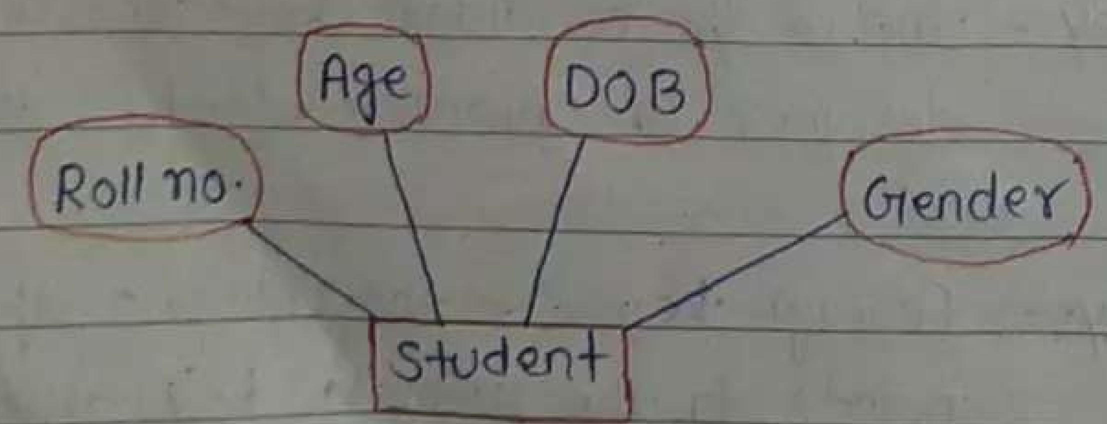
Composite attributes

→ Those which are composed of many other simple attributes.



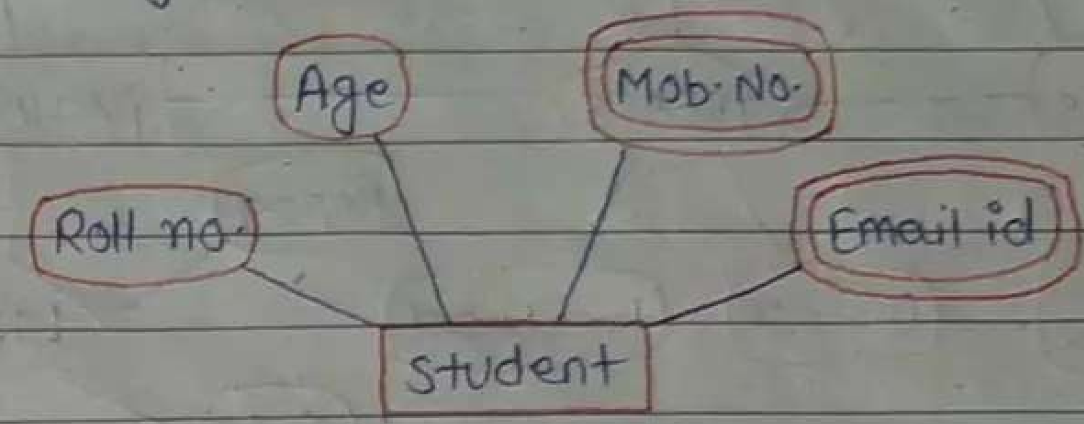
Single Valued attributes

→ Those which can take only one value for a given entity from an entity set.



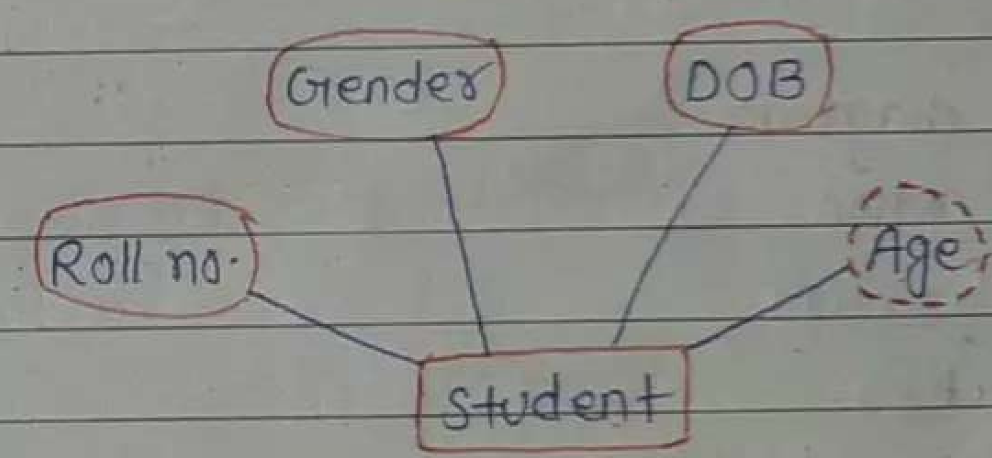
Multi Valued Attributes

→ Those which can take more than one value for a given entity from an entity set.



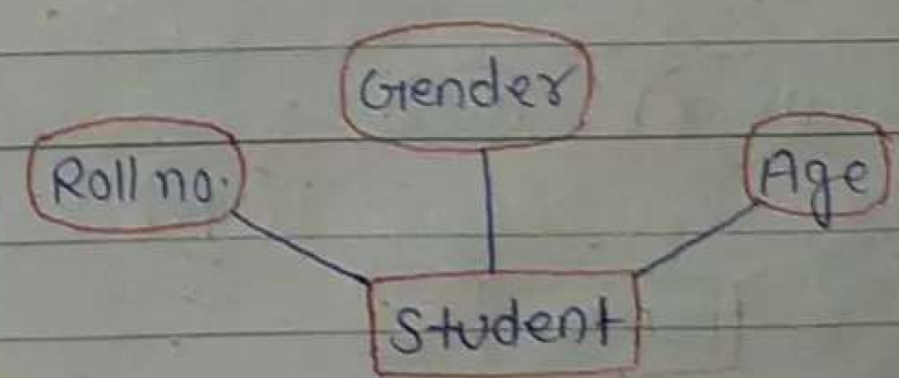
Derived attributes

→ Those attributes which can be derived from other attributes.

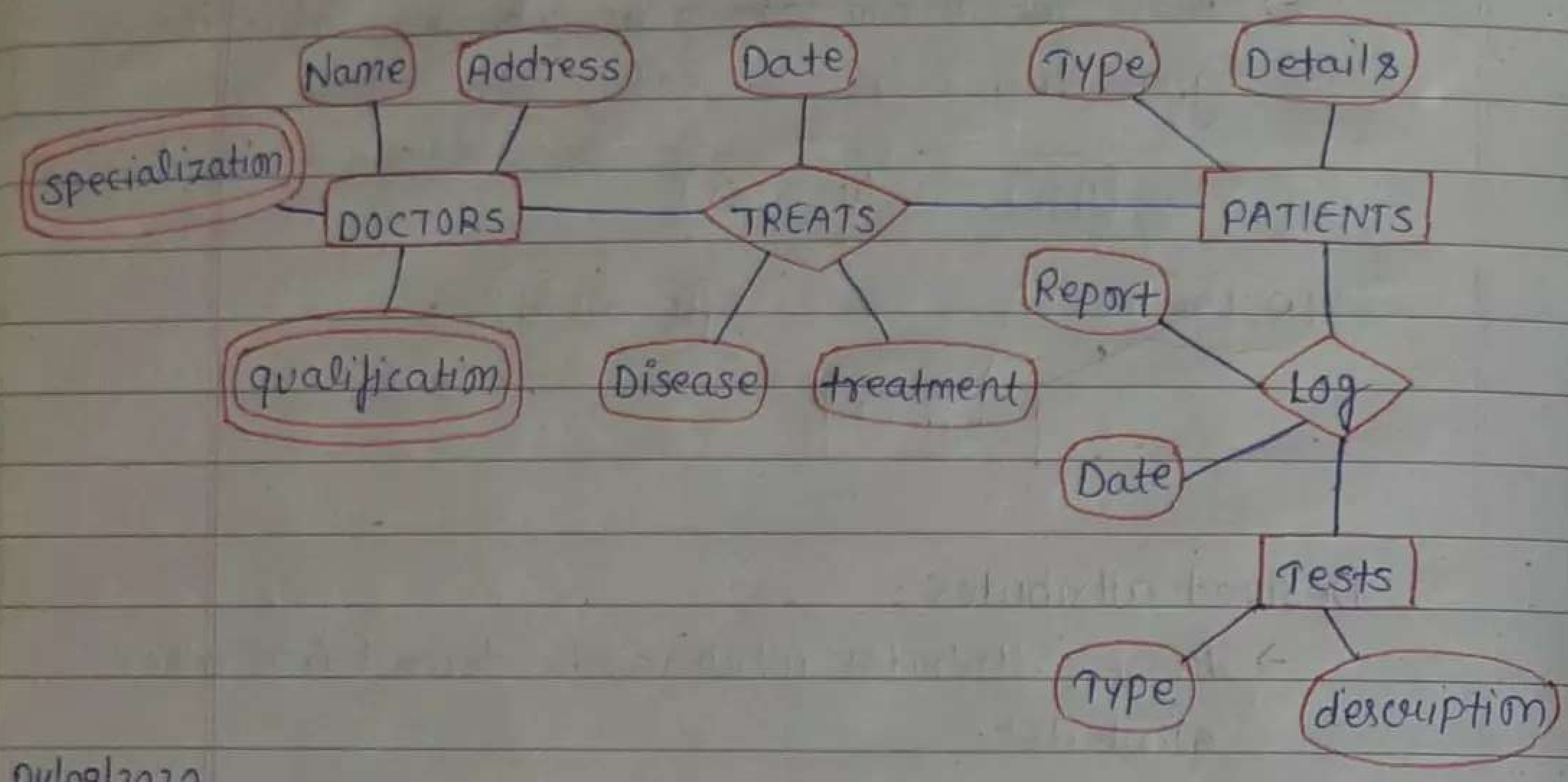


Key attributes

→ Those attributes which can identify an entity uniquely in an entity set.



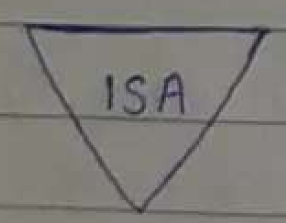
5) Draw an ER diagram for hospital management system.



04/09/2020

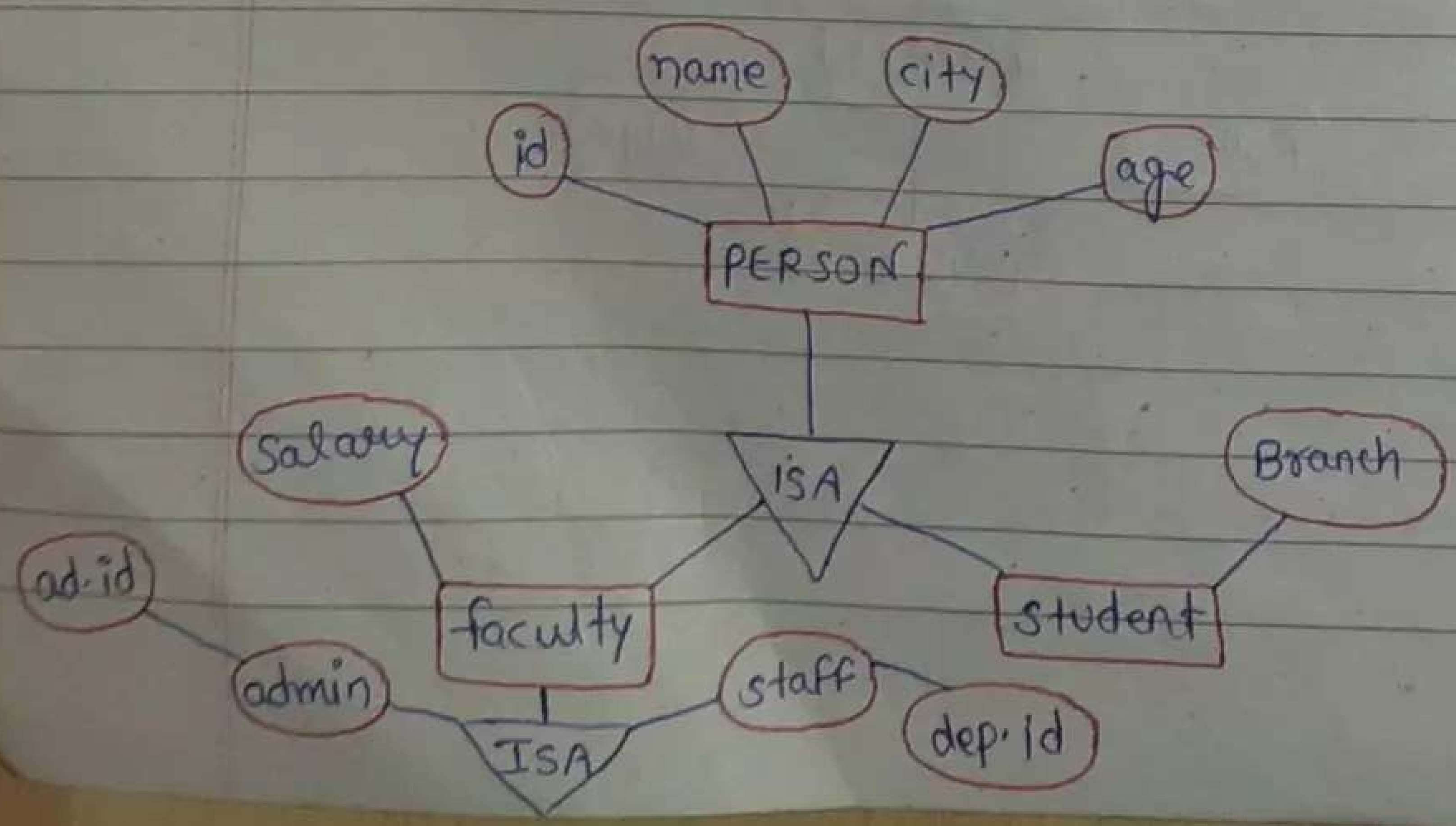
@ Extended ER diagram
1) Specialization
2) Generalization
3) Aggregation

@ Specialization



IS-A relationship

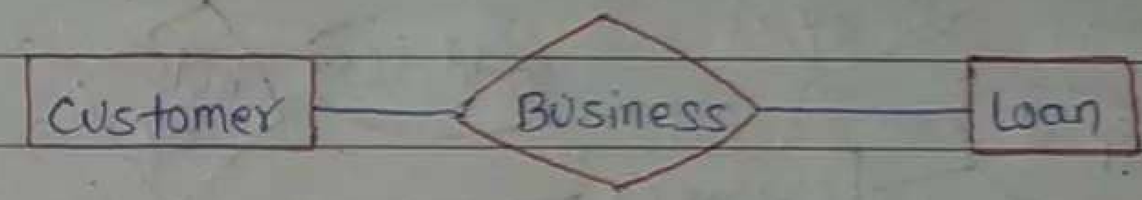
generalize to specialize



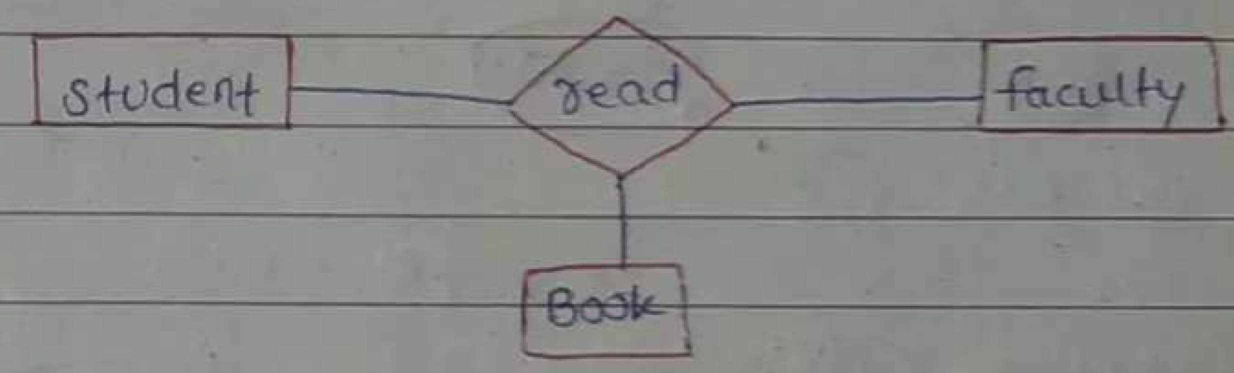
Employee/Faculty (salary, id, name, city, age)
admin (ad.id, salary, id, name, city, age)

@ NOTE - # generalize to specialize = specialization
# specialize to generalize = generalization

@ Types of relationship
1) Unary
2) Binary.
3) Ternary

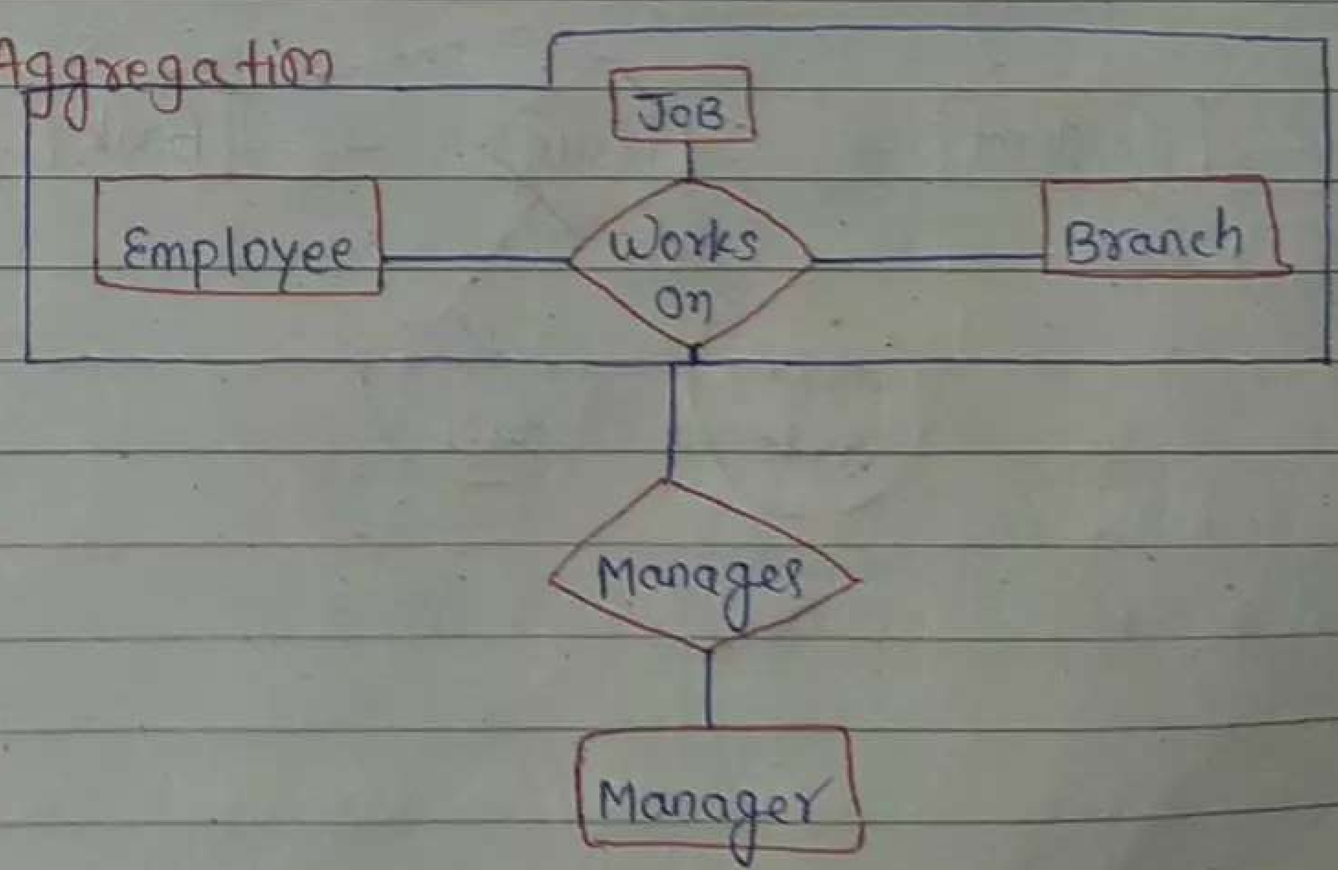


→ Binary.

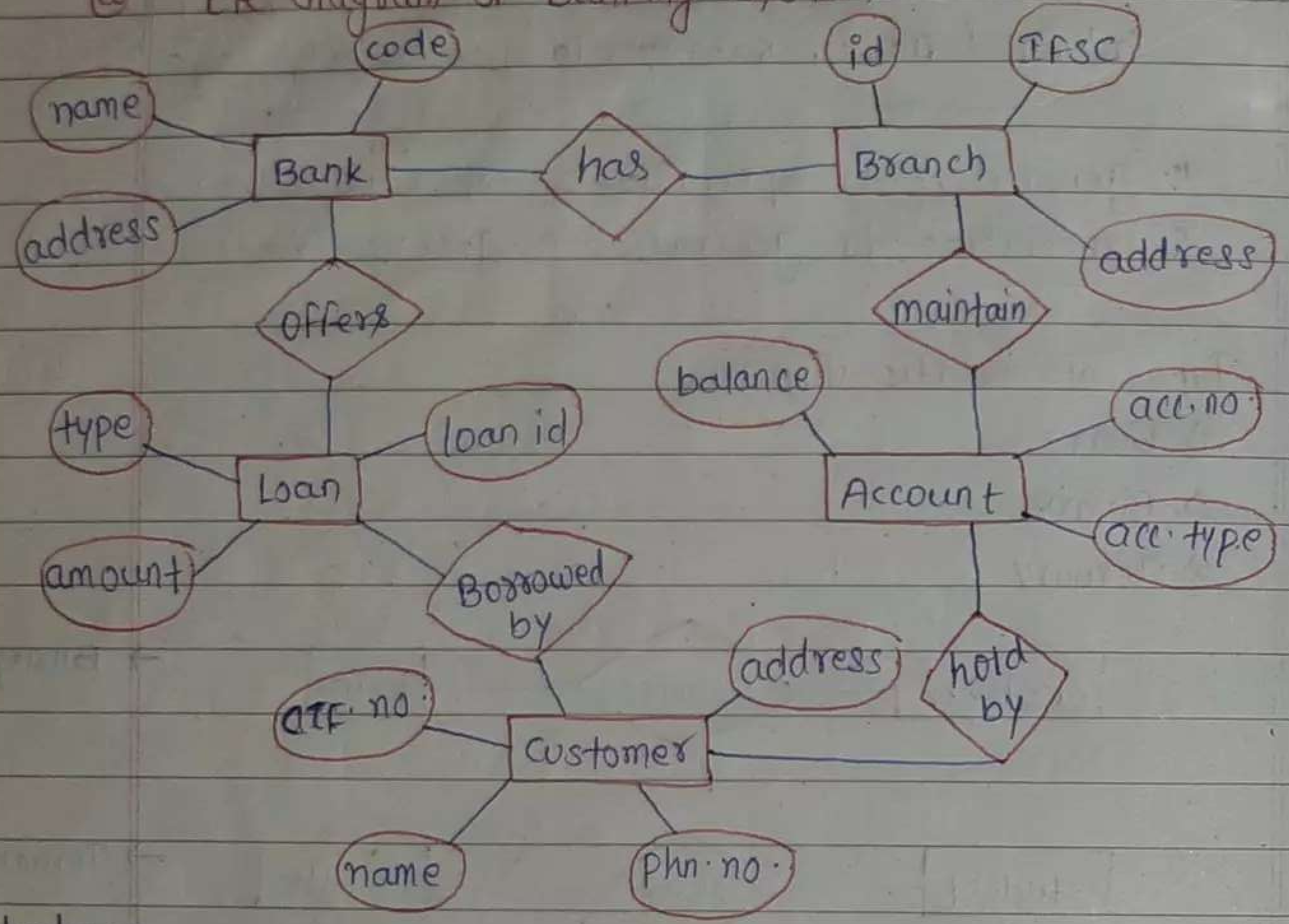


→ Ternary

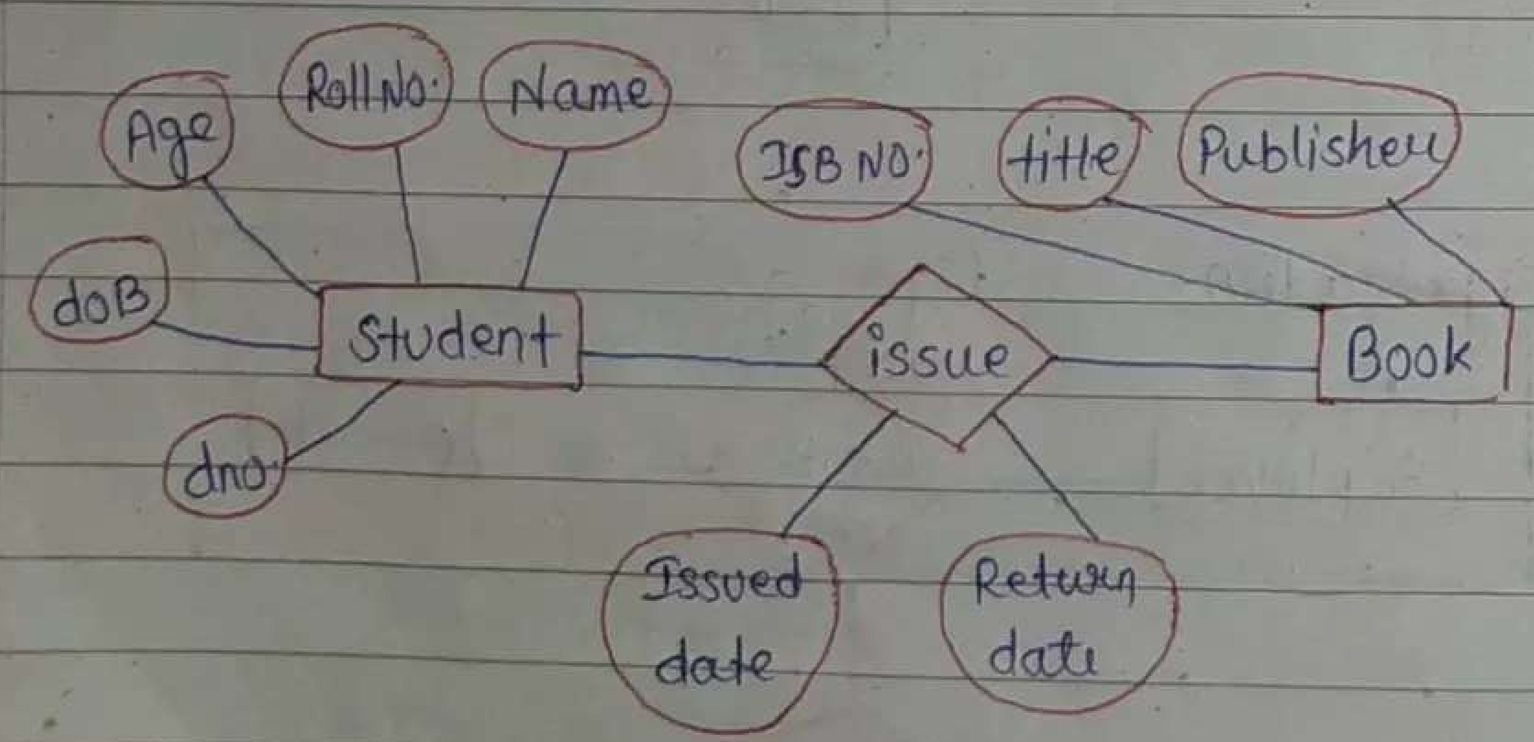
@ Aggregation



@ ER diagram of Banking System



07/09/2020



STUDENT

Name	Roll no.	Age	DOB	dNo.
xx	xx	xx	xx	xx
xx	xx	xx	xx	xx
xx	xx	xx	xx	xx
xx	xx	xx	xx	xx

BOOK

Title	Publisher	ISB No.
xx	xx	xx
xx	xx	xx
xx	xx	xx
xx	xx	xx

ISSUE

Roll no.	Issued date	Return date
xx	xx	xx
xx	xx	xx
xx	xx	xx

@ Relational Model

foreign key

primary key

STUDENT

Roll no.	Name	Age	City	d.no
1	X	21	LKO	d1
2	Y	22	GKP	d2
3	Z	20	GKP	d1
4	X	20	LKO	d3
5	P	20	KNP	d5

Here,
student = relation
each row = tuple
each column = attributes

@ Constraints → Limit

1) Integrity constraints

a) Entity Integrity → (Primary key)

b) Referential Integrity → (Foreign key)

c) Domain constraints →

dept·

primary key ⟵

dno·	dname	dloc·
d_1	CS	A
d_2	EC	B
d_3	ME	C
d_4	CE	D

10|09|2020

Domain constraint

① particular field
② specific field

eg id → int
check $21 \leq Age \leq 30$

@ How to make constraint ?

```
Create table emp
(
#  id  int,        Primary key
   name varchar(100)
   Age  int,
   city varchar(50),
   dno  varchar(10)
        );
```

emp

id	name	age	city	dno·
1	X	21	LKO	d_1
2	Y	23	GKP	d_2

```
# Alter table emp
  add constraint Pri primary key (id)
```

NOTE - Primary Key = Unique key + Not null.

ASSIGNMENT - 2

11|09|2020

1) Create a database SMS & Use it·

```
CREATE DATABASE SMS ;
USE SMS ;
```

2) Create table emp (id, name, age, city, salary)

```
CREATE TABLE emp (
        id int,
        name varchar (50),
        age int,
        city varchar (50),
        salary int
        );
```

3) Make id as primary key (after table creation)

```
ALTER TABLE emp
ADD PRIMARY KEY (id);
```

4) Insert to records in emp table·

int = X
char = ✓

```
INSERT INTO emp (id, name, age, city, salary)
VALUES ('1', 'A', '20', 'LKO', '15000');
VALUES ('2', 'B', '21', 'LKO', '17000');
VALUES ('3', 'C', '22', 'LKO', '17500');
VALUES ('4', 'D', '20', 'GKP', '16000');
VALUES ('5', 'E', '21', 'VNS', '15000');
VALUES ('6', 'F', '20', 'LKO', '12000');
VALUES ('7', 'G', '22', 'VNS', '15000');
VALUES ('8', 'H', '22', 'VNS', '20000');
VALUES ('9', 'I', '20', 'GKP', '22000');
VALUES ('10', 'J', '20', 'LKO', '12000');
```

5) Update salary of an employee whose id = 2

```
UPDATE emp
SET salary = '25000'
WHERE id = 2 ;
```

6) List the name of employees from city Lucknow.

```
SELECT name
From emp
WHERE city = 'LKO';
```

7) Increase salary of employees by 10 % whose salary <=3000

```
UPDATE emp
SET salary = Salary + (0.1 * salary)
WHERE salary <= 30000 ;
```

8) Delete record of employee whose id = 4.

```
DELETE FROM emp
WHERE id = '4' ;
```

9) Add a column dno. in emp table.

```
ALTER TABLE emp
ADD dno. varchar (10) ;
```

10) Drop a column city.

```
ALTER TABLE emp
DROP COLUMN city ;
```

11) Sort the record of all employees according to salary in ascending as well as in descending order.

```
SELECT * FROM emp          | SELECT * FROM emp
ORDER BY Salary ASC        | ORDER BY Salary DESC
```


14/09/2020

employee

id	Name	Age	salary	city
1	Amit	21	35000	LKO
2	Vineet	20	30000	LKO
3	Ajit	22	27500	BPL
4	Karan	21	25000	GKP
5	Saud	20	31000	VNS

```
create table emp
(
id, int ,
name varchar (100),
Age int ,
sal int ,
city varchar (50)
) ;
```

Insert into emp
value ('1', 'Amit', '21', '35000', 'LKO');

Alter employee | Alter table employee
add constraint Pk primary key (id)

create table employee
(id int primary key

Alter employee
add constraint Pk primary key (id, fid) } composite key

delete from employee
where id = 2

select id, name, sal from employee
select * FROM employee

FOR GKP only,
select id, name, sal from employee
where city = 'GKP'

Alter emp
add column dno varchar (10)

select Name from employee
where salary >= 40000 AND Salary <= 50000

select Name from employee
where Age <= 30 AND sal >= 40000

Order by
select * from employee
order by age asc
order by salary asc

order by salary desc
order by age desc

WILDCARDS

Select * from employee
where name like 'A%' → Start with A

where name like '%t' → End with T

where name '__m%' → Exact M

where name '%nee%' → contain nee

18/09/2020

CONSTRAINTS

Select * from student
insert into student values ('1', 'ajay', 23, 'gkp', 'd3', 22000)

alter table student
add constraint pk primary key (roll)

update student
set scholar = 20000 where roll no = 16

update student
set scholar = 20000 where scholar is null

alter table student
add constraint chk check (age <= 25)

select * from student
select * from dept
FOREIGN KEY
alter table student
add constraint fk foreign key (deptno) references dept(dno).
select * from student, dept

21/09/2020

@ Relational Algebra

① Selection

$$\left\{ \begin{array}{l} (\sigma) \} \rightarrow \text{symbol} \\ \sigma_{<condition>}(R) \} \rightarrow \text{syntax} \end{array} \right.$$

σ (emp)

row — $\sigma_{city = 'LKO'}$ (emp)

emp

id	name	age	salary	city
1	X	21	21000	LKO
2	Y	20	25000	GKP
3	P	20	28000	LKO

select * from emp
where city = 'LKO'. } SQL

② Projection

$$\left\{ \begin{array}{l} (\pi) \rightarrow \text{symbol} \\ \pi_{(A_1, A_2, \ldots A_n)}(R) \rightarrow \text{Syntax} \end{array} \right\}$$

column — select name, salary from emp. } SQL

$\pi_{name, salary}$ (emp)

Ques. Find the name and salary of employees who belongs to LKO.

① select name, salary from emp
where city = 'LKO' } SQL

② $\pi_{name, salary}(\sigma_{city = 'LKO'}(emp))$

OR

③ $R_1 \rightarrow \sigma_{city = 'LKO'}(emp)$

$R_2 \rightarrow \pi_{name, salary}(R_1)$

Ques. Find the name of employees whose sal > 25000 and belong to LKO.

① select name from emp
where sal > 25000 AND city = 'LKO'

② $\pi_{name}(\sigma_{sal > 25000 \text{ AND } city = 'LKO'}(emp))$

$\pi_{name}(\sigma_{name \text{ like } 's\%'}(emp))$

24/09/2020

@

UNION

$R \cup S = \{ t \mid t \in R$ or $t \in S \}$

R

A	B
a_1	b_1
a_2	b_2
a_3	b_3

S

A	B
a_2	b_2
a_4	b_4
a_5	b_5

R ∪ S

A	B
a_1	b_1
a_2	b_2
a_3	b_3
a_4	b_4
a_5	b_5

arity must be same.

INTERSECTION

$R \cap S = \{ t \mid t \in R$ and $t \in S \}$

R

A	B
a_1	b_1
a_2	b_2
a_3	b_3

S

A	B
a_2	b_2
a_4	b_4
a_5	b_5

R ∩ S

A	B
a_2	b_2

arity must be same.

SET DIFFERENCE

$R - S = \{ t \mid t \in R$ an $t \notin S$

R

A	B
a_1	b_1
a_2	b_2
a_3	b_3

S

A	B
a_2	b_2
a_4	b_4
a_5	b_5

R − S

A	B
a_1	b_1
a_3	b_3

arity must be same.

@ Cartesian product (CROSS JOIN)

$A = \{\underbrace{a_1, a_2, a_3}_{n}\}$ $B = \{\underbrace{b_1, b_2}_{n}\}$

$A \times B = \{(x,y) \mid x \in A \ \& \ y \in B\}$

$A \times B = \{(a_1,b_1), (a_1,b_2), (a_2,b_1), (a_2,b_2), (a_3,b_1), (a_3,b_2)\}$

$A \times B = m * n$ elements.

#

X_1

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

X_2

D	E
d_1	e_1
d_2	e_2

$(X_1 \times X_2)$

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_3	b_3	c_3	d_1	e_1
a_3	b_3	c_3	d_2	e_2

#

emp

id	name	dno.
1	X	d_1
2	Y	d_1
3	Z	d_2

dept.

dept.no.	deptname
d_1	CS
d_2	EC

emp × dept

id	name	dno.	dept.no.	deptname
1	X	d_1	d_1	CS
1	X	d_1	d_2	EC
2	Y	d_1	d_1	CS
2	Y	d_1	d_2	EC
3	Z	d_2	d_1	CS
3	Z	d_2	d_2	EC

01/10/2020

@ Join

1) Cross join $\boxtimes$
2) ~~Normal join~~ Natural join
3) Outer join a) Left outer join
   b) Right outer join
   c) Full outer join

@ Natural join

emp ⋈ dept (direct filter → Matching)

$\pi_{emp}(\sigma_{dept.dname = 'cs'}(emp \bowtie dept))$

05/OCT/2020

@ Outer join

1) Left outer join → ⟕
2) Right outer join → ⟖
3) Full outer join → ⟗

emp

id	name	sal	dno
1	X	21000	d1
2	Y	92000	d2
3	Z	72000	d3
4	P	46000	d5

dept.

dept no.	dname
d1	CS
d2	EC
d3	ME
d4	CE

left outer join

emp ⟕ dept

id	name	sal	dno	deptno	dname
1	X	24000	d_1	d_1	CS
2	Y	42000	d_2	d_2	EC
3	Z	72000	d_3	d_3	ME
4	P	46000	d_5	NULL	NULL

left relation = all tuples
right relation = matching tuples

right outer join

emp ⟖ dept

id	name	sal	dno	deptno	dname
1	X	24000	d_1	d_1	CS
2	Y	42000	d_2	d_2	EC
3	Z	72000	d_3	d_3	ME
NULL	NULL	NULL	NULL	d_4	CE

left table = matching tuple
right tuple = all tuples

full outer join

emp ⟗ dept

id	name	sal	dno	deptno	dname
1	X	21000	d_1	d_1	CS
2	Y	42000	d_2	d_2	EC
3	Z	72000	d_3	d_3	ME
4	P	46000	d_5	NULL	NULL
NULL	NULL	NULL	NULL	d_4	CE

all tuples

07/OCT/2020

1) Student (<u>rollno</u>, name, age, branch, city)

Book (ISBN, title, author, publication, Y_pub)

Book issue (rollno, ISBN, issue date, returndate)

a) Find the name of students belongs to LKO.

$\pi_{name}(\sigma_{city = 'LKO'}(student))$

b) Find the title & author of books issued before 10 sep 2020.

Book

ISBN	title	Author	Pub	Y_Pub
15001	DBMS	Forth	POR	2016
15002	TAFL	ABC	XY	2017
15003	DAA	EFG	ST	2018

Book-issue

Rollno	ISBN	issue date	return date
1	15001	10 sep 20	-
2	15002	9 sep 18	-

$\pi_{Book.title, Book.Author}(\sigma_{issuedate < 10 sep 2020}(Book \bowtie Book issue))$

c) Find the title & author of book published before 2018.

$\pi_{title, Author}(\sigma_{Y\text{-}pub < 2018}(Book))$

d) Find the name of student who have issued a book of DBMS.

$\pi_{student.name}(\sigma_{Book.title = "DBMS"}(Book \bowtie Book issue) \bowtie student)$

12/10/2020

@ NOTE

Join

- Cross join
  - $A \times B$
  - $m \quad n$
  - $m \times n$
- Natural join
  - $A \bowtie B$
- Outer join
  - A ⟕
  - A ⟖
  - A ⟗

@ Join in SQL

→ SQL

① Equi join

② Non equi join

emp

id	name	age	sal	dno
1	X	21	28000	d1
2	Y	22	27000	d2
3	Z	20	32000	d1

dept

dno	dname
d1	CS
d2	EC

emp X dept

id	name	age	sal	dno	dno	dname
1	X	21	28000	d1	d1	CS
1	X	21	28000	d1	d2	EC
2	Y	22	27000	d2	d1	CS
2	Y	22	27000	d2	d2	EC
3	Z	20	32000	d3	d1	CS
3	Z	20	32000	d3	d2	EC

1) name, sal of CS dept

select * from emp, dept

$\pi_{emp.name, emp.sal}(\sigma_{dept.dname = 'CS'}(emp \times_{emp.dno = dept.no} dept))$

select * from emp, dept

select emp.name, empsal from emp.dept

where emp.dno. = dept.dno.

and dept.dname = 'CS'

ⓐ Cross join

ⓑ inner join

ⓒ outer join

① left outer

② right outer

③ full outer

select emp.name, emp.sal

from emp $\boxed{\text{inner join}}$ dept

on emp.dno. = dept.dno.

where dept.dname = 'CS'

Select emp.name, emp.sal

from emp $\boxed{\text{left outer join}}$ dept

on emp.dno = dept.dno.

14/OCT/2020

1) student (Roll no., name, age, branch)

Book issue (Roll no., ISBN, title, issue date, return date)

List the name of students who have issued books.

student

Roll no.	name	age	branch
1	X	23	CS
2	Y	20	EC
3	Z	20	EN
4	P	21	CE
5	Q	22	ME

Book issue

Rollno.	ISBN	title	issue date	return date
1	B01	DBMS		
4	B02	DSUC		

```
select student name from student inner join Book.issue
on student Roll no = Bookissue.Rollno
where book.issue.title = 'DBMS'
```

@ Subqueries

emp

id	name	sal	dept no.
1	X	21000	–
2	Y	52000	–
3	Z	38000	–
4	P	76000	–
5	Q	54000	–
6	R	28000	–

	sal
0	62000
1	59000
2	51000
3	38000
4	32000
5	21000

general form N-1, 1

2nd highest salary

```
select sal from emp
order by sal desc
limit 1,1
```

3rd highest salary

```
select sal from emp
order by sal desc
limit 2,1
```

1) Find the name of employees belongs to cs dept.

dept.

dno	dname	dloc
d1	CS	A
d2	EC	B
d3	EN	C

emp.

id	name	sal	city	dno
1	X	21000	LKO	d1
2	Y	35000	GKP	d2
3	Z	62000	KNP	d3
4	P	82000	LKO	d4

```
# select empname from
emp inner join dept
on emp.dno = dept.dno
where dept.dname = 'CS'
```

```
# select name from emp
where dno = ( select dno from
              dept where dname
              = 'CS')
```

2) Find the name of employees whose sal is greater than avg. sal.

```
select name from emp
where sal > ( select avg (sal)
              from emp)   50000
```

3) Find the name of employees of cs dept who belongs to LKO or GKP.

4) Find the name of all employees belongs to LKO or GKP.

```
select name from emp
where city = 'LKO' or city 'GKP'
```

```
select name from emp
where city IN ('LKO', 'GKP')
```

① = >, <, <>
② IN, NOT IN
③ ANY, SOME
④ ALL

21/10/2020

@ Subqueries continue ----

emp

id	name	age	sal	city	dno
1	X	21	22000	LKO	d1
2	Y	25	42000	GKP	d1
3	Z	27	80000	GKP	d2
4	P	32	38000	KNP	d3
5	Q	56	70000	LKO	d3
6	R	39	80000	LKO	d2
7	S	42	34000	GKO	d1

dept

deptno	dname	dloc
d1	CS	A block
d2	EC	B block
d3	ME	C block

1) Average sal ?

```
select avg(sal) from emp
    group by dno
      or group by city
```

```
select name from emp
    where sal > (select avg(sal)
                   from emp)
```

2) Find the name of employees whose salary is greater than avg salary of all the department.

```
select name from emp
where sal > All (select avg(sal) from emp
                   group by dno)
```

3) ---- smaller than ----

```
select name from emp
where sal < All (select avg(sal) from emp
                   group by dno)
```



```
select name from emp
  where sal IN (select avg(sal) from emp
                   group by dno)
```

```
select name from emp
  where sal IN (70000, 80000, 34000)
```

```
select name from emp
  where dno IN (select deptno from dept where
                   dname = 'CS' or dname = 'ME')
```

22/10/2020

@ emp (id, name, age, sal, city, dno)
dept (deptno, dname, dloc)

1) Find the name of employees not belongs to Lucknow, Kanpur, Agra.

```
select name from emp where
city NOT IN ('Lucknow', 'Kanpur', 'Agra').
```

2) Find the name of employees not belongs to CS and EC dept.

```
select name from emp where
 dno NOT IN (select deptno from dept where
                dname IN ('CS', 'EC'))
```

→

```
select name from emp where
dno NOT IN (select deptno from dept where
                dname = 'CS' AND
                dname = 'EC')
```

→ select name from emp where
sal > ANY (select avg(sal) from emp
group by dno)

3) Find the name of employees whose salary is greater than avg. sal of atleast 1 dept.

select name from emp where
sal > ANY (select avg(sal) from emp
group by dno).

04/NOV/2020

@ Functional Dependency

R

	A	B	C
t_1	a_1	b_1	c_1
t_2	a_2	b_2	c_2
t_3	a_1	b_1	c_3
t_4	a_3	b_3	c_4

R = relation
A, B, C = attributes

set of tuples = relation

$X \rightarrow Y$ (where, X determines Y)

X, Y are sub set of attributes
of relation R

R {A, B, C}

$\{A\} \rightarrow \{C\}$

$AB \rightarrow C$

$\{AB\} \rightarrow \{C\}$

functional dependency of attributes are represented as $X \rightarrow Y$ (X determines Y) holds if $t_1[X] = t_2[X]$
$\Rightarrow t_1[Y] = t_2[Y]$

eg. if $t_1[A] = t_3[A]$
implies
$t_1[B] = t_3[B]$ $\}$ $A \rightarrow B$ holds

$B \rightarrow A$ hold
$C \rightarrow A$ hold
$A \rightarrow C$ not hold

R

	A	B	C
t_1	a_1	b_1	c_1
t_2	a_2	b_2	c_2
t_3	a_1	b_1	c_1
t_4	a_3	b_1	c_4

$AB \rightarrow C$ hold

$A \rightarrow BC$ hold

trival dependency

$Ax + By + Cz = 0$

$x = y = z = 0$

$X \rightarrow Y$

if $Y \subseteq X$

$AB \rightarrow A$ when RHS is a subset of LHS it is k/a trival dependency.

$A \rightarrow A$ trival dependency (hold)

05/Nov/2020

@ Armstrong Axioms

Consider a relation $R(W, X, Y, Z)$ where $X \subseteq R$, $Y \subseteq R$

① Reflexive

If $Y \subseteq X$

then $X \rightarrow Y$ holds

eg. $X \rightarrow X$

$Y \rightarrow Y$

② Augmentation

If $X \rightarrow Y$

then $XW \rightarrow YW$ also holds

③ Transitive

If $X \rightarrow Y$ & $Y \rightarrow Z$ holds

then $X \rightarrow Z$ also holds

④ Decomposition

If $X \rightarrow YZ$ holds

then $X \rightarrow Y$ & $X \rightarrow Z$ also holds

⑤ Union

If $X \rightarrow Y$ & $X \rightarrow Z$ holds

then $X \rightarrow YZ$ also holds.

⑥ Pseudo transitivity

If $X \rightarrow Y$ & $WY \rightarrow Z$ holds

then $XW \rightarrow Z$ also holds

@ Closure of attributes in a functional dependency set

① closure of attributes in FD set

② closure of FD set

$$\begin{cases} R(A, B, C) \\ F(A \rightarrow B, B \rightarrow C) \end{cases}$$

F^+ = set of all FD that are present in FD set F or can be infered using FD in FD set F.

$$F^+ = \{ A \rightarrow A,\ B \rightarrow B,\ C \rightarrow C,\ A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C \}$$

@ Closure of attributes in FD set

R (A, B, C, D)

F { A → B, C → D }

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

closure of $\{AC\}^+ = \{A\ C\ B\ D\}$

↓

KEY

or

SUPER KEY

06/Nov/2020

eg:

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

F { A → B, B → C, BC → E, D → A }

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

↓

KEY (Minimal key or Candidate key)

closure of $\{E\}^+ = \{E\}$
closure of $\{DA\}^+ = \{DABCE\}$

↓

KEY (Super key)

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

F { C → F, E → A, EC → D, A → B }

(A) CD
(B) EC (✓)
(C) AE
(D) AC

closure of $\{CD\}^+ = \{CDF\}$
closure of $\{EC\}^+ = \{ECAFDB\}$

↓

Candidate key

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

F { A → B, BC → D, E → C, D → A }

Candidate key = ?

closure of $\{EH\}^+ = \{EHC\}$
closure of $\{AEH\}^+ = \{AEHBCD\}$ (✓) → candidate key
closure of $\{BEH\}^+ = \{BEHCDA\}$ (✓) → candidate key
closure of $\{CEH\}^+ = \{CEH\}$
closure of $\{DEH\}^+ = \{DEHACB\}$ (✓) → candidate key

No. of candidate keys = 3

3)

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

(A) XY → Z & Z → Y
(B) YZ → X & Y → Z (✓)
(C) YZ → X & X → Z
(D) XZ → Y & Y → X

09/NOV/2020

1) R(A B C D E F G H)
F { CH → G, A → BC, B → CFH, E → A, F → EG }
Find the candidate keys ?

~~closure of {D}~~

D → D
closure of $\{D\}^+ = \{D\}$
closure of $\{DA\}^+ = \{DABCFHEG\}$ (✓)
closure of $\{DB\}^+ = \{DBCFHEGA\}$ (✓)
closure of $\{DC\}^+ = \{DC\}$
closure of $\{DE\}^+ = \{DEABCFHG\}$ (✓)
closure of $\{DF\}^+ = \{DFEGABCH\}$ (✓)
closure of $\{DG\}^+ = \{DG\}$
closure of $\{DH\}^+ = \{DH\}$

candidates keys → {DA, DB, DE, DF}
No. of possible candidates keys = 04

closure of $\{DGB\}^+ = \{DGBCFHEA\}$
↓
super key

2) R(E F G H I J K L M N)
F { EF → G, F → IJ, EH → KL, K → M, M → N }
Find the candidate keys ?

closure of $\{EFH\}^+ = \{EFHGIJKLMN\}$
↓
candidate key

ⓑ ~~Canonical cover~~

11/NOV/2020

ⓐ Minimal cover

1) MINIMAL FUNCTIONAL DEPENDENCY SET

R(W X Y Z)
F { X → W, WZ → XY, Y → WXZ }

Closure of $(Y)^+ = \{WXYZ\}$
Closure of $(WZ)^+ = \{WZXY\}$
closure of $(WZX)^+ = \{WZXY\}$

① X → W
WZ → X
WZ → Y
Y → W
Y → X
Y → Z

① decompose
② Extra F.D
③ Closure
③ Minimise L.H.S if more than 1 attributes are present.
④

② X → W ✓
closure of $(X)^+ = \{XW\}$
after removing the above closure, Now
closure of $(X)^+ = \{X\}$

WZ → X Extra
closure of $\{WZ\}^+ = \{WZXY\}$
after removing the above closure, Now
closure of $(WZ)^+ = \{WZYX\}$

$WZ \to Y$ ✓

closure of $(WZ)^+ = \{WZYX\}$

after removing the above closure, Now

closure of $(WZ)^+ = \{WZ\}$

$Y \to W$ Extra

closure of $(Y)^+ = \{YWXZ\}$

after removing the above closure, Now

closure of $(Y)^+ = \{YXZW\}$

$Y \to X$ ~~Extra~~ ✓

closure of $(Y)^+ = \{YWXZ\}$

after removing the above closure, Now

closure of $(Y)^+ = \cancel{\{YXZW\}}\ \{YZ\}$

$Y \to Z$ ✓

closure of $(Y)^+ = \{YZXW\}$

after removing the above closure, Now

closure of $(Y)^+ = \{YXW\}$

③ $WZ \to Y$

closure of $(W)^+ = \{W\}$

closure of $(Z)^+ = \{Z\}$

④
$$\left.\begin{array}{l} X \to W \\ WZ \to Y \\ Y \to XZ \end{array}\right\}$$ Minimal Functional dependency set.

1) $R(ABC)$

$F\{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$

Find the minimal FD set.

① $A \to B$ ✗

$A \to C$

$B \to C$

$A \to B$

$AB \to C$

② $A \to C$ Extra

closure of $\{A\}^+ = \{ABC\}$

after removing the above closure, Now

closure of $\{A\}^+ = \{ABC\}$

$B \to C$ ✓

closure of $\{B\}^+ = \{BC\}$

after removing the above closure, Now

closure of $\{B\}^+ = \{B\}$

$A \to B$ ✓

closure of $\{A\}^+ = \{ABC\}$

after removing the above closure, Now

closure of $\{A\}^+ = \{A\}$

$AB \to C$ Extra

closure of $\{AB\}^+ = \{ABC\}$

after removing the above closure, Now

closure of $\{AB\}^+ = \{ABC\}$

③ $A \to B$, $B \to C$ } Minimal functional dependency set.

12/NOV/2020

@ Cover

eg $R(ABCD)$

$F_1 \{A \to B, B \to D, D \to C\}$

$F_2 \{A \to BD, B \to C, C \to D, D \to A\}$

① F_1 covers F_2

$A \to BD$

closure of $\{A\}^+ = \{ABDC\}$ ✓

$B \to C$

closure of $\{B\}^+ = \{BDC\}$ ✓ { F_1 did not cover F_2 }

$C \to D$

closure of $\{C\}^+ = \{C\}$ ✗

② F_2 covers F_1

$A \to B$

closure of $\{A\}^+ = \{ABDC\}$ ✓

$B \to D$ { F_2 covers F_1 }

closure of $\{B\}^+ = \{BCD\}$ ✓

$D \to C$

closure of $\{D\}^+ = \{BCDA\}$ ✓

@ Minimal cover { To find Minimal FD set }

eg $R(ABCD)$

$F_1 \{A \to B, B \to D, D \to C\}$

$F_2 \{A \to BD, B \to C, C \to D, D \to A\}$

F_1 ① $A \to B$

$B \to D$

$D \to C$

② After removing $A \to B$

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

$\{B\}^+ = \{BDC\}$ → after removing $B \to D$

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

$\{D\}^+ = \{DC\}$

after removing $D \to C$

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

③ F_1 is a minimal FD set

F_2 ① $A \to B$ ✓

$A \to D$ ✗

$B \to C$ ✓

$C \to D$

$D \to A$

② i) $\{A\}^+ = \{ABDC\}$

after removing $A \to B$

$\{A\} = \{AD\}$

ii) $\{A\}^+ = \{ABDC\}$

after removing $A \to D$

$\{A\}^+ = \{ABCD\}$

iii) $\{B\}^+ = \{BCDA\}$

after removing $B \to C$

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

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

after removing $C \to D$

$\{C\}^+ = \{C\}$

v) $\{D\}^+ = \{DABC\}$

after removing $D \to A$

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

After above solution, we have new F_2 set

$F_2 = \{A \to B, B \to C, C \to D, D \to A\}$

@ To Find Minimal FD set

F_2 cover F_1

$A \to B$ implied in F_2

$B \to D$ implied in F_2

$\{B\}^+ = \{BCDA\}$

$D \to C$ implied in F_2

$\{D\}^+ = \{DABC\}$

F_2 is minimal cover of F_1

or F_2 is canonical cover of F_1.

ASSIGNMENT

18/NOV/2020

1) Let R = (A, B, C, D, E, F) be a relation scheme with the following dependencies -

$C \to F$, $E \to A$, $EC \to D$, $A \to B$

Find the candidate key.

closure of $\{EC\}^+ = \{ECAFDB\}$

↓

candidate key

2) The following functional dependencies are given -

$AB \to CD$, $AF \to D$, $DE \to F$, $C \to G$, $F \to E$, $G \to A$

Which one of the following options is false ?

(a) $CF^+ = \{ACDEFG\}$

(b) $BG^+ = \{ABCDG\}$

(c) $AF^+ = \{ACDEFG\}$

(d) $AB^+ = \{ABCDFG\}$

closure of $\{AF\}^+ = \{ADEF\}$ {option C}

doesn't contain C and G.

closure of $\{AB\}^+ = \{ABCDG\}$ {option D}

does not contain F.

3) In a schema with attributes A, B, C, D, E, following set of functional dependencies are given -

$A \to B$

$A \to C$

$CD \to E$

$B \to D$

$E \to A$

which of the following functional dependencies is NOT implied by the above set ?

(a) $CD \to AC$
(b) $BD \to CD$
(c) $BC \to CD$
(d) $AC \to BC$

closure of $\{CD\}^+ = \{CDEAB\}$
closure of $\{BD\}^+ = \{BD\}$
closure of $\{BC\}^+ = \{BCDEA\}$
closure of $\{AC\}^+ = \{ACBDE\}$

option (b) $BD \to CD$

4) Consider the relation scheme $R = (E,F,G,H,I,J,K,L,M,N)$ and the set of functional dependencies $\{\{E,F\} \to \{G\}, \{F\} \to \{I,J\}, \{E,H\} \to \{K,L\}, K \to \{M\}, L \to \{N\}$ on R.
What is the key for R ?

(a) $\{E,F\}$
(b) $\{E,F,H\}$
(c) $\{E,F,H,K,L\}$
(d) $\{E\}$

All attributes can be derived from $\{E,F,H\}$

closure of $\{EF\}^+ = \{EFGIJ\} \neq R$
closure of $\{EFH\}^+ = \{EFGHIJKLMN\}$ → Candidate key
closure of $\{EFHKL\}^+ = \{EFGHIJKLMN\}$ → Super key
closure of $\{E\}^+ = \{E\} \neq R$

option (b) $\{E,F,H\}$

5) Consider the relation $X(P,Q,R,S,T,U)$ with the following set of functional dependencies $F = \{\{P,R\} \to \{S,T\}, \{P,S,U\} \to \{Q,R\}\}$
Which of the following is the trivial functional dependency in F+ is closure of F ?

(a) $\{P,R\} \to \{S,T\}$
(b) $\{P,R\} \to \{R,T\}$
(c) $\{P,S\} \to \{S\}$
(d) $\{P,S,U\} \to \{Q\}$

A functional dependency $X \to Y$ is trivial if Y is a subset of X.

option (c) $\{P,S\} \to \{S\}$

19|NOV|2020

(a) Minimal cover continue ----

Consider a relation R & F_1, F_2 are two functional dependency sets of relation R, ~~then~~, if all the functional dependencies of FD set F_1 are implied in FD set F_2, then F_2 covers F_1 $(F_2 \supseteq F_1)$

Similarly, if all the Functional dependencies of FD set F_2 are implied in FD set F_1, then F_1 covers F_2 $(F_1 \supseteq F_2)$

If F_1 covers F_2 & F_1 is minimal FD set, then F_1 is k/a Minimal cover or canonical cover of FD set F_2.

If $F_1 \supseteq F_2$ & $F_2 \supseteq F_1$,
then F_1 & F_2 are k/a equivalent FD set.

1) $R(ABCD)$
$F_1 \{ A \to C,\ C \to B,\ B \to D \}$
$F_2 \{ A \to BD,\ C \to \cancel{BD}\,AD,\ B \to C \}$
are they equivalent ?

F_1 covers F_2

$A \to C$	$C \to B$	$B \to D$
$\{A\}^+ = \{ACBD\}$	$\{C\}^+ = \{CBD\}$	$\{B\}^+ = \{BD\}$

F_2 covers F_1

$A \to BD$	$C \to AD$	$B \to C$
$\{A\}^+ = \{ABDC\}$	$\{C\}^+ = \{CADB\}$	$\{B\}^+ = \{BCAD\}$

F_1 covers F_2 = ✗
F_2 covers F_1 = ✓

so, not equivalent.

@ DECOMPOSITION

R
R_1 R_2

decomposition
↓
ⓐ lossless
ⓑ dependency preserving

$R(ABCD)$

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_2	d_2
a_3	b_3	c_3	d_3

decomposition →

$R_1(AB)$

A	B
a_1	b_1
a_2	b_2
a_3	b_3

$R_2(CD)$

C	D
c_1	d_1
c_2	d_2
c_3	d_3

↓
after combing R_1 & R_2 we get

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_2
a_1	b_1	c_3	d_3
a_2	b_2	c_1	d_1
a_2	b_2	c_2	d_2
a_2	b_2	c_3	d_3
a_3	b_3	c_1	d_1
a_3	b_3	c_2	d_2
a_3	b_3	c_3	d_3

spurius tuples

we cannot get the same relation R.

20/Nov/2020

@ Decomposition continue -----

\# Consider a relation R. If it is decomposed into two sub relations as R_1 & R_2.
then, the decomposition is loosless,
if $R_1 \cap R_2 \rightarrow R_1$
or
$R_1 \cap R_2 \rightarrow R_2$

\# That means the common attribute of relation R_1 & R_2 must be the key of relation R_1 or relation R_2 for lossless decomposition, otherwise the decomposition is lossy.

\# eg. R(ABC)
R_1(AB)
R_2(BC)
$R_1 \cap R_2 = \{B\}$

\# If $R_1 \cap R_2$ is ϕ,
then decomposition is always lossy.

1) R(ABC)
F $\{A \rightarrow B, B \rightarrow C\}$

R_1(AB)	R_2(BC)
$A \rightarrow B$	$B \rightarrow C$
$\{R_1 \cap R_2\} = \{B\}$ $\{B\}^+ = \{B\}$	$\{R_1 \cap R_2\} = \{B\}$ $\{B\}^+ = \{BC\}$

Hence, $\{R_1 \cap R_2\} \rightarrow R_2$
decomposition is lossless.

2) R(ABC)
F $\{A \rightarrow B, C \rightarrow B\}$

R_1(AB)	R_2(BC)
$A \rightarrow B$	$C \rightarrow B$
$\{R_1 \cap R_2\} = \{B\}$ $\{B\}^+ = \{B\}$	$\{R_1 \cap R_2\} = \{B\}$ $\{B\}^+ = \{B\}$

Hence, $\{R_1 \cap R_2\} \rightarrow \phi$
decomposition is lossy

3) R(ABCD)
F $\{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B\}$
(AB) (BC) ~~(BD)~~ (BD)
decomposition is lossless or lossy ?

ABCD
ABC BD
AB BC

R_1(AB)	R_2(BC)	R_3(BD)
$A \rightarrow B$	$B \rightarrow C$ $C \rightarrow B$	$B \rightarrow D$ $D \rightarrow B$

now check lossless

~~R1(AB)(BC)~~

~~yes superkey of R2~~

R1 (AB) R (BC)

yes superkey of R2

R1R (ABC) R3 (B, D)

yes key of R3

So, lossless is garunted

	$\{A\}^+ = \{ABCD\}$
	$\{B\}^+ = \{BCD\}$
	$\{C\}^+ = \{CDB\}$
	$\{AB\}^+ = \{ABCD\}$
	$\{BC\}^+ = \{BCD\}$

23/NOV/2020

@ Lossless decomposition continue ----

R

R1 R2

① $R_1 \cap R_2 = \phi$

② $\{R_1 \cap R_2\} \rightarrow R_1$

or

$\{R_1 \cap R_2\} \rightarrow R_2$

1) R(ABCD)

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

$R_1(AB),\ R_2(BC),\ R_3(BD)$

decomposition is lossless or lossy ?

R(ABCD)

R1(ABC) R2(BD)

R3(AB) R4(BC)

R3 (AB)	R4 (BC)
$A \rightarrow B$	$B \rightarrow C$
$B \rightarrow C$	$C \rightarrow D$
$\{B\}^+ = \{BC\}$	$D \rightarrow B$
	$\{B\}^+ = \{BCD\}$

R1 (ABC)	R2 (BD)
$A \rightarrow B$	$B \rightarrow C$
$B \rightarrow C$	$C \rightarrow D$
$C \rightarrow D$	$D \rightarrow B$
$\{B\}^+ = \{BCD\}$	$\{B\}^+ = \{BCD\}$

So, decomposition is lossless.

@ Dependency Preservation

R(ABC)

$\{A \rightarrow B,\ B \rightarrow C,\ \cancel{C \rightarrow A}\}$

R1 (AB)	R2 (BC)
$F_1 \{A \rightarrow B\}$	$F_2 \{B \rightarrow C\}$
$\{A\}^+ = \{AB\}$	$\{B\}^+ = \{BC\}$
$\{B\}^+ = \{B\}$	

$$R\{F\} \rightarrow R_1\ \{F_1\},\ R_2\ \{F_2\},\ R_3\ \{F_3\} \cdots R_n\ \{F_n\}$$

$\{F_1\}^+ \cup \{F_2\}^+ \cup \{F_3\}^+ \cdots \cup \{F_n\}^+ = \{F\}^+$

$\{F_1\}^+ = \{A \to B,\ A \to A,\ B \to B\}$

$\{F_2\}^+ = \{B \to C,\ B \to B,\ C \to C\}$

$\{F_1\}^+ \cup \{F_2\}^+ = \{A \to B, A \to A, B \to B, B \to C, C \to C\}$

$\{F\}^+ = \{A \to A,\ B \to B,\ A \to B,\ B \to C,\ A \to C\}$

decomposition is lossless
but not dependency preserving

@ Dependency Preserving continue ----
Consider a relation R having FD set F. Relation R is decomposed into $R_1, R_2, R_3 \cdots R_n$ having FD set $F_1, F_2, F_3 \cdots F_n$ respectively. Then, the decomposition is dependency preserving → if $\{F_1\}^+ \cup \{F_2\}^+ \cup \{F_3\}^+ \cdots \cup \{F_n\}^+ = \{F\}^+$.

@ NOTE
decomposition
→ lossless
→ dependency preserving

24/NOV/2020

@ Functional dependency & Normalization

Consider a relation R, a functional dependency $X \to Y$ represents X functionally determines Y or Y is functionally dependent on X, where X and Y are the subset of attribute of relation R.

$X \to Y$ holds if $t_1[X] = t_2[X]$
$\Rightarrow t_1[Y] = t_2[Y]$

that means value of X in tuple t_1 is same as value of X in tuple t_2, then the value of Y in tuple t_1 must be equal to the value of Y in tuple t_2 to hold the functional dependency $X \to Y$.

For different X values in two tuples, the value of Y may or may not be same.

03/12/2020

1) Consider R(ABC) is a relation having FD set F $\{A \to B, B \to C\}$. Relation R is decomposed into two sub relation $R_1(AB)$ & $R_2(BC)$. Then, find that the decomposition is lossless or lossy.

$$R(ABC) \rightarrow R_1(AB)\ F_1\{A \to B\},\quad R_2(BC)\ F_2\{B \to C\}$$

(1) $\{R_1 \cap R_2\} = B$
Hence, $\{R_1 \cap R_2\} \neq \phi$

(2) $\{R_1 \cap R_2\} = B$

closure of $\{B\}^+$ in relation R_1 using FD set F_1.

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

Hence, $\{B\}$ is not a super key of relation R_1.

(3) closure of $\{B\}^+$ in relation R_2 using FD set F_2.

$\{B\}^+ = \{BC\}$

Hence, $\{B\}$ is a super key of relation R_2.

Hence, $\{R_1 \cap R_2\} \rightarrow R_2$ is true & the decomposition is lossless.

5/12/2020

1) Consider a relation R (ABCD) having F.D set $F\{A \rightarrow CD,\ C \rightarrow BD,\ BD \rightarrow A\}$

(1) $A \rightarrow C$

$A \rightarrow D$

$C \rightarrow B$

$C \rightarrow D$

$BD \rightarrow A$

(2) removing F.D $A \rightarrow C$ ✓

$\{A\}^+ = \{AD\}$

Hence, $A \rightarrow C$ cannot removed.

(3) removing FD $A \rightarrow D$ ✗

$\{A\}^+ = \{ACDB\}$

Hence, $A \rightarrow D$ can be removed.

removing FD $C \rightarrow B$ ✓

$\{C\}^+ = \{CD\}$

Hence, $C \rightarrow B$ cannot be removed.

removing F.D $C \rightarrow D$ ✓

$\{C\}^+ = \{CB\}$

Hence, $C \rightarrow D$ cannot be removed.

removing F.D $BD \rightarrow A$ ✓

$\{BD\}^+ = \{BD\}$

Hence, $BD \rightarrow A$ cannot be removed.

(3) The F.D $BD \rightarrow A$ contain more than 1 attributes on L.H.S.

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

∴, $\{B\}^+$ does not contain D.

So, D cannot be removed.

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

∴, $\{D\}^+$ does not contain B.

So, B cannot be removed.

(4) reduced F.D set

$A \rightarrow C$

$C \rightarrow BD$

$BD \rightarrow A$

7/12/2020

Normalization continue ---

1 NF → should be atomic

2 NF → should be 1 NF
not partial dependency

8/12/2020

→ Normalization

Large relations have many anomalies such as redundancy, inconsistency etc.. To avoid these anomalies & manage the database, normalization is reqd..

In Normalization, any relation is decomposed into sub-relat-ions based on some conditions. ~~The decomposition of relation~~ The normal forms are defined on the basis of these conditions such as -

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

→ 1NF

A relation R is in 1 NF, if the attribute of relation R are 'atomic' i.e. for an attribute, there must be a single value at a time.

eg. emp

id	name	age	contact
1	X	21	8946
2	Y	22	2246
			4896
3	Z	21	2828
			2532
			5492

consider a relation emp having attributes id, name, age & contact. The contact attribute is multi-valued. In relational model, each attribute can have a single value at a time.

For 1 NF, the attribute must be atomic.
Hence, emp is in 1 NF, if its all attributes are atomic.
To convert it into 1 NF, there are two methods –

① emp

id	name	age	contact 1	contact 2	contact 3
1	X	21	8946	–	–
2	Y	22	2246	4896	–
3	Z	21	2828	2532	5492

In 1^{st} method, the schema of relation emp is expanded and for contact attribute, multiple contact columns are introduced.
The drawback of this method is the change in schema of original relation. Also several null values may be introduced.

② emp

id	name	age	contact
1	X	21	8946
2	Y	22	2246
2	Y	22	4896
3	Z	21	2828
3	Z	21	2532
3	Z	21	5492

In 2^{nd} method, the schema of original relation remains same but for each contact a new row is introduced.
Hence, in this relation redundancy occurs.

→ 2 NF

A relation R is in 2 NF, if it is in 1 NF and no non-prime attribute should be partially dependent on the primary key. **OR**
Each non-prime attribute must be fully functionally dependent on the primary key of relation R.

Roll no.	dept no.	name	age	dname	dloc.

R

A	B	C	D	E	F

FD1
FD2
Not in 2NF

R_1

A	B	C	D

R_2

B	E	F

R_1 & R_2 are in 2 NF.

09/12/2020

prime attribute → The attributes that are part of candidate key of relation R are k/a prime attributes.
Other attributes that are not the part of candidate key are k/a non-prime attributes.

eg. R(ABCD)
F{ $A \to B$, $B \to D$ }

$\{AC\}^+ = \{ACBD\}$
Hence, {AC} is candidate key of relation R.
Hence, A & C = prime attributes
B & D = non-prime attributes

For the FD $A \to B$, B is a non-prime attributes and is partially dependent on the candidate key. {AC} of relation R.

Hence, relation R is not in 2NF.

11/12/2020

→ **3 NF**
A relation schema R is in 3NF if -
(1) It is in 2NF
(2) No non-prime attribute A in R is transitively dependent on the primary key.

{not in 3NF}

emp

id	name	age	city	dno	dname	dloc

FD1
FD2

{in 3NF}

emp 1

id	name	age	city	dno

dept

dno	dname	dloc

OR

A relation R is in 3NF if whenever a FD $X \to A$ holds in R, then
X must be a superkey
or
A is a prime attribute

1) R(ABCDEPG)
{ $AB \to CD$
$DE \to P$
$C \to E$
$P \to C$
$B \to G$ }
{AB} = primary key.
$\{AB\}^+ = \{ABCDEPG\}$
A & B = prime attribute
C, D, E, P, G = non-prime

For the FD $B \to G$, G is a non-prime attribute and is partially dependent on the primary key. {AB} of relation R.

Hence, relation R is not in 2NF.

2) R(ABCDEFGH)
F{ $CH \to G$, $A \to BC$, $B \to CFH$, $E \to A$, $F \to EG$ }
{DA} = primary key
$\{DA\}^+ = \{DABCFHEG\}$
D & A = prime attribute
B, C, E, F, G, H = non-prime.

For the FD $A \to BC$, BC is a non-prime attribute and is partially dependent on the primary

key {DA} of relation R.
Hence, relation R is not in 2NF.

3) R(rollno, courseno, name, grade) — A = rollno, B = courseno, C = name, D = grade — R(ABCD)

name, courseno → grade — $CB \to D$
rollno, courseno → grade — $AB \to D$
name → roll no. — $C \to A$
rollno → name — $A \to C$

{courseno, rollno} = {rollno, courseno, name, grade} — $\{BA\}^+ = \{B, A, C, D\}$

{courseno, name} = {rollno, courseno, name, grade} — $\{BC\}^+ = \{BCAD\}$

primary key = {BA} & {BC}
{courseno, rollno} & {courseno, name}
prime attribute = {roll no., courseno, name}
non-prime attribute = grade

Hence, relation R is in 3NF.

Transitive dependency → A functional dependency $X \to Y$ in a relation schema R is a transitive dependency if there exist a set of attributes Z in R i.e. neither a candidate key nor a subset of any key of R and both $X \to Z$ and $Z \to Y$ holds.

A B

→ **BCNF**
A relation schema R is in BCNF if -
whenever a non-trivial FD $X \to Y$ holds in R then, X must be a super key of R.

eg

A	B	C

FD1: $AB \to C$
FD2: $C \to B$

R(ABC) is in 3NF but not in BCNF.

22/12/2020

@ **Multivalued dependency**
It is the dependency where one attribute value is potentially a multivalued fact about another.
For multivalued dependency in a relation -
1) There must be three or more attributes. The attribute must be independent of each other.
2) Multivalued attributes must be independent.

r(R)

ename	contact	Hobbies
A	2148	Playing
	5168	Singing
B	1234	Dancing
	~~5826~~	

→

ename (α)	contact (β)	Hobbies (γ)
A	2148	playing
A	2148	singing
A	5168	playing
A	5168	Singing
B	1234	Dancing

For a functional dependency, $\alpha \rightarrow \beta$, we cannot have two tuples with same α value but different value.

WHEREAS

$\alpha \rightarrow\rightarrow \beta$, is defined as if any legal relation, $r(R)$ $\forall$ pairs of tuples $\{t_1 \& t_2\}$ in r such that $t_1[\alpha] = t_2[\alpha]$,
then $\exists$ tuples $\{t_3 \& t_4\}$ in r such that,
$t_4[\alpha] = t_3[\alpha] = t_2[\alpha] = t_1[\alpha]$
& $t_3[\beta] = t_1[\beta]$, $t_4[\beta] = t_2[\beta]$
& $t_4[\gamma] = t_1[\gamma]$, $t_3[\gamma] = t_2[\gamma]$

@ 4NF

A relation R is in 4NF, if
1. It is in BCNF
2. It does not contain multivalued dependency.

A relation R is in BCNF, if for each multivalued dependency $\alpha \rightarrow\rightarrow \beta$,
1. either a trival dependency
   or
2. α must be a super key of relation R.

emp

ename	contact	hobbies
A	2148	singing
A	5168	dancing
A	2148	dancing
A	5168	singing
B	4198	Reading

emp 1

ename	contact
A	2148
A	5168
B	4198

emp 2

ename	hobbies
A	singing
A	dancing
B	reading

@ Join dependency

Let R be a relation schema and $R_1, R_2, R_3 \ldots R_n$ be the decomposition of relation R, then the relation R is said to satisfy the join dependency
$*(R_1, R_2, R_3 \ldots R_n)$.
If & only if, $\boxed{\Pi_{R_1}(R) \bowtie \Pi_{R_2}(R) \bowtie \Pi_{R_3}(R) \cdots \bowtie \Pi_{R_n}(R) = R}$

@ 5NF

A relation R is in 5NF, if
1. It is in 4NF
2. No non trivial join dependency exists.

A relation R is in 5NF with respect to a set F of Functional multivalued & join dependencies, if for every non-trivial join dependency in $\{F\}^+$, every R_i is a super key of R.

emp

name	skill	Job
A	S_1	J_1
B	S_1	J_2
C	S_2	J_3
D	S_3	J_1
E	S_2	J_2

name	skill
A	S_1
B	S_1
C	S_2
D	S_3
E	S_2

name	job
A	J_1
B	J_2
C	J_3
D	J_1
E	J_2

Skill	Job
S_1	J_1
S_1	J_2
S_2	J_3
S_3	J_1
S_2	J_2

name	Skill	Job
A	S_1	J_1
B	S_1	J_2
C	S_2	J_3
D	S_3	J_1
E	S_2	J_2

Join dependency exists
lossless join decomposition.

23/12/2020

ⓐ Transaction

① Transaction refers to a collection of operation that forms a logical unit of work.

② A database system must ensure proper execution of transaction despite failure. Either the entire transcation executes or none of it does.

For eg Transfer of money from one account to another is a transaction consisting of update to each account. It must manage concurrent execution of transcation in a way that avoids the introduc-tion of inconsistency.

Therefore a transaction is a unit of program execution that occurs & possibly updates various data items.

To ensure integrity of data we require that the database system maintains the following properties of the transactions →

ACID

A = atomicity
C = consistency
I = isolation
D = durability

→ Atomicity

Either all operation of the transaction are reflected properly in the database or none are.

For eg. Consider two accounts Ⓐ & Ⓑ, which have balance of ₹ 1000 & ₹ 2000 respectively. If we start the transcation to transfer ₹ 500 from A to B, then changes in both the account must be reflected as A becomes ₹ 500 & B becomes ₹ 2500.

→ Consistency

Execution of the transcation in isolation preserves the consistency of the database. The consistency requirement is that the sum of A & B remains unchanged by the execution of the transaction.

→ Isolation

Even though multiple transaction may executes concurrently, the system gurantees that for every pair of transaction T_i & T_j, it appears to t_i that either t_j finished execution before T_i started or T_j started execution after T_i finished.

Thus each transaction is unaware of other transaction executing con-currently in the system.

→ Durability

After a transaction completes successfully, the changes it has made to the database persist even if there are system failures.

@ Operations during Transaction

1. read (x)

   It transfer the data items (x) from the database to a local buffer belonging to the transcation that executed the read operation.

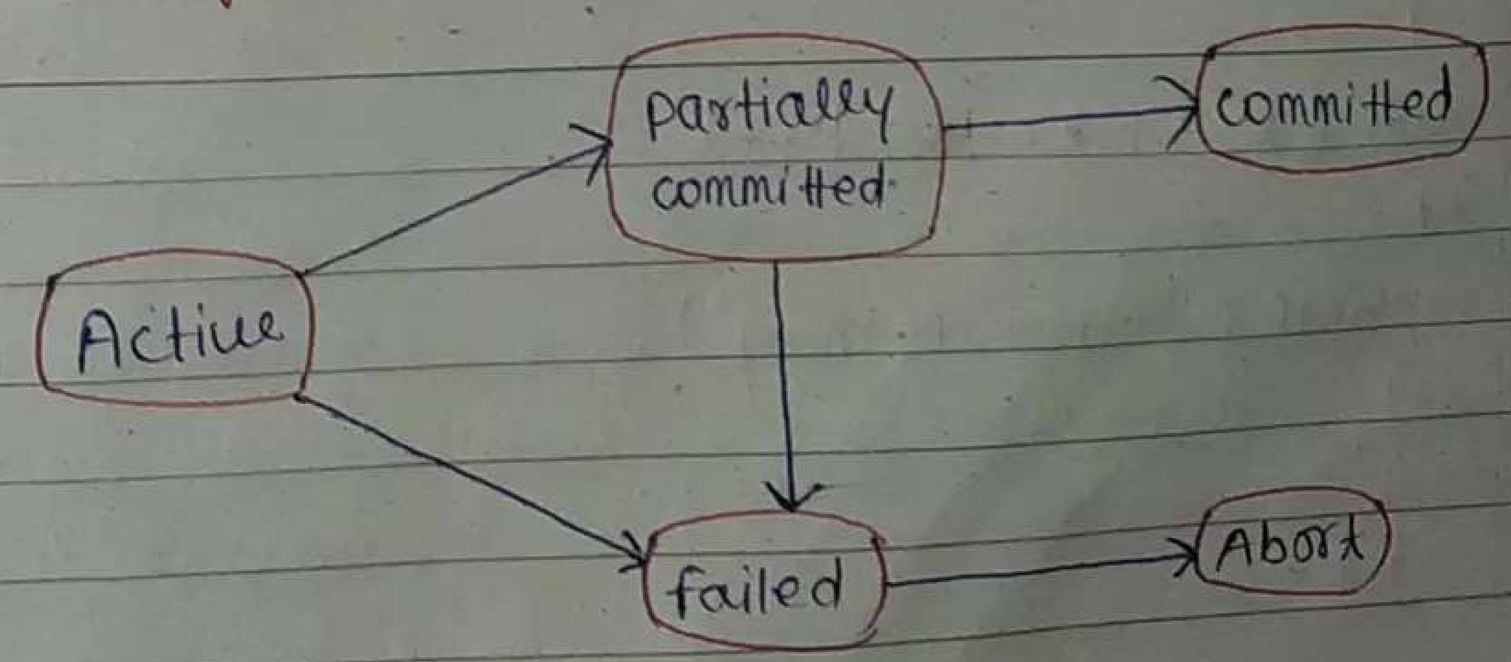
2. write (x)

   It transfers the data item (x) from the local buffer of the transcation that executed the right back to the database.

For eg Consider T_i be a transcation that transfers ₹ 500 from account A to B. This transcation can be defined as -

read (A)
A = A - 500
write (A)
read (B)
B = B + 500
write (B)

@ States of a transcation



1)
```
A()
{
for (i=1; i<=n; i=i*2)
{
printf ("Good Morning");
}
}
```

Time complexity = ?

$O(\log_2 n)$

$i = 1, 2, 4, 8, 16, 32 \cdots n$

$2^0, 2^1, 2^3, 2^4, 2^5 \cdots 2^k$

$2^k = n$

$k = \log_2 n$

$O(\log_2 n)$

2)
```
A()
{
int i, j, k
for (i=1; i<=n; i++)
{
for (j=1; j<=i; j++)
{
for (k=1; k<=100; k++)
{
printf ("Good Morning");
}
}
}}
```


Time complexity = ?

$O(n^2)$

24/12/2020

@ States of transcation continue---

The states of a transaction can be defined as -

1. ACTIVE - It is the initial state. The transaction stays in this state while it is executing.

2. PARTIALLY COMMITTED - The transaction enters into this state after the final statement has been executed.

3. FAILED - The transaction enters into this state after the discovery that normal execution can no longer proceed.

4. ABORTED - In this state, transaction has been rolled back and the database has been restored to its state prior to the start of transaction.

5. COMMITTED - A transaction enters into this state after successful completion.

A transaction start in active state, when it executes the final statement, it enters into partially committed state. At that point, the transaction has completed its execution but it is still possible that it may have to be aborted.

Since the actual o/p may still be temporary residing in main m/y and thus a hardware failure may prevent its successful completion.

A transaction enters into a failed state after the system determines that the transaction has no longer proceed with its normal execution and enters into aborted state. At this point, the system have two options -

① It can restart the transaction. but only if the transaction was aborted as a result of some hardware or software error that was not created through the internal logic of the transaction.

② It can kill the transaction. It usually does so because of some internal logical error such as bad i/p or data not found in the database.

@ Concurrent execution

If the transactions executes concurrently, then we get -

① inproved throughput
② resource utilisation
③ reduced waiting time

Hence, concurrent execution of transaction is reqd.

@ Schedules

The execution sequence of transactions are called schedules. They represent the chronological order in which instructions are executed in the system. Hence schedules for a set of transaction must consist of all the instructions of those transaction & must preserve the order in which the instructions appears in each individual transaction.

Consider a transaction T_1 & T_2 in a schedule S_1 and S_2.

A = 1000
B = 2000

S_1 Serial Schedules		S_2 Concurrent Schedules	
T_1	T_2	T_1	T_2
read(A)		read(A)	
A = A - 50		A = A + 50	
write(A)		write(A)	
B = B + 50			read(A)
write(B)			A = A * 0.5
	read(A)		A = A - t
	t = A * 0.5		write(A)
	A = A - t	read(B)	
	write(A)	B = B + 50	
	read(B)	write(B)	
	B = B + t		read(B)
	write(B)		B = B + t
			write(B)

In serial schedules, the consistency is gurantee but in concurrent schedules, the consistency may or may not be occured.

Consider A = 1000 & B = 2000, after execution of transaction t_1 & t_2 in schedule S_1, we get the final value of A = 475 & B = 2525.

Hence, the total sum (A + B) = 3000, remains unchanged.
In schedule S_2, after execution of transaction we get the final values of A = 475 & B = 2525.

Hence, the total sum $(A+B)$ is preserved.

S_3

T_1	T_2	
read (A)		
$A = A - 50$		
	read (A)	
write (A)	$t = A * 0.5$	Inconsistency
	$A = A - t$	Occurs
	write (A)	
read (B)		
$B = B + 50$		
	read (B)	
	$B = B + t$	
	write (B)	
write (B)		

26/12/2020

(a) Serializability

1. The database system must control concurrent execution of transcation, to ensure that the database state remains consistent.
2. In serial schedules, consistency of database is guaranteed, so concurrent schedules should also give result similar to the serial schedules.
3. Hence, the ability of a concurrent schedules which seems to work ~~be~~ like a serial schedule is called serializability.
4. It can further classified as conflict serializability and view serializability.

(a) Conflict Serializability

1. Let us consider a schedules S in which there are two consecutive instruction I_i & I_j of transaction T_i & T_j respectively $(i \neq j)$.
2. If I_i & I_j refers to different data items, then we can swap I_i & I_j without affecting the result of any instruction in the schedule.
3. However, if I_i & I_j refers to same data items θ, then the order of execution may matter.
4. Since we are dealing with only read & write instruction, there are 4 cases that we need to consider –

(a) If I_i = read (θ) & I_j = read (θ)
The order of I_i & I_j does not matter.
Since the same value θ is read by T_i & T_j regardless of the order.

(b) If I_i = read (θ) & I_j = write (θ)
If I_i comes before I_j, then T_i does not read the value of θ i.e. written by T_j in instruction I_j.
If I_j comes before I_i, then T_i reads the value of θ i.e. written by T_j.
Hence, order of I_i & I_j matters.

(c) If I_i = write (θ) & I_j = read (θ)
~~If~~ The order of I_i & I_j matters because if I_i comes before I_j, then T_j reads the value written by T_i.
If I_j comes before I_i, then T_j cannot read the value written by T_i.

(d) If I_i = write(Q) & I_j = write(Q)

Since both the instructions are write operations, the order of these instruction does not effect either I_i or I_j. However the value obtained by the next read (Q) instruction of schedule S is affected.

If there is no other write(Q) instruction after I_i & I_j in schedule S, then the order of I_i & I_j directly affect the final value of (Q) in the database.

S1

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

S2

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (A)	
write (A)	
	read (A)
	write (A)

28/12/2020

(5) If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instruction. Then, we say that S and S' are conflict equivalent.

(6) Schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

(0) View Serializability:

(1) Consider two schedules S and S' where the same set of transaction participate in both schedule.

(2) The schedule S & S' are said to be view equivalent if the following conditions are met –

29/12/2020

→ For each dataitem 'Q', if transaction Ti reads the initial value of Q in schedule 'S', then transaction Ti must in schedule S', also reads the initial value of Q.

→ For each dataitem 'Q', if transaction Ti executes read (Q) in schedule 'S' and if that value was produce by a write (Q) operation executed by transaction 'Tj', then the read(Q) operation of transaction 'Ti' must in schedule S', also read the value of Q that was produced by the same write (Q) operation of transaction 'Tj'.

→ For each dataitem 'Q', the transaction that perform the final write(Q) operation in schedule 'S', must perform the final write(Q) operation in schedule 'S''.

(3) Condition (i) & (ii) ensure that the transaction reads the same values ~~in the~~ in both schedules and therefore perform the same computation.

condition (iii) coupled ~~th~~with condition (i) & (ii) ensures that both schedules result in the same final state.

Hence, a schedule S is view serializable, if it is view equivalent to a serial schedule.

S1		S2		S3	
T_1	T_2	T_1	T_2	T_1	T_2
read(A)		read(A)		read(A)	
write(A)		write(A)		write(A)	
	read(A)	read(B)			read(B)
	write(A)	write(B)			write(B)
read(B)			read(A)	read(B)	
write(B)			write(A)	write(B)	
	read(B)		read(B)		read(A)
	write(B)		write(B)		write(A)

④ S_1 is view equivalent to S_2
BUT
S_1 is not view equivalent to S_3.

⑤ Since, S_2 is a serial schedule, hence S_1 is view serializable.

⑥ Every conflict serializable schedule is also view serializable but there are view serializable schedule that are not conflict serializable.

i) S

T_1	T_2	T_3
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Blind writes

① Transaction T_2 & T_3 in transcation S perform write(Q) operation without having perform a read(Q) operation.

② Writes of this sort are called Blind writes

③ Blind writes may appear in a view serializable schedule but that schedule is not conflict serializable.

ⓐ Testing of Serializability

① Consider a schedule S, we construct a directed graph called a precedence graph from schedule S.

② This graph consist of a pair $G(V, E)$, where
V = set of vertices
E = set of edges

③ The set of vertices consists of all the transactions participating in schedule S.

④ The set of edges consists of all edges $T_i \to T_j$ for which one of 3 condition holds -
ⓐ T_i executes write(Q) before T_j executes read(Q).
ⓑ T_i executes read(Q) before T_j executes write(Q).
ⓒ T_i executes write(Q) before T_j executes write(Q).

S

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

$T_1 \rightarrow T_2$

precedence graph of schedule S.

⑤ If the precedence graph of schedule S has a cycle, then schedule S is not conflict serializable.

⑥ If the graph contain no cycle, then schedule S is conflict serializable.

1)

S1

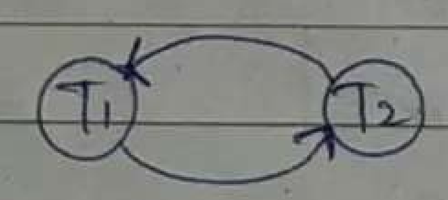
T_1	T_2
read (A)	
A = A - 50	
	read (A)
	t = A * 0.5
	A = A - t
	write (A)
	read (B)
write (A)	
read (B)	
B = B + 50	
write (B)	
	B = B + t
	write (B)

S2

T_1	T_2
read (A)	
A = A - 50	
write (A)	
	read (A)
	t = A * 0.5
	A = A - t
	write (A)
read (B)	
B = B + 50	
write (B)	
	read (B)
	B = B + t
	write (B)

Draw the precedenc graph for both {S1 & S2}

$S_2 \rightarrow$ 

$S_1 \rightarrow$ 

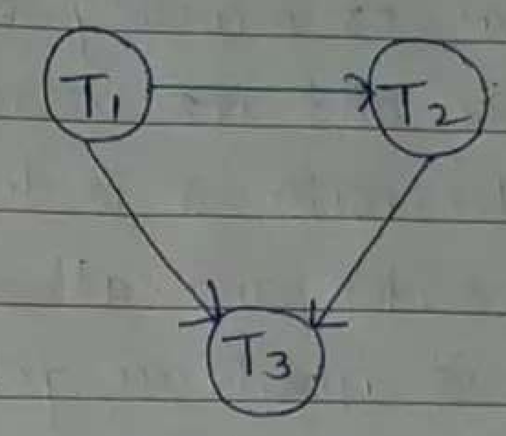
Hence, it is not conflict serializable.

30/12/2020

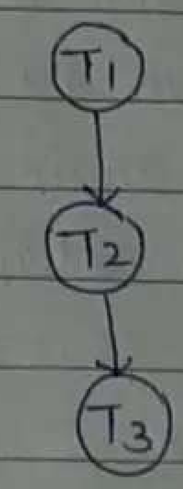
@ Topological sorting

① A serializability order of the transcation can be obtained through topological sorting which determines a linear order.

②



precedence graph

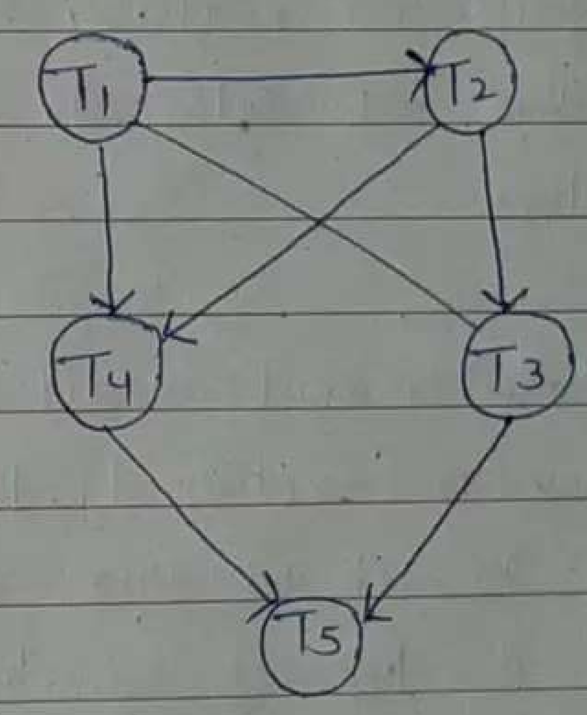


topological sorting

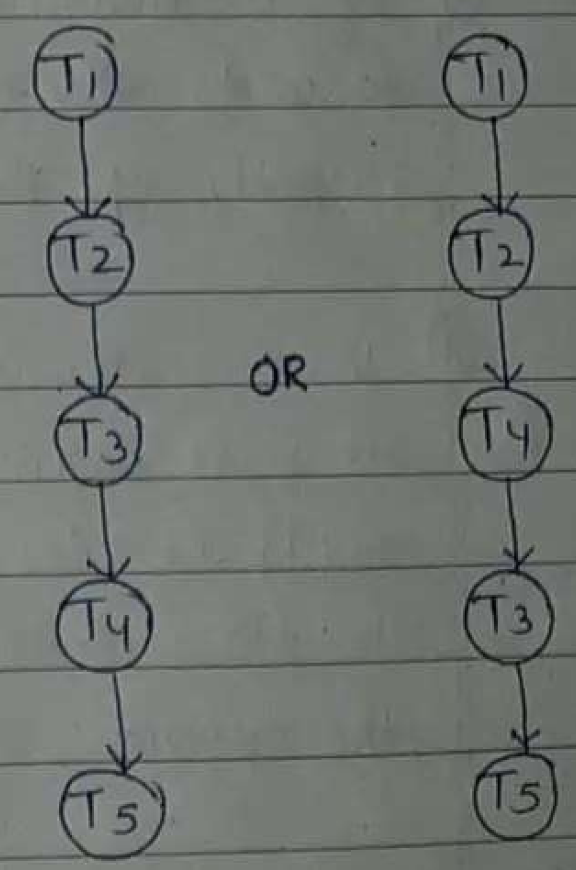
S

T_1	T_2	T_3
–	–	–
–	–	–

③



precedence graph



topological sorting

S

T_1	T_2	T_3	T_4	T_5
–	–	–	–	–
–	–	–	–	–

Topological sorting can be obtained by the precedence graph of schedule S in following steps -

1. Start with the vertex of 0 in degree. i.e. the vertex which has no incoming edge.
2. Remove the selected vertex and its associated edges and again find the vertex with 0 in degree.
3. Again delete the selected vertex and all its associated edges. This process continue until we get a single vertex.
4. Topological sorting order (More than one) can be obtained from a precedence graph of a conflict serializable schedule.

Q Recoverability

1. If a transaction T_i fails for whatever ~~region~~ reason, we need to undo the effect of this transaction to ensure the atomicity property of the transaction.

2. In a system that allows concurrent execution, it is necessary also to ensure that any transaction T_j that is dependent on T_i is also aborted. To achieve this surely, we need to place restrictions on the type of schedule permitted in the system.

Q Recoverable Schedule

1. A recoverable schedule is one where for each pair of transaction T_i & T_j such that T_j reads the data item previously written by T_i.
2. The commit operation of T_i appears before the commit operation of T_j.

S

T_i	T_j
read (A)	
A = A - 50	
write (A)	
	read (A)
COMMIT	A = A + 100
	write (A)
	COMMIT

31/12/2020

Q Cascadeless Schedule

1. Even if a schedule is recoverable to recover correctly from the failure of the transaction T_i, we may have to roll back several transaction in case of cascading schedule.

2. In cascading schedule, transactions read the data written by transaction T_i.

For eg. :
- Transaction T_1 writes the value of A that is read by transaction T_2.
- Similarly T_2 writes a value of A that is read by transaction T_3.
- Suppose at that point, T_1 fails, then T_2 must roll back. Since T_3 is dependent on T_2, hence T_3 also rolls back.
- This phenomenon in which a single transaction failure leads to a series of transaction roll back is k/a Cascading roll back.

S

T1	T2	T3
read (A)		
write (A)		
	read (A)	
	write (A)	
		read (A)
		write (A)

@ Recovery

① A computer system like any other device, is subject to failure from a variety of causes like disk crash, software error, fire etc.

② In any failure information may be lost. Therefore the database system must take action in advance to ensure that the atomicity & durability properties of transaction must holds.

02/01/2021

ASSIGNMENT

1) Consider a schema

emp (id, name, age, sal, city, dno).

dept (dno, dname, dloc)

project (pid, id, pname, dno)

(a) Find the name of employees whose age is greater than avg. age of employees.

(b) Find 2^{nd} highest salary of employees.

(c) Find the name of employees working on project XYZ.

(d) Find the name of all dept located at Yamuna block.

(e) Find the name of employees whose salary is greater than all employees belonging to city 'GKP'.

(f) Find the avg. sal of each dept.

(g) Find the name of employees whose sal lies b/w 40000 to 70000.

(h) Find the name of dept. involved in project XYZ.

(i) Find the total no. of employees of CS dept.

(j) Find the name of employees whose name start with P.

a) select name from emp
where age > (select avg (age) from emp); ✓

b) select sal from emp
order by sal desc ✓
limit 1,1 ;

c) ① select emp. name from emp inner join project
~~where pname = 'XYZ'~~ on emp.id = project.id
where project.pname = 'XYZ';

~~π_{name} (σ pname = 'XYZ' (emp))~~ ② select name from emp
where id IN (select id from project where pname = 'XYZ');

d) Select dname from dept ✓
where dloc = 'Yamuna Block';

e) select name from emp ✓
where sal > All (select sal from emp where city = 'GKP');

f) select avg(sal) from emp ✓
group by dno ;

g) ① Select name from emp ✓
where sal >= 40000 AND sal <= 70000;

② Select name from emp
where sal between 40000 AND 70000;

h) ① select dname from dept inner join project
on dept.dno = project.dno
where project.dname = 'XYZ';

② select dname from dept
where dno IN (select dno from project where
pname = 'XYZ');

i) select count(*) from emp where dno = (select dno
from dept where
dname = 'CSE');

j) select ~~*~~ name from emp
where name like 'P%';

8/10

@ Find the name of employee whose dno is not allocated yet

select name from emp
where dno is NULL;

emp

id	name	dno
e1	X	d1
e2	Y	-
e3	Z	-
e4	P	d2

@ Find the name of employee whose salary is greater than atleast 1 employee belonging to city 'GKP'.

select name from emp
where sal > Any (select sal from emp where city = 'GKP');

04/01/2021

@ Recovery continue ----

Failure Classification

~~There are~~

① • TRANSACTION FAILURE -

There are two types of error that may cause a transaction to fail.

• LOGICAL ERROR -

The transaction can no longer continue with its normal execut-ion because of some internal conditions such as bad input, data not found etc.

• SYSTEM ERROR -

The system has entered into an undesirable state (deadlock) as a result of which a transaction cannot continue with its normal execution.

② • SYSTEM CRASH

There is a hardware problem or a bug in a database software or in the operating system, that causes the loss of content of volatile storage and led a transaction towards failure.

③ • DISK CRASH

A disk block looses its content as a result of either a head crash or failure during a data transfer operation.

Recovery
- log based
  - Deferred
  - Immediate
- shadow paging
- check points

ⓐ Recovery Method

① LOG BASED RECOVERY

The log is a sequence of log records, recording all the update activities in the database.
There are several types of log records. An update log records describe a single database write. It has followings fields –

- **TRANSACTION IDENTIFIER**
  It is a unique identifier of the transaction that perform the write operation.

- **DATAITEM IDENTIFIER**
  It is the unique identifier of the dataitem written.

- **OLD VALUE**
  It is the value of dataitem prior to the write operation.

- **NEW VALUE**
  It is the value of dataitem after the write operation.

Syntax : $\langle T_i, X_i, V_i, V_j \rangle$

$A = 1000$
$B = 2000$

S

T_1	T_2	
read (A)		
A = A − 50		
write (A)		$\langle T_1, A, 1000, 950 \rangle$
read (B)		
B = B + 50		
write (B)		$\langle T_1, B, 2000, 2050 \rangle$
	read (A)	
	A = A + 200	
	write (A)	$\langle T_2, A, 950, 1150 \rangle$
	read (B)	
	B = B + 500	
	write (B)	$\langle T_2, B, 2050, 2550 \rangle$

05/01/2020

- **DEFFERED DATABASE MODIFICATION**

① The deffered modification techniques ensures transaction atomicity by recording all database modification in the log. BUT, differing the execution of all write operations of a transaction until the transaction partially commits. It is assumed that transactions are executed serially.

② The execution of transaction T_i proceeds as follows –
Before transaction T_i starts its execution, a record T_i start {< T_i start >} is written into the log. A write (x) operation by T_i results in the writing of a new record to the log. Finally when T_i partially commits a record T_i commit {< T_i record >} is written to the log.

S

Log	T_1	T_2
< T1, start >		
< T1, A, 1000, 900 >	read (A)	
< T1, B, 2000, 2100 >	A = A - 100	
< T1 commit >	write (A)	
	read (B)	
< T2 start >	B = B + 100	
< T2, A, 900, 1100 >	write (B)	
< T2 commit >		read (A)
		A = A + 200
		write (A)

A = 1000
B = 2000

		Log
T_1	read (A)	< T1 start >
	A = A - 50	
	write (A)	< T1, A, 950 >
	read (B)	
	B = B + 50	
	write (B)	< T1, B, 2050 > < T1, commit >
T_2	read (A)	< T2 start >
	A = A + 200	
	write (A)	< T2, A, 1150 >
		< T2 commit >

① Hence, in deffered database modification, deffered writes are perform after successful completion of a transaction.

② In this technique, only 3 fields are reqd. to create a log.

③ The transaction identifier, dataitem identifier and the new value of dataitem are reqd. in deferred modification.

④ Thus we can simplify the general update log record structure by ommitting the old value field.

⑤ In deferred modification, REDO (T_i) is used. It sets the value of all dataitem updated by transaction T_i to the new values by re-executing the transaction.

• **IMMEDIATE DATABASE MODIFICATION**

① Immediate database modification techniques ensures transaction atomicity by recording all database modification in the log after each write operation.

② Since the database is updated immediately after each write operation, hence it is k/a immediate database modification techniques.

③ The execution of transaction T_i proceeds as -
< Ti start > is written into the log when a transaction Ti start. After each write, < Ti, A, Vi, Vj > is written to log where Ti = transaction identifier
A = dataitem
Vi = old value (before write)
Vj = new value (after write)

A = 1000
B = 2000

		Log	Database
T_1	read (A)	< T1 start >	A = 1000, B = 2000
	A = A - 50		
	write (A)	< T1, A, 1000, 950 >	A = 950, B = 2000
	read (B)		
	B = B + 50	< T1, B, 2000, 2050 > < T1 commit >	A = 950, B = 2050
	write (B)		
T_2	read (A)	< T2 start >	A = 950, B = 2050
	A = A + 200	< T2, A, 950, 1150 >	A = 1150, B = 2050
	write (A)	< T2, commit >	

1. In this technique, database will modify after each write operation.

2. In case of failure, old value of dataitem is reqd. to roll back to previous state.

3. New value is reqd. to update the database after successful write operation.

4. In this technique, REDO (T_i) & UNDO (T_i) are used.

   UNDO (T_i) = restores the values of all dataitems updated by transaction T_i to the old values.

06|01|2021

@ Shadow Paging

1. An alternative to log based recovery is shadow paging. The shadow paging technique is an improvement on shadow copy techniques.

2. A database is partitioned into some no. of fixed length blocks which are reffered to as pages. Assume that there are n pages numbered 1 to n. These pages do not need to be stored in any particular order on the disk.

3. However there must be a way to find the i^{th} page of the database for any given i. We used paged table for this purpose. The page table has n enteries one for each database page. Each entry contain a pointer to a page on disk. The key idea behind the shadow paging technique is to maintain two page table during the life of transaction. These page table are current page table or shadow page table.

4. When the transaction starts, both page tables are identical. The shadow page table is never change over the duration of a transaction. The current page table may be changed when a transaction perform write operation. All input or output operations use the current page table to locate database page table on disk.

5. Suppose that transaction T_j performs a write(X) operation and that X resides on 1 i^{th} page. The system execute the write operation as follows -

→ If the i^{th} page is not already in main memory, then the system issues input (X).

→ If this is the write first perform on the i^{th} page by this transa-ction, then the system modify current page table as follows -

→ If find an unused page on disk, usually the database system has access to a list of unused pages.

→ It deletes the page found in step (1) from the list of free page frames. It copies the content of i^{th} page to the page found in step (1).

→ It modifies the current page table, so that the i^{th} entry points to the page found in step (1).

⑥ It assigns the value of x_j to X in the buffer page.

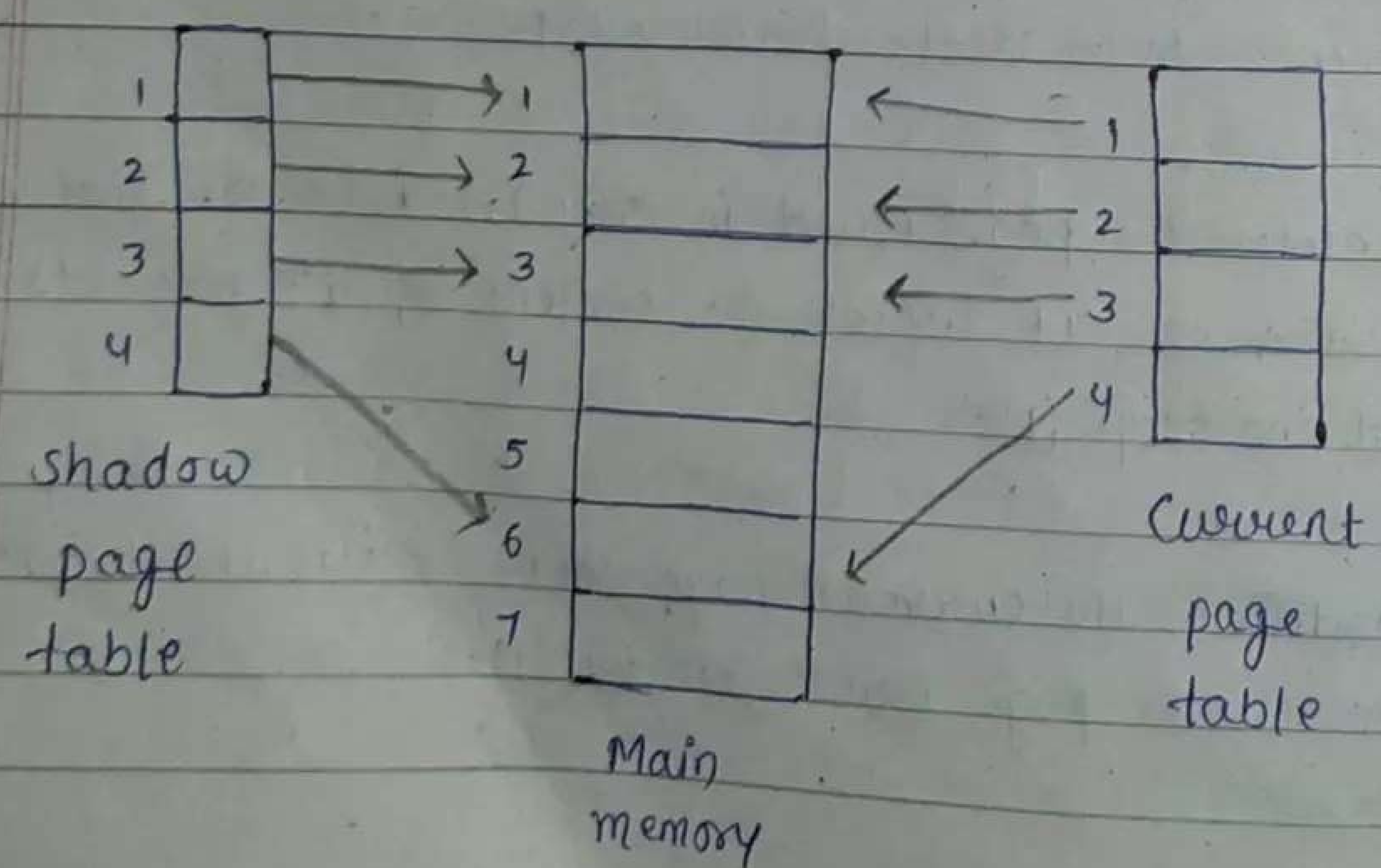
→ The shadow page approach for recovery is to store the shadow page table in non-volatile storage. So, that the state of database prior to the execution of transaction recover in case of system prior or system abort.

⑦ When the transaction committed, then the system writes the current page table to non-volatile storage. The current page table now become new shadow page table and the next transaction is allowed to begin execution.

@ Advantage of Shadow paging
① No log record is to be maintained.
② Recovery from crash disk is significantly faster.

@ Disadvantage of Shadow paging
① Commit over head
② Data fragmentation
③ Garbage collection

07/01/2021



@ Check points
① We used check points to reduce the no. of log records that the system must scan when it recovers from a crash.

② Since we assume no concurrency, it was necessary to consider only the following transactions during recovry –

→ Those transaction that started after the most recent check point.
→ The one transaction if any that was active at the time of most recent check point.

③ The situation is more complex when transaction can execute concurrently. Since, several transaction may have been active at the time of most recent check point.

④ In a concurrent transaction processing system, we require that the check point log record be of the form
< checkpoint L > , where L = list of transactions active at the time of checkpoint.

⑤ Again we assume that transactions do not perform updates either on the buffer blocks or on the log while the check point is in progress.

⑥ When the system recovers from the crash, it constructs two list.
→ The UNDO list consists of transaction to be undone.
while.
→ The REDO list consists of transaction to be redone.

Check point → ① COMMIT → REDO
② START → UNDO

7. Initially both list are empty.

8. The system scans the log backward examining each record until it finds the first <checkpoint> record.

9. For each record, found of the form <Ti commit>, it adds Ti to REDO list.

10. For each record of the form <Ti start>, if Ti is not in the REDO list, then add Ti to UNDO list.

11. Once the REDO list and UNDO list have been constructed, the recovery proceeds as follows -

→ The system rescans the log from the most recent record backward and perform an UNDO for each log record that belongs to transaction Ti on the UNDO list.

Log records of transaction on the REDO list are ignored in this phase. The scan stops when <Ti start> have been found for every transcation Ti in UNDO list.

→ The system locates the most recent <checkpoint L> on the log.

→ The system scans the log forward from most recent <check point L> and perform REDO for each log record.

12. After the system has done all transaction in UNDO list, it redoes the transaction in the REDO list.

13. In this manner, the recovery process may be completed.

08/01/2021

UNIT 5

@ Concurrency control

When several transaction executes concurrently in the database, then isolation property may not be preserved. To ensure the control interaction among concurrent transaction concurrency control technique are used.

@ Lo~~g~~ck based protocals

① One way to ensure serializability is to require that data items must be accessed in mutually exclusive manner. i.e. while one
② transaction is accessing a data item, no other transaction can modify that data item.

③ The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a look on that data item.

@ locks

There are various modes in which a data item may be locked.

① Shared Locks

If a transaction 'Ti' has obtained a share mode lock on data item Q, then transaction 'Ti' can read data item but can not write. It is represented by 'S' and data item 'Q' may be locked in shared mode by using instruction lock - S(Q)

② Exclusive locks -

If a transaction 'Ti' has obtained an exclusive mode lock on data item 'Q', then transaction 'Ti' can read as well as write data item Q. It is represented by X and a transaction Ti can locked a data item Q in exclusive mode by executing the instruction lock - X(Q)

(52)

@ NOTE

Both shared and exclusive mode lock on data item Q can be released by executing the instruction unlock (Q).

–	S	X
S	Yes	No
X	No	No

compatibility Matrix.

@ Shared Mode lock

① Shared mode lock is compatible with share mode but not with exclusive mode for some data item Q. That means at any time several shared mode locks can be held simultaneously on a particular dataitem but a subsequent exclusive mode locks request has to wait until the currently hold shared mode or exclusive mode locks are released.

S_1

T_1	T_2
Lock - S(A)	
read (A)	Lock - S(A)
display (A)	read (A)
unlock (A)	lock - S(B)
	read (B)
	display (A+B)
	unlock (A)
	unlock (B)

S_2

T_1	T_2
Lock - X(A)	
read (A)	Lock - X(B)
A = A - 50	B = B + 100
write (A)	write (B)
unlock (A)	unlock (B)
	lock - X(A)
lock - S(B)	A = A + 200
read (B)	write (A)
display (B)	unlock (A)
unlock (B)	

(a) Two phase locking protocal

One protocol that ensures serializability is two phase locking protocal. This protocal requires that each transaction issue lock and unlock request in two phases -

① Growing Phase

A transaction may obtained locks but may not release any locks.

② Shrinking Phase

A transaction may release locks but may not obtained new locks.

eg.
```
Lock-X(A)   } growing
Lock-X(B)   }  phase
read(A)
A = A-50
write(A)
read(B)
B = B+50
write(B)
unlock(A)   } shrinking
unlock(B)   }  phase
```

09|01|2021

ASSIGNMENT

① Write the relational algebra query

emp (id, name, age, city, dno, salary)

dept (deptno, dname, dloc)

project (pid, empid, pname, dno)

a) Find the name of employees belongs to city Lucknow.

b) Find the name of department working on the project 'xyz'.

c) Find the name of employees of CSE department who are working on the project 'XYZ'.

d) Find the name of employees who salary lies b/w 40000 to 50000.

e) Find the avg. salary of employees.

f) Find the total no. of employees of CSE department.

② Consider a relation R (ABCD), having FD set F = $\{A \rightarrow B, C \rightarrow AD, D \rightarrow C\}$

Find the candidates keys of relation R.

③ Consider a relation R (ABCD), having FD set F_1 = $\{A \rightarrow B, C \rightarrow AD, D \rightarrow C\}$ and $F_2 = \{A \rightarrow BD, C \rightarrow A, D \rightarrow C\}$

Find F_1 & F_2 are equivalent or not.

④ Consider a relation R (ABCD), having FD set F = $\{A \rightarrow BD, CD \rightarrow E, E \rightarrow AC, B \rightarrow D\}$

Find the minimal cover FD set.

⑤ Consider a relation R (ABCD) having FD set F = $\{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$

It is decomposed into R_1, R_2, R_3

R (ABCD)

R_1(AB) R_2(BC) R_3(CD)

Find the decomposition is lossless or lossy.

⑥ Consider a relation R (ABCDEF) having FD set F = $\{AB \rightarrow D, B \rightarrow E, E \rightarrow F, C \rightarrow E\}$

Find the highest normal form which relation R follows.

@ Two phase locking protocol continue ----

① Two phase locking protocol ensures conflict serializability.

② The point in the schedule where the transaction has found obtained its final lock is known as the lock point of the final transaction.

③ Two phase locking does not ensures freedom from deadlock.

S

T_1	T_2
lock-x(B)	
read(B)	
B = B-50	lock-s(A)
write(B)	read(A)
	display(A)
	lock-s(B)
lock(A)	

$T_1 \rightarrow B$

$T_2 \rightarrow A$

Deadlock

④ Cascading rollback may occur under two phase locking. Cascading rollbacks can be avoided by a modification of two phase locking called strict two phase locking. protocol.

- **Strict two phase locking**

① This protocal is not only a two phase locking i.e. locks are allocated and released in two phases, growing phase and shrinking phase respectively., BUT also that all exclusive mode locks taken by a transaction held until that transaction commits.

② This protocol ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits, preventing any other transaction from reading the data.

- **Regerous two phase locking**

① In this protocal, all the locks either shared mode or exclusive mode will be held until the transaction commits.

ASSIGNMENT SOLUTION

① a) $\pi_{name}(\sigma_{city = 'Lucknow'}(emp))$

b) $\pi_{dname}(\sigma_{pname = xyz}(dept \times project))$

c) $\pi_{name}(emp \times (\sigma_{pname = xyz}\ project) \times (\sigma_{dname = cse}\ dept))$

d) $\pi_{name}(\sigma_{salary \geq 40000\ AND\ salary \leq 50000}(emp))$

e) $_{id}\pi_{avg(salary)}[emp]$

f) $\pi_{count(id)}(\sigma_{dname = CSE}(emp \times dept))$

(2) R(ABCD)

F { A → B, C → AD, D → C }

closure of $\{A\}^+ = \{AB\}$

closure of $\{B\}^+ = \{B\}$

closure of $\{C\}^+ = \{CADB\}$

closure of $\{D\}^+ = \{DCAB\}$

Candidate keys → {D}, {C}

No. of possible candidate keys = 02

(3) R(ABCD)

F_1 { A → B, C → AD, D → C }

F_2 { A → BD, C → A, D → C }

F_1 covers F_2

A → BD	C → A	D → C
$\{A\}^+ = \{ABDC\}$	$\{C\}^+ = \{CABD\}$	$\{D\}^+ = \{DCAB\}$

F_2 covers F_1

A → B	C → AD	D → C
$\{A\}^+ = \{ABDC\}$	$\{C\}^+ = \{CADB\}$	$\{D\}^+ = \{DCAB\}$

F_1 covers F_2 = ✓

F_2 covers F_1 = ✓

Hence, they are equivalent.

(4) R(ABCDE)

F { A → BD, CD → E, E → AC, B → D }

Minimal cover FD set?

a) A → B

A → D

CD → E

E → A

E → C

B → D

b) A → B

closure of $\{A\}^+ = \{ABD\}$

after removing the above closure,

closure of $\{A\}^+ = \{AD\}$

A → D (Extra)

closure of $\{A\}^+ = \{ABD\}$

after removing the above closure,

closure of $\{A\}^+ = \{ABD\}$

CD → E

closure of $\{CD\}^+ = \{CDEAB\}$

after removing the above closure,

closure of $\{CD\}^+ = \{CD\}$

E → A

closure of $\{E\}^+ = \{EACBD\}$

after removing the above closure,

closure of $\{E\}^+ = \{EC\}$

E → C

closure of $\{E\}^+ = \{EACBD\}$

after removing above closure,

closure of $\{E\}^+ = \{EABD\}$

B → D

closure of $\{B\}^+ = \{BD\}$

after removing the above closure,

closure of $\{B\}^+ = \{B\}$

$$\left.\begin{array}{l} A \to B \\ CD \to E \\ E \to AC \\ B \to D \end{array}\right\} \text{Minimal functional dependency set}$$

(5) R(ABCD)

F { A → B, B → C, C → D, D → A }

R(ABCD)

R_1(AB) R_2(BC) R_3(CD)

$R_1(AB)$	$R_2(BC)$
$A \to B$	$A \to C$
$B \to C$	$C \to D$
$C \to D$	$D \to A$
$D \to A$	$A \to B$
$\{B\}^+ = \{BCDA\}$	$\{B\}^+ = \{BCDA\}$

$R_2(BC)$	$R_3(CD)$
$B \to C$	$C \to D$
$C \to D$	$D \to A$
$D \to A$	$A \to B$
$A \to B$	$B \to C$
$\{C\}^+ = \{CDAB\}$	$\{C\}^+ = \{CDAB\}$

Here, $R_1 \cap R_2 = \phi$

$R_2 \cap R_2 = \phi$

$\{R_1 \cap R_2\} \to R_1$ or $\{R_1 \cap R_2\} \to R_2$

$\{R_2 \cap R_3\} \to R_2$ or $\{R_2 \cap R_3\} \to R_3$

So, loseless is gauranted. Hence proued.

⑥ R (ABCDEF)

$F\{AB \to D, B \to E, E \to F, C \to E\}$

Highest normal form ?

$\{ABC\}^+ = \{ABCDEF\}$ = candidate key of relation R.

A, B, C = prime attributes

D, E, F = non-prime attributes

Here, no no-prime attribute is partially dependent on candidate key of relation R.

2NF confirm (✓)

56

$\{AB\}^+ = \{ABDEF\}$ $AB \to D$ (neither AB is super key nor D is prime attributes)

$\{B\}^+ = \{BEF\}$ $B \to E$ (neither B is super key nor E is prime attributes)

$\{E\}^+ = \{EF\}$ $E \to F$ (neither E is super key nor F is prime attributes)

$\{C\}^+ = \{CEF\}$ $C \to E$ (neither C is super key nor E is prime attributes)

Hence, the relation is not in 3rd normal form.

So, the highest normal form of relation will be 2nd Normal form.

Graph Based protocols

1. The two phase locking protocol is both necessary & sufficient for ensuring serializability. BUT if we wish to develop the protocal that are not two phase, then we need aditional information on how each transaction will access the database.

2. To acquire such prior knowledge, we impose a partial ordering on set $D = \{d_1, d_2, d_3 \cdots d_n\}$ of all data items.

3. If $d_i \to d_j$, then any transaction accessing both d_i & d_j must access d_i before accessing d_j.

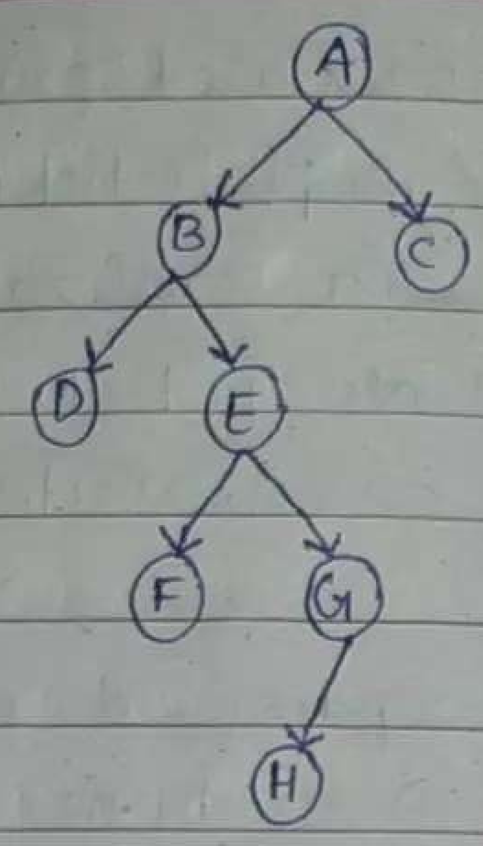
4. The partial ordering emplies that the set $d = \{d_1, d_2, d_3 \cdots d_n\}$ may now be viewed as a directed acyclic graph called database graph.

5. Here we use only those graph that are rooted trees. Hence it is called tree protocol.

Tree protocol

1. In tree protocol, only lock instructions allow is lock-X.
2. Each transaction T_i can lock a dataitem atmost once and must observe the following rules.
   - → The first lock by transaction T_i may be on any dataitem.
   - → Subsequently a dataitem Q can be locked by transaction T_i only if the parent of Q is currently locked by T_i.
   - → Dataitems may be unlock at anytime.
   - → A dataitem that has been locked and unlocked by T_i, cannot subsequently be relocked by T_i.

T_1	T_2
lock-X(E)	
lock-X(G)	
unlock-(E)	
unlock-(G)	
	lock-X(E)
	lock-X(F)
	unlock ~~X~~(E)
	unlock (F)



12/01/2021

3. Tree protocol ensures conflict serializability and freedom from deadlock.
4. It does not ensure recoverability and cascadelessness.
5. To ensure recoverability and cascadelessness, the protocal can be modify to not permit release of locks until the end of the transaction.

DRAWBACK

1. A transaction may have to lock dataitems that it does not access. It is called locking overhead.

13/01/2021

@ Time - Stamp based protocols

1. Time Stamp

- With each transaction T_i in the system we associate a unique fixed time stamp denoted by ~~Ts~~ TS(T_i).
- This time stamp is assigned by the database system before the transaction T_i starts execution.
- If a transaction T_i has been assigned time stamp TS(T_i) and a new transaction T_j enters into the system, then TS(T_i) < TS(T_j).

- There are two methods for implementing this scheme -
  - @ Use the value of **system clock** as the time stamp when the transaction enters into the system.
  - (b) Use a **logical counter** that is incremented a new time stamp has been assigned.
- If TS(T_i) < TS(T_j), then the system must ensures that the produce schedule is equivalent to a serial schedule in which transaction T_i appears before transaction T_j.
- To implement this, each dataitem Q is associated with two time stamp values. -
  - @ **W timestamp (Q)**
    It denotes the largest timestamp of any transaction that executed write(Q) successfully.
  - (b) **R timestamp (Q)**
    It denotes the largest timestamp of any transaction that executed read(Q) successfully.
- These time stamp are updated whenever a new read(Q) or write(Q) instruction is executed.

2. **Time stamp ordering protocol**

- It ensures that any conflicting read & write operations are executed in time stamp order.
- This protocol operates as follows -
  - @ Suppose that transaction T_i issues read(Q) -

* if W timestamp (Q) > TS(Ti),
then Ti needs to read the value of Q that was already overwritten. Hence the read operation is rejected and Ti is rolled back.

* if W timestamp(Q) ≤ TS(Ti),
then read operation is executed and R timestamp (Q) = max { R timestamp (Q), TS(Ti) }

(b) Suppose that transaction Ti issues write(Q),
* if TS(Ti) < R timestamp (Q),
then the value of Q that Ti is producing was needed previously & the system assumed that the value was never produced. Hence, the system rejects the write operation and transaction Ti rolles back.

* if TS(Ti) < W timestamp (Q),
then Ti is attempting to write an absolute value of Q. Hence the system rejects the write operation and Ti rolles back.

(c) * Otherwise transaction executes the write (Q) operation and set W timestamp (Q) = TS(Ti)

15/01/2021

TS(T1) 100 TS(T2) 200 TS(T3) 300

1)

T1	T2	T3
R(A) ✓		
	R(B) ✓	
W(C) ✓		
		R(B) ✓
R(C) ✓		
	W(B)	
		W(A) ✓

Find the no. of ~~instructions~~ transaction rolled back.

	A	B	C
RT	100	300	100
WT	300	0	100

(1) R(A) T(1)
WT(A) > TS(T1)
0 > 100 ✗

max { 0, 100 } = 100

(2) R(B) T(2)
WT(B) > TS(T2)
0 > 200 ✗

max { 0, 200 } = 200

(3) W(C) T(1)
RT(C) > TS(T1)
0 > 100 ✗

WT(C) > TS(T1)
0 > 100 ✗

WT(C) = TS(T1)
100

(4) R(B) T(3)
WT(B) > TS(T3)
0 > 300 ✗

max {200, 300} = 300

(5) R(C) T(1)
WT(C) > TS(T1)
100 > 100 ✗

max { 0, 100 } = 100

(6) W(B) T(2)
RT(B) > TS(T1)
300 > 200 ✓
rolles back

~~WT(B) > TS(T1)~~
~~100 > 100 ✗~~

(7) W(A) T(3)
RT(A) > TS(T3)
100 > 300 ✗
WT(A) > TS(T3)
0 > 300 ✗

$WT(A) = TS(T_3)$
$= 300$

16/01/2020

@ Thomas Write Rule

① Consider a schedule S where timestamp $TS(T_1) = 100$ & $TS(T_2) = 200$

S (T_1: 100, T_2: 200)

T_1	T_2
read(Q)	
	write(Q)
write(Q)	

② write(Q) operation of transcation T_1 is rejected & T_1 must be rolled back.

③ Although, ~~roll~~ rollback of T_1 is reqd. by timestamp odering protocol but it is unnecessary.

④ This observation leads to a modified version of timestamp odering protocol in which absolute write operation can be ignored under certain circumstances.

⑤ The protocol rules for read operation remain unchanged, the protocol rules for write operation are slightly different from timestamp odering protocol.

⑥ This modification is k/a Thomas write rule.

→ Suppose transaction T_i issues write(Q) -

~~@ If W timestamp (Q) > TS(T_i)~~

@ If R timestamp(Q) > TS(T_i),
then the value of (Q) that T_i is producing was previously needed & it has been assumed that the value would never be produced. Hence system rejects write operation

and transaction T_i rolles back.

@ If W timestamp (Q) > TS(T_i),
then transaction T_i is attempting to write an absolute value of Q. Hence the write operation can be ignored.

@ Otherwise system executes the write operation & set W-timestamp (Q) = TS(T_i)

@ Validation Based protocol

① Assume that each transaction T_i executes in 2 or 3 different phases in its lifetime depending on whether it is ~~only~~ a read only ~~phase~~ or an update transaction.
The phases are in order -

① READ PHASE -
- During this phase, the system executes the transaction T_i. It reads the values of various dataitem and stores them to a local buffer of transaction T_i.
- It performs all write operations on temporarily local variable without updating the actual database.

② VALIDATION PHASE
- Transaction T_i performs a validation test to determine whether it can copy to the database • the temprarily local variables that hold the result of write operation without causing a violation of serializability.

③ WRITE PHASE

- If transaction T_i succeeds in validation, then the system apply the actual updates to the database, otherwise the system rolles back T_i

→ To perform validation test, we need to know when the various phases of transaction T_i tooks place.

→ Therefore, 3 different timestamp are associated with transaction T_i.

① $Start(T_i)$ → when transaction T_i started

② $Validate(T_i)$ → when transaction T_i finish read phase and started validation phase

③ $Finish(T_i)$ → when transaction T_i finish its write phase.

→ We determine the serializability order by timestamp ordering techniques using the value of timestamp $validate(T_i)$.

→ For validation test, it is assumed that $TS(T_i) < TS(T_j)$ and one of the following two condition must hold -

① $Finish(T_i) < start(T_j)$

Since, T_i completes its execution before T_j started, the serializability order is maintained.

② The set of dataitems written by T_i does not intersect with the set of dataitems read by T_j AND T_i completes its write phase before T_j starts its validation phase.

i.e. $start(T_j) < Finish(T_i) < Validate(T_j)$

This condition ensures that the writes of T_i & T_j do not over-lap, since the write of (T_i) do not affect the read of (T_j) and since (T_j) cannot affect the read of (T_i). AND the serializability order is maintained.

18/01/2020

ⓐ Multiple Granularity protocol.

① In normal concurrency control techniques, we have used individual data item as the unit on which synchronization is performed.

② There are circumstances where it would be advantageous to group several dataitems & treat them as individual unit.

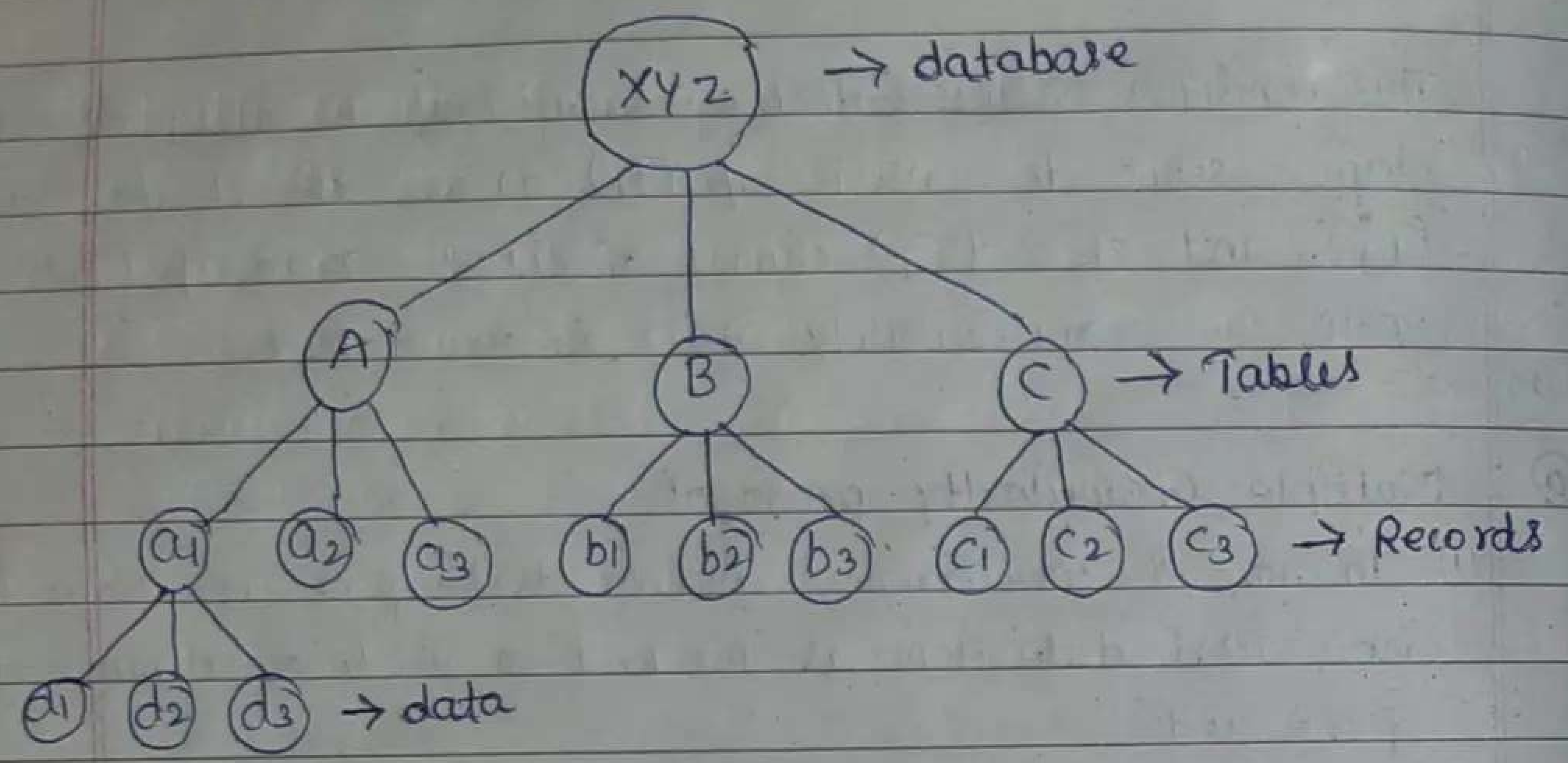
③ For eg If a transaction T_i needs to access the entire database & a locking protocol is used. Then T_i must lock each dataitem in the database.

④ Executing all the locks is time consuming, therefore it would be better if T_i could issue a single lock request to lock the entire database.

⑤ On the other hand, if transaction T_j needs to access only the few dataitems, then it should not require to lock the entire database since the concurrency is locked.

⑥ Therefore a mechanism is needed to allow the system to define the multiple level of Granularity.

⑦ We can make one by allowing dataitem to be of various sizes & defining the heirarchy of data granularities where the small granularity are nested within larger once. Such a heirarchy can be represented graphically with a tree.



8. When a transaction lock a node in either shared or exclusive mode, it also has emplicitly locked all the descendent of that node in the same lock mode.

9. Intention lock are introduced if a node is locked in intention-shared mode (**IS mode**), explicit locking is being done at a lower level of the tree but with only shared mode locks.

10. If a node is locked in intention exclusive mode (**IX mode**), explicit locking is being done at a lower level with exclusive modes or shared mode locks.

11. Finally a node is locked shared intention exclusive mode (**SIX mode**), then the sub tree rooted by that node is locked explicitly in shared mode and that explicit locking is being done at a lower level with exclusive mode locks.

20/01/2021

@ Compatibility Table

	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

- Multiple Granularity protocol ensure serializability.
- Each transaction Ti can lock a node Q by following rules –
  1. It must observe the lock compatibility function.
  2. It must lock the root of the tree first & can lock it in any mode.
  3. It can lock a node Q in exclusive (X), shared intention exclusive (SIX), intention exclusive (IX) mode, only if it currently have the parent of Q locked in either IX or SIX mode.
  4. It can lock a node Q in S or IS mode only if it currently have the parent of Q locked in either IX or IS mode.
  5. It can locked a node if it has not previously unlocked any node.
  6. It can unlock a node Q only if it currently has none of the children of Q lock.

- Multiple Granularity protocol requires that locks can be acquired in top down order & when locks are released, then it must be released in Bottom up order.

22/01/2020

(a) Multi-version protocol

① Multi-version timestamp protocol

- In multi-version concurrency control techniques, each write(Q) operation creates a new version of (Q).
- When a transaction issues a read(Q) operation, the concurrency control manager selects one of the versions of (Q) to be read.
- The concurrency control scheme must ensure that the version to be read is selected in a manner that ensures serializability.

- Multi-version time stamp ordering involves or works as follows -
  ① with each transaction T_i, a timestamp $TS(T_i)$ is associated.
  ② the database system assigns the timestamp before the transaction (T_i) starts execution.
  ③ with each dataitem (Q), a sequence of versions $\langle Q_1, Q_2, Q_3 \ldots \rangle$ is associated.
  Each version 'Q_k' containing 3 data fields -
    (a) Content is the value of version 'Q_k'.
    (b) W timestamp Q_k is the timestamp of transaction that created version 'Q_k'.
    (c) R timestamp Q_k → largest timestamp of any transaction that successfully read 'Q_k'.

- Transaction T_i creates a new version 'Q_k' of dataitem 'Q' by issuing a write(Q) operation.
- The content field of the version holds the value written by transaction T_i.
- The system initialises W timestamp & R timestamp to $TS(T_i)$.
- It updates the R timestamp Q_k, whenever the transaction T_j reads the content of Q_k and R timestamp$(Q_k) < TS(T_j)$

- Suppose T_i issues a read(Q) or write(Q) operation - Let Q_k denotes the version of (Q) where write timestamp is the largest timestamp.
- If T_i issues read(Q), then the value return is the content of version Q_k.
- If T_i issues write(Q) & $TS(T_i) <$ R timestamp (Q_k), then the system rolles back T_i.
- On the other hand, if $TS(T_i) =$ W timestamp (Q_k), then the system overwrites the content of Q_k, otherwise it creates a new version of 'Q'.

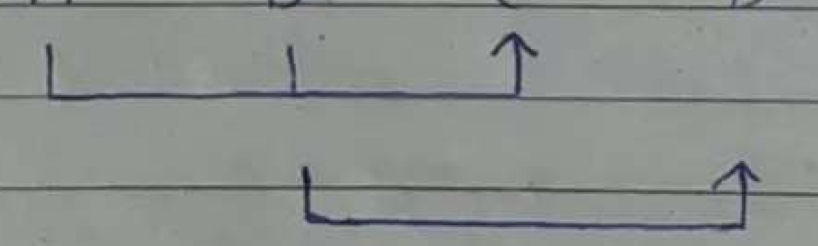
02/02/2021

1. Write the armstrong axioms.
2. Explain dependency preservation.
3. What do you mean by partial functional dependency?
4. What is conflict seralizability?
5. What is blind write?

1. Armstrong axioms –
   - Reflexive → $X \to Y$, $X \to X$, $Y \to Y$ $Y \subseteq X$ → holds.
   - Augumentation → $X \to Y$, $XW \to YW$
   - Transitive → $X \to Y$, $Y \to Z$ → $X \to Z$
   - Pseudo transitivity → $X \to YZ$ & $YZ \to W$ → $XY \to W$
   - Union → $X \to Y$, $X \to Z$ → $X \to YZ$
   - Decomposition → $X \to YZ$ → $X \to Y$ & $X \to Z$

2. Dependency preservation → If R is a relation having F.D 'F' and it can be decomposed into $R_1, R_2, R_3 \dots R_n$ and their F.D set $F_1, F_2, F_3 \dots F_n$, then, if $\{F_1\}^+ \cup \{F_2\}^+ \cup \{F_3\}^+ \cup \{F_4\}^+ \dots \cup \{F_n\}^+ = \{F\}^+$ is known as dependency preservation.

3. A B C D E



$C \to AB$ Here C determines AB
$D \to B$ D determines B

no non-prime attribute is partial dependent on the ~~primary~~ candidate key of the relation.

Here, B is partial dependent on C.
This is known as partial functional dependency.

4. Conflict serializability states that if a schedule S can be transform into serial schedule by swapping of non-conflict operation.

5. When write operation perform without having performed the read operation, this phenomenon is known as blind write.

S1

read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Serial schedule

S2

read (A)	
write (A)	
	read (B)
	write (B)
read (A)	
write (A)	
	read (B)
	write (B)

~~Conflict~~ Concurrent schedule

Conflict serializability occurs in four conditions –
1. read(Q) – read(Q) } non-conflicting.
2. read(Q) – write(Q)
3. write(Q) – read(Q)
4. write(Q) – write(Q)

} conflicting

03/02/2021

ⓐ Deadlock

System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another in the set.

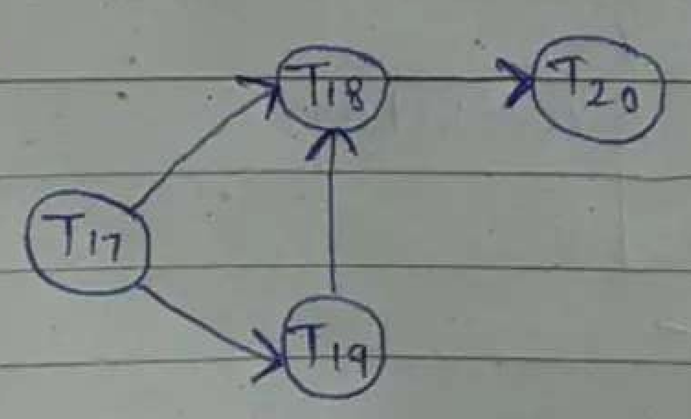
① Deadlock prevention

- Ensures that the system will never enter into a deadlock state.
- Some prevention strategies are -
  - → Require that each transaction locks all its data item before it begins execution.
  - → Impose partial ordering of all data items and require that a transaction can lock data item only in the order specified by the partial order.

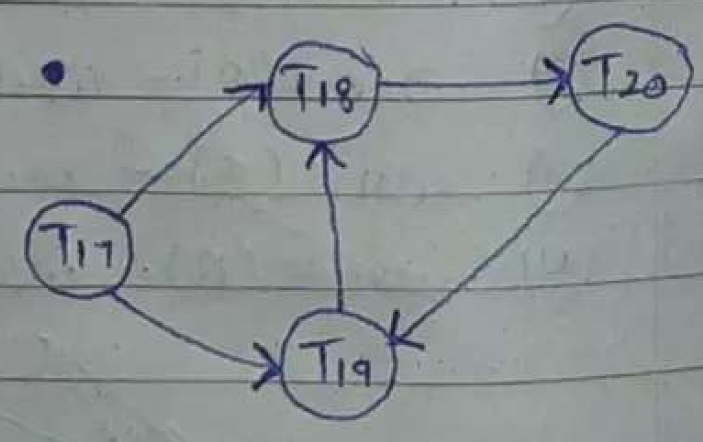
- Following schemes use transaction timestamp for the sake of deadlock prevention alone.

- wait-die scheme — non-preemptive
- wound-wait scheme — preemptive
- timeout based schemes

② Deadlock detection

- 

wait-for graph without a cycle

- 

wait-for graph with a cycle

- Deadlocks can be described as a wait-for graph, which consists of a pair $G = (V, E)$
  - → V = set of vertices
  - → E = set of edges

③ Deadlock recovery

- When deadlock is detected :
  - → Some transaction will have to rolled back to break deadlock. Select that transaction as victim that will incur minimum cost.
  - → Rollback - determine how far to roll back transaction.
    - Total Rollback : Abort the transaction and then restart it.
    - More effective to roll back transaction only as far as necessary to break deadlock.
  - → Starvation happens if same transaction is always choosen as victim. Include the no. of rollbacks in the cost factor to avoid starvation.