DBMS

20/7/23

Text book
1. Introduction DBMS by Kapt
2. by Navathe

Data: Raw materials, variable parts are data
Information = collection of datas meaningful
In a software system
Data → Information → knowladge (Human x ; now → AI)
↓
S/W (software), programming

Way of storing infos/datas
tree, graph, XL → this
all are data models

Anything that stores data is a database.
Everything is database but everything isn't management. i.e everything is not dbms if it is not well organized/systemed.
It should be detect two things -
1) How data should be organised.
2) How to efficiently access the data.

RDBMS (Relational DBMS → the default dbms = RDBMS)

If DBMS is relational then, it is called RDBMS.
It is basically a program that allows us to create, delete, update a relational db. It is a database system that stores & retrieves data in a tabular format organized in the form of rows & cols. It is the smallest subset of dbms.

Data model
1) Heirarchical (tree form)
2) network (graph form)
3) Relation (tabular (table) form)

{1, 2, 3} → set
{<1, 2, 3>} → sequence
{<A:1>, <B:2>} → ordered pair
{<a, b, c>, <d, e, f>} → set of tuples or relation

a	b	c
e	f	g

entity → tuples.
any 2

Ex***

DBMS – A database management system (DBMS) is a software system that is designed to manage & organize data in a structural manner. It allows users to create, modify, & query a database, as well as manage the security and access controls for that database. [query = It is used in DBMS to retrive or modify data stored in the database].

Procedural language – A Procedural language is a type of computer programming language that specifies a series of well-structured steps & procedures a program. It contains a systematic order of statements, functions & commands to complete a computational task or program. Ex- C, Java etc. Here users has to specify "what to do" & also "how to do".

Non-Procedural language – Here, the user has to specify only "what to do" & not "how to do". It is also known as an applicative or functional language. Ex – SQL etc.

Structure Query Language (SQL)

RDBMS → oracle, mysql, MS SQL Server, DB II, MS Access

Language that can implement some logic

- SQL is a non-procedural language.

Procedural & Non Procedural - definition

what to do, how to do → Procedural lang.
what to do, × → non Procedural lang.

Student

Stud.N	Roll	Add	R.No

select * from table;
to Print the full table -

SQL

Advantage - 1) very easy to understand

Disadvantage - 1) we should be very careful as we can't get the proper logic, so it can be lost.
2) No. comp. lang. can understand SQL.

- For every n.p.l. there should be a p.l. that will represent it in executable form.

Relational algebra - It is a procedural lang & it is for SQL.

Disadvantage of writing Programme in C

i) Not platform independent.
ii) ~~Every~~ Lack of exception handeling.
iii) The data types used in that programme range of isn't known to other who can't see the programe.
iv) It doesn't run in all machines (compilers), whether it is 16 bit or 32-bit compilers.

Cl Relational algebra 25/7/23

this are called attributes/fields

Student

Stream	Sem	Roll	Name	address
CSE	5	13	XJZ	Rol=63 ← entry
ECE	3	11	ABC	Rol-42

tuple

collection of/set of

- keys are attributes.
- key is a set of attributes which can uniquely identify tuples.

any 2

- Any superset of superkey is also a superkey
- Any superset is superkey

Keys
- super key
- candidate key
- primary key
- Foreign key

key attributes = stream, sem, roll

Superkey = stream, sem, roll, address
~~subset of~~

candidate key = minimal of the superkey
i) stream, sem, roll (c_1)
ii) Reg. no. (c_2)

- A relation of a table can have multiple candidate keys.

Primary key = Any one of the candidate key should be considered as primary key.

- A table cannot have multiple primary keys
- The primary key should have to be unique, but here there is no ~~example~~ example because there is no such rule.

Here Reg. no is a primary key (the one key that ~~can~~ is enough to find a thing).

Comp	Theory
field	attribute
table	relation/table

[The attribute that functionally determines the functionality of all other attributes, primary key]

Emp

Emp id	Name	Sal	dept ~~Emp~~ id
01	AA	10K	d2
02	BB	12K	d1
03	CC	11K	d1

Non-key attribute (not part of superkey (c-key))

foreign key (FK) (the key that is common in both tables / create a link in two tables)

Dept

Dept id	Name	Dept loc
d1	Sales	Ground
d2	A/C	1st floor

Primary key of Emp = Emp id
Dept = Dept id

Foreign key = If a primary key of one table is used as a non key attribute in another table then it is called a foreign key in that table.

Diff. between 4 types of keys

Super key	Candidate key	Primary key	Foreign key
i) A combination of cols that uniquely identifies any row within a relational database ms (roam) table.	i) A specific type of field in a relational database that can identify each unique record independently of any hierarchy data.	i) The col or cols that contain vals that uniquely identify each row in a table.	i) A col or group of cols in a relational database table that provides a link between data in two or more tables

iii) If we add any attribute with C.K & after that it remains C.K then it's not C.K not it is S.K. iv) Any subset of S.K is also C.K.	ii) minimal of S.K. iii) A table can contain many C.K's. It should be unique & can be null.	ii) Any one of C.K should be considered as P.K. The attribute that functionally determine the functionality of other attributes is called P.K. iii) A table can't contain multiple P.K's. It should be unique but it can't be null.	ii) If a P.K of one table is used as non key attribute in another table then this is known as foreign key. iv) The key that is common in 2 or more tables.

Imp

Relation algebra (RA) → It is a sequence of operators & operand

operand → Relation
→ attribute

operators

→ Π (Projection), σ (selection), $\cup$, $\cap$, '−' (set diff.), $\times$ (cartesian product), $\bowtie$ (natural join), ρ (Rename operator), $\mathcal{G}$ (aggregate as well as ~~group by~~ group by operator)

- $\Pi_{emp\,id,\,name}(emp)$ = give emp id, name for all the employees. (It is an unitary operator)
- $\sigma_{sal>10k}(emp)$ = give all the tuples above Sal 10k (It is also an unitary operator)

[The operators that operates on only one operand]

RA	SQL
Π	select
σ	where

① Find the name of the employee whose Sal. is greater than 10k.

Ans → $\Pi_{name}(\sigma_{sal>10k}(emp))$ [Procedural lang]

Equ SQL → select name from emp where sal > 10000; [Non-Procedural lang]

Saving

A/c	name	deposite

loan

loan no	name	amount	Pd amount

② Find the name of the person (considering this 2 table) who has taken a lone as well as deposited in the bank.

Ans — $\Pi_{name}(Saving.S) \cap \Pi_{name}(loan)$

$A \cap B$
$A - (A - B)$
Has name in both table 'n'

Equ SQL → Select saving.name from saving join loan on saving.name = loan.name;

$\cap$ = AND = *

③ Find the name of the person who has either taken a lone or deposited in the bank.

Ans — $\Pi_{name}(Saving) \cup \Pi_{name}(loan)$

U = OR = +

Has name in either one 2 table, 'U'

$\boxed{\text{eqv SQL}}$ → select name from saving
union select name from loan;

(4) Find the name of the person who has taken a loan but not deposited in the bank.

Ans – $\Pi_{Name}(loan) - \Pi_{name}(savings)$

$\boxed{\text{eqv SQL}}$ → select distinct loan.name from loan where loan.name not in (select name from saving);

27/7/23

CW

emp

	empid	Name	sal	deptid
(1)	01	AA	10K	01
(2)	02	BB	20K	01
(3)	03	CC	15K	02
(4)	04	DD	12K	02
(5)	05	FE	13K	02

(emp.deptid = dept.id)

dept

dept id	dname	dloc
(1) 01	Sales	Ground floor
(2) 02	IT	3rd floor

10 rows will ~~etc~~ be created here, but a row is valid when dept id will be same. ∴ there will be 5 ~~too~~ final rows. after 'X'. & the row will be like (↑)

(*) What is the sal of the person 'BB'?

$\Pi_{sal}\left(\sigma_{Name='BB'}(emp)\right)$

$\boxed{\text{SQL}}$ → select sal from emp where name = 'BB';

Cartesian Product

A ~~(1,2)~~ B → ~~(1,3)~~

a_1, a_2 (datas) b_1, b_2, b_3 (datas)

have to arrange in that formats

A × B
$a_1\ b_1$
$a_1\ b_2$
$a_1\ b_3$
$a_2\ b_1$
$a_2\ b_2$
$a_2\ b_3$

have to arrange in that form

(2,1)

A B → (3,1)

a_1 — b_1, b_2, b_3

a_2 — b_1, b_2, b_3

$A \times B = (6,2)$

→ when we have to make one table from 2 tables then we have to do Cartesian Product

$R_1 = (p, q)$

$R_2 = (m, n)$

$R_1 \times R_2 = (p*m, q+n)$

$(emp \times dept) = 10, 7$

emid	name	sal	deptid	dName	dloc
02	BB	20K	01	Sales	Ground floor

→ valid row, when dept id is same.

$\Pi\left(\sigma_{emp.dept.id = dept.deptid}(emp \times dept)\right)$ →

name	Sal	dname
02	20K	Sales

name, sal, dname ↓

$\boxed{\text{SQL}}$ → select name, sal, dname from emp, dept where emp.deptid = dept.deptid;

dname is here so emp.dept

⑤ Find the name of the employees who work in IT dept and earn more than 12th thousand. (or, EE)

$\wedge$ = and
$\vee$ = or
$\neg$ = not
⇓
If you have multiple conditions cix them

$\Pi_{name}\left(\sigma_{sal>12k \wedge dname}(emp \times dept)\right)$

$\Pi_{name}\left(\sigma_{emp.deptid = dept.deptid \wedge sal>12k \wedge IT}(emp \times dept)\right)$ SQL

select name from emp, dept where emp.deptid = dept.deptid and sal>12k and dname = 'IT'

⑥ Find the name of the person who earn the max salary.

$\Pi_{name}\left(\sigma_{emp.deptid = dept.deptid \wedge sal = max(20k)}(emp \times dept)\right)$

this table will consist those chose sal < d.sal

$\Pi_{name}(emp) - \Pi_{name}\left(\sigma_{sal<d.sal}(emp \times \rho_d(emp))\right)$

SQL → select name from emp where sal = (select max(sal) from emp));

$\sigma_{-dept = dept}(emp \times dept)$

common attribute

$(emp \bowtie dept)$

natural join

sal. is an attribute we can't 'x' them

we can only 'x' the table, not attributes

Here, we have nothing to do with 'dept' table so 'x', (emp x emp) but this is ambiguous so we have to rename one of 'emp' table & have to 'x' it, so that we can't get confused. so emp x ρ(emp)

⑦ Find out the office location of the person named 'EE'?

$\Pi_{dloc}\left(\sigma_{name='EE'}(emp \bowtie dept)\right)$

SQL → select dloc from dept where deptid = (select deptid from emp_02 where name='EE');

Don't have attributes

⑧ Aggregate operator (G) – count() → count(*)

sum()
min()
max()
avg()
} takes only name in attribute

i) It returns value (scalar value). But the other operators return relation (bcoz attributes) (diff. b/w aggregate & other operators)

⑧ What is the avg sal of dept IT?

$G_{avg(sal)}\left(\sigma_{name='IT'}(emp \bowtie dept)\right)$

SQL → done (pto)

⑨ Find the emp's strength of IT dept.

$G_{count(empid)}\left(\sigma_{name='IT'}(emp \bowtie dept)\right)$

inserction operator

SQL → select count(*) from emp where deptid = (select deptid from dept where dname = 'IT');

x is a no. but $y \leftarrow \Pi_{name, sal}(emp)$, it's table, i.e we can't define the data type.

(10) Find ~~the emp~~ the name of the person who earn the max sal. (advance query of qn-(6))

Imp

Imp

$$\boxed{\cancel{\mathcal{G}_{max(sal)}(emp \bowtie dept)}}$$

$$x \leftarrow \mathcal{G}_{MAX}\ \cancel{(sal)}\ (sal)(emp\ \cancel{\Pi(emp)})$$

$$\Pi_{name}(\mathcal{G}_{sal = x}(emp))$$

Sol → done (mr)

(11) Find the name of the person who earn more than the Avg salary of the sales dept.

Imp

Imp

$$x \leftarrow \mathcal{G}_{AVG}(sal)\left(\mathcal{G}_{dname = 'Sales'}(emp \bowtie dept)\right)$$

$$\Pi_{Name}(\mathcal{G}_{sal > x}(emp))$$

Sol → select name from emp_02 where sal > all (select AVG(sal) from emp_02 group by deptid);

• ~~Aggr~~ Group by operation →

$$x \leftarrow \mathcal{G}_{AVG(sal)}(emp)$$

(It is the relation now not scalar value)

$$\mathcal{G}_{deptid}(emp)$$

avg sal of each dept

~~G (group by) half cartesian product but it will also give a single table like it~~

It is actually for ALL dept group by's working ~~exp to x's working~~

$\mathcal{G}_{AVG(sal)}(emp)$ ← emp because we are calculating avg on sal that is on 'emp' table

dept	Avg(sal)
01	15K
02	10K

(12) Find the name of the dept with highest emp strength.

Imp

Imp

$$\mathcal{G}(emp \bowtie dept)$$

$$dept_count \leftarrow {}_{deptid,\ dept_name}\mathcal{G}\ count(*)(dept_count)$$

$$x \leftarrow \mathcal{G}\ max(count(*))$$

$$\Pi_{dept_name}(\mathcal{G}_{count(*) = x}(dept_count))$$

emp strength = count(*)

dept name	emp no.
S1	10
S2	5
S3	15

Not that → (But it isn't incorrect)

Sol → select dname from dept join (select deptid, ~~dname~~ count(*) as emp_count from emp group by deptid order by emp_count desc limit 1) emp on dept.deptid = emp.deptid;

(13) Find the dept with highest employee strength.

$$\boxed{\mathcal{G}_{max}(\mathcal{G}(emp\ \mathcal{G}_{Name = 'empid'}(emp)))}$$

$$\mathcal{G}_{max}(emp\ \mathcal{G}_{max}(emp\ id))$$

Here we are not doing (*) so the no. of dept id = count(*) = no. of emps

Count(*) on emp means count any attributes of employee table

$$dept_count \leftarrow \mathcal{G}_{dept\ id\ count(*)}(emp)$$

$$x \leftarrow \mathcal{G}_{max}(dept_count)$$

$$x \leftarrow \mathcal{G}_{count(*)}(\mathcal{G}_{Name = 'dept\ id'}(dept))$$

$$\mathcal{G}_{emp\ id > x}(emp)$$

$$\Pi_{dept\ id}(\mathcal{G}_{count(*) = x}(dept_id))$$

$$\mathcal{G}_{count(*) = x}(dept_count) \rightarrow x \leftarrow \mathcal{G}_{max(dept.count(*))}(dept_count)$$

dept-id	count(*)
01	2
02	3

$\mathcal{G}$ (emp ~~⋈~~ dept)

dept_count ← dept_name count(*) (dept_count)

deptid,

$x \leftarrow \mathcal{G}$ max(count(*)) (dept_count)

$\mathcal{G}_{count(*)=x}$ (dept_count))

Π dept_name

Not that → (But it isn't incorrect) deptid, ~~dname~~ dept_count

Not that

(11) SQL → select name from emp where sal > (select AVG(sal) from emp) emp where dept id = (select deptid from dept where dname = 'Sales');

are n't doing '*', so the no. of dept id = count(*) = no. of emps

count(*) on emp means count any attribute of employed table

$x \leftarrow \mathcal{G}$ count(*) (emp) / $\mathcal{G}$ empid > x (emp) / $\mathcal{G}$ name = 'deptid' (DEPT)

$\mathcal{G}_{count(*)=x}$ (dept_count) Π dept_id

dept_count	
deptid	count(*)
01	2
02	3

$x \leftarrow \mathcal{G}$ (dept_count) max(dept.count(*))

(dept_id))

It is actually for ALL dept

Group by's working etc to X's working

$\mathcal{G}$ (emp) ← emp

Avg(sal)

because we are calculating

SQL → select dept.deptid

SQL

Create the table

Create table emp(

emp id Number(3) Primary key, → this is temporary (not for you can't write it also)

name varchar(50),

↓ occupy exact 50 chars ig more than it } → o/p: Blank table

Sal Number2(6)

dept id varchar2(3));

↓ latest version of varchar

Want to see the o/p =

Desc emp;

Insert data

Insert into emp values(

1, 'AA', 10K, '01');

→ Statement one record is successfully created

How to modify a table

i) Alter table emp add

Sal Number(5); } → new col will added in an existing table

ii) Alter table emp modify

dept id Number(3); } → you forget dept id should be no. instead of varchar.

iii) Alter table emp ~~from~~

drop Caloum sal; } → delete the sal col.

Relational algebra

– It refers to a procedural query lang. that accepts a relation as input and outputs another relation. [Theoritical foundations for relational databases and SQL are provided by relational algebra] It has basic/fundamental operators ~~used~~ :-

1) selection(σ)
2) Projection(Π)
3) union(∪)
4) set difference(−)
5) set Intersection(∩)
6) Rename (ρ)
7) Cartesian Product (×)
8) Natural ~~operators~~ join(⋈) [Derivative] → 9) G or group by

Operation – It is a reserved word or a char. that is used to query our database in a SQL expression. In SQL dbms we have this operations.

Operand –

An operator manipulates individual data items & returns a result. the data items are called operands or arguments. These are represented by special characters or by keywords [They are of 2 types – i) relation, ii) attributes]

1) selection(σ) - [It is used to select required tuples of the relations. [It only selects the required tuples but does not display them. For display, the data projection operator is used.]"

2) Projection(π) - It is used to project (show) the required column data from a relation. By default, it removes duplicate data.

3) Union(∪) - It is the same as union operation in set theory. The only constraint in the 'U' of two relations is that both relations must have the same set of Attributes.

4) set Difference(−) - set diff. ~~is~~ in r.a is ~~as~~ the same set diff. of as in set theory. The only constraint in the set Diff. b/w 2 relations is that both relations must have the same set of Attributes.

5) set Intersection(∩) - It is the same set intersection of in the set theory. The only constraint in the set ~~Diff~~ intersection b/w 2 relations is that both relations must have the same set of attributes

6) Rename (ρ) - It is a unary operation used for renaming the required attributes of a relation.

7) Cross Product (×) - Cross-Product b/w 2 relations. Let's say A & B, so the Cross Product b/w A×B will result in all the attributes of A followed by each attribute of B. [Each record of A will pair with every record of B. [If A has 'n' tuples & B has 'm' tuples then A×B will have 'n+m' tuples]"

|| Derived operators - They are derived from the fundamental operators -

8) Natural join(⋈) - It is a binary op. natural join b/w 2 or more relations will result in a set of all combinations of tuples where they have an equal, common attributes.

Aggrigate or group by operator

→ This function is used to perform the calculation on multiple rows of a single col. of a table. It returns a single value. It is also used to summarize the data.

~~Aggrigat func~~ → count(*), sum

→ This statement is used for organizing similar data into groups. The data is further organized with the help of fn func, i.e if different rows in a ~~place~~ precise col have the same values it will arrange those rows in a group.

→ count (*) — It is used to count the no. of rows in a database table. It can work on both numeric & non-numeric data types. It considers duplicate & NULL.

→ SUM() — It is used to calculate the sum of all selected cols. It works for numeric fields only.

→ AVG() — — —

Name	Sal 10K	deptid
A		01
B	120K	01
C	15K	02
D	12K	02

⋈

deptid	max(sal)
01	12K
02	15K

Name	Sal	dept	max(sal)
A	10	01	12
B	12	01	12
C	15	02	15
D	12	02	15

the salary of a person of a dept can't match with the max(sal) of others dept as it only counts for its own

(16) Find the name of the persons who earn more than the Avg Sal of a dept. (more well pay his dept) (reasoning)

$r \leftarrow {}_{deptid}\mathcal{G}_{AVG(sal)}(emp)$

$\Pi_{name}\left(\sigma_{emp.sal > r.\cancel{sal}\ AVG(sal)}(emp \bowtie r)\right)$

SQL → select name from emp-02 where sal > (select Avg(sal) from emp-02 where emp-02.deptid = dept-02.deptid); Not that.

Select name from emp ~~group by deptid~~ where sal > (select Avg(sal) from emp) group by Avg(sal), deptid);

what you are selecting must have to be in group by

Delete from emp where sal > 20000; → delete

Drop table emp; → entire table will be deleted

Update emp set sal = sal * 1.1 ~~where~~ where empid = 10; → this will update the following

next of the conditions of the row

(18) Increase the sal. by 10% for the empid 10. $emp \leftarrow (emp - \sigma_{empid=10}(emp)) \cup \Pi_{empid, sal*1.1, name, did}(\sigma_{empid=10}(emp))$

(19) Increase the sal. of IT dept by 10%.

Sol: Err of emp ⋈ dept:

Select * from emp, dept; where emp.deptid = dept.deptid;

Don't 'I' anything Producing everything (select *)

Ans (19)

Err Sol →

Single / multiple line, all are allowed

Update emp set sal = sal * 1.1 where dept id = (select dept id from dept where dname = 'IT')

(17) Find the Avg sal of 'IT' dept.

Query → [(Alternative (without joining))] $\mathcal{G}_{Avg(sal)}(\sigma_{name='IT'}(emp \bowtie dept))$

Err Sol → select Avg(sal) from emp where dept id = (select dept id from dept where dname = 'IT')

~~(18) Ans (without joining)~~ [no need of this (in-18) is already said Ans] → [same for, if you replace deptid with empid, the main point is the common attribute must have to be same]

(18) Ans → Equivalent sql

~~increase~~ update emp set sal = sal * 1.1 where deptid = (select ~~the~~ deptid from emp where empid = '10')

[not work in mysql]

~~Correct to join~~

Insert, delete update ~~RA~~ by 3/8/23

Chw

sql

① insert into emp values('10', 'xyz', 20k, d1)

↓ will insert a full row

query →

emp ← emp ∪ {10, 'xyz', 20000, 'd1'}

② sql → delete from emp where empid = '10'

delete the whole row where empid = '10'

emp ← emp − $\sigma_{empid='10'}$ (emp) ← query

③ Increase the sal of IT dept by 10% → (in-19)

query ⇒ emp ← (emp − $\sigma_{dname='IT'}$ (emp ~~dept~~)) ∪ $\Pi_{name, empid, sal*1.1, did}$ ($\sigma_{dname='IT'}$ (emp ~~dept~~))

[update = combination of insert & del]
[we delete some old data & insert some new data.]

sql - Done

emp

[while using group by operators / aggregate operators on joining 2 tables & ~~also~~ performing some tuples, we have to write tuples like this → (emp ⋈ dept), sql on (emp × dept) [as per need] AND [Also add the sql for group by of each if needed]]

Functional dependency (FD)

(1) (imp)

- R(A1, A2 ... A10) — n rows
- x → y can be

A1 → A10

A1, A3, A5 → A2 A4 A10

x → y [x, y = set of attributes]

{A1, A2, A3} → {A5, A10}

The existance of y is dependent on x

[x, y can contain any values]

[x → y read x determines y]

t1 [...]

t2 [ ]

if t1[A1] = t2[A1]

then, t1[A10] = t2[A10]

~~For any two tuples have matched then~~

[If t[A1] determines t2[A1] then it is right & then t1[A1] & t2[A10] must have to be same.]

- If any two attributes matched, then 1st attribute → 2nd attribute (Here, A1 → A10)

eg - Name & Reg. No.

If Reg. No determines the name, then ~~the name is same but the reverse one isn't true~~. it is right but the vice verse isn't same, so it isn't correct. (i.e, name is dependent on Reg. no. but the vice verse isn't correct)

t_1 [ ]

t_2 [ ]

if $t_1[A_1] = t_2[A_1]$

then, $t_1[A_{10}] = t_2[A_{10}]$

~~For any two tuples have matched then~~

[If $t_1[A_1]$ determines $t_2[A_1]$ then it is right & then $t_1[A_{10}]$ & $t_2[A_{10}]$ must have to be same.]

If any two attr

values]

[$x \rightarrow y$ Read x determines y]

$$emp \leftarrow \left(\cancel{dept}\ emp - \Pi_{(empid, name, sal, deptid)} \left(\sigma_{dname = 'IT'} (emp \bowtie dept) \right) \right) \cup \left(\Pi_{empid, name, sal = 1.1 * sal, \cancel{deptid, dname = 'IT'}} \left(\sigma_{dname = 'IT'} (emp \bowtie dept) \right) \right)$$

Same, so it isn't correct (i.e Name is dependent on Reg. no. but the vice versa isn't correct)

Imp Imp

Armstrong's Axioms (Inference Rules)

this will help us to find out the closure

Rules –

① IR I: (Reflexive)

if $y \subseteq x$ then $x \rightarrow y$

Name ⊆ Reg. no
Reg. no → Name

② IR II: (Augmentive)

if $x \rightarrow y$ then
$wx \rightarrow wy$

Roll → ~~Add~~ Name
Roll, Add → Name, Add

③ IR III: (transitive)

if $x \rightarrow y$
&& $y \rightarrow z$
then $x \rightarrow z$

~~Name~~ Roll → dept && dept → dept loc
Roll → dept loc

Prove – given $t_1[x] = t_2[x]$ $[x \rightarrow y]$
$t_1[y] = t_2[y]$ → can overlook this
$t_1[z] = t_2[z]$ $[y \rightarrow z]$

∴ if $t_1[x] = t_2[x]$ && $t_1[z] = t_2[z]$

∴ $x \rightarrow z$ (Proved)

Imp

④ IR IV: (Union)

Imp if $x \rightarrow y$ && $x \rightarrow z$
then $x \rightarrow yz$

⑤ IR V: (decomposition) [Rev of IV]

Imp Imp if $x \rightarrow yz$
then, $x \rightarrow y$, $x \rightarrow z$ [④, ⑤, ⑥ → Derived / complex]

⑥ IR VI: (Pseudotransitive) ① ② ③ → Basic / simple / fundamental / primes]

Imp Imp if $x \rightarrow y$
&& $yw \rightarrow z$
then $xw \rightarrow z$

Prove –
$x \rightarrow y$
$yw \rightarrow z$
$xw \rightarrow yw$ [By IR II]
$yw \rightarrow z$
∴ $xw \rightarrow z$ (By IR III)

Here $x * x = x$ (as x is an attribute)

⑤ Prove – ~~$x \rightarrow yz$~~
$x \rightarrow y$
$x \rightarrow z$
~~$xz \rightarrow yz$ [By IR II]~~
~~$xy \rightarrow yxy = y$~~

$x \rightarrow y$
$y \rightarrow yz$
yz

$x \rightarrow xy$ [By II]
$xy \rightarrow zy$ [By II]
∴ $x \rightarrow yz$ [By IR III]

⑤ prove -

~~x → (yz) → x~~ $x \not\to yz$

$\Rightarrow \cancel{x \to x} \to yz$

~~yz ⊂ x~~ yz

$y \quad z$ ← subset

$x \to yz \quad yz \to y$ (IR I)

~~[yz → z (IR I)]~~ //

$\therefore x \to y$ [IR III]

$x \to yz \quad yz \to z$ (IR I)

$\therefore x \to z$ [IR III]

Closure ~~Imp~~ Imp

closure to A (A^+)

↓

It will contain all the attributes of A as well as derived attributes. It is a combine set [derived by A^+ combine with A]

A = set of attributes [It can be one/multiple sets]

A^+ = closure of A

~~eto~~ I/P

$R(A_1 \ldots A_n)$

FD = { }

O/P - A^+

→ Algo for it

DO

Closure := A → [Initialization]

for each functional dependency $x \to y$

if $x \subseteq$ closure && $y \notin$ closure

then closure := closure ∪ y

~~to~~ until no change → [If there is any change repeat this iteration again, if there isn't any change then the loop will exit]

I.g - R(A, B, C, D)

FD = { A → B, B → D, C → D } ip

O/P = The closure of any attribute

Let find the closure of A.

Closure = A (outer loop has changed (A, B, C))

↓

closure = A, B, D (outer loop isn't changed)

A^+ = {A, B, D} [this isn't a Primary key]

$(AC)^+$ = {A, C, B, D} → this can be possible candidate key as well as Primary key.

↓

A — A^+ doesn't contain all attributes

C — C^+ also doesn't contain all attributes

[AC contains all attributes so it is a Primary key]

[AC is a CK as A^+ & C^+ isn't containing all attributes

AC is the minimal, so it is candidate key]

Imp

Imp

Imp

Imp

Consider a

① $R = \{A, B, C, D, E\}$, $FD = \{AB \rightarrow C, A \rightarrow D, D \rightarrow E, AC \rightarrow B\}$, what could be the Primary key?

$R = \{A, B, C, D, E\}$

$FD = \{AB \rightarrow C, A \rightarrow D, D \rightarrow E, AC \rightarrow B\}$

[If, A or B or C is a C.K & AD is also a CK or AB is also a C.K. then (AD) the combination of A & D isn't a CK as C.K. is minimal & A & D is already a CK or A & B is already a CK. If we add anything with CK it is superkey]

Closure: AB

↓

Closure: AB, CDE

$\therefore (AB)^+ = \{ABCDE\}$ → candidate key as well as primary key (as it doesn't contain any null values)

Closure: AC

↓

Closure: ACBDE

$\therefore (AC)^+ = \{ACBDE\}$ → candidate key as well as Primary key (")

[In exam you have to get all the candidate keys but can't P.K enough to find

closure: A

↓

closure: A, D, E

$\therefore A^+ = \{A, D, E\}$

Not a CK as not combining all attributes

closure: B

↓

closure: {AC, B} [If we are told to find all C.K then have to find closure for all relations given in R, i.e R (A, B, C, D). find + for A, B, C, D. In between them you can mention only one PK.]

$\therefore B^+ = \{AC, B\}$

closure: C

↓

closure: ?

② $R = (A, B, C, D, E)$, $FD\{A \rightarrow B, AC \rightarrow D, B \rightarrow E\}$ (same ques)

$R = (A\ B\ C\ D\ E)$

$FD = \{A \rightarrow B, AC \rightarrow D, B \rightarrow E\}$

closure: AC

↓

Closure: ACBDE

$\therefore (AC)^+ = \{ACBDE\}$ → AC is the candidate key as well as Primary key

~~Search from net, the queries & sol~~ & sol^n

* Every PK is CK but the vice versa is n't correct.

Functional dependency - the functional dependency is a relationship that exists between two attributes ~~It~~ or we can say a relationship that exists when one attribute uniquely determines another attribute. It typically exists between the primary key & non-key attribute within a table.

~~Eg~~ If R is a relation with attributes x & y, a functional dependency between the attributes is represented as $x \rightarrow y$, which specifies y is functionally dependent on x. Here $(x \rightarrow y)$, the left side of FD is known as a determinant, the right side of the production is known as a dependent.

// Function - ~~The~~ FDs are used to mathematically express relations among database entities & are very important to understand advanced concepts in relation database system.

// Valid FD - roll-no → dept_name since roll-no can determine whole set of {name, dept_name, dept_loc}, it can determine its subset dept_name also (as the prev ex.)

// Invalid FD - name → dept_name Students with the same name can have different dept_name, hence this isn't a valid FD (as, prev ex. (vice versa)).

Arm Strong Axioms

[The term Armstrong Axioms refers to the sound & complete set of inference rules or axioms, introduced by William W. Armstrong, that is used to test the logical implication of FD]. It helps us to find out the closure. Armstrong Axioms are a set of rules, that are ~~when~~ applied repeatedly to generates a closure of functional dependencies.

Closure - The set of all those attributes which can be functionally determined from an attribute set (or) is called as a closure of that attribute set.

closure of a set F of FDs is the set of F^* of all FDs that can be inferred from F. / ~~Eg~~ closure of an attribute x is the set of attributes that ~~are~~ has functional dependency on x with respect to F, denoted by x^* (that means (x^* means) = what x can determine.)

helps us [illegible] a set of
-wing Axioms are applied repeatedly to generates a
are ~~when~~ closure of functional dependencies.

Closure - The set of all those attribu
tes which can be functionally determined
[illegible] is called as a

the sils are not depended on Corresponding Tuines set
here, im writing the sils after learning
the overall Operations (I have used aggregate ~~are~~ has
operators, the most map by operations, not rename, insertion etc) respect
functional de[illegible] means) =
to F, denoted by X^* (that [illegible]
what X can determine.

10/8/23

Canonical covers (F_c) → F is ~~set~~ fd set

Step I: write each fd in standard form.

e.g $A \rightarrow BCD$

is written as

$A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$ (in a decomposed way)

Step II: Remove extraneous attribute.

if for $A \rightarrow B$, any subset of A also determines B, then remove the rest part of A

$A_1 \rightarrow A_3$

$A_1 A_2 \rightarrow A_3$

remove this (extraneous attribute)

Step III: Remove redundant fd.

if $A \rightarrow B$ in F then,

$F' = F - A \rightarrow B$

Now, if $A_{F'}^{+}$ contains B

then $A \rightarrow B$ is redundant

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

$A \rightarrow C$ ✗

$A \rightarrow C$ → this is redundant

$F' = \{A \rightarrow B, B \rightarrow C\}$

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

Canonical cover = minimum of original that holds all attributes of original

Step IV: Union the common dependent.

if $A \rightarrow B$ & $A \rightarrow C$

$A \rightarrow BC$

Emp id ename sal deptid dname dloc

↓

this would cause 3 types of problems

↳ Unnormalized data (unstructured data)

1) Anomalies

- → Insertion anomaly
- → Deletion
- → Update

Primary key can't be NULL

Splitting is the only way to get rid of denormalization

Anomalies – "A database anomaly is an inconsistency in the data resulting from an operation like an update, insertion, or deletion. There can be inconsistencies when a record is held in multiple places & not all of the copies are updated. It is of 3 types.

Normalization

It is a process of filtering denormalised data into normalized data.

denormalized
1NF
2NF
3NF
BCNF
normalized

→ layers [NF = Normal form]
[output of 1NF is i/p of 2NF) without passing any NF you can't go to the next NF]

1) 1NF — A relation is said to be in 1NF iff it has no multi value attribute. [If it has multi value attribute, the way to overcome is split]

~~Std id~~ s In other words the value of each cell should be ~~atom~~ atomic (only one, no another part).

Std id	Std name	Std ph
x	x	987x
x	x	1201

→ id, name (key)

→ id, phn (key) (as 1 Std. can have multiple phn-no)

no P.K here that can be also another (state is foreign key)

Partial functional dependency

(~~It is just like~~ The rest part of A is called extraneous)

for any fd $A \rightarrow B$ if any subset of A also determines B, then $A \rightarrow B$ is partial FD. otherwise it is known as complete FD.

(if $A \rightarrow B$, & subset of $A \rightarrow B$)

2) 2NF — A relation is said to be in 2NF if it is already in 1NF & all the functional dependencies are complete functional dependency.

emp
dept

new table — project (proj-id, proj-name, proj-loc)

emp id, proj id, dur, proj loc

proj (as one emp can work in many projs) — hm key attributes

002	P_1	4hrs	Lond.
002	P_{10}	2hrs	Kol

→ Already in 1NF

Proj-id can determine proj-loc itself so, emp id is partial function. To overcome it we need to split & we have to delete the column proj-loc.

As it already exists.

- If the Primary key is a single attribute then without checking 2NF's properties we can tell that if there is any partial func. or not (as no subset will be possible the)

3rd 3rd
3NF — the relation is said to be in 3NF if it is already in 2NF & a: there is no transitive dependency.

or,

b: there is no dependency between 2 non key attributes.

or,

c: For each functional dependency $A \rightarrow B$ either A is a superkey or B is a primary attribute (part of ~~pri~~ candidate key)

dependency between 2 non key attribute.

R1 (empid, name, sal, (deptid, dname, dloc))

key, either, deptid is a superkey

transitive dependency

(split

R2 (deptid, dname, dloc)

key

empid → deptid
deptid → dname
empid → dname
transitive relation b/w 2 non key attributes

deptid = not superkey, dname = not prime attribute

2nd 2nd
Boyce Codd normal form (BCNF) — A relation is said to be in BCNF if it is already in 3NF & for each functional dependency $A \rightarrow B$, A is a superkey.

Reg No. (A), course (B), mentor (C)

As reg no. will be repeated

as multiple students will do B.tech/M.tech/Ph.d

So p.key will be the combination

AB → C (superkey)

C → B (superkey no)

C (not superkey) → Co. BCNF

(C isn't a superkey, but B is a superkey (so, 3NF no).

there will not be any functional dependency between 2 non key attributes

non key attribute can't be a superkey

R1: A, C (composite p.k)
R2: C, B } : (A, B) (All the info's are preserved)

(as one student can come twice)

(one teacher can do for one course not multiple)

needs more than 1 attri to make primary key = composite p.k

- BCNF is stricter than 3NF as every BCNF is 3NF but the vice versa isn't true. Even if we see the NF then we can see that BCNF comes after 3NF. So it is more stricter.

(1) Find the canonical cover

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

~~Set~~ Step 1

$A \to B, \quad A \to C$

$B \to C,$

$AB \to C$

~~$A \to C, B \to C$~~

Step 2

$A \to B$
$A \to C$
$AB \to C$ [B = extraneous attribute]
$B \to C$

$\therefore$ $A \to B$
$A \to C$
$B \to C$

Step 3 $A \to B$
$A \to C$ ✗
$B \to C$

~~Step 3~~

$F' = \{ \cancel{A \to B, B \to C} \}$
$A \to B, B \to C$

$\therefore A_F^+ = \{A\ B\ C\}$ ← Optional (can write A_F^+, as can't also)

Step 4

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

Canonical covers — (→ Intuitively) It is a set of f.d.s that is minimal, irreducible & equivalent to the original set of dependencies. A minimal cover is a set of dependencies that is minimal but not necessarily irreducible. It is also known as irreducible set.

A canonical cover for F is a set of f.d.s (Fd) such that —

- F logically implies all dependencies in Fd &
- Fd logically implies all dependencies in F, &
- No functional dependency in Fd contains as extraneous attribute &
- Each left side of functional dependency in Fd is unique. ⊛

Extraneous attributes — An attribute is considered to be an extraneous if it can be removed from the left hand side without changing the closure of dependencies.

Denormalized data — Denormalization is a database optimization technique used by database administrators to optimize the efficiency of their database by adding redundant (duplicate) data to one or more tables. Denormalized data means unstructured/unorganised data.

Insertion anomaly – The term "insertion anomaly" is used to describe when a new row is added to a table and it causes an inconsistency. ~~update~~

Deletion anomaly – A deletion anomaly occurs when you delete a record that may contain attributes that shouldn't be deleted. For instance, if we remove information about the last account at a branch, such as account A-101 at the Downtown branch in//

Update anomaly – It is a data inconsistency that results from data redundancy and a partial update. For ex- each employee in a company has a dept associated with them as well as the student group they participate in. Emp id, name, dept.

Normalization – It is a process of structuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity. ~~It has~~ It has forms like 1NF, 2NF, 3NF, BCNF etc.

17/8/23

CW

To check a decomposition is lossless or not

(lossless decomposition)

Let $R(A_1, A_2 \text{ --- } A_n)$

$R_1 \; (A_1 \text{ --- } A_x)$
$R_2 \; (A_5 \text{ --- } A_y)$
$\vdots$
$R \; (\text{ --- } A_n)$
} subsets of R

Algo

Step-I: A relation R having ~~each~~ no. of attributes is splited into m no. of tables/ relations. F is a set of fd of the tables

$F = \{fd^1, fd^2 \text{ --- }\}$

Algo

Step-I – Create a m×n matrix where rows represent this splited tables & columns are the original attributes.

	A_1	A_2	...	A_n
R_1				
R_2				
$\vdots$				
R_m				

→ m×n matrix

Step-II – ~~For each fd in F~~ ~~x → y~~ ~~y in F. If x value of 2 rows~~ For each row if the attribute is present in that table,

write, a_j

otherwise write,

b_{ij}

E.g, $R_1(A_1, \ldots A_n) \rightarrow (A_1, A_n)$

$R_2(A_5, \ldots A_y) \rightarrow (A_1, A_2)$

$F = \{fd^1, fd^2 \ldots\} \rightarrow (A_1 \rightarrow A_2, A_1 \rightarrow A_n)$

	A_1	A_2	...	A_n
R_1	a_1	b_{12}		a_n
R_2	a_1	a_2		b_{2n}
R_n				

i = row
j = col

Step-III – Take each fd $x \rightarrow y$ in F. If LHS is same for any two rows & in RHS we have one 'a' value another 'b' value then replace the 'b' value with 'a' value.

$A_1 \rightarrow A_2$

$(x \rightarrow y)$

$A_1 \rightarrow A_n$

	A_1	A_2	...	A_n
R_1	a_1	~~b_{12}~~ a_2		a_n
R_2	a_1	a_2		~~b_{2n}~~ a_n
R_n				

we will keep on changing until the state of no change.

Step-IV – Check if any row of the matrix has only 'a' values, then the decomposition is lossless otherwise the decomposition is non-lossless. (lossy decomposition)

① R = {empid, name, sal, deptid, dname, dloc}
A B C D E F

Sol

$F = \{A \rightarrow B, C, D, E, F$
$D \rightarrow E, F\}$ — start at 3NF

$R_1 = \{A, B, C, D\}$

$R_2 = \{D, E, F\}$

	A	B	C	D	E	F
R_1	a_1	a_2	a_3	a_4	~~b_{15}~~ a_5	~~b_{16}~~ a_6
R_2	b_1	b_{12}	b_{13}	a_4	a_5	a_6

∴ this is a lossless decomposition

✓ D → E, F

(12)

R AS, G (emp)

R r ← deptid, dname count(*)

common attribute

But this logic isn't correct, there should be one attribute

By giving the common attribute 'deptid' ~~time~~ we are merging 2 tables in 1 so, no need of '⋈'. we are just putting another attribute rather than 'deptid' (dname) in the merged relation 'r'.

② Let $R = \{A_1, A_2, A_3, A_4, A_5\}$ is decomposed into $r_1(A_1, A_2, A_3, A_5)$, $r_2(A_1, A_3, A_4)$ & $r_3(A_4, A_5)$.

FD is $F = \{A_1 \rightarrow A_3, A_5\}$, $A_5 \rightarrow A_4$

$A_1 \rightarrow A_3$
$A_1 \rightarrow A_5$

~~$F_2 = \{A_5 \rightarrow A_4\}$~~

	A_1	A_2	A_3	A_4	A_5
r_1	a_1	a_2	a_3	$\cancel{b_{14}}$	a_5
r_2	a_1	b_{22}	a_3	a_4	$\cancel{b_{25}}$ a_5
r_3	b_{31}	b_{32}	b_{33}	a_4	a_5

∴ The lossless decomposition.

lossless decomposition – It is a decomposition of relation R into relations R1, R2 such that if we perform a natural join of relation R1 & R2, it will return the original relation R. This is effective in removing redundancy from databases while preserving the original data

Transaction

A transaction is a set of instructions that access a database & probably update the data base. [Transactions refers to a set of operations that are used for performing a set of logical work.]

$A \xrightarrow{100} B$ (set of instructions as)

I_1: R(A) [R = Read operation]
I_2: R(B) [W = write operation]
I_3: A := A − 100 → update
I_4: W(A)
I_5: B := B + 100
I_6: W(B)

Inconsistant state = A := A − 100 but B := B + 100 not happen.

- ACID → this 4 properties of a transaction should be satisfied
  - A → Atomicity
  - C → Consistancy
  - I → Isolation
  - D → Durability

Atomicity – A transaction is full done or not done. In database systems, atomicity is one of the ACID transaction properties. An atomic transaction is an indivisible & irreducible series of database operations such that either all occurs or nothing occurs

Condn

Query	SRU
(emp X dept) →	(emp ⋈ dept)
(emp ⋈ dept) →	(emp X dept)

select dname, min(sal), sum(sal), AVG(sal) from dept_02 join emp_02 on emp_02.deptid = dept_02.deptid group by dept_02.deptid;

As we are joining we can interchange

'from emp_02 join dept_02'

interchange

22/8/23

dw

Ex – R(A)
A = A – 100
W(A)
R(B)
B = B + 100
W(B)

If there is a failure in one instruction then all the instructions will be undone. (roll back)

A + B = Const – Consistency

Isolation – No 2 transaction should run in concurrent way. Even if 2 transactions are running concurrently it should appear that 1 transaction has ~~should~~ finished its task. It is known as isolation.

Durability – The changes made to the database should be persistant.

Commit; → to ensure durability
→ Command (Saving)

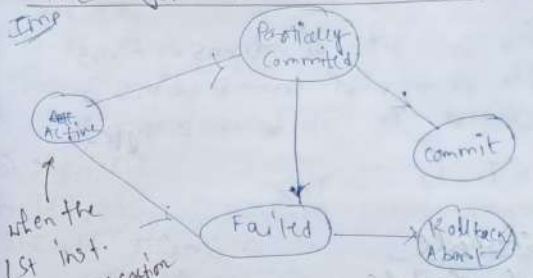
Once commited you cannot rollback. (Opposite of rollback) [As, if we rollback once we can commit it again after rollback]

Durability ensures that changes made to the database (transactions) that are successfully committed will survive permanently, even in the case of system failures.

Ex – If a flight booking reports that a seat as successfully been booked then the seat will remain booked even if the system crashes

Life cycle of transaction

Imp



when the 1st inst. of a transaction is executing / is in execution. It will remain in active state until the execution of last state

• All the dbms supports autocommit

T_1

→ R(A) [R(A) done, auto commit]
→ W(A) [W(A) done, autocommit]
→ R(B) [R(B) done, autocommit]
→ W(B) [W(B) done, auto commit]
→ commit; [But it is been advised to write commit command at the end for more safety.]

Consistency - Consistency in database systems refers to the requirement that any given database transaction must change affected data only in allowed ways. Ex - Data is in a consistent state when a transaction state & when it ends. for eg, in an application that transfers funds from one account to another, the

Consistency property ensures that the total value of funds in both the accounts is the same at the start & end of each transaction.

<u>Isolation, example</u> - In an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

Active State – The active state is the 1st state of every transaction. In this state, the transaction is being executed.

Partially Committed State – In this state, a transaction executes its final operation, but the data is still not saved to the database.

Commited state – A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state – If any of the checks made by the database recovery system fails, then the transaction is said to be in failed state.

Aborted/rollback – If ~~any of check~~ the transaction has reaches a failed state then the database recovery system will make sure that if the db is in its previous consistant state. If not then it will abort or roll back the transaction to bring the db into a consistent state. After this state, the db recovery module will select 1 of the 2 op: 1) Re-start the transaction 2) Kill the transaction.

for insertion,
SQL general format,

select name from emp02 where sal = (select max (sal) from emp_02);

[Insertion will be a combination of many select, where] if there are more subqueries then the no of select & where may increase (you may have to use group by op too

Coming insertion operation

for insertion,
~~SQL~~ general format,
select name from emp02 where :
max (sal) from emp_02);

Coming insertion operation

[Insertion will be a combination of many select, where] if there are more subqueries then the no of select & where many increase, you may have to use group by op too

for qn - 10
like that
other exs
---

not permanent
(can change for others)

(12) SQL → select dname from dept join emp on emp.deptid = dept.deptid group by dname order by ~~AV~~ count(*) desc limit 1;
(case to MySQL)

(13) SQL → select deptid from emp group by deptid order by count(*) desc limit 1;

Shedule -

It is a set of transactions where the order of transactions is also defined. It can be of 2 types.

$S = \langle T_1, T_2 \ldots T_n \rangle$

I) Serial shedule

II) Concurrent shedule

In serial shedule instructions belonging to same transaction appear together. whereas, in concurrent shedule instructions of different transactions are overlapped in time slots.

(S1)

	T1	T2
t_1	R(A)	
t_2	W(A)	
t_3		R(B)
t_4	W(B)	
t_n		W(A)

Concurrent shedule

(S2)

	T1	T2
t_1	R(A) W(B)	
t_2		
t_3		W(A) R(A) R(A)
t_n		

It start & end between t_1 to t_n

Serial shedule

Properties of shedule

Shedule can be classified as recoverable shedule & non recoverable shedule.

If any failure of one transaction in a shedule is fully recoverable by rollback operation then it is a recoverable shedule otherwise it is non recoverable shedule.

If rollback of one transaction leads to a chain of rollback of others transactions then it is known as cascading shedule, otherwise cascadeless shedule.

T1	T2	T3
R(A)		
A=A+100	R(A)	
	A=A+200	
	W(A)	
W(A)		

If we rollback T1 we need to rollback T2 & T3 too.

R(A) W(A) ← Cascading shedule

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

← Cascadeless shedule

If

T1	T2
R(A)	R(A)
A=A+100	A=A+A*20
W(A)	W(A)
	Commit

← Non recoverable shedule

T1	T2
A=A+100	R(A)
	A=20
	W(A)
Commit	

← recoverable shedule

Serialization

Conflict — View

Conflict serialization

STEP-I — If T_1 & T_2 are 2 transactions accessing a common resources ~~queue~~ Q, then 4 cases may arise.

i) R(Q) R(Q) → non conflict

ii) R(Q) W(Q) → conflict

iii) W(Q) R(Q) → conflicting

iv) W(Q) W(Q) → non conflict

STEP-II

Two instructions are said to be conflicting if changing order of them affect the transactions otherwise they are non conflicting. For ex — R(Q) R(Q) is non conflicting & W(Q) W(Q) are all the other cases are conflicting.

STEP-III

We interchange the time slots of all the non conflicting instructions. If the new shedule is a serial shedule, then we call it conflict serializable & the process is called conflict serialization

Ex-1

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

Concurrent shedule = there is a chance of inconsistancy

← Concurrent shedule

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

← Serialization is possible

this is serial shedule

23/8/23

View serialization

STEP I — If S is a concurrent shedule & S' is it's equivalent shedule then whoever reads the initial value of ~~queue~~ Q in S should also read the initial value of Q in S'

	T1	T2
1	R(A)	
2	W(A)	
3	R(B)	R(A)
4	R(B)	W(A) R(B)
5	R(B)	W(A)
6	W(B)	
7		R(B)
8		W(B)

	T1	T2
1	R(A)	
2	W(A)	
3	R(B)	R(A)
4	R(B) W(B)	W(A) R(A)
5	R(B) W(B)	W(A) R(A)
6	W(B)	W(A)
7		R(B)
8		W(B)

R(A)
W(A)
R(B)
W(B)

R(A)
W(A)
R(B)
W(B)

R(Q) R(Q)
W(Q) W(Q) → non conflicting

R(Q) W(R)
W(Q) W(R) → not conflicting

R(Q) W(Q)
W(Q) R(Q) → conflicting

R(Q) R(R)
W(Q) W(R) → non conflicting

Step-II – If T_i writes a value of Q which is read by T_j in S, then the same order should be followed in S'.

Step-III – Whoever writes the final value of Q in S should also write the final value of Q in S'.

Step-IV – For all the other cases time slots of the transaction of S could be interchanged to create S'.

Ex-1

S

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

S'

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

This is a conflict as well as view serialization.

Ex-2

S

T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)

S'

T_1	T_2	T_3
R(A)		
W(A)		
	W(A)	
		W(A)

(view serializable) but not conflict

- All ~~conflict~~ view serializable schedule are conflict ~~view~~ serializable but the vice versa isn't true. (which are is semicolon conflict or ~~the~~ view). (write)

(~~Give~~ ~~Proof~~ Prove it through examples like ①, ②) [view S, S' ≈ S & S' is serial schedule already & if we want we can make S serial too, but in conflict we have to make conflict schedule into serial schedule]

Locking Protocol

two types of lock –

- Shared (S) (Allows multiple locking at same transaction)
- Exclusive (X) (If one transaction locks the resource no other transactions can lock the resource)

	S	X
S	T	F
X	F	F

Lock manager (who will lock, whom have to lock is managed by) (transaction, resource, type) → action → T/F

Ex

T1	T2
lock x(A)	
R(A)	lock - x(A)
	R(A)
W(A)	
	W(A)
R(A)	unlock (A)
unlock (A)	

T	R	W
T1	A	

T2 will not start (until the unlock (A)) will happen, so we have to make the entire block / we have to forcefully make it sequential / serial schedule

lock - S(A)	
R(A)	
unlock (A)	
lock - X(A)	
W(A)	
unlock	

- If we will lock, unlock as per our need then the turnout time will be less.

Two phase locking protocol (2PL)

In a 2PL the phases are growing & shrinking phase. In growing phase the transactions may attain new lock but cannot unlock any resource. In shrinking phase transaction can unlock resources but cannot attain new lock. There are 2 varieties of 2PL — i) Strict & ii) Rigorous. In strict 2PL all the exclusive mode locks are held until commit. In rigorous 2PL all types of locks are held until commit.

Advantage — It avoids cascading rollback.

Time stamp based locking protocol

Read TS

Write TS

Read TS — RTS(Q) is a number which is equal to the time stamp of the transaction who has made the last successful read operation on Q.

Write TS – WTS(Q) is a number which is equal to the time stamp of the transaction who has made the last successful write operation on Q.

T_{10}	T_{20}
R(A)	
	R(A)
	W(A)
W(B)	
	R(B)

RTS(A) → 20
WTS(B) → 10
WTS(A) → 20

Ex –
(2PL) –

T_1	T_2
0 Lock-S(A)	
1	Lock-S(A)
2 Lock-S(B)	
3 ~~Lock-X(B)~~	–
4 Unlock(A)	
5	Lock-X(C)
6 Unlock(B)	
7	Unlock(A)
8	Unlock(C)
9 –	–

T_1, growing phase : from step 1–3
shrinking " : " 5–7
Lock point : at 3
T_2, growing " : " 2–6
shrinking " : " 8–9
Lock point : at 6

20/8/23

CW

T_{10}	T_{11}	T_{20}
R(A)		W(B)
	R(B)	
R(B)	W(A)	

RTS(B) → 10
WTS(A) → 11

$TS(T_i)$
RTS(Q) → time stamp of resource
WTS(Q) → time stamp of transaction

There are 2 cases of TS based locking protocol.

Case - I

Let, T_i want S Read operation on Q

if $TS(T_i) < WTS(Q)$ then

T_i is rolled back

else

$TS(T_i) \geq WTS(Q)$ then

T_i is allowed & to read

$RTS(Q) = TS(T_i)$

$T_{10} < T_{11}$
old — new transaction
greater than 2 newer
lesser than = older

[when we want to R then check who W last & vice versa]

Case - II

If T_i wants to write on Q.

$TS(T_i) < RTS(Q)$ then

T_i is rolled back

if $TS(T_i) < WTS(Q)$ then

T_i is rolled back

Otherwise,

$WTS(Q) = TS(T_i)$

as T_i wants to write so to accept T_i we have to change

- No locking protocol is free of a problem

called deadlock (A is dependent on B & vice versa)

Deadlock handeling

- → deadlock Prevention
- → deadlock ~~Prevention~~ detection & recovery

Imp

Prevention

① wait - die (non preemptive)

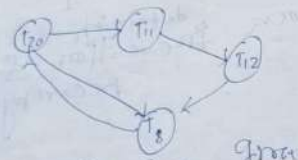
② wound - wait (preemptive)

wait_die is a non preemptive algo. If T_i wants a resource Q which is currently held by T_j, then if $TS(T_i)$ is $<$ $TS(T_j)$, T_i should wait, otherwise T_i is rolled back.

wait is a preemptive algo. If T_i wants a resource Q which is currently held by T_j & $TS(T_i) > TS(T_j)$ then T_i should wait. Otherwise T_j is rolled back. [If T_i is older then T_j it will wound T_j & take T_i's place]

Deadlock detection

Wait for graph - It is a graph of transaction dependency.



If a wait for graph ~~is~~ from a cycle then it goes to deadlock.

Recovery - ① Rollback (~~the~~ A ~~one of~~ ~~only~~ ✓ recovery way of)

② Selection of victim

③ Starvation

[✓ It happens when a transaction is repeatedly denied access to a data item because of other transaction holding locks on it]

(we can do ① & ② but we have to take care of starvation)

③

①

9/3/23

Entity Relationship Diagram

Entity = Real life examples

E.g. Everyone of this class, among the entity is student

SRS

Scheme = tables, attributes, data type, type etc.

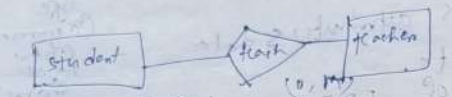
type (common characteristic share)

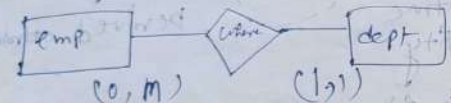
Building blocks of ERD

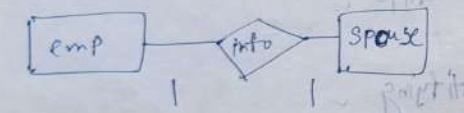
- Entity — Entity
- Weak entity
- Associative entity
- Relationship
- Identifying relationship
- Mandatory relationship
- Optionals
- Partial Participation
- Total
- Attribute
- Key attribute
- Attribute (Key attribute)
- multivalued attribute (reg attribute)
- Derived attribute

[Emp details are dependent, so it is weak relationship]

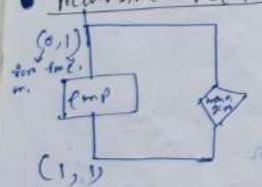
cardinality 1:1, 1:m, m:1, m:m







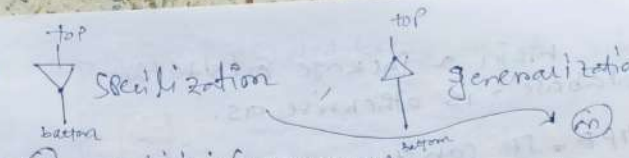
- recursive relationship



[manager has no head so, 0]
[Emp has 1 head]

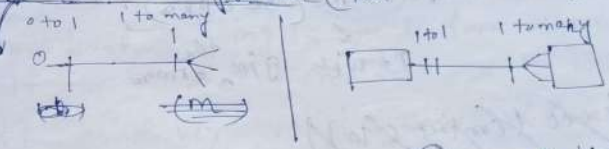
- age is derived attribut [Storing birth no need to store age then]

- ~~TOP Specialization Down~~



- (d) → disjoin (no common)
  (o) → overlapping (not disjoin)

- Uml class diagram (Not in syllabus)



③ • 14/9/23

Class

PL/SQL

DDL = Data defination language
DML = " manupulation "
DCL = " Control "
TCL = Transaction " " (save point, Partial commit)
DQL = Data query language

→ they aren't procedural (Gen generation language)

:= → assignment (here)

(Unscop → Pointers to a Context

Imp Diff between Procedure & func...

Procedure → No return type
func → ~~may~~ must have return type ~~may not~~

" " " is & as → only one func for a particular work

there are many funcs like this

write within a package outside the database - is otherwise as.

RowTYPE - It can store entire row as datatype.

|| — separator

dbms_output.put_line (SQL Error);

It will give the error

~~pvm~~

~~e.g. / exec (Contact (10)).~~

Trigger - If a particular con. satisfied then it will run automatically.

12/10/23

Class

Indexing & Hashing

Order of the index key same as the order of the file is primary indexing

sort by empid

index	
E12	loc F
E01	F108

Empid	name	Sal
E12	AA	10000
E01	BB	12000
F07	CC	10000
F13	CC	11000

secondary indexing — pointers loc

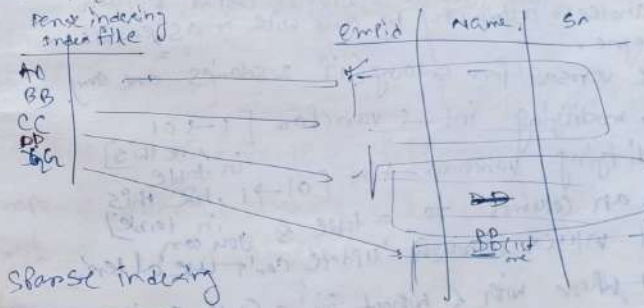
Another index is created like -

AA	loc F
BB	
CC	
CC	

→ Primary indexing

Imp Imp Dense & Sparse index (Primary indexing)

- multilevel indexing is possible for dense index. If two changes then data will be lost
- If search key val is same, we take only the 1st index.



Sparse indexing

- AA
- CC

If DD doesn't create a new bucket we don't change the index file we will only change the original table.

or DD asking is not the 1st id of that bucket → creates then place it both. If 1 elem from that bucket is present then replace.

- only LSB can be used as hash prefix.
- Dynamic or extendable hashing (4)

In case of alter,

1) If empid has 2 same val (empid isn't a PK) & that 2 values have 2 diff. salary: if you want to update empid, you can go through salary as it is different [but it isn't necessary as other attributes can also contain diff values for that 2 same empids].

2) vice versa

3) If you want to update any attributes through an attribute ~~that itself~~ empid, through empid ~~&~~ then that includes empid i.e. through empid and another attribute by 'and' with it as empids are same.

4) vice versa for salary (if 2 salaries are same).

5) ~~alter~~ modifying int → varchar [1 → 01 in table, like this]

6) modifying varchar → int [01 → 1, like this in table]

7) add an column to a table & you can give it values through 'update' can't use 'alter'.

8) Select where with & without joining (both possible)

9) update

10) Aggregate — — — — (both —)

11) Join 2 tables based on FK,

i) natural join → i) If select * then will print both attributes of 2 tables, ii) If select required attributes name then it is a perfect table

2) Kartecian product → i) If select * from t1 → will show attributes from t1 only though you performed joining t2 → ...

3) Join that is applicable for both of ①, ② → i) like ①'s

DBMS (CT) → 2023

① State & Proof IV, V, VI [Done]

② Define & give eg of 2NF, 3NF, BCNF

Defination - [Done]

Eg - 2NF, f →

empid	proj id	proj loc
002	05	NYC
002	10	Kol

f is not completely FD, so it isn't in 2NF. To make it 2NF split the table like,

empid	proj id
002	05
002	10

f1 →

f2 →

proj id (FK)	proj loc
05	NY
10	Kol

Rest of egs [Done]

3) write RA for the following consider

emp = {empid, name, sal, deptid}

Dept = {deptid, dname, dloc}

① Find the 2nd highest salary. salary in

SQL → select ~~max~~ from emp where (select (not needed) ~~salary~~ max(sal) ~~from~~ emp where sal ~~salary~~ < (select max(sal) from emp));

RA → $\pi_{sal}(\sigma_{sal < max(sal)}(emp))$

$\pi_{sal}(\sigma_{sal}$ $\rho \leftarrow \pi_{sal}(\sigma_{sal < max(sal)}(emp))$

$\pi_{sal}(\sigma_{sal = max(sal)}(\rho))$

If you have to add null values then add like this

insert into table-name values(' ', ', NULL);

② Print name of each dept with name of emp, who earn the minimum salary in that dept.

SQL (not needed) → select dname, name from emp join dept on emp.deptid = dept.dept where sal = (select min(sal) where deptid = emp.deptid)

RA → $\pi_{dname, name}(\sigma_{sal = min(sal)}(emp \bowtie dept))$

select dname, name, min(sal) from emp join dept on deptid where deptid = emp.deptid group by dname, name, min(sal);

③ Find the total no. of emp work in '1st floor'.

SQL (not needed) → select count(*) from emp join dept on emp.deptid = dept.deptid where dloc = '1st floor'.

RA → $\pi_{count(*)}(\sigma_{dloc = '1st\ floor'}(emp \bowtie dept))$

④ $F = \{A \to C, C \to D, AB \to DE\}$

Calculate F^c & PK of $R(A, B, C, D, E)$ from F^c.

F^c / F^+ (Canonical Cover)

Step-1

$A \to C$
$C \to D$
$AB \to D$
$AB \to E$

Step 2

No extraneous attributes

Step - 3

No redundant attributes

Step 4

$A \to C$
$C \to D$
$AB \to DE$

$\therefore F^c = \{A \to C, C \to D, AB \to DE\}$

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

Closure: AB

↓

Closure: ABCDE

$\therefore (AB)^+ = \{ABCDE\}$

$\therefore$ AB is a PK

PYQ (DBMS) - 2022

G-A

i) select the best alternative -

i) $A \rightarrow B$ is a trivial functional dependency if $A \supseteq B$ $A \supseteq B$

(ii) $A \cap B$ is same as - $A-(A-B)$

(iii) Project Operator in RA is equivalent to select in SQL.

(iv) After group by we can project - B & C (B - on which it is grouped, C - Aggregated result)

v) An example of an aggregate function in SQL is - ~~all~~ min

vi) Grant Statement is an example of - DCL

vii) After successful execution of the last instruction of the transaction, the state is committed.

viii) Rollback ensures - ~~Atomicity~~ consistency

ix) Second level indexing is - Dense

(x) Starvation is associated with - Deadlock recovery

G-B

2) What are the advantages of using DBMS over file based systems. Describe the three level architecture of DBMS.

Advantages of DBMS over file system:

i) Data Integrity & Consistency. DBMS ensures data integrity by enforcing data constraints & rules.

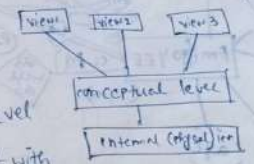
ii) Data Sharing & accessibility.

iii) Data manipulation & query optimization.

iv) Data security & access control.

v) Data backup & recovery.

3-level architecture of DBMS. The 3-level architecture of DBMS seperates the data & the logic that accesses the data.



the 3 levels are

1) External level - This is the level at which the users interact with the database. It defines the various views of the data & hides the complexities of the internal level from the users.

2) Conceptual level - This level is also known as the "logical level" because all the logical work is done in this level. It also designs the whole database & defines the relationship between the tables in the database.

3) Internal level - This level has an internal schema, which describes the physical storage structure of the database.

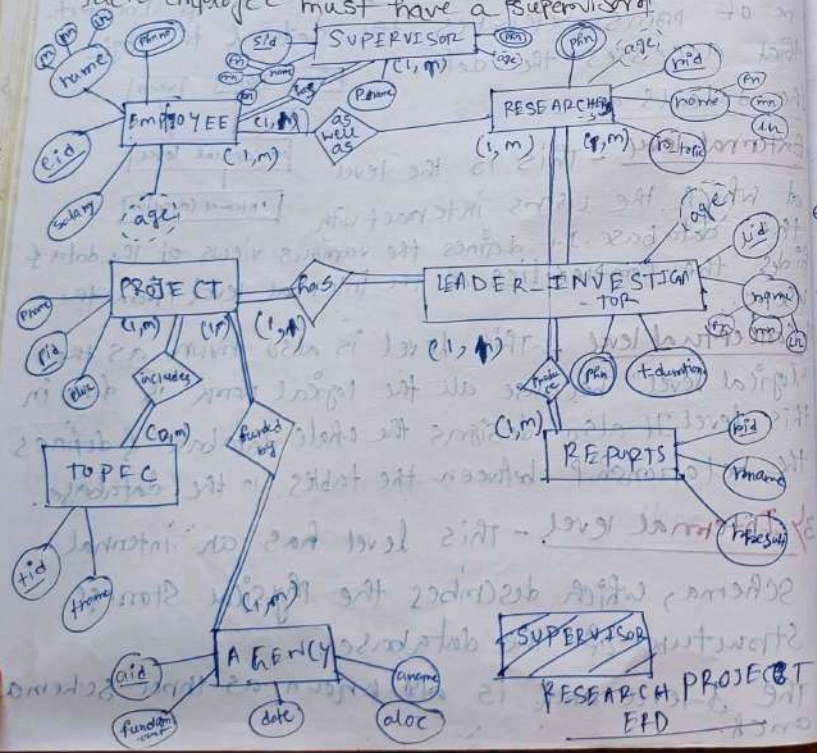
The 3-level arch is also known as three-schema arch.

3) Prove the Armstrong axioms IR iv, IR v & IR vi using the fundamental rules. Define different types of keys with suitable example.

Done, Done

4) Considering the following features of a RESEARCH PROJECT & design a ERD to represent the cardinalities of the entities:

- Some employees are researchers
- Every project has a leader investigator
- Every project must be funded by an agency
- A project may include several topics
- A topic could appear in several projects
- Researchers must produce report(s).
- Each employee must have a supervisor



RESEARCH PROJECT ERD

5) Find the Primary key of the table R(A,B,C,D,E,F,G,H); considering the functional dependency set F = {A → CD, AB → EG, C → G, D → EF, AC → D, G → FH}. Discuss phases of query optimization.

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

F = { A → CD
AB → EG
C → G
D → EF
AC → D
G → FH }

$(AB)^+$ = {A B C D E F G H}

Primary key of the table = AB & it is the only one candidate key here.

Phases of query optimization –

1) Parsing & translation – i) The query is parsed to ensure it adheres to the syntax rules. ii) It is then translated into an internal representation.

2) Optimization – i) The query optimizer explores various execution plans. ii) It aims to find the most efficient plan, considering factors like indexes, join methods, & access paths.

3) Query rewrite – i) Some queries can be rewritten to equivalent forms that may have better performance. ii) This phase involves transformations based on properties of algebraic expressions.

4) Plan generation – i) The chosen plan is converted into an executable form. ii) This involves specifying the orders of operations & access methods.

5) Plan execution - i) The database system executes the optimized plan. ii) Actual data retrieval & processing occurs during this phase.

7) Define ACID Properties. What is cascade less schedule? What is irrecoverable schedule? What is the significance of Precedence graph? Define 2PL Protocol.

Done, Done, Done, Done

~~Significance of precedence graph~~ - In DBMS a precedence graph is significant for representing the dependencies among transactions. It helps ensure the consistency & isolation of transactions in a concurrent processing environment. Specifically, it assists in identifying & avoiding conflicts between transactions ensuring that they execute in a coordinated manner to maintain the integrity of the database.

7) Considering the tables Emp(empid, name, sal, deptid) & Dept (deptid, dname, dloc), write Relational algebra for following queries:

i) Find the name of the persons who earn below the average salary of their department.

$r \leftarrow {}_{deptid}\mathcal{G}_{Avg(sal)}(emp)$

$\Pi_{name}(\sigma_{emp.sal < r.Avg(sal)}(emp \bowtie r))$

ii) What is number of employee who works at 'Ground floor'?

$\mathcal{G}_{count(*)}(\sigma_{dloc='Ground\ floor'}(emp \bowtie dept))$

iii) Who earn the maximum salary in 'Sales' department?

$\Pi_{name}\ x \leftarrow \mathcal{G}_{max(sal)}(\sigma_{dname='sales'}(emp \bowtie dept))$

$\Pi_{name}(\sigma_{sal=x}(emp))$

iv) Find name of the department with maximum employee strength.

$r \leftarrow {}_{dname}\mathcal{G}_{count(*)}(emp \bowtie dept)$

$x \leftarrow \mathcal{G}_{max(count(*))}(r)$

$\Pi_{dname}(\sigma_{count(*)=x}(r))$

7) Find the name of the person who earns the highest salary.

$h \leftarrow \mathcal{G}_{max(sal)}(emp)$

$\Pi_{name}(\sigma_{sal=x}(emp))$

8) Explain conflict & view serialization schemes with following schedule.

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	R(B)

Conflict

Sl	T_1	T_2	T_1	T_2
1	R(A)		R(A)	
2	W(A)		W(A)	
3	R(B)	~~R(A)~~	R(B)	
4	~~R(B)~~ W(B)	~~W(A)~~ ~~R(A)~~	W(B)	
5	~~R(B)~~ ~~W(B)~~	~~W(A)~~ R(A)		R(A)
6	W(B)	W(A)		W(A)
7				R(B)
8		R(B)		W(B)
		W(B)		

We know,

Step-I

If T_1 & T_2 accessing common resources (A, B), then this are the following case that may arise.

for A,

	T_1	T_2
i)	R(A)	R(A)
ii)	R(A)	W(A)
iii)	W(A)	R(A)
iv)	W(A)	W(A)

for B

	T_1	T_2
i)	R(B)	R(B)
ii)	R(B)	W(B)
iii)	W(B)	R(B)
iv)	W(B)	W(B)

Step-II

Now, two ins. are said to be conflicting if chang order of them affect the transaction, otherwise they are non-conflicting.

Here in this case, R(A) R(A), R(B) R(B), W(A) W(A), W(B) W(B), R(B) W(A) [As they aren't accessing same resource], W(B) R(A), W(B) W(A), R(B) R(A) — they are non conflicting & all the others cases are conflicting.

Step-III

We interchange the time slots of all non conflicting instructions. We can see that the new schedule is serial schedule, so we can call it conflict serializers & the whole process is called conflict serialization

view → S (Concurrent schedule) | S' (view serial schedule)

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

We know,

Step-I ~~we know,~~

If S is concurrent schedule & S' is its equivalent serial schedule, then whoever reads the initial value of A in S should also read the initial value of A in S' & the same will be happen for B too. We can see that T_1 reads the initial value of A both in S & S'.

& also

T_1 reads ------ B ------

Step-II - If T_1 writes the value of A which is read by T_j in S, then the same order should be followed in S'. & the same will be happen for B too. Here we can see, T_1 writes the value of A that is read by T_2 in S, & ~~the same~~ S' is also following the same order.

& also

T_1 ------ B ------ T_2 ------

Step-III

whoever writes the final value of A in S should also write the final value of A in S', & the same will be happen for B too.

Here we can see,

T_2 reads the final of A both in S & S'

and also

T_2 ------ B ------

Step-IV

For all other cases time slots of the transaction of S ~~can~~ could be interchanged to create S'. Here we can see, there is no such cases.

~~For~~ For the given schedule all 4 cases are satisfied, so we can call it view serialization & this whole process is called view serialization.

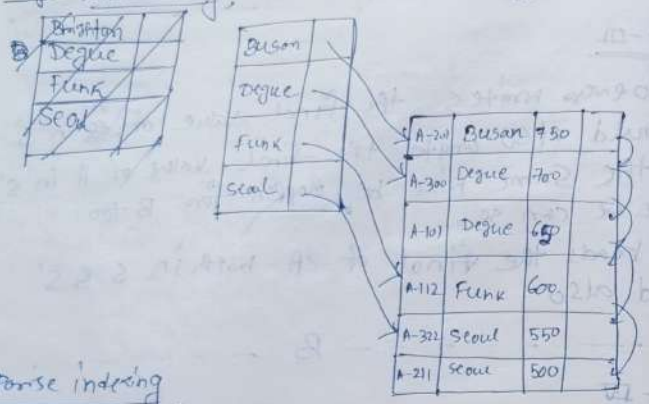
Q) Explain time-stamp based locking protocol. Describe preemptive & non preemptive schemes of deadlock avoidance.

Done, Done.

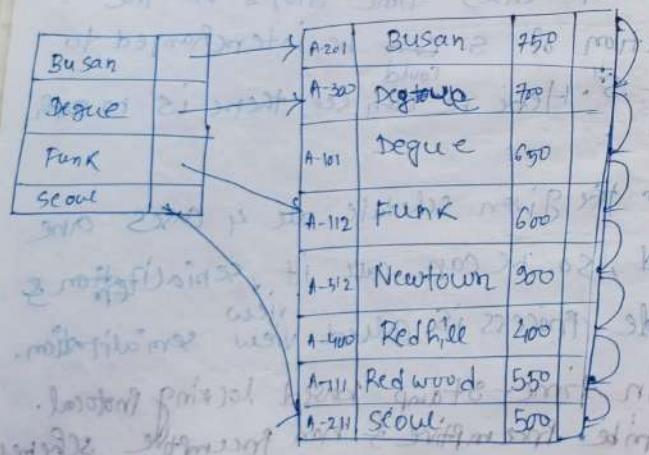
10) Explain dense & sparse indexing. Discuss dynamic hashing technique.

Dense

E.g. Dense indexing,



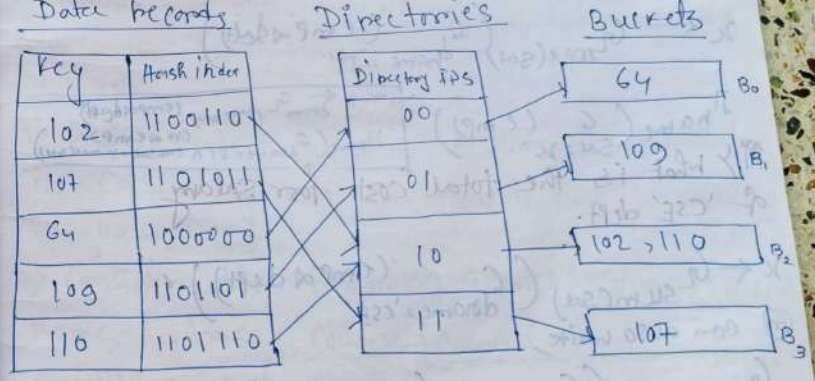
Sparse indexing,



Dynamic hashing technique - Dynamic hashing is a mechanism for dynamically adding and removing data buckets on demand. It is also known as extendible hashing, used to handle database that frequently changes data sets. The way as the numbers of data records varies, the buckets will also grow & shrink in size periodically.

~~Possibilities~~ Technique -

Eg - If global depth (the no of bits in each directory id) : k=2, the keys will be mapped accordingly to the hash index. k bits starting from LSB will be taken to map a key to the buckets. That leaves us with following 4 possibilities; 00, 11, 10, 01



Dynamic hashing

We can see in the above image the k bits from LSBs are taken in the hash index to map to their appropriate buckets through directory IDs. Here the buckets B0, B1, B2, B3 are taken dynamically on demand.

DBMS (Doms) - 2021

1) Let emp{empid, name, sal, deptid} & dept{did, dname, dloc} are 2 relations. Answer the following query in R.A.

i) Find the name of employee who earn more than the avg sal of their dept.

$$r \leftarrow {}_{deptid}\mathcal{G}_{Avg(sal)}(emp)$$

$$\Pi_{name}({}_{emp.sal > r.Avg(sal)}\sigma(r \bowtie emp))$$

ii) find the name of the employee who earn the highest sal in 'IT' dept.

$$x \leftarrow \mathcal{G}_{max(sal)}({}_{dname='IT'}\sigma(emp \bowtie dept))$$

$$\Pi_{name}({}_{sal=x}\sigma(emp))$$

$$r \leftarrow {}_{dname}\mathcal{G}_{max(sal)}(emp \bowtie dept)$$

$$\Pi_{name}({}_{dname='IT' \wedge emp.sal=r.max(sal)}\sigma(r \bowtie emp))$$

iii) What is the total cost for salary of 'CSE' dept.

$$x \leftarrow \mathcal{G}_{sum(sal)}({}_{dname='CSE'}\sigma(emp \bowtie dept))$$

(or) can also write,

$$\mathcal{G}_{sum(sal)}({}_{dname='CSE'}\sigma(emp \bowtie dept))$$

iv) which dept has highest employee strength?

$$r \leftarrow {}_{dname}\mathcal{G}_{count(*)}(emp \bowtie dept)$$

$$h \leftarrow \mathcal{G}_{max(count(*))}(r)$$

$$\Pi_{dname}({}_{count(*)=x}\sigma(r))$$

v) What is the avg sal of the employees working at '3rd floor'.

$$h \leftarrow \mathcal{G}_{Avg(sal)}({}_{dloc='3rd\ floor'}\sigma(emp \bowtie dept))$$

(or)

$$\mathcal{G}_{Avg(sal)}({}_{dloc='3rd\ floor'}\sigma(emp \bowtie dept))$$

2) write an algorithm to find out candidate key(s) from the functional dependency set. Find the primary key of the table R(A,B,C,D,E,F,G,H), considering F={A→CD, AB→EG, C→G, D→EF, AC→D, G→FH}.

Done, Done

3) Define 2NF, 3NF, & BCNF with example. State & Prove the Armstrong's axioms V & VI.

Done, Done

4) Describe ACID properties of transaction? What is cascading rollback? How it can be avoided?

Done, Done, Done

5) Consider the following tables: Student{Roll, Name, Address}, Course{Course_id, Course_name, Credit}, Student_course{Roll, Course_id} & write the following SQL statements

i) write SQL to create the above tables, assuming the appropriate data types.

```
(create table Student{Roll number(6) Primary key
    name varchar(10),
    Address varchar(20));
```

Create table course{ course_id varchar(10) PK,
course_name varchar(20),
credit number(4));

Create table student_course{ Roll Number(6),
course_id varchar(10));

iii) Find the student name who has taken 'DBMS' but not 'Networking'.

RA: $\Pi_{name}(\sigma_{course.name = 'DBMS'}(\text{(course} \bowtie \text{student_course)} \bowtie \text{Student})) - \Pi_{name}(\sigma_{course.name = 'Networking'}(\text{(course} \bowtie \text{student_course)} \bowtie \text{Student}))$

SQL: Select name from Student (Join Student_course on Student_course.Roll = Student.Roll) join course on course.course_id = Student_course.course_id) where ~~name~~ course_name = 'DBMS' and name not in (select name from student (Join Student_course on Student_course.Roll = Student.Roll) join course on course.courseid = student_course.course_id) where course_name = 'networking');

(or) Π_{name} ——— $- \Pi_{Roll}$ (— —)

SQL: select name from student join student_course on Student.Roll = Student_course.Roll join course on Student_course.course_id = course.course_id where course_name = 'DBMS' and student.Roll not in (select student_course.Roll from course join s_c on c.c_id = s_c.c_id where c.name = 'NW');

(iii) Find the total credit of the courses taken by 'Ramiz'.

RA: $\sigma_{sum(credit)}(\sigma_{name='Ramiz'}(\text{(course} \bowtie \text{student_course)} \bowtie \text{Student}))$

SQL: select sum(credit) from course ((Join Student_course on Student_course.courseid = Course.course_id) join student on student.Roll = Student_course.Roll) where Name = 'Ramiz';

(or) select sum(course.credit) from student join student_course on student.Roll = Student_course.Roll join course on Student_course.c_id = course.c_id where Name = 'Ramiz';

iv) Which course has been taken by the max students.

RA: $x \leftarrow \sigma_{\text{course_name}, \cancel{\text{Roll}}, \cancel{\text{sum(Roll)}}, \text{count(Roll)}}(\text{(course} \bowtie \text{student_course)})$ (in what)

$x \leftarrow \sigma_{max(count(Roll))}(x)$

$\Pi_{course_name}(\sigma_{count(Roll) = x}(x))$

SQL: (select course_name from course join Student_course on s_c.c_id = c.c_id group by course_name order by count(*) desc
count(Roll)
limit 1 (for my sql)

for oracle, select course_name from(— — — — — — — count(*) desc) where rownum = 1;
(Roll)

5) Print name & address of the students who has have taken at least 2 courses.

RA: $\pi_{name, address}((student \bowtie (\pi_{Roll}(\sigma_{count(c_id) \geq 2}(S_C \bowtie student)))) \bowtie S_C)$ or $\left[r \leftarrow \sigma_{count(c_id) \geq 2}(S_C \bowtie student) \quad \pi_{name, address}(r \bowtie student) \right]$

Sql: select name, address from student join (select Roll from student_course group by Roll having count (course_id) >= 2) ~~on~~ S_C on ~~S.R~~ student.Roll = S_C.Roll;

6) What is serialization in schedule? Explain different types of serialization with example.

Serialization in scheduling is the arrangement of tasks or events in a specific order, ensuring they are executed sequentially to meet dependencies or for efficient data handling.

Done

7) Define an algorithm for checking loss-less decomposition. Test the following case: Given a relation schema R = {EMPID, NAME, PROJ_NAME, PROJ_Loc, HOURS} is decomposed into the following tables.

R1 = {EMPID, NAME}

R2 = {PROJ_NAME, PROJ_Loc} ~~PROJ_ID~~ PROJ_ID}

R3 = {EMPID, PROJ_ID, HOURS}

The functional dependencies are:

FD = {EMPID → NAME, ~~PR~~

PROJ_ID → {PROJ_NAME, PROJ_Loc}

{EMPID, PROJ_ID} → HOURS}

Done

	A	B	C	D	E	F
R1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R3	a_1	$\cancel{b_{32}}$ a_2	a_3	$\cancel{b_{34}}$ a_4	$\cancel{b_{35}}$ a_5	a_6

m = 3
n = 6

R3 ~~row~~ has only 'a' values so this decomposition is lossless decomposition

8) Describe time stamp based locking protocol. What are the methods used for deadlock prevention?

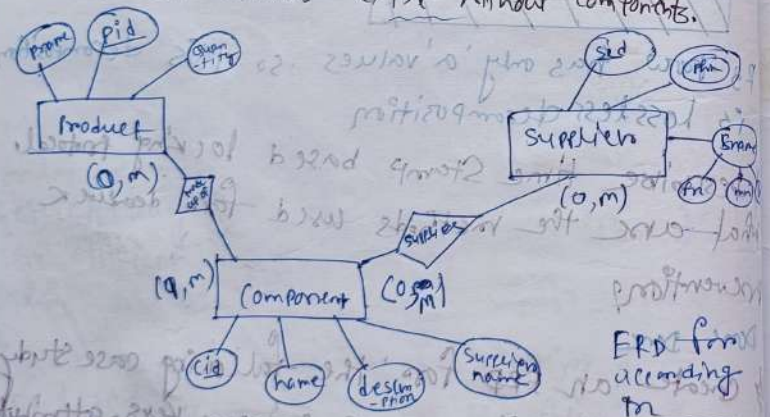
Done, Done

9) Create an ERD for the following case study showing entity names, primary keys, attributes for each entity, relationship between the entities & cardinality.

Optional - Participation

A manufacturing company produces products. The following Product information is stored: product name, product ID & quantity on hand. These products are made up of many components. Each component can be supplied by one or more suppliers. The following component & Each Component can be supplied by one or more Suppliers. The following component information is kept: Component ID, name, description, suppliers.

- A supplier can exist without providing components.
- A component doesn't have to be associated with a supplier.
- A component does not have to be associated with a product. Not all components are used in products.
- A product cannot exist without components.



ERD for according to

As participation showing isn't mentioned in ques, so we are not doing it.

Q) What is Precedence Graph? Explain 2-phase locking protocol. Describe the lifecycle of a transaction.

Done, Done, Done.

10) Write short notes on any 2:

i) Data Independence - Data independence refers to the seperation of the logical structure of data from its physical storage, allowing changes in one aspect without affecting the other. It enhances flexibility & simplifies maintenance. There are 2 types of data independence, logical data independence & physical data independence.

ii) Role of Database Administrator (DBA) -

A DBA plays a crucial role in managing & maintaining a database system. It includes designing, implementing, optimizing databases etc. It also manage user access, monitor performance, & plan for scalability, contributing to the overall reliability.

(iii) Different types of keys in database design - done

~~Extra~~ Extra Questions (mostly asked)

1. Diff. b/w DBMS & RDBMS
2. Constraints & their types
3. Diff. b/w Delete, Drop & Truncate (Imp)
4. Diff. b/w group by & order by
5. Types of joins (Imp)
6. Diff. b/w nested subquery & correlated subquery (Imp)
7. Pattern matching
8. Find nth highest salary (Imp)
9. SQL vs MySQL & SQL vs NoSQL (Imp)
10. varchar vs varchar2