Introduction

Database
collection of logically interrelated data & description of this data
eg: Bank database
Railway reservation
College database < student table
faculty table

Features of data in a database
1. Shared: b/w many people i.e user of data & applications of data.
2. Persistence: Data should be permanent.
3. Validity :- data should be valid, correct & it should remain.
4. Consistency :- remain same throughout the scope of program.
5. Non-redundant :- should not be replicated any where means should remain at one place.
6. Independence

DBMS: system used to manage database
database + software eg: IRCTC.

It is collection of interrelated data & a set of program to access these data.

The collection is referred to as database, contain info about or related to enterprise.
The primary work of DBMS is to store, retrieve database that is convinient & efficient (within time).

* Applications of DBMS

1. Enterprise Info
   a) Sales - customer product
      Accounting - payment receipt, balance
   b) Human resources → data of employes
   c) Manufacturing → making factories unit, then dispatch - - -
   d) online retailers - online shopping

2. Banking & finance
   a) Banking operation :- accounts, loans, banking transactions, credit card transaction
   b) Finance - info of holidays, sales, purchase of policies etc.

3. Universities
   - student, faculties, employees database.

4. Airlines :- flight reservation (1st used DBMS) used distributed database.

5. Telecommunications :- mobile companies, bills the first

* Purpose of DBMS

1. Data redundancy & inconsistency
   - will be removed using DBMS
2. Difficulty in accessing data
   In case of text file → sentences present → we write app. program → then string matching → tedious whereas in DBMS → change query.
3. Data isolation
   in file system is ~~complex~~ present while in DBMS is easy because of presence of shared data.
4. Integrity problem
   of file system by using constraints that will be specified along with queries so can overcome the problem.
5. Atomicity problem either transaction done completely or not done.

	A	B
Balance	1000	2000
Debit	1000	
	− 50	
	950	

& now system stops & B is not credited with 50 ∴ problem can overcome using DBMS.

6. Concurrent Access anomalies
   - file already in use by one user then other user can't use it in file system.

7. Security problems :- Not every data user should be given access to whole data
eg :- To check balance in bank we can do but can't credit or debit yourself, we need to give query to banker for crediting or debiting or through system.

* Comparison between File system & DBMS

File management	DBMS
- It is used in small system where data is less. (C++)	- It is used in vast / large system (oracle is used)
- It is cheap (C++ is freeware)	- It is expensive (oracle is licensed software)
- It has simple structure	- It has complex structure
- Low level design	- High level design
- Not secured	- Secured
- Single user	- Multi user
- Isolated data app1 app2 data1 data2	- Shared data same data is shared among various users.
- Simple Backup mechanism	- Complex.

* View of data

DBMS provides abstract view of data, it hides the way how data is stored.

3 levels
(a) Physical level
(b) Logical level
(c) View or external level.

Physical level :- how data is stored in DBMS
- data type
- DS
- size

Logical level :- what data is stored
(Abstraction is present) → student - name, class, RNo. branch

View level :- granting access as much required like in university to check result → roll no. given → no other info given which are already stored in DBMS.

eg:
```
type instructor(5);
    ID : char(5);
    name : char(20);
    salary : number(7);
end;
```
Physical level

instructor, ID, name, salary ⇒ logical level

view level:- how many levels are viewed either 2, or 3 - - - -

* Schemas & Instances

Schemas :- overall design of database
- only 1 for database

collection of info in a database system at a particular moment ← Instances :- at present what data is present, many of 1 single database

eg:- Table of employees ⇒ schema

Emp	cname	eid	emobile
1992	A	1	
	B	2	
1996	C		
	D		
	E		
	F		

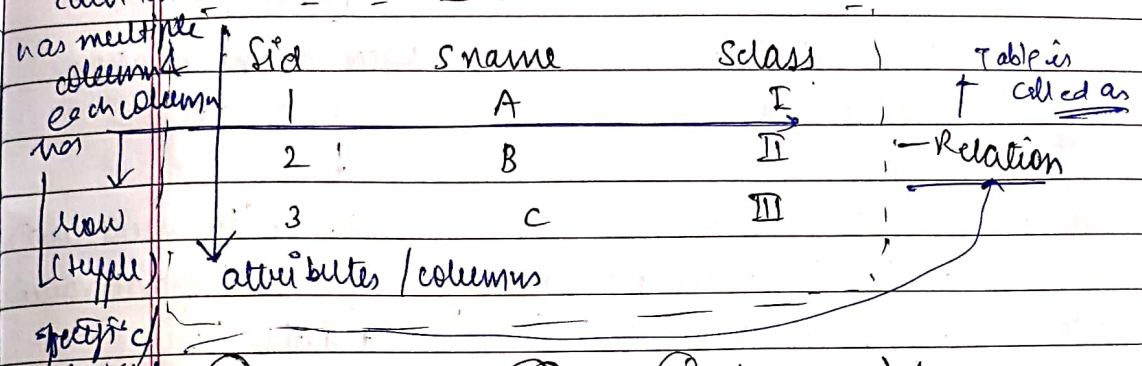
In 1992 → 2 records means instance
In 1996 → if 4 more added then instance will be 6 records.

Subschema:- part of total schema. app. or users point of view.

They are collection of conceptual tools for describing data, data relationship, data semantics & constraints
↑ 2 parts → mathematical notation exist b/w data & relation that data
↳ set of operation to manipulate

* DATA MODELS

1. Relational models:- relationship between data & database.
   - It is example of record based model (info related to one particular thing)



(1) Id of student (A) name (I) class → Record based model

2. ER Model (Entity relationship)
   collection of entities & relationship among those object.
   Relationship among entity are identified
   ER diagram → tables.
3. Object based data model (Extended ER model)
   Data is object oriented. ER model + encapsulation + methods + object identity
   object → store, maintain, manipulate
4. Object - relational data model
   Combination of object oriented data model + Relational model.
5. semi structured data model give specification of data where individual data items → diff. data length possible (attributes)
   eg: XML having same type name different set of attributes.

- insert new data,
- modify data
- retrieve info
- delete

* Database languages

Languages of DBMS

DDL
Data definition language
↓
create (table)
alter (add, delete in already present table)
drop (delete table)
↓
schema i.e sid, sname will also be deleted

create, modify, delete database structure but not data

DML
Data manipulation language
↓
insert (to insert row)
update (update value)
delete (will delete content of schema but not schema i.e sid, sname will remain)

allow changing of data within database

↓

Procedural DML
How data is fetched is told
eg:- total program, link, string matching etc.

It requires a user to specify what data is needed

Non-Procedural DML
What data is fetched without specifying how to get those data

* DC CO
It is component of SQL statement to control access to

*

*

*

Non-procedural

- Commercial query language → SQL
- Formal query language → i.e relational algebra, tuple calculus

* DCL (Data control language)
  will control data, how much access is given. It includes :- command like grant (to give access) Revoke (to take the request) of showing or accessing

  It is component of SQL statement to control access to data & database

* DQL (Data query language)
  command :- select → "retrieve data from database".
  select already stored data & show

* Transaction language

commands: commit
. save point
. Roll back (at time of atomicity)

to complete any transaction :- commit
back up :- save point

→ commit : save work done
→ Roll back : restore database to original since last commit

Ability to modify schema design at one level without affecting schema at next higher level

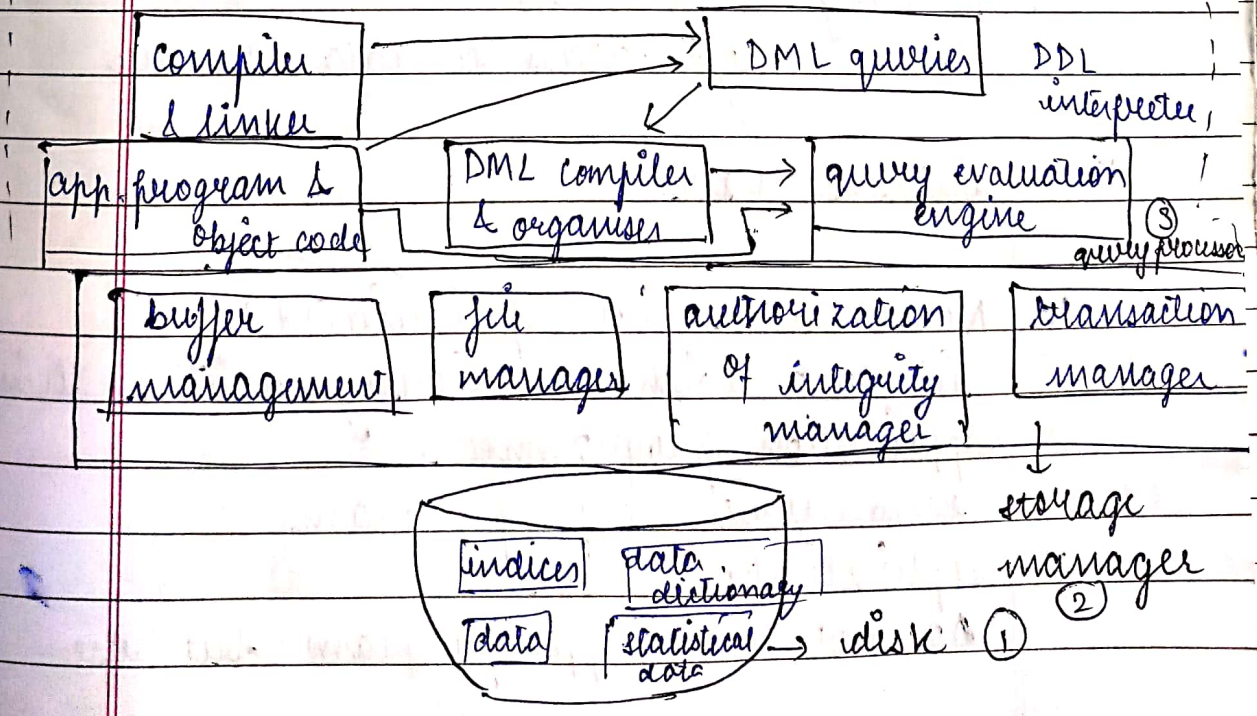
* Data independence
Based on level it is of 3 types

(i) Physical data independence] main
(ii) Logical data independence]

difficult than because app. programs are very much dependent on logical structure of data.

Physical independence	Logical independence
changes at physical level without affecting logical / external level.	change can be at lower level but not on external level
- Physical storage structure, devices can be changed without affecting conceptual schema	conceptual schema can be changed without affecting external schema
- improves performance	- structure of database is altered.
- data is copied	- Difficult as change at logical level
- immunity to logical or external level	- It provides immunity to external level.

* components of data base



(4) Naive users → application interfaces
(5) Application programs → app. progra.
(6) sophisticated user → query table
(7) DBA → administrative program

DBA (Database administration)
- person or group of person having overview of 1 table or entire table to view entire database.

FUNCTION OF DBA

1. Defining conceptual database schema definition
2. Storage structure & access method like arrays etc
3. Security & integrity checks authorization of data, no data leakage
4. Backup and recovery strategies weekly, monthly, at how much interval & how backup is to be taken.
5. granting user access how much access to data to user

* Database Users

1. Naive users (unsophisticated) interact with previously written things
2. Application programmers who write app. programs
3. sophisticated users do not write app. program but use

query to fetch the program.

4. specialised user
eg :- knowledge or expert system.

* storage manager

1. authorization and integrity manager
it test for integrity of constraints.
check whether query is valid or not.

2. Transaction manager
data is in consistent state or not
→ before & after transaction it should be consistent.

3. File manager :- how to store info,
how to represent in file.

4. Buffer manager :-
how much data will be stored
in different memories like cache, RAM
or disk.

* Query processor
Process query & pass to storage manager

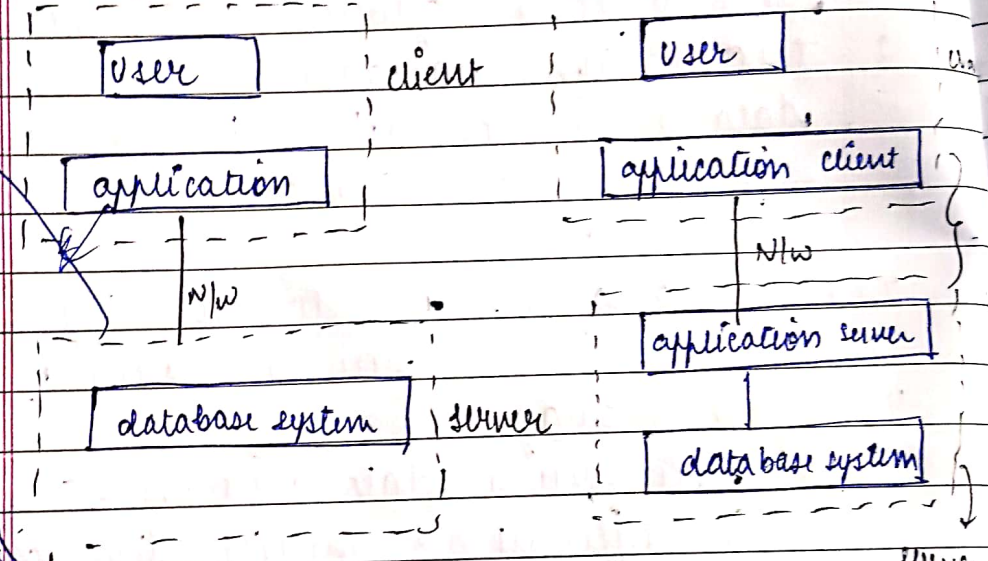
1. DDL compiler - schemas related query are compiled. (static commands are compiled)

2. DML interpreter - ① interpret
low execution ← ② query optimization
cost.
Here we need to select
& to handle data ∴ use interpreter
in a way to execute query → low
cost.

3. Query evaluation engine
at present execution of instruction/
query given to it.

* DB architecture (IMP)

Two tier architecture



App. invoke database system through query lang.

- when no. of user ↑ then this architecture doesn't work
- Not used now.
- Problems like scalability.

Three tier archi.

- used mostly
- Processing at server level in 2-tier then given to Business layer
- application server ↳ query resolved here & interact with database system.

Business level/ layer

Scanned by CamScanner

super key:- it is a set of one or more attributes to uniquely identify row/ tuple in a relation
eg: ID

* Relational model

Data is in proper tabular form

Sid	Sname	branch
S_1	A	CSE
S_2	A	IT
S_3	B	CSE

→ Relational instance

set of attributes that forms superkey but none of its subsets is a superkey.

attributes — domains.

Cardinality → no. of rows.

* **Keys** No two tuples can have same value of all attributes
It allows to uniquely identify set of attributes or min no. of attributes in a relation that is used to differentiate tuples of relations.

attributes are underlined

(i) Primary key → to denote a candidate key by database designers to identify tuple in a relation
(ii) candidate key → Minimal superkeys
(iii) Alternate key
(iv) Secondary key
(v) super key
(vi) Foreign key
(vii) composite key

{name, dept-name} & {ID}
combination

{ID, name} → ~~candid~~
because ID alone is a candidate key.

it is one or more column to identify a row in a table
- Values of attributes should not changed & it should be null.

] is chosen in such a way that

student id	course id
1.	A
	B
	C
2.	A
	B
	C

Here (sid cid) ⇒ taken together as a key.

enrollNo	Sid	Sname	PassP No.	License No.	DOB	Fname
	S_1		NULL/unknown			Y
	S_2		P_4			X
	S_3		NULL			Z
	S_4		P_3			
	S_5		NULL			

candidate key uniquely identify row/tupple, minimal key.

In above table 3 → enrollno, sid, LNo } can be uniquely identify

work as candidate key

combination (Sname + Fname) ∈ candidate key

Suppose we consider Sid as primary key.

Other keys except primary key i.e sid will be alternate key

Let us consider LNO as secondary key

(We have to specify them at time of indexing)

superkey:- super set of any key.

Indexing done at primary key

Borrowing of attribute = Foreign key

* constraints for primary key

* CREATE TABLE student
(sid varchar(10) primary key, Sname varchar(20) NOT NULL,
PassPNo varchar(20) UNIQUE, LNo varchar(10) UNIQUE, NOT NULL,
DOB date NOT NULL, FName varchar(30)) NOT NULL,
unique (DOB, Fname));

Q The maximum number of superkeys for the relation schema R (E, F, G, H)
E is the key is _____
(R = table name; E, F, G, H = attribute)

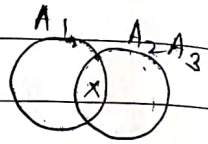
* No. of superkeys = 2^{n-1} where n = total no. of attributes in any relation
$= 2^{\text{no. of non-key attributes}}$
1 = no of key attributes

* We have relational schema R ($A_1, A_2, \ldots A_n$)
then the total no. of super keys = ?
the candidate key = A_1
super key = A_1, A_1A_2, $A_1A_2A_3$, $A_1A_2A_3A_4$ .... n-1 times

No of superkeys = 2 × 2 × 2 × ---- (n-1) times
$= 2^{n-1}$

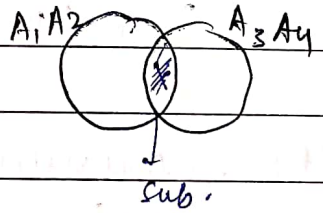
Q We have relational schema R($A_1, A_2, A_3, \ldots A_n$)
candidate key = A_1, A_2
No. of superkeys = 2^{n-2}
candidate key = A_1, A_2, A_3
No. of superkeys = 2^{n-3}

Q If the candidate key = $(A_1, A_2 A_3)$



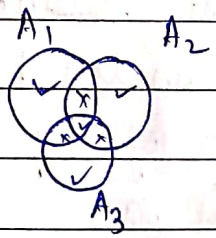
No. of superkeys =
superkeys due to A_1 + superkeys due to $A_2 A_3$ - common superkeys (A_1, A_2, A_3)
$= 2^{n-1} + 2^{n-2} - 2^{n-3}$
$= 2^{n-3}\left(\frac{2^{n-1}}{2^{n-3}} + \frac{2^{n-2}}{2^{n-3}} - 1\right) = 2^{n-3}(2^2 + 2 - 1)$
$= 2^{n-3} \cdot 5$

Q candidate key = (A_1, A_2, A_3, A_4)



No. of superkeys
$= 2^{n-2} + 2^{n-2} - 2^{n-4}$
$= 2^{n-1} - 2^{n-4}$

Q Relation R(A, B, C, D, E)
candidate key = (A, BC)
= (A, BC)
= (A_1, A_2, A_3)

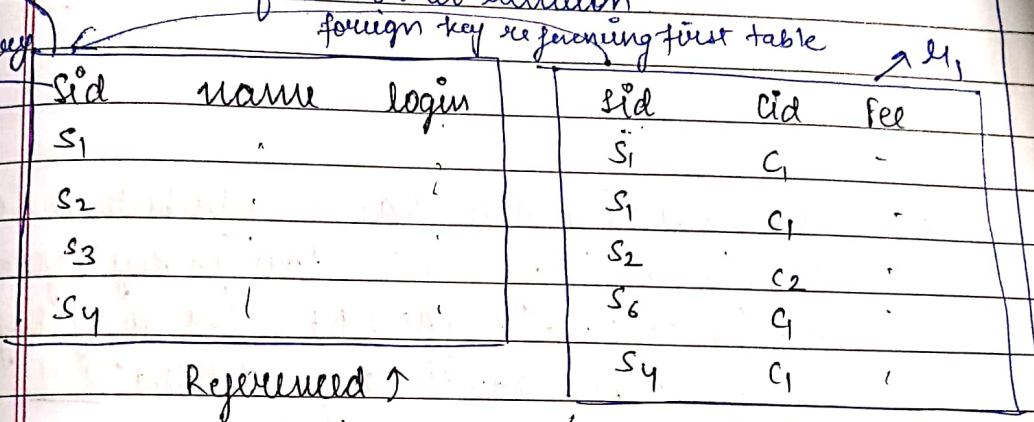


Referential integrity constraint requires that the value appearing in specified attributes of any tuple in referencing relation also appear in specified attributes of at least one tuple in referenced relation

Let us consider a relation R_1 may contain among its attribute the primary key of other relation (R_2). This attribute is foreign key from R_1, referencing R_2.

* Foreign Key (Referential key)

If a relation doesn't have unique key, it will borrow from other relation



Sid	name	login
S_1		
S_2		
S_3		
S_4		

Referenced relation
student table

Sid	Cid	Fee
S_1	C_1	
S_1	C_1	
S_2	C_2	
S_6	C_1	
S_4	C_1	

(Here Sid is foreign key
Referencing relation
Student enrolled table

CREATE TABLE Enrolled
(sid varchar(10),
cid varchar(10),
Fee integer,
PRIMARY KEY (sid, cid),
Foreign key (sid) references student (sid)
ON DELETE NO ACTION

R_1 is referencing
R_2 is referenced relation.
- can be left NULL
→ it follows referential integrity rule

// means if we delete any tuple from enrolled table should not delete anything in student table.

foreign key (sid) references student(sid)

One more ON DELETE SET NULL → set sid on student table to NULL.
ON DELETE CASCADE

this is referencing ∴ called self referential integrity

Employee → supervisor

Eid	supid
E_1	NULL
E_2	E_1
E_3	E_1
E_4	

as E_1 of supid references a particular Eid i.e E_1 in same table.

* The following table has 2 attributes A & C
A = Primary key & C = foreign key referencing A with ON DELETE CASCADE. The no. of tuple that must be additionally deleted with 2, 4 is

Primary → A | foreign → C

A	C
2	4
3	4
4	3
5	2
7	2
9	5
6	4

delete

To prevent from referential integrity delete 2 from (e column) ∴ additionally 2 will be deleted

due to above Eid deleted ∴ supid no supervisor ∴ that will be deleted.

ON DELETE NO ACTION → no other tuple will be deleted.

Constraints of referential integrity.

1. Insertion :- no violation
2. Deletion :- violation may be possible in reference table on delete no action (by default)
" " cascade

Scanned by CamScanner

Employee
Eid — Sunid

The value of primary key cannot be NULL
Entity integrity.

(deletion from both table)
on delete set null

Referential integrity
any value of FK is not NULL

* Entity relationship model (ER model)
High level database design
↓
ER diagram
↓
ER table

Entity :- any object which can be differentiated from other object in real world.

Entity set :- set of similar entities.

Attributes / column → representation

Primary key

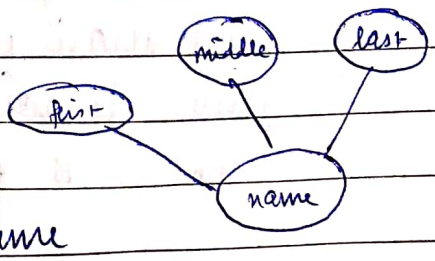
Multivalue attributes

Multivalued

Sid	Sname	Cname
1	A	C / C++
2	B	C / JAVA

Compound attributes -

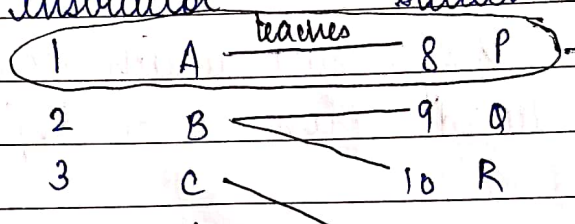
name
/ | \
First Middle last name



* Derived attributes
We know DOB → age can be derived.

(DOB) ⋯ (Age)

* Relationship
is an association among entities.
for eg:-



relationship b/w instructor & student.

* Relationship instance
one record of above table

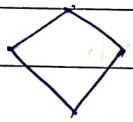
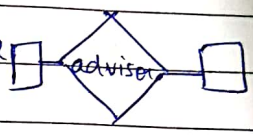
* Role
function that an entity play here.

* Recursive relationship set
If some entity participate in a relation more than one & both times or many times it will be in diff. role

ex: B → 9 } advisor is B
B → R } for both

The Entity set E in relationship set R is said to be total if all entities in E participates in atleast one relationship in R. If some participates → Partial

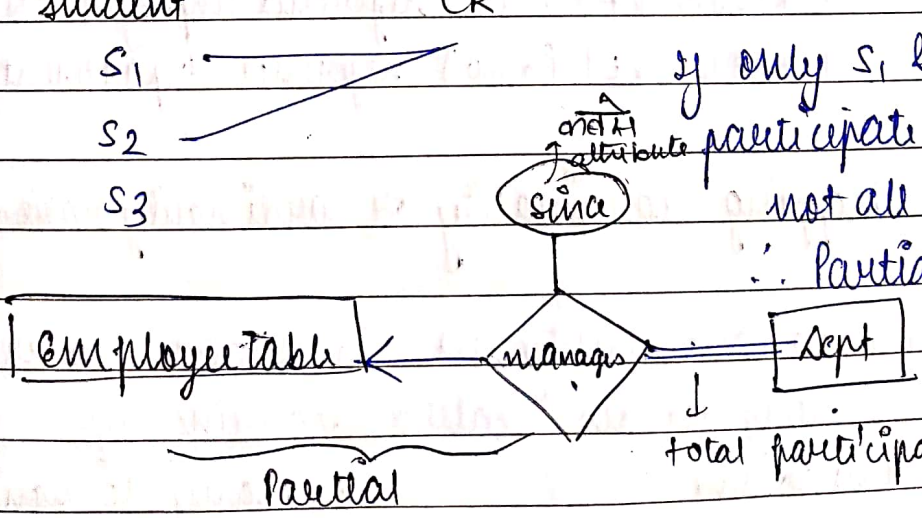
* Binary relationship
  It is between 2 entities, 2 entity set
* Degree of relationship set :- No. of entities set that participate in a relationship set

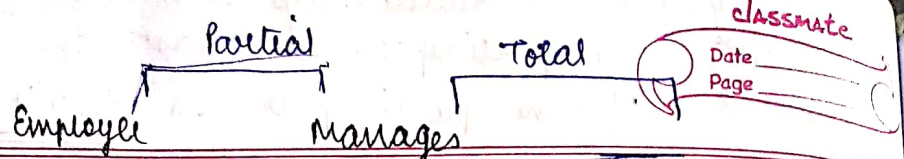
* Relationship set  example 

- constraints of relationship set
1. Total participation (═══) (=) Every entity in entity set participate in relation.
2. Partial participation (——) - some entities participate.

b/w 2 entities ⇐ is a ⇒ relation

student	Sub option	If student have to participate in atleast 1 sub : total
S_1	Opt_1	
S_2	Opt 2	
S_3	Opt 3	





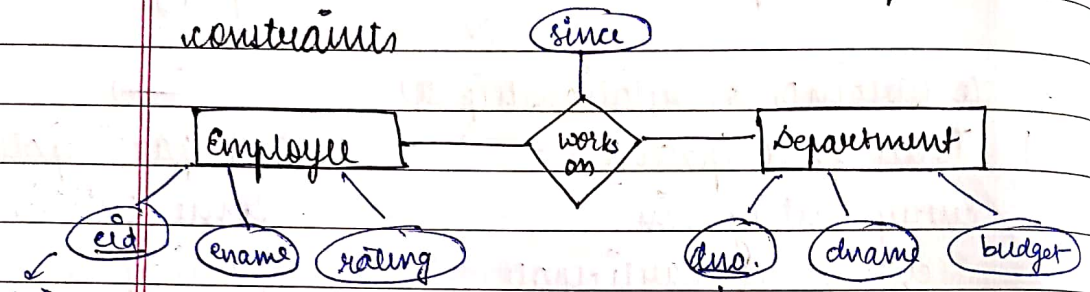
E_1 —— 10/10 —— d_1

E_2 —— 12/10 —— d_2

E_3 —— 11/10 —— d_3

E_4

* How to create table in relationship set constraints



* attribute with relationship set = descriptive attribute.

Create table workson
(eid varchar(10),
dno varchar(5),
since date,
FOREIGN KEY (eid) reference Employee ON DELETE CASCA
FOREIGN KEY (dno) reference Department ON DELETE
NO ACTIO

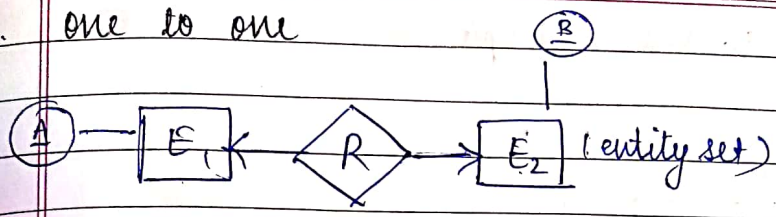
* Mapping cardinality or cardinality ratios

How one entity set is related to another entity set is called cardinality.

1. one to one
2. one to many
3. Many to one
4. Many to many

1. one to o
2. one to
3. Many
4. Many

1. one to one

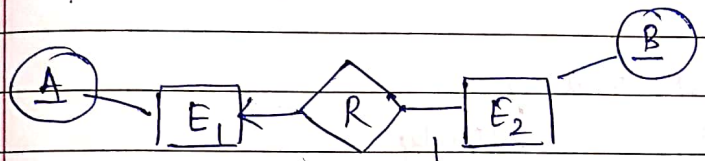


(entity set)

Any one

candidate key = (A, B)

2. one to many



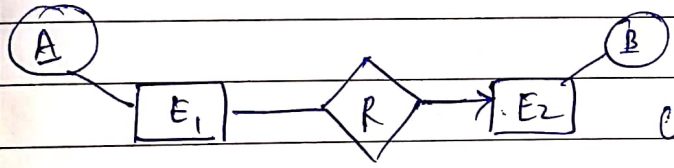
candidate (many)

key = B

No arrow
i.e many

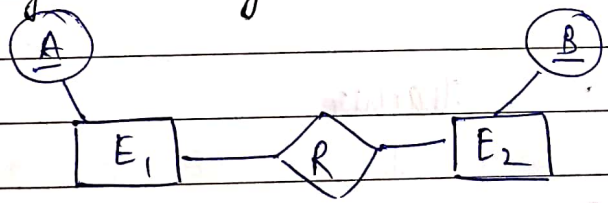
when one to many we take candidate key of many side.

3. Many to one



candidate key = A

4. Many to many



candidate key
= (A, B)
i.e (AB)
combination.

one to many example

Eno.	did
E_1	d_2
E_2	d_3
E_3	d_1

* One to one example

dept		HOD
CSE	⟷	A
IT	⟷	B
ECE	⟷	C

* one to many

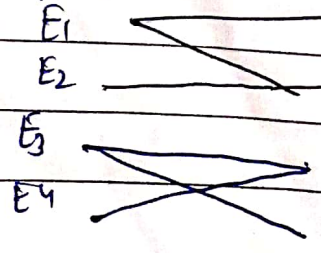
HOD	faculty
A	m no. of faculties 1
	2
	3
	4
B	1
	2
	3
	4

* Many to one

student	teacher
60 students	1

* Many to many

Employee — Project



Project 2 has E1 & E2
E1 working on P1 & P2

Non-binary / Ternary

* Entity relationship design issues
Broadly 2 | 3 issues

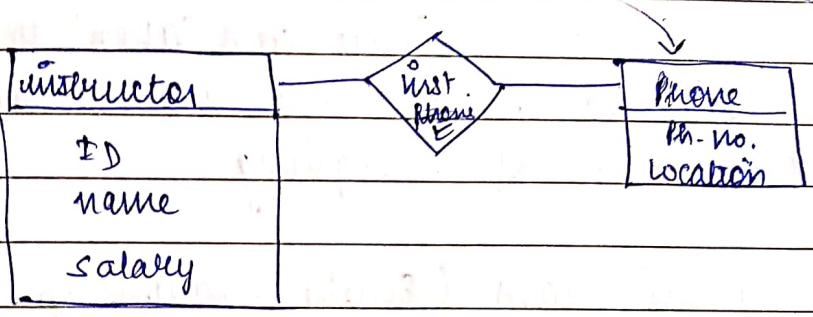
eg:-

instructor table
ID
name
salary
Phone no.

table name

attributes

If we want to store diff. info about phone then we don't take it as attribute we take it as set.



on the basis of design

1. Use of entity set Vs attribute

- single value → take as attribute
- Multiple " → take as entity set
- decided at time of design.

2. Use of entity set Vs relationship set

When it is not clear that whether the object is best expressed

It is not clear to use something as entity set or relationship set.

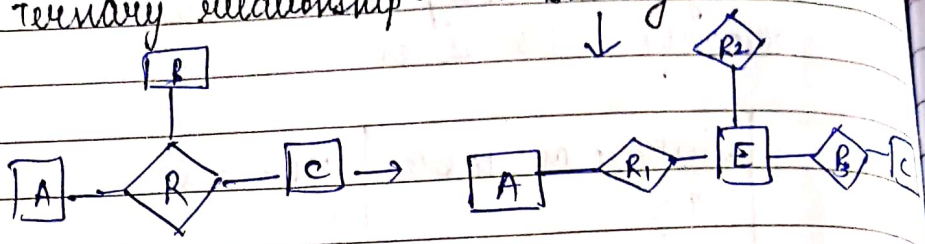
Like instructor phone → entity
→ relation or

Most of relationships in database are binary. Some relationships that appears to be non-binary can be expressed easily in terms of binary relationships.

3. Use of binary or n-ary relationship.

Pg 293

Ternary relationship ⟶ Binary



We decide at time of design

4. Placement of relationship attributes.

Attributes of one to one or one to many relationship set can be associated with one of the participating entity set rather than with the relationship set.

Descriptive attribute can taken to many side as a attribute has to decided at time of designing.

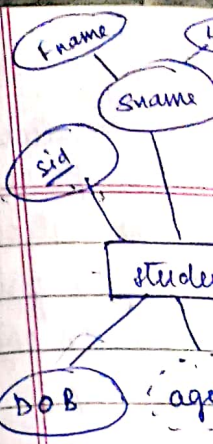
IMP

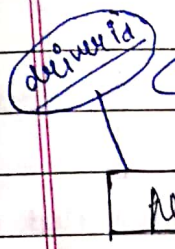
* ER diagram (Entity relationship diagram) to explain overall logical structure of database graphically

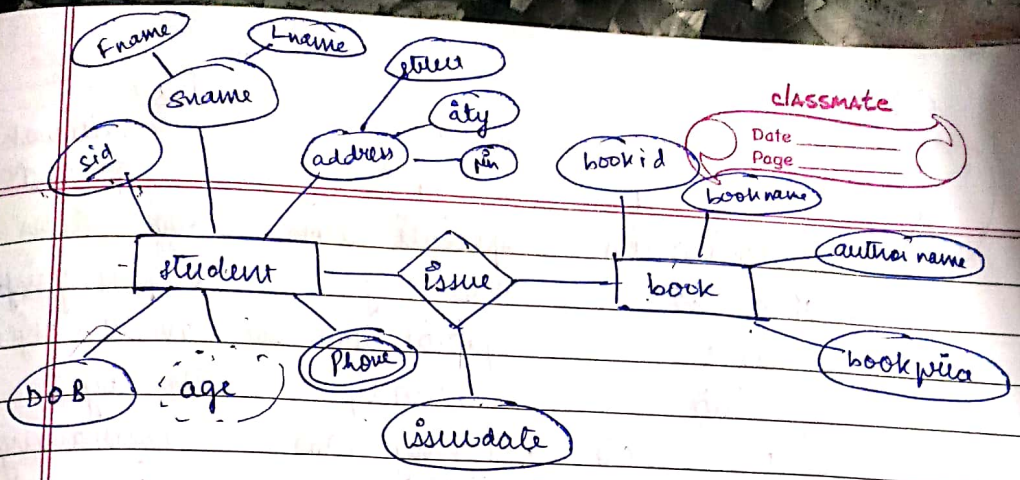
Major component
1. Rectangle
2. Ellipse
3. Diamonds
4. Lines :- Links
5. Double ellipse - Multivalued attribute
6. Dashed ellipse - derived attribute
7. Double lines - Total
8. Double rectangle - weak entity set.

Dashed lines → links attributes of a relationship set to a "

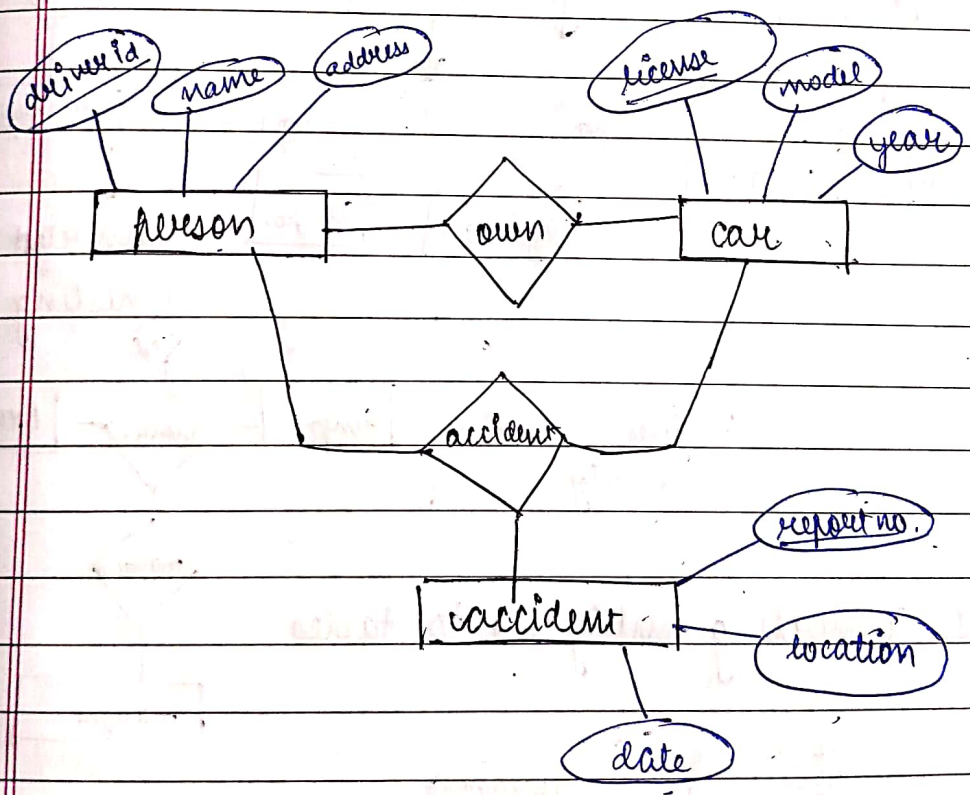
* ER diagram for issuing book from the library.







1. Construct ER diagram for a car insurance company whose customer owns 1 or more car. Each car associated with 0 to any no. of recorded accidents.



* Extended ER features

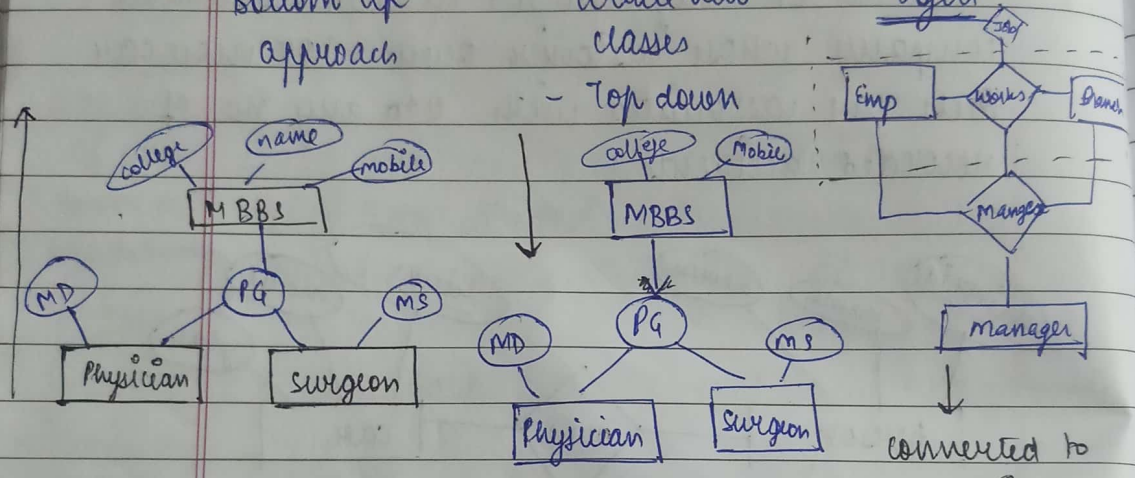
* Generalization
Abstracting view of object as single general class. Bottom up approach.

DMS

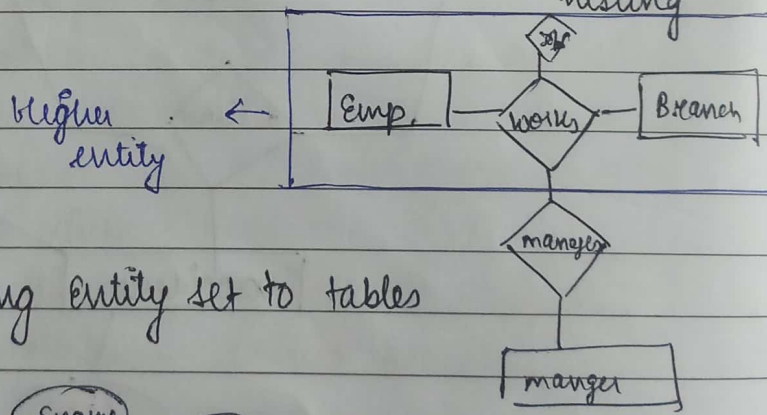
same ER but diff. process

classmate has a relationship or is part of

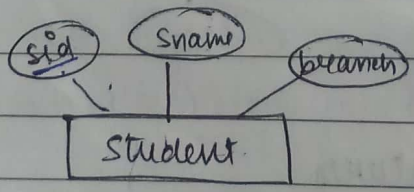
Generalization	specialization	Aggregation
- Abstracting process of viewing object as single general class - Bottom up approach	Adding of special characteristics to already existing class of object to create new classes - Top down	compiling info on an object, thereby abstracting higher level object



converted to nesting



* 1. Converting entity set to tables



Student

sid	Sname	branch

Scanned by CamScanner

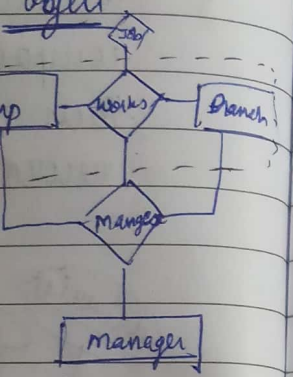
classmate

has a relationship

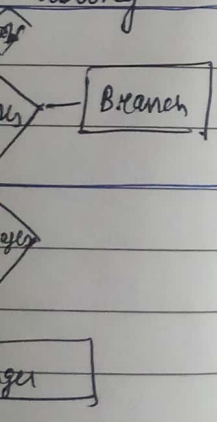
or is part of

Aggregation

Combining into an object, thereby abstracting higher level object



converted to

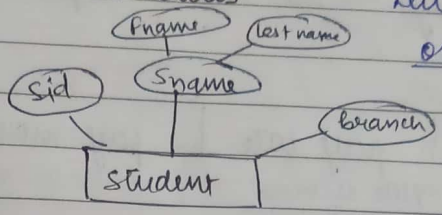


Partial ER

Subclass inherits attributes not only of its direct superclass but also of its predecessors

Superclass

2. Composite attributes

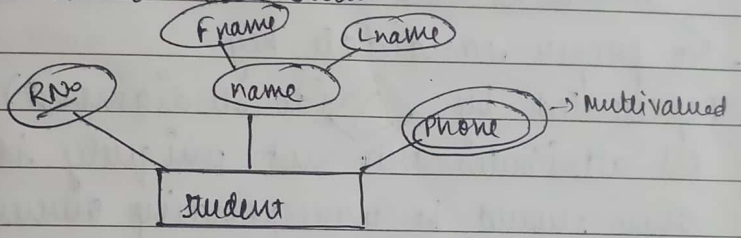


Name the composite one & the write its part.

student

Sid	Fname	Lname	Branch

3. Multivalued attributes



2 tables → otherwise redundancy

Student

R.No	Fname	Lname
1	RAM	Kr
1	RAM	Kr

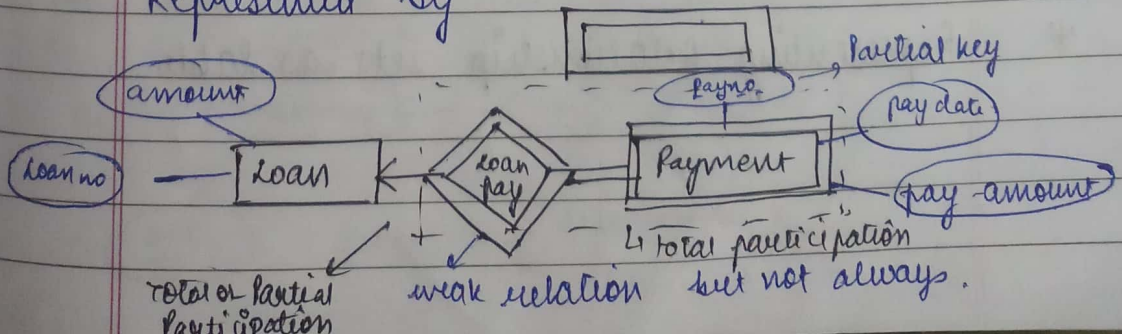
R.No.	Phone
1	9832
1	9999

} redundancy if we combine the two tables

* Weak entity set

No primary key only partial key

Represented by



Weak entity → the relationship → weak

payment

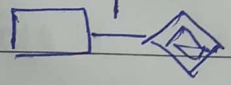
loanno.	payno	pay date	payamount
1	1		
1	2		
1	3		
2	1		
2	2		
2	3		

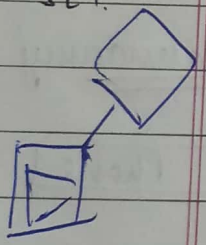
→ complete serves as primary key

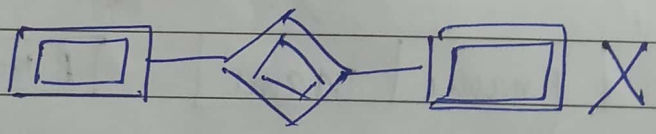
→ partial key

→ member is called as sub-ordinate entity

POINTS:- FOR WEAK ENTITY SET

- No proper candidate key
- Represented by  (double diamond)
- All attributes are not uniquely identified
- There should be corresponding owner entity set i.e 

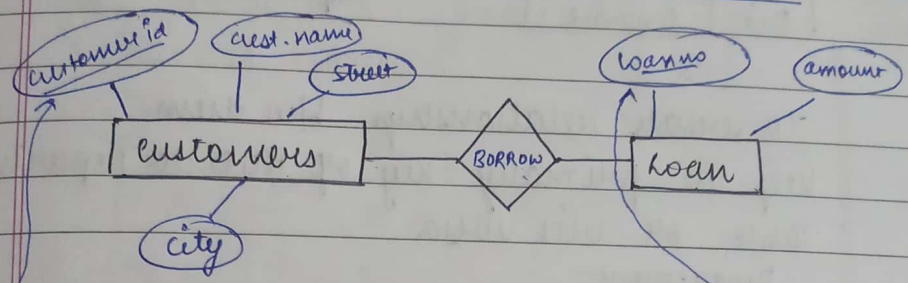




- Partici
- Total participation b/w weak entity set & weak owner
- cardinality → one to many
  many " "
- Database table should be formed

* Representing Relationship sets as table

1. Many to Many (single line with no arrow)

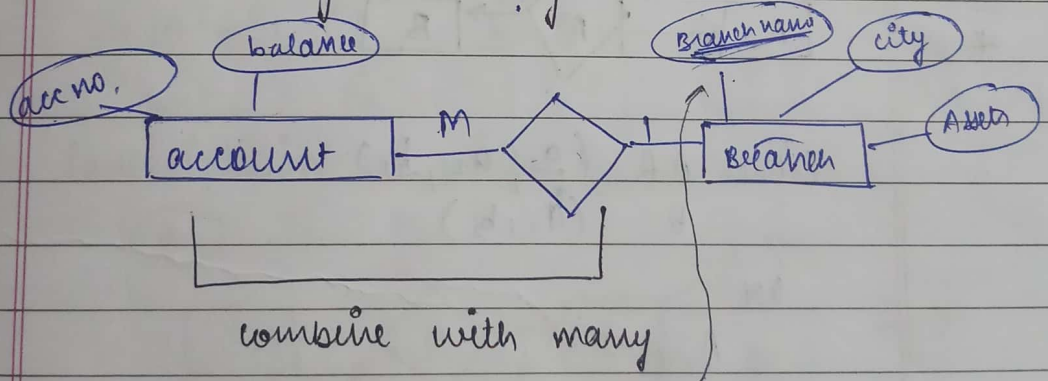


3 Tables ① customers
② Loan
③ Borrow (additional)

Borrow

cust id	loan no
FK	FK

2. One to many or many to one

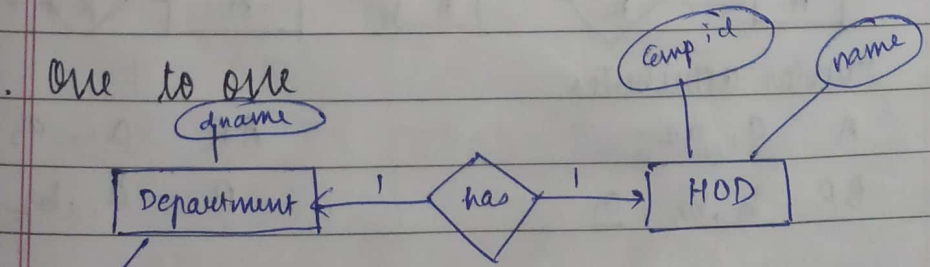


combine with many

account

acc no.	Balance	Branch name
		FK

3. One to one



Did

2 tables — department
HOD

Department

Did	Dname

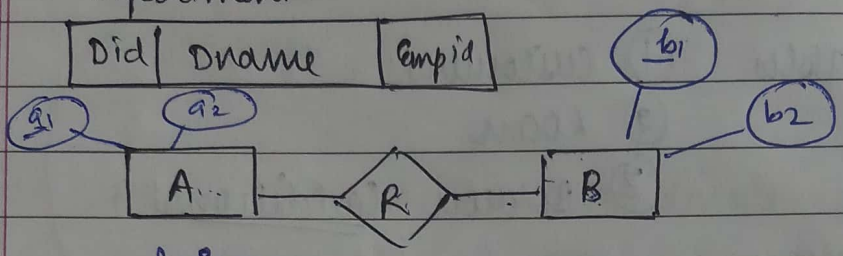
HOD

Empid	name

To create relationship b/co them keep the primary key of HOD to Department table or vice versa

Department-

Did	Dname	Empid

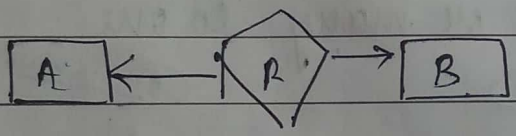


Primary key

A a_1

B b_1

R a_1, b_1



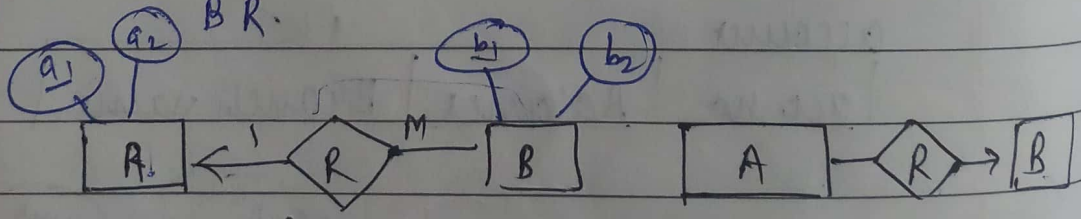
Either AR $(\underline{a_1}, a_2, \underline{b_1})$

B $(\underline{b_1}, b_2)$

Or

A

BR.



2 tables attributes

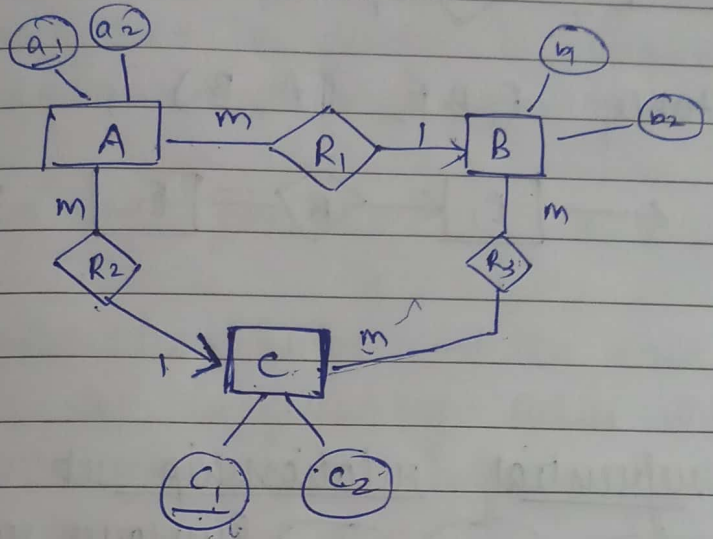
A $\underline{a_1}, a_2$

BR $b_1, b_2, \underline{a_1}$

AR $\underline{a_1}, a_2, \underline{b_1}$

B $\underline{b_1}, b_2$

* Find no. of tables that are possible when you translate the given ER diagram into relational model.



C (C1, C2)

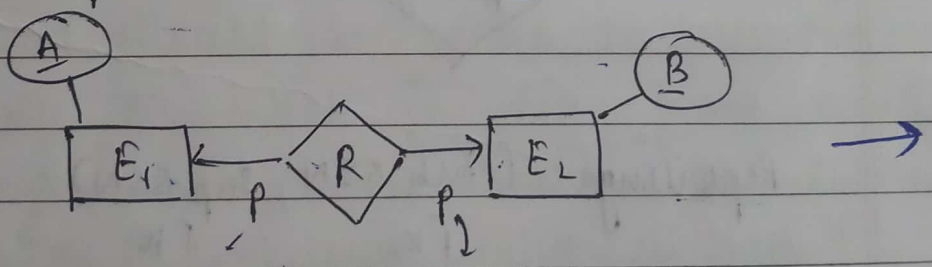
B (b1, b2)

R3 (b1, c1) FK

→ AR1R2 (a1, a2, b1, c1)

Pk

* Participation set with E.-constraints



Partial participation

3 tables

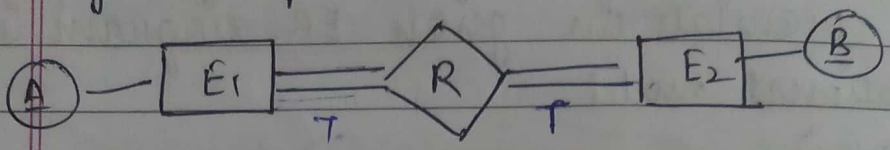
E1 (A)

R (A, B) → attributes

E2 (B)

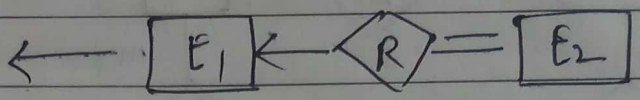
Special Cases

I If total participation



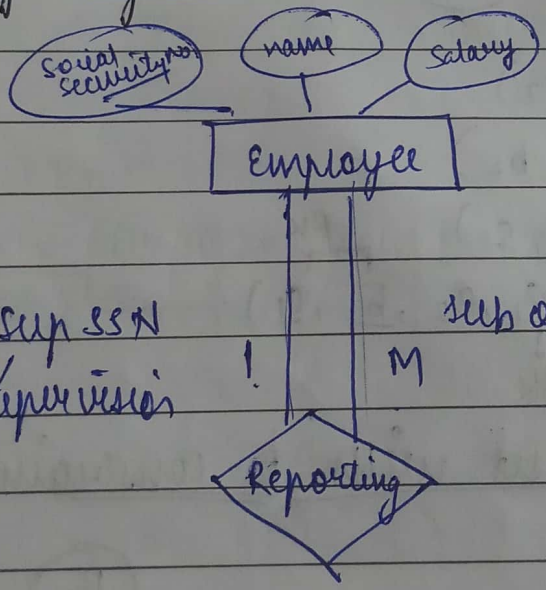
One table E_1RE_2 (A, B)

- E_1R or RE_2



II

* Self referential relationship set



Employee reporting will be done by employee

Reporting (Sub SSN, Sup SSN)
FK FK

UNIT-2

Relational Model VS Relational DBMS

- Relational model is concerned with what is required which separates it from how the model is implemented.
- Represent data in form of table.

CODD's rule for DBMS

1) Information Rule :- All the data must be presented in form of tables i.e. all the data represented in form of rows & columns.

2) Guranteed Accessible :- All data can be accessed without any ambiguity.
- Table name, Primary key & attribute name.

3) Systematic treatment of NULL values :-
A field should be allow to remain empty or NULL is diff. from empty string. "NULL value should treat the missing value or unknown value in table".

4) Dynamic Online catalog based on relational values :-
Active online catalog :- structure of database must be present or stored in online catalog which can be queried by authorized users.

database should be accessible by any language supported for definition, manipulation, transaction management operations

5) comprehensive data sublanguage
Relational data base must support atleast 1 language like DDL, DML, DCL
example:- Data definition language
- Data definition
DML → - Data manipulation
DCL → - Data control
- Data authorization
- View definition
- Integrity constraints
- Transaction control
} Features of language supporting database

6) View updating Rule Different views having changes must be updated at all levels.
Various purposes should be automatically updated by the system.

7) High level insert, Update & Delete.
selected data → operation performed eg:-

Relational model should support insert, delete, update at all the levels of relation. Also other operation union, ∩, minus

8) Physical and (9) logical data independence

changes at physical level without change in logical & external level	changes at logical level without change in external level/view level → app. program eg: Bank, IRCTC

9) Integrity independence
constraints that language uses, must be supported.
Integrity constraints at database level should not enforce modification at app level.

DBMS —atleast 7 rules supported→ RDBMS

11) Distribution independence ⇒ distribution of data at various places should not be known to end users.
User does not know about the database present at a place.

12) Non-subversion Rule
weakened data
Data base security must not be weakened. Bypassing the security features of database by user should not be true.

DBMS	RDBMS
- Doesn't support client - server architecture	support
- doesn't support distributed database	support
- In DBMS, no tight security of data.	Multiple levels of security such as logging at operating system level, command level, object level etc.
- Support less than 7 rules of CODD	- More than 7 rules are supported.
- Less speed of operation	- High speed of operation
- Less requirement for hardware & software	- More requirement.

* Excel never used for database but can be used for personal database

- Platform used in DBMS.	DOS, UNIX, VMS etc.
- Uses 3GL i.e 3rd Generation language	- Uses 4th generation language. i.e 4GL.
- Facilities & utilities offered are limited	- Are high.

CONSTRAINTS

1) Domain constraints
values associated with particular attribute means if char type data then that attribute will have all character type data.

2) key constraints
- Each table must have a key
- Value of key ≠ NULL

3) Entity integrity

If we have entities then relationship establish b/w them using keys.
(value of key ≠ NULL)

4) Referential integrity
In case of foreign keys.
- value of foreign key referencing other key in table should not be NULL.
- No. of tuples in foreign key & referenced key should be same.

5) Orphan
Customer (~~order ID~~, cust ID, ~~order data~~), order
Order - (~~order Id~~, ~~cust ID~~,
Customer (cust name, cust ID)
Order (order Id, cust ID, order date)
FK

5) Semantic integrity constraints
salary of employee should not exceed manager's salary.

6) Functional dependency constraint
Functional relationship b/w constraints of 2 attributes.

* Relational Algebra define set of operation on relation.

(i) Select operation :- select data given by
σ (condⁿ) (R)
↳ relation or table name

<, >, <=, >=, =, ! or <> or ≠

Connectors used in SQL = $\wedge$, $\vee$, $\neg$

$\sigma_{\theta_1}(\sigma_{\theta_2}(R)) = \sigma_{\theta_2}(\sigma_{\theta_1}(R))$ // commutative.

σ in relational algebra = where clause (condition)
(select)

example:-

eid	ename
1	A
2	B
3	C

$\sigma_{eid=1}$ (emp)

eid	ename
1	A

will be shown

example:- ID name deptname salary
table name = instructor

Query :- Retrieve the rows where deptname is Physics
$\sigma_{deptname = 'Physics'}$ (instructor)

(2) Find all instructors with salary > 9000
$\sigma_{salary > 9000}$ (instructor)

(3) Find all departments whose name is same as their building name.

$\sigma_{deptname = building}$ (instructor)

(ii) Project operation :- retrieve data from particular attribute
Represented by :- π

Syntax $\pi_{A_1, A_2, \ldots, A_N}(R)$

attribute from relation

Query: List al
donor

π ID

(2) Find
physi

π

(iii) Set
(i)
(a)
b

→ Rel
(2

→ Th
of

No. of
attribut
=

$\Pi_{i,d,name,\ldots}(I)$

query: List all instructor ID, name & salary but donot care about deptname.

$\Pi_{ID, name, salary}$ (instructor)

(2) Find the name of all instructors in the physics department

$\Pi_{name}(\sigma_{deptname = 'Physics'}$ (instructor))

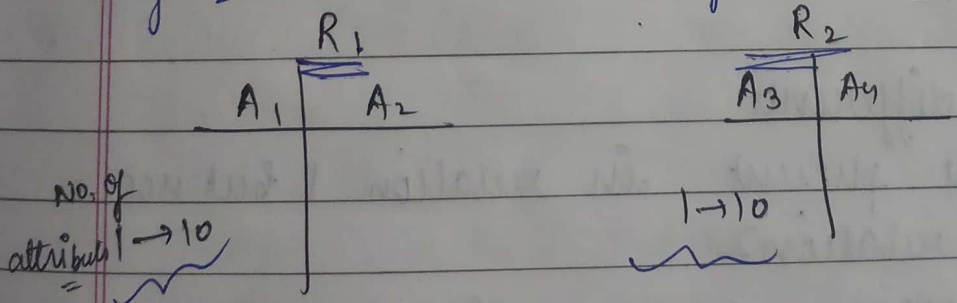
(iii) Set operations :-

(i) Union :-

(a) union compatible :- R_1, R_2 → two relation, both relation must have same no. of attributes, domain must also be same means i

→ Relation R_1 & R_2 must be of same arity (degree) (same attributes)

→ The domain of i^{th} attribute of R_1 & i^{th} attribute of R_2 must be same for all i.



R

ID	name	age
1.	A	20
2.	B	21

S

ID	name	age
3	C	23
4	D	24

R ∪ S (tuples from R, S, or both)

id	name	age
1	A	20
2	B	21
3	C	23
4	D	24

(duplicate tuple removed)

(2) show the name of all person who are in R or S

$\Pi_{name}(R) \cup \Pi_{name}(S)$

(iii) Intersection

only common attribute shown

R

id	name	age
1		
2		
3	C	23

S

id	name	age
3	C	23

shown:

id	name	age
3	C	23

(iv) Set difference

value present in relation 1 but not in relation 2

Student

id	name
1	A
2	B
3	C

P Student

id	name
1	A
2	B

attribute → Π

Q Name of all student which are not F student

$= \Pi_{name}(student) - \Pi_{name}(F student)$

Q Find all courses taught in fall 2009 sem but not in spring 2010

section

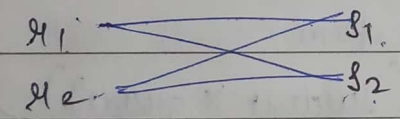
course id sem year

$= \Pi_{course\ id}(\sigma_{sem='Fall' \wedge year=2009}(section)) - \Pi_{course\ id}(\sigma_{sem='spring' \wedge year=2010}(section))$

(V) Cartesian product of 2 tables (X) ⇒ 2 tables & we give all possible combination of 2 tables. for joining of 2 tables

name find possible combinat

R S



$R \times S = r_1 \cdot s_1$

$r_1 \quad s_2$

$r_2 \quad s_1$

$r_2 \quad s_2$

R_1	R_2	$R_1 \times R_2$
x	y	x+y

No. of attributes

$R_1 \times R_2 =$ x+y no. of attributes.

tupples n_1 n_2 $n_1 \times n_2$ = no. of tupples.

R		S	
A	B	C	D
α	1	x	9
β	2	y	10

RXS

A	B	C	D
α	1	x	9
α	1	y	10
β	2	x	9
β	2	y	10

Query Find the names of all students having grade A

student	
name	RollNo
RAM	1
SHYAM	2
MOHAN	3

Grade	
Roll no	Grade
1	B
2	B
3	C

$\pi_{name}(\sigma_{grade='A'}(\underbrace{student \times grade}_{\text{from}}))$

(vi) Rename operation (ρ)

Rename attributes & tablename (Relation)

Find the name of student

$\rho_{studentgrade}(\pi_{name}(\sigma_{grade='A'}(student \times grade)))$

OR

ρ_{a_1/a_2} → new attribute name

→ old attribute name

difference b/w Join & cartesian product

Join
filtering property
then combine
less costly

cartesian product
all multiply then
condition applied.
More costly

* Types of joins

(a) Inner join
only matching
tuples

(b) outer join
all tuples

(a) Theta Join ($\bowtie_\theta$)
<, >, =, >=, <=,
≠

(b) Equi Join
special case of theta
= join with '=' sign

(c) Natural Join (∞)
Based on common
attribute in both
tables.

(a) Left outer join ($\bowtie$)
→ left relation tuple

(b) Right outer join ($\bowtie$)
→ right relation tuple

(c) Full outer join ($\bowtie$)
→ all tuple

Mobile

model	price
Nokia	10k
samsung	20k
iphone	50k

laptop

model	price
Dell	30k
Acer	20k
Lenovo	10k

$\bowtie_{(mobile.price < laptop.price)}$

1. Purchase both mobile & laptop but mobile price should be less than laptop price.

[filtering]

mobile $\bowtie_{(mobile.price < laptop.price)}$

2. mobile & laptop price should be same.

mobile $\bowtie_{(mobile.price = laptop.price)}$ ⇒ Equijoin

Natural Join

Forms cartesian product of two arguments performs selection & forcing equality on those attributes that appear in both relation schema & finally remove duplicate attribute.

Emp

Name	Emp Id	Dname
A	1	Finance
B	2	sales
C	3	sales
D	4	CE

Dept

Dname	Manager
Finance	M_1
sales	M_2
MKt	M_3

will not join in outer join

if written then no join

Emp ⋈ Dept

Name	EmpId	Dname	Manager
A	1	Finance	M_1
B	2	sales	M_2
C	3	sales	M_3

Date
Page

* Outer Join

Left outer join

(i) Emp ⟕ Dept = all tuples in left will come

o/p	Name	Eid	Deptname	Manager
	A	1	F	M_1
	B	2	S	M_2
	C	3	S	M_2
	D	4	CSE	NULL
⟕ ⇒	NULL	NULL	MKt	M_3

with A, B, C excluding D

i.e

A	1	F	M_1
B	2	S	M_2
C	3	S	M_2
NULL	NULL	MKt	M_3

⟖

A	1	F	M_1
B	2	S	M_2
C	3	S	M_2
D	4	CSE	NULL
NULL	NULL	MKt	M_3

* Division operator

$R_1(x) = R_1(z) \div R_2(y)$

Result contains all tuples appear in $R_1(x)$ in combination with $R_2(y)$ in

$x \cup y$

R_1

Course #	Name
10	B-tech
12	MBA
14	MCA

R_2

course #
10

$R_1 \div R_2$

Name
Btech

see same values in common attribute i.e course #

Find the names of student who failed in all the subjects.

Student

name	sub - failed	Subject
A	X	X
A	Y	Y
B	Z	
B	W	

Student ÷ subject

o/p ⇒ A as duplicate values → A

Π_{name} (student ÷ subject)

Generalized projection Π (used on Arithmetic operation)
- This operation is different
used on particular list

eg:- credit_info (customer_name, limit, credit)
Determine difference of limit & credit of each customer

$\Pi_{custname, limit - credit}$ (credit_info)

Aggregate functions $\mathcal{G}$

take collection of values eg: sum, min, max, avg, COUNT, COUNT DISTINCT (similar values eliminated)
Find out sum of salaries of all instructor

$\mathcal{G}_{sum(salary)}$ (instructor)

Find the total (sum) no. of instructor who teach a course in the spring 2010 sem.

$\mathcal{G}_{count\ DISTINCT(ID)}$ ((teaches) ∇ semester = spring $\wedge$ year = 2010 (instructor))

* Assignment operator

Find the names of all customer who have account in all branches of Delhi.

$r_1 \leftarrow \pi_{branchname}(\sigma_{branch\text{-}city = \text{"DELHI"}}(Branch))$

$r_2 \leftarrow \pi_{custname, branchname}(Depositor \times account)$

$r_2 \div r_1$ → customer corresponding to branch name in both will be the o/p.

Q Find the average salary in each department

dept. name $\mathcal{G}_{average(salary)}$ (instructor)

Q Find average salary all instructor

$\mathcal{G}_{average(salary)}$ (instructor)

Multiset Relational algebra

R_1		R_2
A	B	
1	a	2
2	a	3
		3

$\pi_B(R_1) = a, a$

$\pi_B(R_2) = a_2, a_3, a_3$

$\pi_B(R_1) \times R_2 = a_3, a_3$

* Features of SQL

1. High level language
2. Free format syntax
3. Authorization

decimal precision ↑ scale →

Number (p, s)

22.06

(4,2)

↙ 4 digit

varchar2 (size)

if size => 20 used 10 then 10 free

when var

char → if size = 20 fixes remaining 70 also if 10 used.

SQL command

1)	DDL	DML	DCL
-	Create	- Select	- revoke
-	Alter	- update	- grant
-	Drop	- insert	
		- delete	

* create table dept

(dept name varchar (20), primary key,

building varchar (15), UNIQUE, ← write directly here

budget numeric (12, 2),

primary key (dept name)); // to make primary key.

exactly same as in parent table

or - - -

create table instructor

(ID varchar (5),

name varchar (20) NOT NULL,

→ dept_name varchar (20), ← DML

salary numeric (8,2)

primary key (ID), foreign key (dept_name) references (dept));

- CHECK constraint ↳ eg:- either male or female.

CHECK (field name) in ('M' 'F')

eg:- CHECK ('budget) IN ('2010' & 2011')

changes

modify & alter table

① To add new column
② To add new integrity constraints
③ To modify existing columns.
④ → to expand length
⑤ → to decrease length but all values in the col. should be NULL.
⑥ to drop the integrity constraint

Create the table supplier | id number 3
name varchar 20
scode number 4

Q1 Add new columns pincode & city to table supplier
Q2 change the width of pincode to 5.
Q3 Add a primary key after table is created
Q4 Remove the " "
Q5 ADD not NULL constraint
Q6 Drop " " "
Q7 ADD constraint CHECK to deposit column.
Q8 To drop this check constraint.
Q9 To add primary key "
Q10 Drop " " "

ALTER TABLE supplier
ADD (pincode varchar (3)),
CITY varchar (5));

```
ALTER table supplier
MODIFY ( PINCODE varchar(5));
ADD PRIMARY KEY (ID);

DROP PRIMARY KEY;

MODIFY NAME NOT NULL;
MODIFY NAME NULL;

ADD CONSTRAINT CKDEP        name of constraint
CHECK (deposit BETWEEN 4000 and 5000);

DROP CONSTRAINT CKDEP;

ADD CONSTRAINT PID
PRIMARY KEY (ID);

DROP CONSTRAINT PID;


INSERT INTO CUSTOMER VALUES
(&cust_no, '&LNAME', '&FNAME');

SELECT FNAME || ' ' || LNAME FROM CUSTOMERS;
          concatenation
```

* To sort the data

```
SELECT LNAME, FNAME, PIN FROM CUSTOMER
ORDER BY STATE DESC;
        UPPER (STATE)
```

% percentage (as many character you want)
_ underscore (specific character " ")

* To retrieve all rows where customer no.

```
SELECT from Emp          check RAJ (3 character)
where FNAME LIKE 'RAJ%'
                  for one character
                  'RAJ_'
```

* Update command

```
UPDATE CUST SET PIN = "10001", WHERE ID = '2';
```

* View

```
CREATE VIEW EVIEW AS
SELECT empno, e-name, d-name
from emp, dept
WHERE emp deptno. = dept.deptno
```

"To hide complex query = use of view"

IMP:- Order By clause can't be used with 'create view' statement.
View can't be updated if it contain join, set operator, group fn, group by clause & distinct.

* Number function
* Aggregate "
* character "
* conversion "
* Date "

ABS absolute
SELECT ABS (200) FROM DUAL; || temporary table

COUNT
→ count (*) it counts no. of rows in a table including NULL
→ count DISTINCT → remove duplicates & tell total no.
→ COUNT → leave NULL & will tell.

INIT CAP
LENGTH
TO_CHAR (____)
To show only o/p ⇒ NVL fn.

SELECT ENO, BASIC, NVL(COMISSION, 500) FROM CAE;

Decode f^n is used to decode name
like DEL → Delhi

select city DECODE('CITY', 'UDP', 'UDUPI', 'BNG', 'BANGALORE'
'CITY NAME'

CITY	CITY NAME
UDP	UDUPI
BNG	BANGALORE

ADD MONTHS
TO DATE change in ddmm yyyy format.

* DELETE FROM CUSTOMER WHERE STATE='DEL';
COMMIT; // to save. if forget write ROLL BACK;
to save]
SAVE POINT D1
DELETE —
SAVE POINT D2
DELETE —
ROLL BACK TO D1;

SELECT EMP.NO, SUM(BASIC)
FROM SAL GROUP BY EMP.NO
ORDER BY SUM(Basic)

ID	ENO	BASIC	COMM	DEDUCT
	1000	2000		
	1002	1000		
	1003	3000		
	1004			
	1001	2000		

EMP No.	SUM
1001	4000
1002	2000
1002	2000
1001	4000

} to apply condition we use HAVING clause rather than WHERE clause

HAVING :- group
WHERE :- on a row

alias (another name)

```
SELECT S.EMP_NO, EMP_NAME, AVG(BASIC)
FROM SALARY S, EMP
(alias)
WHERE S.EMP = EMP.EMPNO
GROUP BY S.EMP, EMP.NAME
HAVING AVG (BASIC) = 5500
ORDER BY AVG (BASIC);
```

Nested queries, subqueries

must return a single column

- The result can contain column references without nor
- It return single row at time of [illegible] operator query
- We can't use between operator *
- We can use them with insert, delete update statement etc.

*

① Display employee who earn less than avg salary in organization.

```
SELECT * FROM SALARY WHERE
BASIC < (SELECT AVG (BASIC) FROM SALARY)
```

② List the name of employees who donot work in marketing department.

```
SELECT EMP_NO, ENAME FROM EMP WHERE DEPT NOT IN
(SELECT EMP.NO. FROM SALARY WHERE DEPT = 'MKT')
```

③ Select name of employees where deduction is < than 500.

```
SELECT EMP.NO, FROM EMP
(SELECT EMP.NO FROM SALARY WHERE DEDUCTION
                                < 500)
```

④ Display name of employees

```
SELECT EMP.NO, E.NAME, FROM EMP WHERE DEPT NOT IN
SELECT EMP_NAME FROM EMP WHERE EXISTS
(SELECT * FROM SALARY WHERE EMP.EMP.NO = SALARY.
                                EMP.NO
        AND DEPARTMENT = 'CSE';
```

Reports

Write set of program to generate salary report in SQL plus.

```
SET FEEDBACK OFF
Edit SALARY REPORT
SET PAGESIZE 30
SET LINESIZE 75
TTITLE CENTER 'SALARY REPORT'
BTITLE 'END OF SALARY REPORT'
COLUMN EMP_NAME FORMAT A10 TRUNC
COLUMN "NET SAL" FORMAT 9,999.00
SELECT S.EMP_NO, EMP_Name, BASIC, COMMISSION,
DEDUCT, BASIC + COMMISSION - DEDUCTION "NETSAL"
FROM SALARY S, EMP
WHERE S.EMP_NO = EMP, EMP_NO;
TTITLE OFF
BTITLE OFF
CLEAR COLUMNS
```

By default left aligned

Notepad

top

Bottom

By default center aligned

format column

compulsory

package name

SALARY REPORT

EMPNO	EMPNAME	BASIC	COMMISSION	DEDUCT	NETSAL

END OF SALARY REPORT

2 rows selected

To run → SQL> START SALARYREPORT <filename>

PL/SQL superset of SQL

* Anonymous block Name block

- does not have anyname
- executed only once.

Name block: subprograms → Triggers, procedures, functions

↓

DECLARE
variables, cursors, exceptions (user defined), functions

BEGIN
SQL & PL/SQL statement
EXCEPTION
trap the errors

END; (compulsory)

Anonymous to add to numbers

```
set serveroutput on; // to see output
DECLARE
i integer := 10;
j integer := 20;
c integer;
BEGIN
   c := i+j;
   DBMS_OUTPUT.PUT_LINE(c);
END;
/ (To run the PL/SQL code.
```

package name ← DBMS_OUTPUT

PUT_LINE → procedure name

if named file, to start
SQL> ED filename
start filename.

Q Write a PL/SQL code to retrieve employee name, join basic & designation of employee whose no. is i/p by user.

Q write a PL/SQL code to run a for loop in reverse.

```
set serveroutput on;

BEGIN
DBMS_OUTPUT.PUT_LINE('For loop...');

FOR I IN REVERSE 1...10 LOOP
DBMS_OUTPUT.PUT_LINE('LOOP INDEX'||I);
END LOOP;
END;
/ (to run)
SET SERVER OUTPUT OFF;
```

if straight then
FOR I 1..10 LOOP

* If-else

```
IF (condition)
 THEN
ELSE IF

END IF
```

* case expression

CASE exp WHEN exp1 THEN statement
WHEN exp2 THEN "
:
ELSE DEFAULT

END.

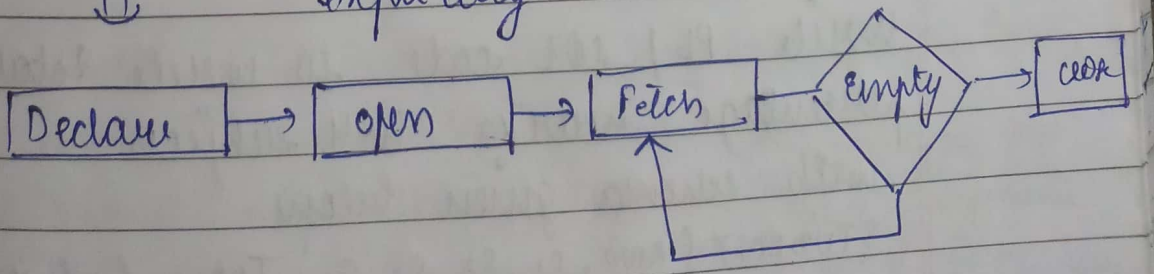
* Cursor

Context area → to gain more control over this " we write cursor
It is a pointer to context area.

All info needed to process all SQL statement using cursor

2 types of cursor

- implicit automatically created
- explicit written by user & declared explicitly



* 4 to 5 attributes

① % found i.e % FOUND
it returns true if row is returned

② % not found % NOT FOUND when table empty

③ To check whether cursor is open or not
% ISOPEN

④ % ROWCOUNT = 0 (when cursor opened)
otherwise return no. of rows in it.

Record ← type variable where all attributes in table will be made int it.

Uses

syntax
DECLARE
CURSOR < cursorname >
IS < SELECT statement >
(To open)
OPEN < cursorname >
(To store data in cursor)
FETCH < cursorname > INTO VAR1, VAR2
or
FETCH < cursorname > INTO < RECORD name >.
CLOSE < cursor name >.

Write PL/SQL code to write total percentage marks in 4 subjects with schema given below
STUDENT (RNO, S1, S2, S3, S4, Total, Percentage)

```
DECLARE
 T NUMBER;
 PER NUMBER;
 CURSOR C1          ||selected & value stored)
 IS SELECT* FROM STUDENT;
 REC C1     %ROWTYPE; || C1 will have variable similar to row in record of student
BEGIN
 OPEN C1;
 LOOP
  FETCH C1 INTO REC;
  EXIT WHEN C1 %NOTFOUND;
  T:= Rec.S1 + Rec.S2 + Rec.S3 + REC.S4;
  PER:= T/4;
 UPDATE STUDENT SET TOTAL=T, PERCENTAGE
 =PER .WHERE RNO = REC.RNO.
END LOOP;
CLOSE C1;
END;
```

Record type variable where all records attribute in table will be made int it.

(*)

```
DECLARE

CURSOR C1 IS SELECT emp no, emp name from Emp;
V_empno  Emp.eno %Type;
V_ename  Emp.ename %Type;

FETCH C1 INTO V_eno, V_ename;
EXIT WHEN C1 %NOTFOUND;
```

जो data type Emp table में empname की है वो इस table name में दी data type बनजाए.

```
DBMS_OUTPUT.PUT_LINE (V_eno || ' ' || V.enam);

END LOOP;
CLOSE C1;
END;
```

* Triggers
define particular action.

(*) They are required to enforce rules that cannot be coded through referential integrity.

(*) To enforce certain business rules, validation in an app. to maintain audit changes made to date, to derive column values.

Syntax

```
CREATE [OR REPLACE] trigger <trigger name>
[Before/ after]
[DELETE/ insert/ update ] [of column]

[on USER].table
[for each row]
[when condition]
[PL/SQL statement]
statement level trigger.
```

- (1) they are associated with the table & are automatically fired when user user DML like insert, update, delete on the table (2) there can be many triggers for one table
(3) there can be only one trigger of a particular type i.e update, insert, delete.
(4) only one table can be specified in triggering statement.
(5) Trigger code can be typed at SQL code, or any editor like notepad.
inside the trigger the correlation name new, old can be used
to refer to data on command line & to data in it respectively.
(6) Trigger does include commit, ROLLBACK, SAVEPOINT.

Write a code to create a trigger so that no operation can be performed on employee table on tuesday.

```
CREATE Trigger EM_TUE
BEFORE INSERT OR UPDATE OR DELETE ON EMP
BEGIN
IF RTRIM (UPPER (TO_CHAR (SYSDATE, 'DAY'))
                = 'TUESDAY' THEN
RAISE_APPLICATION_ERROR (-20101, 'Cannot use table on tuesday!');
END IF;
END;
```

error no. ↓ app. statement ↓

START EM_TUE

DROP Trigger

ALTER TABLE EMP DISABLE ALL TRIGGER;
EM_TUE

" " " EM_TUE ENABLE
EM_TUE DISABLE

* PL-SQL procedures → must return a value
subprogram ← Procedure & f^n. diff. → subprogram coded to
to perform a particular named block perform specific task compute values.
action;
Parameter list
IN passing parameter in
OUT " " out
INOUT " " in then out.

they don't return value

→ Write a SQL procedure which will update the salary whose emp.no is passed as parameter.

CREATE [OR REPLACE] PROCEDURE
raise sal (E.NO. emp.empno %Type)
Is
BEGIN
UPDATE EMP
Set Sal = sal+1000; WHERE
emp_no = eno;
END raise_sal;
raise sal (10);

(consider variable sal)

DROP PROCEDURE < Procedure name >

* f^n returns a value.

write a PL-SQL function to return square of a given no.

```
CREATE [OR REPLACE] FUNCTION SQR(N NUMBER)
RETURN NUMBER
IS                    datatype
BEGIN
RETURN N*N;
END;
/
set serveroutput ON;
BEGIN
DBMS_OUTPUT.PUTLINE ('5 sq is :' SQR(5));
END;
/
Set serveroutput off;
```

returns datatype return value

→ Returns f^n using RETURN keyword.

* PL-SQL f^n → from table. IN, OUT X

```
CREATE [OR REPLACE] FUNCTION find sal (eno
                                      emp.empno
                                      %Type)
RETURN NUMBER
IS
V_sal emp.sal %Type
BEGIN
SELECT sal into V_sal from emp
Where emp_no = eno;
Return V_sal;
END;
```

(consider variable as)

* Functional dependency

It is an association b/w 2 attributes of the same table.

A	B
1	4
1	5
2	8
3	9

not unique
∴ no functional dependency exist.

Q Consider a relation which of following FD is incorrect?

A	B	C
1	2	3
4	2	3
5	3	3

(a) $A \rightarrow B$ correct

(b) $B \rightarrow C$ correct

(c) $BC \rightarrow A$ incorrect

$2,3 \rightarrow 1$

$2,3 \rightarrow 4$

$3,3 \rightarrow 5$

(d) $AC \rightarrow B$ correct.

classification

1. trivial functional dependency

functional dependency $X \to Y$ is trivial if & only if Y is a subset of X. ($Y \subseteq X$)

2. Non-trivial functional dependency

If there is atleast one attribute in RHS that is not part of LHS.

$AB \to BC$ one attribute C which is not part of LHS

3. Full functional dependency

An attribute Y is fully functional dependent if it is functional dependence on X but not on proper subset of X.

$X \to Y$

fully dependent on X (not on a part)

$AB \to C$
$A \not\to C$
$B \not\to C$

then not fully functionally dependent.

* 4. Partial dependency

Given relation R with functional dependency F defined on attributes of R & k is the candidate key. If x is a proper subset of k and if $F \not\Rightarrow x \Rightarrow A$ then A is said to be partial dependent on A.

R(ABCD)

(candidate key) C.k = AB

$A \rightarrow C$ (C depends on A & it is part of CK)

∴ it is partial..

5. Transitive dependency

If $X \rightarrow A$, $A \rightarrow Y$

$\Rightarrow X \rightarrow Y$

Y is transitively dependent on X.

6. Candidate functional dependency

It is a " " that includes all attributes of table.

Dependency diagram (DD) has atleast 1 candidate functional dependency.

*

If on table has only 1 candidate key & that will be primary key & it is

Primary functional dependency.

(key attributes) ⟶ (Non-key attributes)

- transitive FD

(Non-key attribute) ⟶ (Non-key attribute)

- Boyce Codd FD

(Non-key attributes) determine (key attribute)

- Multivalued dependency (MVD)

It occurs

B/w two or more multivalued fact about same attribute occur b/w same table

B, C

A ⟶ B

A ⟶ C

* Join dependency (JD) - exist if

If a relation R is equal to the join of projection of

X, Y, Z where X, Y, Z are " " R

* Inference Rules / Closure properties / Armstrong's Axioms

(i) Reflexivity Rule

if $X \supseteq Y$ (Y is subset of X)
then $X \rightarrow Y$

for eg:

$AB \supseteq A$

A is subset of AB then
$AB \rightarrow A$

Imp.
(ii) Augmentation :-

if $X \rightarrow Y$ then
$XZ \rightarrow YZ$ augmenting attribute Z on both sides of FD

Imp (iii) Transitivity :-

if $X \rightarrow Y, Y \rightarrow Z$ then
$X \rightarrow Z$

Imp
(iv) Union

if $X \rightarrow Y$ and $X \rightarrow Z$
then $X \rightarrow YZ$

(v) Decomposition

if $X \rightarrow YZ$
then $X \rightarrow Y$
$X \rightarrow Z$

(vi) Pseudo transitivity

If you have $A \rightarrow B$

$BC \rightarrow D$

then $AC \rightarrow D$

closure of a set of FD's (F+)

it is a set of all FD's that can be determined using given set of FD's F. (that are logically implied FD)

Total FD's in f closure

F+ for R(AB) $= 2^{2n} = 2^{2\times2} = 16$

attributes

n = no. of attributes.

X	Y	
ϕ	ϕ	$\phi \rightarrow \phi$,
A	A	$\phi \rightarrow A$
B	B	$\phi \rightarrow B$
AB	AB	$\phi \rightarrow AB$
		$A \rightarrow \phi$
		$A \rightarrow B$
		$A \rightarrow AB$

imp:

F+ of

* R(ABCDE)

FD:- {AB → C, CD → E, DE → B}

(a) Is AB a candidate key? If not if is ABD explain?

$AB^+ = ABC$ ✗ as attributes not covered ∴ Not C.K (ABCDE)

$ABD^+ = ABDCE$ ∴ it is a candidate key

↳ → C ↳ CD → E

* Closure of attributes X^+
Set of all attributes that can be determined using a given set X

eg: $A \longrightarrow B$
$B \longrightarrow C$

Closure of A: $A^+ = ABC$
$B^+ = BC$
A से जाने जा सकता है

Applications of attribute closure

(i) Finding keys of a relation
If attribute closure contains all attribute of a relation then it is called superkey of R.

X^+ x → superkey of relation R
attributes

Ques You are given with relation R(ABCDE) and Functional dependencies set as { $A \to BC$, $CD \to E$, $B \to D$, $E \to A$ }
Find candidate key of R.

$A^+ \longrightarrow ABCDE$ (candidate key)
$A \to BC$, $B \to D$, $CD \to E$

$B^+ \longrightarrow BD$ (not " ")
$C^+ \longrightarrow C$ (not " ")
$D^+ \longrightarrow D$ (not " ")
$E^+ \longrightarrow EABCDE$ (candidate key)

as both A^+ & E^+ both are candidate key :. we will not combine them means don't check A & E combinations

$BC^+ = BCDEA$ (candidate key) } No need to make their further combination
$CD^+ = CDEAB$ (" ") }
$BD^+ = BD$ (not " ")

Now triple combination : but as no triple pair can be made

∴ candidate key = {A, E, BC, CD}

* (ii) To find additional functional dependencies.

Q R(ABCD)
FD's { A → BC , B → CD , D → AB }
Find whether (AD → C) ? additional functional dependency

$AD^+ = ADBC$
↳ A → BC

We need to check here whether the attribute on RHS i.e C is covered in closure of AD or not.
∴ as here covered & hence it is additional FD.

Q R (ABCDE)
FD's { A → B, A → C, CD → E, B → D & E → A }

Q Which of the following FD is not implied by the above set ?

(i) $CD \rightarrow AC$

(ii) $BD \rightarrow CD$

(iii) $BC \rightarrow CD$

(iv) $AC \rightarrow BC$

Soln $(CD)^+ = CDEAB$ it includes AC ∴ it is correct

$BD^+ = BD$ Not implied

$BC^+ = BCDEA$ it is correct

$AC^+ = ACBDE$ " " "

* How to check equivalent FD:-

Q $F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\}$ check if F & G are equivalent. $G = \{A \rightarrow DC, E \rightarrow A\}$

Soln

F: $A \rightarrow C$

$AC \rightarrow D$

$E \rightarrow AD$

$E \rightarrow H$

G: $A \rightarrow DC$

$E \rightarrow A$

$A^+ = DC$

$E^+ = EADCX$

$AC^+ = ACD$

$A^+ = ACD$

$E^+ = EADHC$

These FD's are equivalent.

Q $F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\}$

$G = \{A \rightarrow DC, E \rightarrow AH\}$

F: $AB^+ = ABCD$

$B^+ = BX$

$C^+ = CD$

G: $AB^+ = ABCD$

$C^+ = CD$

$G \subseteq F$ but $F \subseteq G$ so they are not equivalent
$F \supset G$ as $F \not\subseteq G$

Consider 2 sets F & G with FD's as below:-

$F = \{ A \to B,\ AB \to C,\ D \to ACE \}$
$G = \{ A \to BC,\ D \to AE \}$

Soln:- F: $A^+ = \underline{ABC}$
$AB^+ = ABC$
$D^+ = \underline{DAEBC}$

G: $A^+ = \underline{ABC}$
$D^+ = \underline{DA}\underline{CE}B$

∴ F & G are equivalent.

Q:- F: $A \to B$
$B \to C$
$AB \to D$

G: $A \to B$
$B \to C$
$A \to C$
$A \to D$

F: $A^+ = \underline{ABCD}$
$B^+ = \underline{BC}$
$AB^+ = \underline{ABCD}$

G: $A^+ = \underline{ABCD}$
$B^+ = \underline{BC}$

They are equivalent.

Page

Q F: $A \to B$
$B \to C$
$AB \to D$

G: $A \to B$
$B \to C$
$A \to C$
$A \to D$

Q F: $A \to B$
$B \to C$
$A \to C$

G: $A \to B$
$B \to C$
$A \to D$

* Minimal set of FD / canonical cover / standard set of FD.

A set of FD is minimal if it satisfies following condition :-

(i) Every dependency in F has a single attribute for RHS

(ii) No dependency $X \to A$ in F can be replaced by dependency $Y \to A$ where Y is a subset of X ($Y \subset X$) & still have a set of dependencies that is equivalent to F.

(iii) We cannot remove any dependency from F but still can have set of dependencies that is equivalent to F.

NOTE: Used for removing redundant FD

1. Given set F as $A \rightarrow B$, $B \rightarrow C$, $AB \rightarrow C$

if $A \rightarrow C$ eliminated whether it is equivalent or not. or to tell whether $A \rightarrow C$ is redundant or not.

* Extraneous attribute

It is one which can be removed ~~to~~ it without changing closure of set of ~~attribute~~ FD.

for eg: $AB \rightarrow C$
$A \rightarrow C$

Here, B is extraneous attribute

* F: $\{AB \rightarrow C, A \rightarrow B\}$ G: $\{A \rightarrow C, A \rightarrow B\}$

find extraneous attribute

$(AB)^+ = ABC$
$A^+ = ABC$

$A^+ = ACB$

"B is extraneous attribute"

as $AB^+ = ABC$
& $A^+ = ABC$

∴ B is extra in AB^+

Q Find minimal set of given set of FD.

$A \rightarrow BC$

$B \rightarrow C$

using inference rules.

(i) using decomposition

$A \rightarrow B$

$A \rightarrow C$

$B \rightarrow C$

$A \rightarrow B$

$B \rightarrow C$

$A \rightarrow C$

$\{ A \rightarrow B$

$B \rightarrow C \}$ (transitivity)

(ii) $AB \rightarrow CD$

$BC \rightarrow D$

TUT Q Let R be a relation

$R = ABCDEF$ having

FD's $F = \{ A \rightarrow BC$

$B \rightarrow E$

$C \rightarrow EF$

$E \rightarrow CF \}$

Find closure of AB over given set of FD's.

Q2 Let R(ABCDEFG) & FD's (ii)
Find out candidate keys of R

A → B
BC → D
BC → E
AEF → G
B → G

(iii)
Q FD: A → BC R(ABCDE)
CD → E
B → D
E → A

Q1 → AB^+ = ABCEE

Q2 (i) R = ABCDEFG

A^+ = ABG
B^+ = BG
C^+ = C
D^+ = D
E^+ = E
F^+ = F
G^+ = G

AB^+ = ABG
BC^+ = BCDEG
CD^+ = CD
DE^+ = DE
EF^+ = EF
FG^+ = FG
AC^+ = ABCGDE
AD^+ = ADBG
AE^+ =

As no closure includes all the attributes
∴ no candidate key.

Adv. of Minimal set

(i) It eliminates redundancy present in any determinant (LHS)

(iii) It eliminates any extraneous FD

(iii) $A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

$A^+ = ABCDE$ ✓

$B^+ = BD$

$C^+ = C$

$D^+ = D$

$E^+ = EABCDE$ ✓

$BC^+ = BCDEA$ ✓

$BD^+ = BD$

~~$BE^+ = BED$~~

$CD^+ = CDEABC$ ✓

$\{A^+, E^+, BC^+, CD^+\}$

Q To prove :-

$AB \rightarrow CD$, $BC \rightarrow D = AB \rightarrow C$, $BC \rightarrow D$

$AB \rightarrow C$

$AB \rightarrow D$, $BC \rightarrow D = AB \rightarrow C$, $BC \rightarrow D$

*

Q Find the minimal cover
$A \to BC$, $B \to C$, $C \to B$ using Armstrong Axioms.

1) First of all, single element on RHS
2) check for extraneous attribute → remove
3) there should not be redundant functionalities.

$A \to BC$
$B \to C$
$C \to B$

$\Rightarrow \{ A \to B$ (decomposition)
$A \to C$
$B \to C$
$C \to B \}$

$\Rightarrow$ Applying transitivity

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

$= \{ A \to C, A \to B, B \to C, C \to B \}$ (transitivity)
(removed)

or
$= \{ A \to B, B \to C, C \to B \}$

(ii) $ABCD \to E$
$E \to D$
$A \to B$
$AC \to D$

(1) ✓ Single attribute on RHS
(2) Extraneous attribute on LHS
(3) Armstrong rules
Remove
→ B extraneous attribute.
$ACD \to E$
$E \to D$
$AC \to D$
$A \to B$
i.e B

$AC \to E$, $E \to D$, $A \to B$, $AC \to D$ ✗

$\{AC \to E, E \to D, A \to B\}$ (transitivity)
minimal set or canonical cover.

* $A \to BC$, $B \to C$, $A \to B$, $AB \to C$

$[A \to B, A \to C, B \to C, A \to B, AB \to C\}$
remove it.

$\{A \to B$
$A \to C$
$AB \to C\}$

→ B is extra

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

Q $A \to BC$, $B \to AC$, $C \to AB$

$\{A \to B$ $B \to A$ $C \to A$ (decomposition)
$A \to C$ $B \to C$ $C \to B\}$

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

$\{A \to B, A \to C, B \to AC, C \to AB\}$
$\{A \to B, B \to A, B \to C, C \to AB\}$
→ $C \to A$
$C \to B$

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

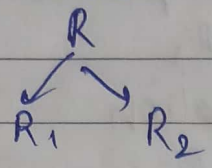
Lossy decompo

classmate
Date
Page

* **Decomposition**

when we normalize

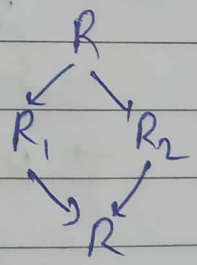
Divide a table into different tables



$R_1 \cup R_2 = R$

No extra tuple generated, no info loss. while joining decomposed relation

A relation R can be decomposed into a collection of relation schemas to eliminate anomalies in original relation R.
- updation
- insertion



Properties

(i) Lossless join decomposition

Let R is a relation & has a set of FD's F over R. The decomposition of R into R_1 & R_2 is loss less wrt to F

$$R_1 \bowtie R_2 = R$$

Natural Join.

(ii) Lossy decomposition :-

when extra tuple generated.

(ii) Attribute preservation
All attributes of original schema R must appear in decomposition
$R = R_1 \cup R_2$

(iii) Dependency preservation

for eg :-

Enrol (sno, cno, date_enrolled, room_no, instructor)
(implies)

enroll 1 (sno, cno, date_enrolled)
Enrol 2 (date_enrolled, room_no, instructor)

FD is preserved in either R_1 or R_2 by using rules.

Q Suppose we decompose schema R(ABCDE) into $R_1(ABC)$, $R_2(ADE)$
$\{A \to BC, CD \to E, B \to D, E \to A\}$

→ $R_1 \cap R_2 = A$ is candidate key ∴ Lossless decomposition

∴ Lossless decomposition

Q R(ABCD)
set of FD's $\{A \to B, C \to D\}$
To show whether decomposition → lossless & lossy
" " is preserved or not

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

⇒ $R_1 \cap R_2 = \phi$ ⇒ Lossy decomposition.

as $A \to B$ is preserved in $R(A, B)$
$C \to D$ " " " $R(C, D)$
∴ preserved.

Q $R(ABCDE)$

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

(1) Determine whether the following decomposition R is lossless or not. (2) Is dependency preserved

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

$R_1 \cap R_2 = A$ (is candidate key)

∴ lossless decomposition

$A^+ = ACDEB$

∴ A is superkey

if FD RHS all covered then it is a superkey

on RHS of FD's

Decomposition is lossless.

If all attributes of either decomposed relation is covered then also

Closure of $R_1 \cap R_2 = R_1$ or R_2

→ FD's are not covered in both $R_1(A, B, C)$ & $R_2(A, D, E)$ ∴ not preserved. like $B \to D$ not in any of it.

→ consider relation schema $R(U, V, X, Y, Z)$

FD's are $Z \to U$, $V \to Y$, $XY \to Z$, $U \to VX$

whether following decomposition of R

$R_1 = U, V, X$

$R_2 = U, Y, Z$

& dependency preserved or not?

$R_1 \cap R_2 = U$

closure of U i.e $U^+ = UVXYZ$

↓

candidate key ∴ lossless decomposition

→ Not preserved dependency

$V \rightarrow Y$ ✗

Q R(ABCD)

FD: {A → B, A → C, C → D}

$R_1 \rightarrow ABC$

$R_2 \rightarrow CD$

Whether lossless or lossy

$R_1 \cap R_2 = C$

$C^+ = CD$ ~~(not cat~~

$C^+ = CD = R_2$ ∴ lossless

I Method

	A	B	C	D	(C → D)
R_1	✗	✗	✗	✗	
R_2			✗	✗	

all attributes

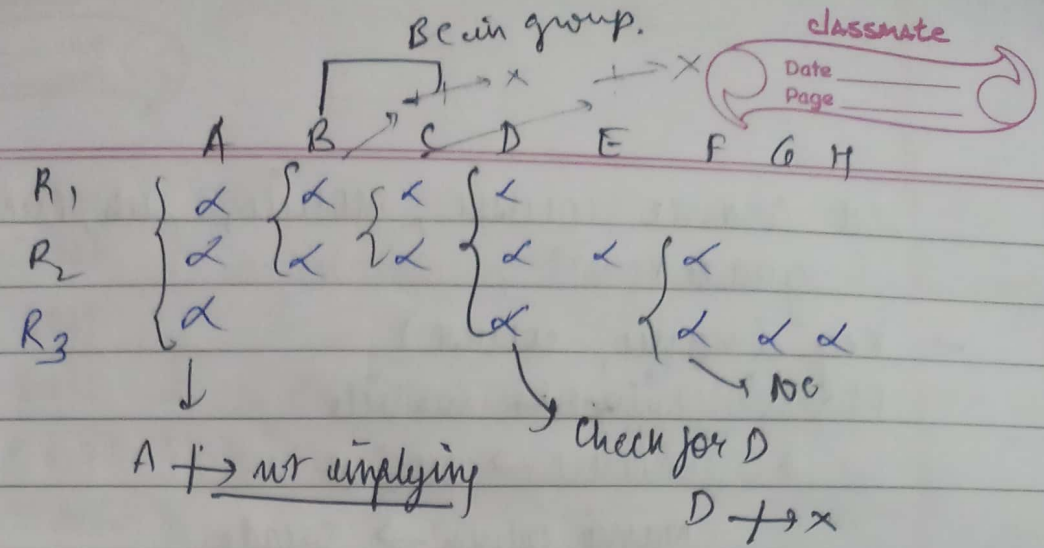
* Schema R (ABCDEFGH)

FD's = {AB → C
BC → D
E → F
G → F
H → A
FG → H}

$R_1 = A, B, C, D$

$R_2 = ABCDEF$

$R_3 = ADFGH$



The one which is present in ~~atleast~~ more relation → check check to which is implies in given FD

3 ∝ then 2 ∝

∴ it is lossy.

→ If any row ~~has~~ is completely filled then lossless decomposition.

IMP

* NORMALIZATION

- Removing redundancy from the tables. It is set of rules that.

⇒ It is a process of analyzing given relational schema based on their FD's and primary keys to achieve desired properties of minimizing ~~redundancies~~ & hence " the insertion, deletion & update anomalies.

std (name, course, mobile, sub, prof, grade)

→ Key (name, course)

FD's : Name → mobile
: Name → subject
Name course → grade
course → prof.

same mobile no. ∴ redundancies

A — dbms — 99957 - —
A — java — 999957 - —

Update anomaly :- 5 times mobile no. updated if 5 courses taken by A

Insertion " :- Student can't insert prof name until he has registered name, course, mobile.

Deletion anomaly :-

NORMAL FORM

It is used to eliminate or reduce redundancy in relation or set of rules to perform decomposition.

decompose into two or more tables

Types of Normal form

1NF
2NF
3NF
BCNF
4NF
5NF

* First
A re
if the
attr

Eno
1
2

E
1

NOTE:

1NF
2NF } dependent on single value dependency
3NF }
BCNF } ⇒ 0% redeendancy due to " " " But MVD may exist
4NF } " " multivalued " exist
5NF } " " Join "

* First Normal form (1NF)
A relation is said to be in 1NF if the values in the domain of each attribute of relation are atomic.

Eno.	Ename	Contact
1	A	99--, 98---
2	B	199--, 100---

⇓

convert to 1NF

Eno	Ename	Contact
1	A	99--
1	A	98--
2	B	199--
2	B	100--

values are atomic

NOTE:- Highest redundancy in 1NF

multivalued
↑
MV attribute not allowed
↑

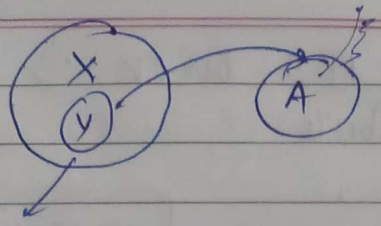
NOTE :- RDBMS is by dyault in 1NF. acc to CODD's rule.

* Second Normal Form (2NF)
Based on fully FD (FFD)

A relation R is in 2NF if
(i) It is in 1NF
(ii) If every non-prime attribute of R is not fully FD on the key of R.
or
no non-prime attribute should be determined by the part of candidate key i.e no partial dependency.

eg : R (ABCD)
(CK) Candidate key = AB
Prime attributes :- A B (attributes of CK)
Non-" " :- C D

AB → C ✓ it is in 2NF as AB is FFD on candidate key AB
AB → D ✓
AB → CD ✓
B → C ✗
A → C ✗ } partial dependency on candidate key.



Y is subset of X
Not possible in 2NF

X = CK
Y = proper subset of CK
A = Non-prime attribute

NOTE:- RHS → CK there will be redundancy for sure.

Q R(ABCD)
FD:- AB → C
B → D
Find all Normal form.

CK ⇒ (AB)⁺ = ABCD ✓
B⁺ = BD ✗
AB is CK

① AB → C This is in 2NF as AB is FFD on CK.

② B → D Partial dependency ∴ not in 2NF
∴ overall relation is not in 2NF.
∴ the given relation is in 1NF

Q R(ABC)
FD: A → B
B → C
find the 2NF?.
A⁺ = ABC → candidate key ✓
B⁺ = BC → " " ✗

for big ques
NOTE RHS पे जो attribute नहीं होगा then it will be of the part of CK ∴ Here CK = A will be part

(A)

① $A \to B$ it is FFD on CK ⇒ 2 NF

A = prime attribute

B, C = Non " ".

if prime attribute ⟶ non-prime attribute (along with FFD) then 2NF

if non " " ⟶ non-prime " "

$B \to C$ then 2NF

∴ the given relation is in 2NF

* R(ABCD)

FD: $AB \to C$

$A \to D$

decompose it into 2 NF

① Find out CK

$AB^+ = ABCD$ it is CK

$A^+ = AD$

② $AB \to C$ LHS is prime attribute

RHS " Non " "

∴ dependent on FFD

∴ it is in 2 NF

③ $A \to D$ Prime attribute ⟶ Non-attribute but not FFD as partial dependency ∴ is not in 2 NF

④ Decompose R(ABCD)

$R_1(ABC)$ $R_2(AD)$

∴ dependency preserved

$R_2(AD)$ $A \rightarrow D$

$A^+ = AD$ Non-prime attri dependent on " " ∴ PFD in 2NF

1. $R(ABCDE)$

FD: $AB \rightarrow C$

$A \rightarrow D$

$B \rightarrow E$

decompose into 2NF

$AB^+ = ABCDE$ (CK)

$A^+ = AD$ ✗

$B^+B = BE$ ✗

(1) $AB \rightarrow C$ prime → non-prime & prime is FFD ∴ 2NF

(2) $A \rightarrow D$ partial ∴ not in 2NF

(3) $B \rightarrow E$ " ∴ " " "

$R(ABCDE) \rightarrow R_1(ABC), R_2(AD), R_3(BE)$

$R_2(AD)$

$A \rightarrow D$

$A^+ = AD$ (CK)

& ∵ prime → non-prime

prime is FFD ∴ it is in 2NF.

Ily for $R_3(BE)$

$B \rightarrow E$ $B^+ = BE$ & as $B \rightarrow E$ (Prime) (Non-prime) FFD ∴ 2NF

* Third Normal form (3NF)

⇒ Real time database is normalized upto 3NF. We leave the database normalized " " or BCNF

imp. A relation schema R is in 3NF if it satisfies

(i) It is in 2NF.

(ii) No non-prime attribute of R is transitively dependent on the key of R.

or A relation is in 3NF if

FD $X \to Y$ satisfies any one of the following condition

any condition true then in 3NF

(i) $X \to Y$ is a trivial FD.
i.e Y is a subset of X.
$\boxed{Y \subseteq X}$

(ii) $X \to Y$ then X is a superkey (SK)
(LHS)
for eg:- $AD \to B$
if superkey then in 3NF

(iii) if $X \to Y$ then $(Y - X)$ is a prime attribute.

NOTE:- $X \to Y$ $Y \to Z$ (transitive)
SK SK SK Non-key ✓ 3NF
is allowed in 3NF

$X \to Y$ $Y \to Z$ $\to X$ 3NF
SK Non-key attribute Non-key attribute

eg:- R(ABCD)
CK ⇒ AD

C → D

(i) X → Y, is trivial X
(iii) (C → D)
X
(iii) D − C = D D is prime attribute as in CK ∴ is in 3NF.

Q R(ABC), the set FD given is A → B
B → C

R is in which NF

A+ = ABC → CK ∴ CK = A
B+ = BC

(i) A → B prime → non-prime with prime FFD ⇒ 2NF

B → C is ~~not~~ in 2NF

→ for 3NF

(i) A → B (i) X trivial
(ii) ✓ ⇒ 3NF

→ B → C (i) C is not subset of B X
(iii) B is superkey X as A ∴ key
(iii) Y − X i.e C − B = C here C is not prime attribute

∴ not in 3NF

but is in 2NF as non-key → non-key

∴ overall relation is in 2NF.

Q R(ABCDE)
FD:- AB → C
B → D
D → E

Decompose it into 3NF

① AB^+ = ABCDE ∴ AB is CK

② AB → C
(a) C is subset of AB
(b) AB is a superkey - 3NF ✓

B → D
(a) Not trivial
(b) B → not superkey
(c) B − B = D, D is not prime
→ X 3NF

D → E
(a) Not trivial
(b) D is not superkey
(c) E − D = E which is not in prime
→ X 3NF

decomposed
R(ABCDE)
R_1(ABC) R_2(BD) R_3(DE)

3NF : LHS → ~~superkey~~

* R(ABC)

FD: { AB → C
C → A

RHS: B ✗

∴ B is part of ck but not ck as $B^+ = B$

$AB^+ = ABC$

Here ck is AB^+

$AB^+ = ABC$

$BC^+ = BCA$

ck = {AB, BC}

prime attributes ABC

① ~~Trivial FD~~

AB → C ~~C ⊆ AB~~

✓ LHS is AB is super key

∴ in 3NF

② C → A

C is not ck but part of ck ∴ 2nd condition ✗

~~A - C~~ = A = prime attribute ✓ ⇒ 3NF

∴ resulting relation is in 3NF.

* create relational schema R(ABCDE) with primary key AB such that R is in 2nd Normal form but not in 3NF.

Key AB

① AB → C 2NF ✓ FFD
3NF ✓

② C → D C is not part of CD
non-key non-key non-prime → NP ∴ in 2NF ✓ but not in 3NF

BCNF (Boyce codd NF)

It is strict version of 3NF or extension of 3NF on strict terms.

A relation R is in 3NF if atleast 1 condition hold

same as 3NF
(i) ~~it is in 3~~, $X \to Y$, $Y \subseteq X$
(ii) $X \to Y$ then X should be a superkey.

Q R(ABC)
FD: {AB → C, C → A}
find whether R is in BCNF or 3NF.

$AB^+ = ABC$ RHS = B
CK = AB, BC ∴ take combination of B
$BC^+ = ABC$

(1) AB → C AB is superkey
3NF ✓
BCNF ✓

(2) C → A A − C = A ⇒ prime ∴
3NF ✓
BCNF ✗

↓ eg.

IMP. * Every BCNF is 3NF but not vice versa

Q R & set of attributes R(ABCDEFGHIJ)
F: AB → C
A → DE
B → F
F → GH
D → IJ

Decompose it into BCNF

AB^+ = ABCDEFGIJ ∴ AB is CK.

(1) AB → C LHS is superkey R_1
(2) A → DE part of CK but not superkey R_2
(3) B → F " " " " " " R_3
(4) F → GH ~~not~~ CK = RH R_4
(5) D → IJ " " " R_5

R_1(ABC) R_2(ADE) R_3(BF) R_4(FGH)
R_5(DIJ)

DMP:- BCNF sometimes violate dependency preservation &
- Because of BCNF decomposition, sometime the original Relation R then CK may not that result in failure of dependency preservation. & hence decomposition is lossy.
- If there is any dependency like (proper subset of CK) → (proper subset of CK) then dependency preservation is violated.
- Let relation R consist of simple CK i.e no compound CK then R is always in 2NF but may or may not be in 3NF or BCNF.

if candidate key = ABC
A → 2NF
B → 2NF
simple CK

- If relation R consist of only prime attribute then R is in 3NF but may (3rd case) or may not be in BCNF. If every attribute is a superkey then R is in BCNF.

→ Relation R with no non-trivial FD then R is in BCNF

NOTE: Binary relation is always in BCNF

→ R(ABCDE)

AB → C
B → D
D → E

Decompose into 3NF

AB^+ = ABCDE is CK = AB

AB → C LHS is superkey ∴ 3NF ✓

B → D partial (NA in 2NF) 1✗ 2✗ 3 D-B=D pairs ✗ ∴ not in ✗ 3NF

D → E Non-prime → Non-prime ∴ 2NF not in 3NF.

Correct Alternate Answer

* B → D B & its closure ✗ 3NF

R_1 (B D E)

B → D ✓ B is superkey ∴ 3NF
D → E D is not " ∴ violating ∴

remove D.

R_{11} (BD)

R_{12} (DE)

* D → E already covered ↑
* CK should not be decomposed ∴ R_2 (ABC)

∴ total no. of Relations in 3NF = 3.

Q Relation R(ABCDEFGHIJ)

FD:- AB → C
AD → GH
BD → EF
A → I
H → J

Decompose into 3NF

RM: ABD not present

$(ABD)^+$ = ABDCGHEFIJ ✓ CK

AB → C LHS is partial ∴

R_1(ABCI)

(AB is its closure)

AB → C
A → I

check for FD :- AB → C A → I

AB is superkey as AB is CK ∴ break

A is partial in K(ABCI)

R_{11}(ABC)

R_{12}(AI)

AD → GH LHS is partial ∴

R_2(ADGHIJ)

transitive:
- AD → GH
- A → I (already covered)
- H → J

as H is not superkey nor J is prime ∴ not in 3NF

R_{21}(ADGH), R_{22}(HJ)

To decompose for 3NF create Relation R (LHS & its closure) for each FD given.

BD → EF

R_3 (BDEF) is in 3NF alone

Now check for CK i.e ABD is it covered in any of the above relation
∴ now consider new R_4 (ABD)

∴ total 5 relation

Decompose into BCNF

R (ABCDEFGHIJ)

AB → C

B → F

A → DE

~~AB → E~~ F → GH

~~AD → I~~ D → IJ

AB^+ = ABCFDEGHIJ ∴ AB is CK

① AB → C A is not superkey it is partial ∴ not in 2NF

CK is covered ∴ directly

~~R (ACDEIJ)~~

R_1 (AC) R_1 (ABC)

∴ no need for its closure

② B → F B is partial ∴ R_2 (B & its closure)

R_2 (BF)(GH)

③ A → DE R_3 (ADE)(IJ)

④ F → GH non-prime → non-prime

~~R_4 (FGH)~~

$R_1(ABC)$ is in 3NF ✓

$R_2(ADEIJ)$

$A \to DE$

$D \to IJ$

part of superkey — RHS is superkey

$A^+ = ADEIJ$

break as alone will be in 3NF but not combined.

$\therefore R_{21}(ADE) \to$ 3NF ✓

$R_{22}(DIJ) \to$ 3NF ✓

$R_3(BFGH)$

$B \to F$ (3NF) B is CK for this relation

$F \to GH$ (x 3NF with combination ✓ ✓ ∴ break)

$\therefore R_{31}(BF)$ $R_{32}(FGH)$

Total relation = 5

$R_1(ABC)$ ∴ in BCNF (A is superkey)

$R_{21}(ADE)$ A is superkey

$R_{22}(DIJ)$ D is "

$R_{31}(BF)$ B " "

$R_{32}(FGH)$ F is "

∴ all in BCNF

* $R(ABCDEFGH)$

$AB \to C$

$AC \to B$

$AD \to E$

$B \to D$

$BC \to A$

$E \to G$

RHS :- H missing F.

∴ CK = candidate key = H will be part of it

Decompose this relation. ?

Q R(ABCDEH)

$A \rightarrow B$

$BC \rightarrow D$

$E \rightarrow C$

$D \rightarrow A$

missing

Q1 3 CK — ABFH
↳ BCFH
↳ ACFH

AB → C violating as it is partial

Partial part of CK ∴ not 3NF.

∴ create R (AB lhs closure)

R_1 (ABCDEG)

AB → C Here AB is CK ∴ break them

AC → B

AD → E

R_{11} (ABC) R_{12} (ACB) R_{13} (ADE) R(BD) R(BH) R(EG)

R(ABFH) R(BCFH) R(ACFH) will be added.

* 4 NF

Multivalued

- The 4th NF
- A relation R is in 4NF if & only if the following condition is satisfied

(i) R is in 3NF or BCNF

(ii) If it contains no multivalued dependencies.

Multivalued dependency
where 1 attribute value is potentially a multivalued fact about another.

for eg:- FD: $\alpha \rightarrow \beta$
$\alpha \rightarrow \beta_1$
$\alpha \rightarrow \beta_2$
$\alpha \twoheadrightarrow \beta$ (α is multivalued fact for β)

A	B	C	D
1	4		
1	3		
2	6		

attribute must be independent of each other.

Person (P)	Mobile (M)	Food like (F)
P_1	M_1	F_1
	M_2	F_2
P_2		F_2

$P \twoheadrightarrow M$
$P \twoheadrightarrow F$

2 table
P & M and P & F depending on multivalued dependency.

consider the following table student with the attributes name, computer & language.

Name (N)	computer (C)	language (L)
Aman	windows / Apple	Eng / Hindi
Mohan	Linux	Eng / spanish

① table

$N \twoheadrightarrow C$: 2 table N, C

$N \twoheadrightarrow L$ ② N, L

* Fifth normal form (PJNF = Projected join normal form) (5NF)

Due to Join dependency

A relation R is in 5NF if following condition

① Relation R is in 4NF

② It can not be further non-lossy decomposed

Rarely found in table.

JOIN DEPENDENCY

Let R be relational schema & $R_1, R_2 \ldots R_n$ be the decomposition of R, R is said to satisfy the join dependency $*(R_1, R_2 \ldots R_n)$ if & only if

(Projection) $\pi(R_1) \bowtie \pi(R_2) \bowtie \pi(R_3) \bowtie \ldots \bowtie \pi(R_n) = R$

if & only if every legal instance $r(R)$ = join of its projection on $R_1, R_2 \ldots R_n$

(A)	(C)	(P)
Agent	Company	Product
Aman	C_1	Pendrive
Aman	C_1	MIC
Aman	C_2	Speaker
Mohan	C_1	Speaker

R_1 R_2

Agent	Company

Agent	Product

$R_1 \bowtie R_2 = R$ then ⇒ Join dependency

R_1

A	C
Aman	C_1
Aman	C_2
Mohan	C_1

R_2

A	P
Aman	Pendrive
Aman	MIC
Aman	Speaker
Mohan	Speaker

R_3

C	P
C_1	Pendrive
C_1	MIC
C_1	Speaker
C_2	Speaker

$\neq (R)$ → original table ∴ NO JD ∴ can't be 2 NF. Can't be decomposed into 5NF.

$R_1 \bowtie R_2$ (Natural join)

A	C	P	
Aman	C_1	pendrive	✓
"	C_1	MIC	✓
"	C_1	Speaker	✗ (extra tuple)
"	C_2	Pendrive	✗
"	C_2	MIC	✗
"	C_2	Speaker	✗
Mohan	C_1	"	✗

$R_1 \bowtie R_2 \neq R$ ∴ No join dependency ∴ further can be decomposed

Q A relation R is in 5NF

Pname	Skill	Job
Aman	DBA	J_1
Mohan	Tester	J_2
Rohan	Programmer	J_3
Sohil	Analyst	J_1

R_1		R_2		R_3	
PName	Skill	PName	Job	Skill	Job
Aman	DBA	Aman	J_1	DBA	J_1
Mohan	Tester	Mohan	J_2	Tester	J_2
Rohan	Programmer	Rohan	J_3	Prog.	J_3
Sohil	Analyst	Sohil	J_1	Analyst	J_1

Ans $R_1 \bowtie R_2 = R$ (it is redundant)

PName	Skill	Job
Aman	DBA	J_1
Mohan	Tester	J_2
Rohan	Prog	J_3
Sohil	Analyst	J_1

$R_1 \bowtie R_2$: it is giving original relation

$(R_1 \bowtie R_2) \bowtie R_3 = R$: " "

∴ Join dependency

∴ not in 5NF

Candidate key.

→ It is a minimal superkey i.e. can uniquely identify tuple & a key is made ⇒ Primary key

To convert it into 5NF break it into 3 Relation i.e R_1, R_2, R_3

* Queries of sessional

(i) flight (flightno, from, to, distance, depart, arrival)
aircraft (aid, aname, cruising range)
certified (eid, aid)
Employees (eid, ename, salary)

(i) $\pi_{eid}(\sigma_{aname \neq 'Boeing'}(Aircraft \bowtie certified))$

(ii) $\pi_{ename}(\sigma_{aname = 'boeing'}(Aircraft \bowtie certified \bowtie empl))$

(iii) $\pi_{aid}(\sigma_{cruisingrange > distance \wedge from = 'Bom' \wedge to = 'Madras'}(flight \bowtie aircraft))$

(iv) $\pi_{flightno}(\sigma_{salary > 1,00,000}(flight \bowtie Aircraft \bowtie certified \bowtie Emp.))$

(v) $\pi_{ename}(R_1, \pi_{eid}(\sigma_{cruisingrange > 3000}(Aircraft \bowtie certified))$

$\pi_{ename}(Employee \bowtie (R_1 - \sigma_{aname = 'Boeing'}(Aircraft \bowtie certified))$

UNIT-IV

Transaction Management

It is a set of operation or instruction = transaction.

money from I to other

A → B

Read (a, A)

a = a - 50

~~write~~

written (A, a)

It is a logical unit of database processing that must be completed in its entirety to ensure correctness.

IMP. A C I D properties of transaction

(1) Atomicity :- either All or None either completed fully or not occur.

(2) consistent : means correctness of database, means database should be correct state.

sum (A, B) $\xrightarrow[\text{execution}]{\text{transaction}}$ sum (A, B)

(3) Isolation : Each & every transaction should be performed in system as if it is alone executed.

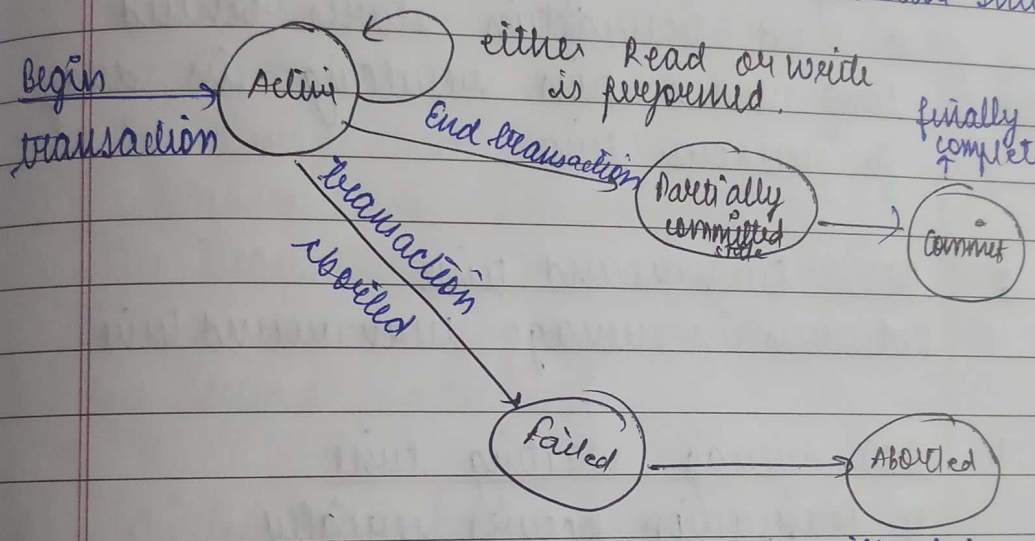
④ Durability:- changes made in system will always be there. (Permanent data)

* states of transaction

5 main states

1) Active state

when transaction start it is in active state.



* Concurrent execution of transaction

Executing more than 1 transaction at same time using interleaving technique.

Advantages of concurrent transaction

1) Increased CPU utilisation

If CPU executes more than 1 transaction at same time then CPU utilization will increase, also we can parallely do both i/o & CPU activity, this increases

average no. of transactions completed in given time.

system [throughput]

2. Better average response time

Interleaved execution of short transaction with a long transaction usually allows a short " to complete quickly In serial execution a short transaction starts behind long transaction resulting in delay in response time.

3. Better Turnaround time
Reduction in average turnaround time.

4. Better averag waiting time
If transaction occurs serially as more " occurs in lesser time
∴ average waiting time reduces.

5. Better disk utilization
overall improvement in resource.

* Problems during concurrent transaction

1) Lost update problem (write - write conflict)
W - W

Scanned by CamScanner

	T_1	T_2
₹ 1000	R(A)	
950	A = A − 50	
		R(A) 1000
		A = A + 100 1100
this update is lost // 950	W(A)	
		W(A) 1100 ↑ write

→ same data item (A)
It occures when two transaction access the same database item & their operations in a way that makes the value of some database item in system.
The Transaction T_1, T_2 both read records & then updated, first update overwritten by second update.

2) Dirty Read [W-R conflict] // temporary update problem.

	T_1	T_2
100	R(A)	
120	A = A + 20	
	W(A)	
		R(A) // 120
		A = A + 10 // 130 updated
		W(A) // 130
	R(B)	This transaction is commit as completed

In some cases if fail then roll back to initial state making A = 100

Dirty read problem.

3) Unrepeatable Read [write - read conflict]

	T_1	T_2	
100	R(A)	~~R(A)~~	
		R(A)	100
		A = A + 200	300
		W(A)	300
		commit ✓	
Inconsistant value of A will be there everytime we read A in T_1,	R(A) ↓	future processing	

4) Incorrect summary problem

	T_1	T_2
		sum = 0
		R(A)
blank ← (interleaved)		sum = sum A
		R(Y)
		sum = sum + Y
	R(Y)	⋮
	Y = Y + 100	⋮ here sum is in processing
	W(Y)	commit ⇒ it will reflect old value of T_1
		∴ inconsistent

conflict]

Schedule

" of transaction is ordering of execution of transaction

$T_i \rightarrow T_j$ means T_j is performed after T_i not implying.

$T_{ix} \rightarrow T_{iy}$

or $T_i(x) \rightarrow T_i(y)$ data items are x, y

schedule is represented 'S' for transaction ion is ordering of transaction in ~~the~~ chronological order

T_1	T_2
R(x)	
	W(x)

$T_1 \rightarrow T_2$

T_1 first start & end.

Total possible ~~transaction~~ schedule for n- transaction $= n!$

Types of schedules

1) Complete Schedule

If in transaction there is end or commit.

A schedule that contain either commit or about for each transaction whose actions are listed in it is called complete. It must contain all the actions of every transactions that appears in it

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
commit	
	Abort

2) serial schedule

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

one after another transactions are written serially

Each serial schedule consist of sequence of instruction from various transaction where ,, belonging to one single 4 appear to that in schedule means transactions are not interleaved from start to finish.

→ lead to consistent state
read

3) Non-serial schedule
operation from set of operations are interleaved.

4) Equivalent schedule

2 schedule S_1 & S_2 are said to be equivalent schedule if they produce the same final database state.

* Result Equivalent Schedule
* Conflict " "

* Result ES :- produces same final database state for same initial value of data.

for eg :- Schedule S_1 & S_2 -

	S_1	S_2
100	R(A)	R(A)
110	A → A + 10	A = A × 1.1
110	W(A)	W(A) 110

* conflict ES : 2 schedules are said to be " " if all the conflicting operations in both the schedule must be executed in same order.

5) Serializable Schedule

Non-serial " that is equivalent to some serial execution of transaction.

* Conflict ES

1) I = Read (Q) } then operation
J = Read (Q) } does not matter

2) I = Read (Q) order does matter
J = Write (Q)

3) I = Write (Q) J = Read (Q) " "

4) I = Write (Q) J = Write (Q) " "

check for conflict ES

S1 $R_1(A), R_2(B), W_1(A), W_2(B)$

S2 $R_2(B), R_1(A), W_2(B), W_1(A)$

writing S_1 & S_2 in tabular form

S_1 Schedule

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

A is not accessed by diff. transaction
Hence B " "

∴ S_1 is conflict equivalent to S_2

$S_1 \equiv S_2$

S_2

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

Q $S_1 = R_1(A), W_1(A), R_2(B), W_2(B), R_1(B)$

$S_2 = R_1(A), W_1(A), R_1(B), R_2(B), W_2(B)$

S_1

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

S_2

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

$\therefore S_1 \subseteq S_2$

→ A schedule S of n transaction is serializable if it equivalent to some serial S of same n transaction

conflict serializable
if it is conflict equivalent to some schedule

S_1

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

S_2

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

S_1 ∴ R(A) in T_1 is before R(A) in T_2
W(A) W(A)

Ily in S_2 ∴ $S_1 \subseteq S_2$

Ily for R(B) in T_1 is before R(B) in T_2
W(B) W(B)

∴ $S_1 \subseteq S_2$

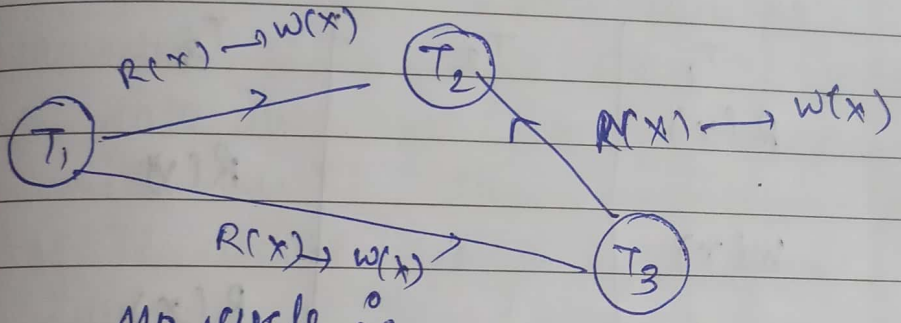
* Precedence graph

1) Directed graph V = Vertices
E = Edges

Algo
- Create a node for each transaction
- A directed edge T_i to T_j if T_j reads value of an item written by T_i
$T_i \rightarrow T_j$ if T_j wants to write a value in item after read by T_i.

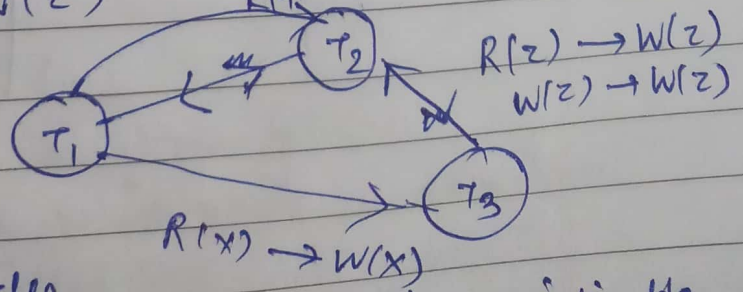
W → R
R → W
W → W

T_1	T_2	T_3
R(x)		
	R(z)	
R(z)		
		R(x)
		R(y)
		W(x)
	R(y)	
	W(z)	
	W(x)	



no cycle ∴ it is conflict serializable.

T_1	T_2	T_3
R(x)		
		R(z)
		W(z)
	R(y)	
R(y)		
	W(y)	
W(x)		
W(z)		R(y) → W(y)
W(y)		



2 cycles

∴ it is not conflict serializable.

check for conflict serializability

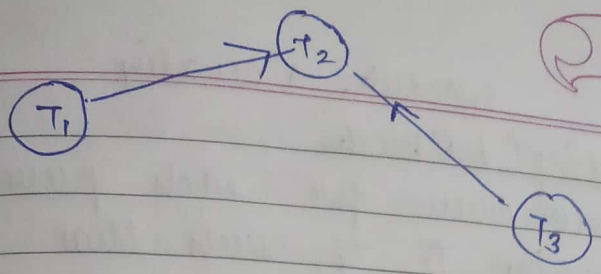
S_1	S_2
$R(x)$	$R_3(x)$, $R_2(x)$, $W_3(x)$, $R_1(x)$,
$R_3(x)$	$W_1(x)$
$W_1(x)$	
$R_3(x)$	
$W_2(x)$	

S_1

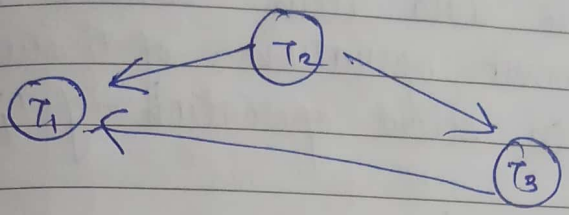
T_1	T_2	T_3
$R(x)$		
		$R(x)$
$W(x)$		
		$R(x)$
	$W(x)$	

S_2

T_1	T_2	T_3
		$R(x)$
	$R(x)$	
		$W(x)$
$R(x)$		
$W(x)$		



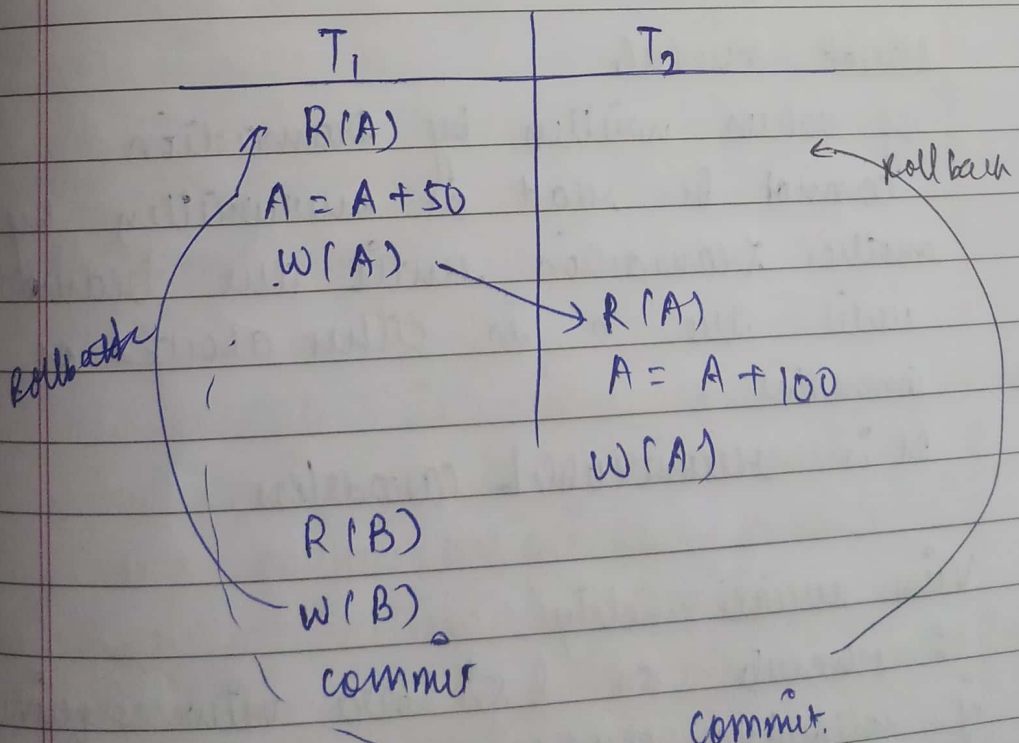
No cycle



No cycle

* Recoverable schedule

It is the one in which each pair transaction T_i, T_j such that T_j reads a data item that was previously written by T_i, then commit operation of T_i should appear before commit operation of T_j.

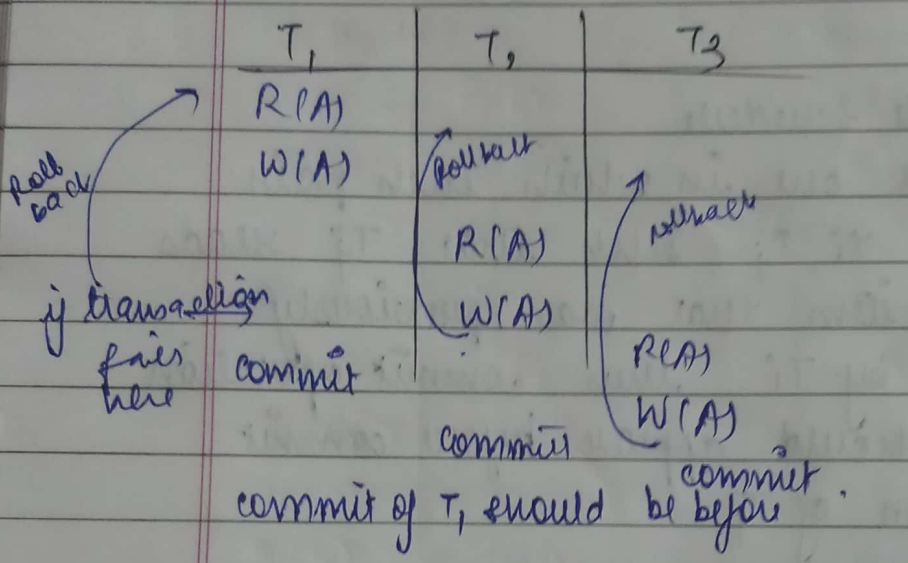


→ The transaction which write the data item will be committed first like here T_1.

Schedule → cascading

cascading ^ Rollback

is one when for each pair of transaction T_i, T_j, such that T_j reads the data item written by T_i then commit operation of T_i should appear after read operation of T_j



commit of T_i should be before commit.

Strict schedule

If values written by transaction cannot be read or overwritten by another transaction until the transaction until the it is either aborted or committed.

It is recoverable & cascadeless.

View serializibility

2 schedule S_1 & S_1' are view equivalent if following condition hold.

for each data item Q.

If Ti reads initial value of Q in schedule S then Ti in S' also reads initial value in Q.
If Ti executes read Q in S & that value was produced by Tj then Ti in schedule S' also reads value of Q that was produced by Tj
→ for each data item Q the transaction perform final write Q in schedule S should also perform " " Q in " S'

S

	Ti	Tj
Q (1)	R(Q)	
(2)	W(Q)	R(Q)
(3)		W(Q)

If final write performed by Ti then also Ti will perform

S'

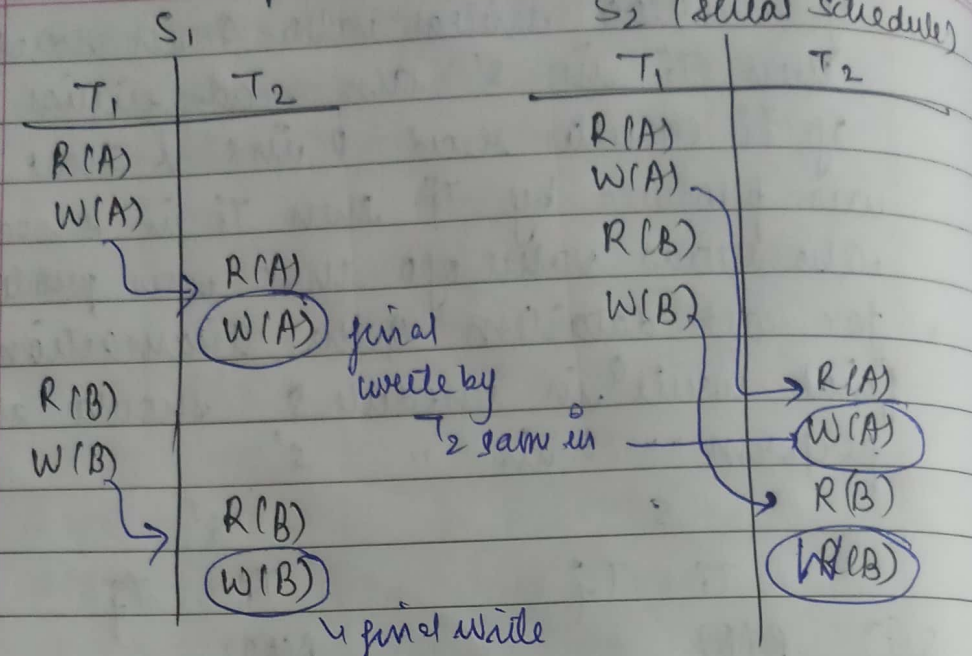
	Ti	Tj
	R(Q)	
	W(Q)	R(Q)
	W(Q)	

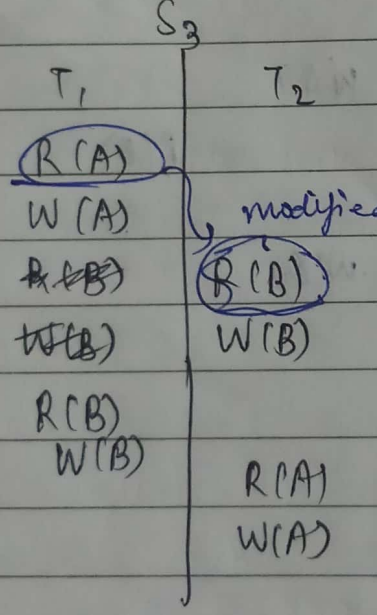
- A schedule is view serializable if it is view equivalent to a serial schedule.
- Every conflict serializable schedule is view equivalent & hence it is view serializable but vice-versa is not true.

conflict ser ⊂ View ser.

Data item are A, B

View equivalent





S_1 & S_3 are not view equivalent.

Here initial data item is read by

R(A) → R(B) } not same

while in S_1

R(A) → R(A)

$S_1 \neq S_3$

* Concurrency control.

It is process of managing the simultaneous execution of transaction in a shared database to ensure the serializibity of transaction.

Purpose

1) To enforce isolation (only 1 transaction executes not interfered)
2) To preserve database consistency
3) To resolve read-write or write-write conflicting operation)

Protocol

① Lock based protocol

Lock → operation → unlock data item

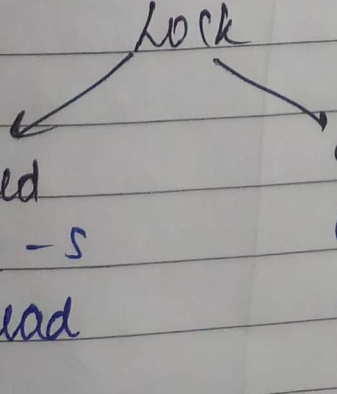
Lock S (Shared) → (Read)
Lock X (Exclusive) → (write)

eg:
(data item) A

A lock guarantees exclusive use of data item to a current transaction (only 1)

Steps

① to access data item → lock is required
② the lock is released.
(all data items are accessed in mutual exclusive)



Lock compatibility matrix

current transaction \ for coming transaction	S	X
S	T	F
X	F	F

SS → T
SX → F
XS → F } transaction waits for the first transaction to get unlock.
XX → F

1) Request for lock goes to concurrency control manager. → whether lock is granted if compatible
   → or wait for the lock (deny)

T_1	T_2
lock-X(B)	
R(B)	
B = B - 350	
W(B)	
unlock(B)	
lock-X(A)	
R(A)	
A = A + 30	
W(A)	
unlock(A)	
	lock-S(A) //shared mode
	R(A)
	unlock(A)
	lock S(B)

read B
unlock(B)
display(A+B)

as T_2 only in shared mode ∴ only read operation can be performed.

* conversion of lock.

shared mode :- 100's of transaction
exclusive " → only 1 "

→ conversion

(1) UPGRADING
shared → Exclusive
S → X

(2) DOWNGRADING
Exclusive → shared
X → S
so that other transaction can also use them
otherwise in exclusive mode only for 1 transaction

* implementation of lock

(1) Transaction → request is send to concurrency control manager CCM → Shared / Exclusive

(2) CCM reverts back either in granting lock or wait for lock (deny)

(3) If data item received it will unlock it.
To manage all the locks in table in all modes ⇒ Lock table

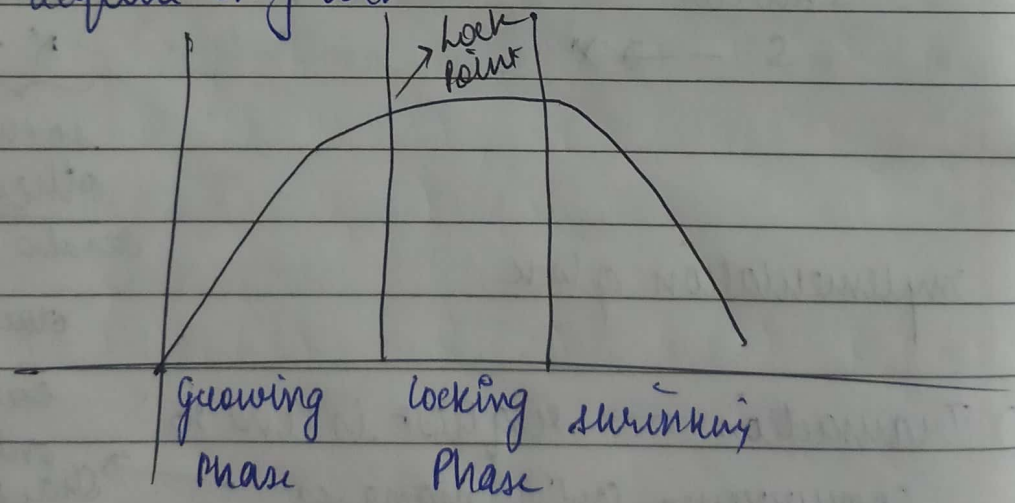
(2) * Two Phase locking Protocol

1) Growing Phase – locks are acquired
2) shrinking phase – " " released.

* assumes that the transaction can be only in one of two phases i.e locking & unlocking is done in two phases → growing
→ shrinking

Growing / Expanding phase
Transaction can only acquire lock but cannot release any lock.

shrinking Phase
Transaction release lock but can't acquire any lock



The transaction reaches a point where all locks it need are acquired

- enforces serializability

(1) Holding the lock unnecessarily in growing phase ⇒ penality to other requirement.

Variation of 2 phase locking

1) strict 2 phase locking protocol
2) Rigorous " " " "
3) static " " " "

↳ the lock which we have acquired will not release lock until transaction committed. ⇒ cascading rollback will not be there.

(2) will not release any type of lock (S, X) until transaction is committed. adv:- No cascading rollback.

(3) Execute when all the locks are acquired

adv (1) No cascading rollback
(2) Deadlock free — No deadlock in system

BINARY LOCKS

0 1

⇒ If lock A is equal to 1 lock(A) = 1 then lock granted
lock(A) = 0 not

Binary lock

It has 2 states locked & unlocked

if lock (A) = 1 then item A cannot be accessed & transaction is forced to wait

else if lock (A) = 0 then item A can be accessed.

disadv It is easy to implement but restricted to yield optimal concurrency condition means database will not allow 2 transaction to read same database even if neither of transaction updates database

① Given 3 schedule

$S_3 \Rightarrow R_1(X), R_2(Z), R_1(Z), R_3(X), R_3(Y), W_1(X), C_1, W_3(Y), C_3, R_2(Y), W_2(Z), W_2(Y), C_2$

$S_4 \Rightarrow R_1(X), R_2(Z), R_1(Z), R_3(X), R_3(Y), W_1(X), W_3(Y), R_2(Y), W_2(Z), W_2(Y), C_1, C_2, C_3$

$S_5 \Rightarrow R_1(X), R_2(Z), R_3(X), R_1(Z), R_2(Y), R_3(Y), W_1(X), W_1(X), W_2(Y), C_3, C_2$

S_3

T_1	T_2	T_3
$R_1(X)$		
	$R_2(Z)$	
$R_1(Z)$		
		$R_3(X)$
		$R_3(Y)$
$W_1(X)$		
C_1		
	~~$W_2(Z)$~~	$W_3(Y)$
		C_3
	$R_2(Y)$	
	$W_2(Z)$	
	$W_2(Y)$	
	C_2	

* Graph based protocol

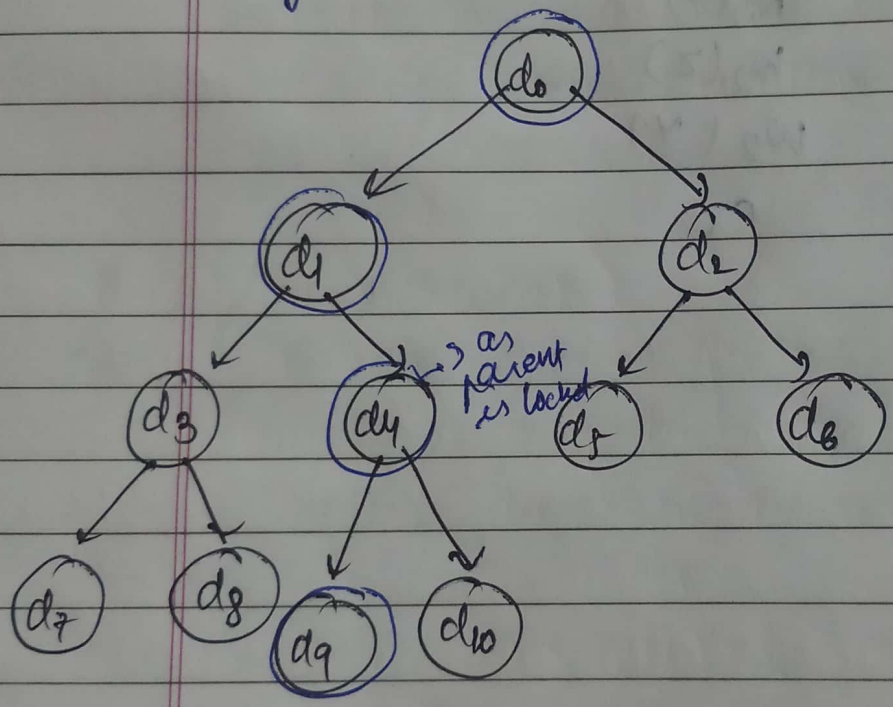
⇓

tree protocol → all are conflict serializable schedule

① Rule 1st lock :- can be on any data item
Parent lock → then data base can be lock otherwise can't.

②

③ We will unlock data item any time.
once data item locked can't be unlocked anytime.



We have 3 transaction T_1, T_2, T_3

T_1 needs locks on

T_1 : d_0, d_1, d_4, d_9

T_2 : d_1, d_4

T_3 : d_3, d_7, d_8.

can lock d_0

T_1	T_2	T_3
lock (d_0)		
lock (d_1)		
lock (d_4)		lock (d_3)
lock (d_1)		lock (d_7)
lock (d_9)	lock (d_1)	lock (d_8)
unlock (d_4)	:	Unlock (d_3)
	lock (d_7)	

Advantage

1. Concurrency can ↑ as ~~the~~ shorter wait
2. deadlock free, no roll back required.

Disadv.

* Time stamp protocol (T_S)

⇓ as time stamp → to indicate which transaction came first, which came after that

① We use counter
② use system time

counter

1	T_1	9:00	T_1 T_5
	T_2		1 5
	T_3		$T_S(T_1) < T_S(T_5)$
	T_4		Old time can be
5	T_5	9:05	seen

older time, younger time

Date
Page

Time stamp values

- W time stamp → latest time when written on transaction
- read time stamp i.e R - time stamp
  ↳ सबसे बाद मे किसने read किया

  latest read stored here

Time stamp ordering protocol

(1) Ti issues read

$T_s(t_i) < W(Q)$

→ it is oldest transaction, it is writting that value that is already been written by transaction

~~Ts(ti)~~

→ (2) $T_s(t_i) > W(Q)$

[It can read

time at which this transaction is reading the transaction.

→ Ti issue write(Q)

$T_s(t_i) \geq W(Q)$ & $T_s(T_i) \geq R(Q)$.

$T_s(t_i)$ becomes new value of $W(Q)$

$T_s(t_i) = W(Q)$

$T_s(t_i)$	$W(Q)$
9:05	9:00
	$R(Q)$ 9:03

then it can modify the value & write the value then value of $W(Q) = T_s(t_i)$

otherwise
$T_s(t_i) < R(Q)$
- no updation allowed

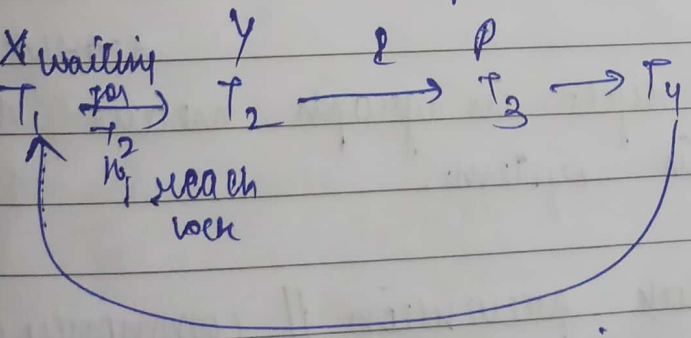
if $T_s(t_i) > R(Q)$
$R(Q) \leq T_s(t_i) \leq W(Q)$
- no updation of value of Q is allowed

Thomas write rule
Modification of time stamp ordering protocol.

1. $T_s(T_i) < R(Q)$
It will not allow to write & the transaction is rolled back i.e T_i is rolled back

2. $T_s(T_i) < W(Q)$
T_i attempting to write in obsolete write

Deadlock:
If there is existing such transaction



Here all transaction wait where no transaction can execute.

① Detect deadlock
② Recover deadlock

① Deadlock prevention
② Deadlock detection & Recovery

~~Weighted wait graph~~ wait-for graph
$T_i \rightarrow T_j$ is directed edge when transaction T_i is waiting for source held by resource T_j

Transaction | Data item | Lock Mode | grant

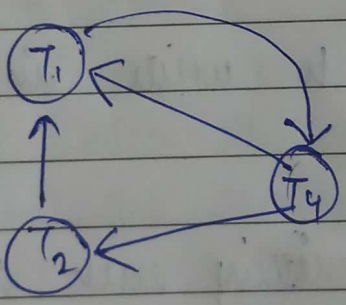
T_1 — Q — S — ✓
T_2 — Q — X — ✓
T_3 — Q — ~~X~~ S — No
T_4 — Q — ~~X~~ — X
T_1 — Q — X — X

XX → incompatible

compatible

check for incompatible mode as incompatible
Q → S
Q → X

Ex.



2 cycles in graph
∴ there is deadlock

If cycle in graph means deadlock in system.

* Deadlock prevention // commonly used when deadlock occurs ~~no~~ again & again
It ensures that system ~~does~~ is not in deadlock

4 things should not occur:

1) Mutual exclusion.
2) Hold & Wait
   Each transaction hold data & then wait for next "
3) No preemption
   1 transaction is executing if high priority " come it will abort low
   " " & give data item to high tr
4) Circular wait
   All waiting for next transaction.

DEAD LOCK PREVENTION METHODS

1) True Protocol
2) Each Transaction lock data item : no deadlock
3) Preemption & transaction Rollback
   will roll back high order transaction & execute itself.

→ Timestamp use in deadlock prevention

1) Wait - die scheme

$$T_s(T_i) < T_s(T_j)$$

T_i is older than T_j.

↳ T_i is allowed to wait for T_j to release lock

but if $T_s(T_j) < T_s(T_i)$

then T_i is not allowed to wait.

~~T_i aborts~~

T_i abort

(2) Wound-wait scheme

$T_s(T_i) < T_s(T_j)$ (T_i is older than T_j)

above T_j it will roll back to initial state but time stamp of transaction T_j will remain same so that everytime same transaction does not roll back

* Recovery from deadlock

System recovers from deadlock by breaking one or more transaction

1) selection of victim.

the transaction which involves min cost-

(1) How long transaction has been executed

T_1 (6)	T_2 (4)	T_3 (10)	T_4 (2)
10	20	5	40

(2) How many data items (data item used)

(3) Make data item are needed.

(4) How many transaction are involved in rollback.

2) Roll back

- Partial: roll back to initial point move to that state cause deadlock break then rollback
- Full

DBMS 7th

* starvation
* granularity of locking

fine grain locking : we lock only that data item on which we want lock
coarse " "

Failure classification
- Transaction failure
  (a) logical error → bad i/p
  (b) system error → deadlock
- system crash
  Hardware malfunction
  Harddisk error
- Software failures
- Network failures net off can't reserve ticket
- Natural disasters
  earthquakes etc. data loss.
- sabotage
  internal corruption of data.
  Hacking

What is affected
Volatile storage :- during system crash like in main memory
Non-volatile storage - not lost
Stable storage - magnetic disk data.

Phantom Read problem

T_1	T_2
R(A)	
	R(A)
Delete (A)	
	R(A) ↑

trying to read ← Phantom Read
that value that doesn't exist.
Unrepeatable read → value exist.

* Transaction Recovery Procedure

1) Deferred database Modification → changes in local buffer & when committed → stored in hard disk.
2) Immediate database Modification
   ↳ UNDO / NO REDO

at every step changes are made in disk

⇒ NO UNDO / REDO
no need to reach initial state.

UNDO: changing old value.

REDO: changing the new value.

Start Recovery
< T_i, commit > // add to redo list → as transaction is already committed
< T_i > // add to undo list

- Require current shadow page access than log record
- Maintain 2 page table

Advantages of shadow paging.

- log record overhead is removed.
- recovery is faster as no undo & redo

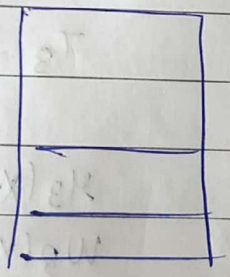
Disadvantages of shadow paging

- for single transaction we are accessing actual data block, current page table, disk
- commit overhead absent.

Data fragmentation

Data divides its fragment.

Garbage collection



modified data ✓
unmodified is also present
↓ garbage.

A) ARIES

full journ + work

TUT Which of the following schedule is (conflict) serializable? For each serializable schedule, determine the equivalent serial schedule?

Three transactions T_1, T_2, T_3, they are executed concurrently & produce a schedule S. S is serializable if it can be equivalent as at least one serial schedule ($T_1 \sim T_2 \sim T_3$ or $T_1 \sim T_3 \sim T_2$ or $T_2 \sim T_1 \sim T_3$ or $T_2 \sim T_3 \sim T_1$ or $T_3 \sim T_1 \sim T_2$ or $T_3 \sim T_2 \sim T_1$)

(a) $r_1(x)$; $r_3(x)$; $w_1(x)$; $r_2(x)$; $w_3(x)$

T_1	T_2	T_3
$r_1(x)$		
		$r_3(x)$
$w_1(x)$		
	$r_2(x)$	
		$w_3(x)$

⇒ This schedule is not serializable because T_1 reads x ($r_1(x)$) before T_3 but T_3 reads x before T_1 writes x ($w_1(x)$) where x is data item
The operation $r_2(x)$ of T_2 does not affect schedule at all so its position is irrelevant As $w_1(x)$ comes after $r_3(x)$
∴ not possible.

(b) $r_1(x)$; $r_3(x)$; $w_3(x)$; $w_1(x)$; $r_2(x)$

T_1	T_2	T_3
$r_1(x)$		
		$r_3(x)$
		$w_3(x)$
$w_1(x)$		
	$r_2(x)$	

This schedule is not serializable
T_1 reads x before T_3 but T_3 writes x before T_1. The operation $r_2(x)$ of T_2 does not affect the schedule, so it's position is irrelevant. Here, $r_3(x)$ & $w_3(x)$ must come after $w_1(x)$ to make correct one.

(c) $R_3(X)$; $R_2(X)$; $W_3(X)$; $R_1(X)$; $W_1(X)$

T_1	T_2	T_3
		$R_3(X)$
	$R_2(X)$	
		$W_3(X)$
$R_1(X)$		
$W_1(X)$		

This is a serializable schedule as all conflicting operations of T_3 happens before T_1, T_2 has only one operation which does not conflict with other operation ($T_2 \rightarrow T_3 \rightarrow T_1$) serial schedule

(d) $R_3(X)$; $R_2(X)$; $R_1(X)$; $W_3(X)$; $W_1(X)$

T_1	T_2	T_3
		$R_3(X)$
	$R_2(X)$	
$R_1(X)$		
		$W_3(X)$
$W_1(X)$		

This is not serializable schedule as
- T_3 reads X before T_1 & T_1 reads X before T_3

but for serial schedule $R_1(X)$ will happen after $W_3(X)$

Draw serializibility (precedence graph) for S_1 & S_2 & state whether each schedule is serializable or not

T_1 : $r_1(x), r_1(z), w_1(x)$

T_2 : $r_2(z); r_2(y); w_2(z); w_2(y)$

T_3 : $r_3(x); r_3(y); w_3(y)$

S_1 : $r_1(x); r_2(z); r_1(x); r_3(x); r_3(y); w_1(x); w_3(y); r_2(y); w_2(z); w_2(y)$

S_2 : $r_1(x), r_2(z), r_3(x), r_1(z), r_2(y), r_3(y); w_1(x); w_2(z); w_3(z); w_2(y)$

T_1	T_2	T_3
$r_1(x)$		
	~~$r_2(z)$~~	$r_2(z)$
$r_1(x)$		
		$r_3(x)$
		$r_3(y)$
$w_1(x)$		
		$w_3(y)$
	$r_2(y)$	
	$w_2(z)$	
	$w_2(y)$	

* Database recovery

(i) Log based recovery can bring to previous state before starting transaction.

(a) start log record
<Ti, start> i.e transaction started

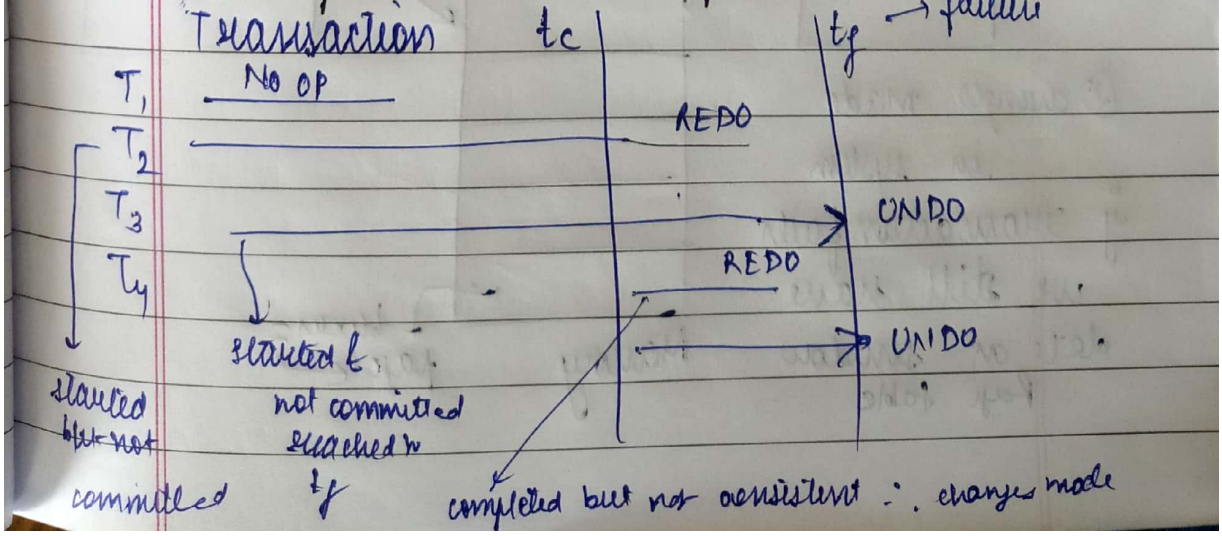
(b) update log record
<Ti, x_j, V_1, V_2> → after value
↓ old value before write
data item to be modified

(c) commit log record
<Ti, commit>
if fail <Ti, abort>.

eg: <T_1, start>
<T_1, x_2, 10, 15>
<T_1, commit>

Write ahead log strategy
log is updated before changes are made in database.
↳ transaction check point t_c

→ check points or save points



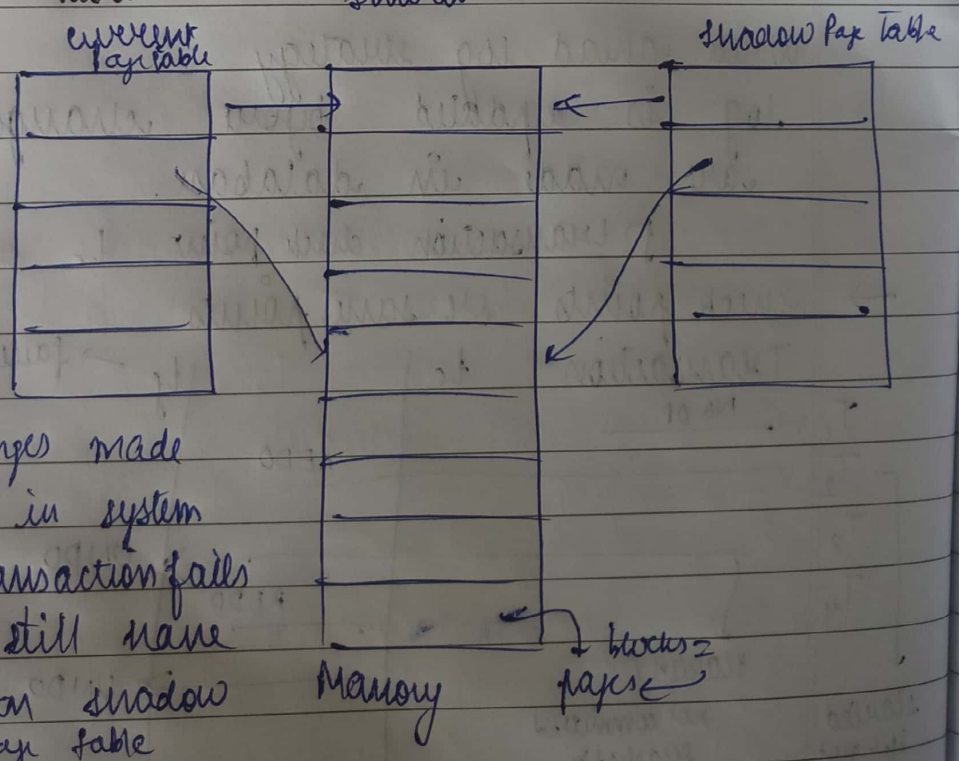
Point of synchronization b/w transaction & log file check point.

check pointing.

1) o/p on stable storage
   - all log records currently residing in main memory
2) o/p on disk
   - changes on buffer block
3) o/p on stable storage
   - log record on check point

(iii) Shadow ~~paging~~ Paging

(1) ~~Shadow~~ page current page table copied to shared "



(2) changes made in system if transaction fails we still have data on shadow Page table